

SanDisk
Secure Digital (SD) Audio
Software Development Tool Kit
(SDDK-07)

User's Guide



SanDisk Corporation
140 Caspian Court
Sunnyvale, CA 94089
TEL: 408-542-0500 FAX: 408-542-0503
URL: <http://www.sandisk.com>

SanDisk® Corporation general policy does not recommend the use of its products in life support applications where in a failure or malfunction of the product may directly threaten life or injury. Per SanDisk Terms and Conditions of Sale, the user of SanDisk products in life support applications assumes all risk of such use and indemnifies SanDisk against all damages.

The information in this document is subject to change without notice.

SanDisk Corporation shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

All parts of SanDisk documentation are protected by copyright law and all rights are reserved. This documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from SanDisk Corporation.

SanDisk and the SanDisk logo are registered trademarks of SanDisk Corporation.

Product names mentioned herein are for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

© 2001 SanDisk Corporation. All rights reserved.

SanDisk products are covered or licensed under one or more of the following U.S. Patent Nos. 5,070,032; 5,095,344; 5,168,465; 5,172,338; 5,198,380; 5,200,959; 5,268,318; 5,268,870; 5,272,669; 5,418,752; 5,602,987. Other U.S. and foreign patents awarded and pending.

Lit. No. 80-36-00144 Rev. 1 2001

Printed in U.S.A.

Revision History

- Revision 1 – initial release.*

Table of Contents

1.0	Overview.....	5
2.0	Introduction.....	6
2.1	Components.....	6
3.0	Source Code.....	8
4.0	SanDisk SD-Audio SDK Applications Programming Interface (API).....	9
4.1	Introduction.....	9
4.2	SDK APIs For Supporting Portable Devices.....	10
4.2.1	SdInit_AudioSystem.....	10
4.2.2	SdMountAudio.....	11
4.2.3	SdUMountAudio.....	12
4.2.4	SdEjectCard.....	13
4.2.5	SdGetPlayListCount.....	14
4.2.6	SdGetPlayLists.....	15
4.2.7	SdGetTrackTitle.....	16
4.2.8	SdGetTrackInfo.....	17
4.2.9	SdOpenTrack.....	18
4.2.10	SdPlayTrack.....	20
4.2.11	SdForward.....	21
4.2.12	SdNextTrack.....	22
4.2.13	SdStopPlay.....	23
4.2.14	SdPauseTrack.....	24
4.2.15	SdResetPlayList.....	25
5.0	Sample Playback Program.....	26
5.1	Overview.....	26
5.2	Initialization.....	26
5.3	Retrieval of Play Lists and Track Titles.....	27
5.4	Retrieval of Specific Track Information.....	27
5.5	Selecting a Track.....	27
5.6	Playing a Track.....	27

SanDisk SD-Audio (Playback) Software
Development Tool Kit User's Guide

1.0 Overview

The Secure Digital (SD) memory card leverages the benefits of the MultiMediaCard, and provides additional features that facilitate copy protection for licensed material such as digital music and electronic books. In addition to MultiMediaCard command functionality that allows the SD Memory Card to operate as a high speed MultiMediaCard, a sophisticated architecture ensures that content can only be accessed by licensed software that has intimate knowledge of this new security scheme.

At the heart of this approach are levels of protection that draw upon licensed keys that are used as part of an authentication process and encryption strategy. A protected area of the card, also referred to as the authentication area, is only made visible after the accessing software is given clearance to do so. Gaining access is accomplished through an "authentication key exchange" (AKE). Once "authenticated," the software uses special SD Memory Card commands to read and write the authentication area. This new set of commands, which supports authentication and protected area access, has been added "behind" the MultiMediaCard standard.

When the SD memory card contents are viewed by a host operating system, only the normal user data area is visible. The card looks just like a MultiMediaCard, or any other block device that uses a FAT file system. Applications can freely read or write files in this area. When the SD Memory Card is used to protect content such as copyrighted music, the data is encrypted and stored in files visible in this area. Encrypted "Title Keys" and Copy Control Information (CCI) associated with each file are stored in the authentication area of the card.

Storage formats are application-specific. The SD-Audio standard describes how audio data is treated by the SD Memory Card. Per this specification, Advanced Audio Codec (AAC) data streams are viewed as multiple audio frames that are encrypted using the title key. Encrypted files are stored in the visible portion of the SD Memory Card using a naming convention that associates each file with its title key. Title keys are tracked and stored within a designated file that is housed in the authentication area. The structure of this file, known as the Title Key Manager, contains elements that include title key entries and other control information. AAC is one of the three supported CODECs. MP3 and MS Audio are also supported in SD Audio.

Given the scope of these security features, it's not surprising that the software that contributes to this overall strategy is extremely complex. Multiple layers are required including: a low-level SD Memory Card device driver, a FAT file system, encryption/decryption functions, and a high-level module which uses intimate knowledge of SD-Audio internals to navigate all portions of this elaborate standard.

The SanDisk SD-Audio Software Development Kit (SDK) contains all of the above software infrastructure and provides an Applications Program Interface (API) that easily integrates with host platforms. With the SD-Audio SDK, the task of supporting the SD Memory Card becomes almost trivial. All of the complexity of SD-Audio and the associated security aspects of the card are encapsulated within this portable package. For music players, tasks such as retrieving song titles and playing encrypted music files are handled with ease using the provided APIs.

2.0 Introduction

The SanDisk SD-Audio SDK contains everything developers need to integrate a host platform such as a music player with an SD Memory Card. Tailoring a few of the SDK configuration options and calling the appropriate SDK API functions are all that is necessary to enable a music player to retrieve song titles and initiate playing music files. The following sections provide an outline of the SD-Audio SDK components, a listing of the source files, and a description of the actual API functions that can be called from within a music player application.

2.1 Components

The SanDisk SD-Audio SDK includes the following components:

SD-Audio Software Layer: This layer contains intimate knowledge of the SD-Audio standard. It formulates requisite security key values, parses related files in the user and authentication areas, and handles all aspects of decrypting/playing protected content. It carries out security-related processing by calling into the SanDisk Security Manager (SSM) software.

Applications Program Interface (API): All functionality required by a typical music player is easily achieved through this API library. These functions include the ability to: mount/unmount volumes; retrieve play lists and track titles; play/stop/pause tracks; and skip forwards/backwards within a given musical track.

Sample Playback Program: This sample application, written in Visual C++ , demonstrates how the SD-Audio SDK services can be easily incorporated into a music player application. It provides a sample GUI via which audio content can be viewed and played.

Figure 2-1 shows how these modules communicate with each other, depicting a high level architectural view of the SD-Audio SDK and its interface with the SDDK-06 and SDDK-05.

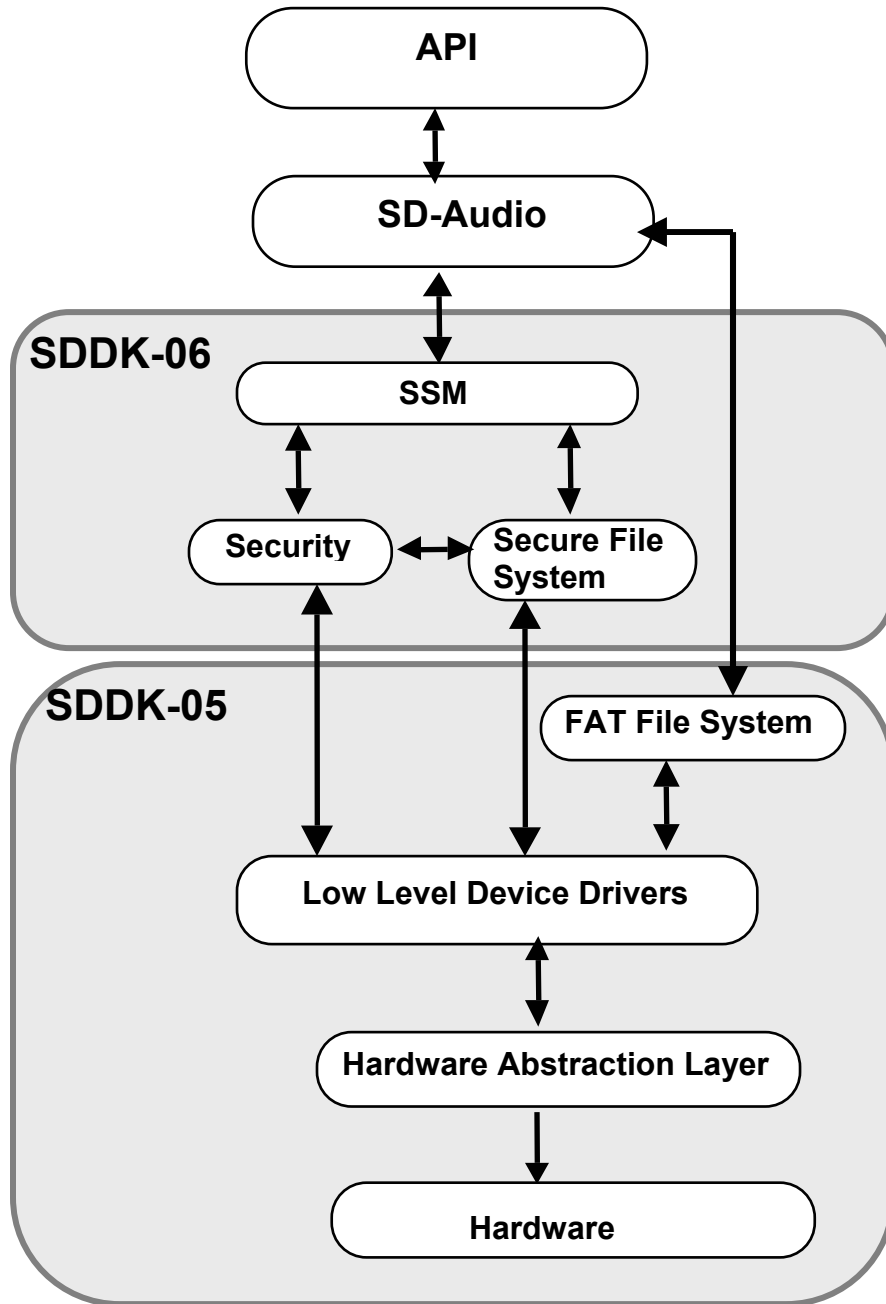


Figure 2-1 SD-Audio SDK Architecture Design

3.0 Source Code

Complete source code is provided for all of the SDK components and the sample application. What follows, are descriptions of some of the most important source files.

Sample Playback Program

SDAUDIO.CPP	Starts up application and displays main music player dialog box
SDAUDIOD.CPP	Music player GUI and function handlers
SDAUDIO.H	Definitions for music player dialog box class

SD-Audio Software Layer

SDAUDAPI.C	Provides APIs such as sdGetPlayLists, SdPlayTrack, etc.
SD_PLM.H	Play list manager structures
SD_TKM.H	Track related structures
SDAUDAPI.H	Prototypes for API functions and definitions for associated data structures
SDAPI.H	Prototypes for FAT file system API functions
SDAUDIO.H	Structures and prototypes for calls to SSM
SSMAPI.H	SSM function code definitions
SD_TYPES.H	Defined data types for the SanDisk SD-Audio SDK

4.0 SanDisk SD-Audio SDK Applications Programming Interface (API)

4.1 Introduction

The use of APIs shields host application software from the complexity of SD-Audio. Music player software which controls the display and playing of play lists can accomplish these tasks by calling the API functions. There's never any need to deal directly with the internals of the SD-Audio files and data structures. To illustrate this, the requirements to retrieve all of the SD-Audio play lists for display are provided below. The music player software calls API functions as follows:

```
/*Sample host music player software*/

#include sdaudapi.h

    UINT8        m_nDrives;
    mediaHandle  m_mediaHandle;
    aud_dev      m_auddev;
    char         m_mediatitle[256];
    pList        m_plist;
    UINT8        m_playlists;
    UINT8        listcount;
    int          i;

/*initialize system (also determines number of SD Memory Cards attached to system)*/
    SdInitAudioSystem(&m_nDrives);
    m_auddev.security = NO_SECURITY;

/*typically only one card - use first one*/
    m_auddev.drivenum = 0;

/*" mount" SD memory card and retrieve default play list information */
    m_mediaHandle = SdMountAudio(&m_auddev, m_mediatitle); /*mount

/*retrieve the number of play lists on the SD memory card*/

    SdGetPlayListCount(m_mediaHandle, &m_playlists);
    m_plist.index = 0;

/*retrieve play list information for all play lists present on the card*/

    for(i = 0; i < m_playlists; i++)
    {
        m_plist.index = i;
        listcount = 1;
        SdGetPlayLists(m_mediaHandle, &m_plist, i, &listcount);
    }

/*all playlists have now been retrieved. Host application can call upon platform-specific
functions to present the lists to the end user*/
```

Notice that in this example, only four SD-Audio SDK API functions are required to initialize the API services and retrieve the play list information. Within the SDK, the SD-Audio software layer calls upon the built-in FAT File System (provided in SDDK-05) to pull the appropriate play list information from the play list manager file, SD-AUDIO.PLM, which resides in the user area of the SD Memory Card. Similarly, the track manager file SD-AUDIO.TKM will be accessed to retrieve track information when API functions such as SdGetTrackTitle are called.

When the actual playing of musical tracks is requested via the SdPlayTrack API function, the SD-Audio SDK also performs all of the required functions. A great number of complex steps take place transparently as follows: Using the title index taken from SD-AUDIO.TKM within the user area of the card, logic that carries out the API function calls upon its protected-area FAT file system to retrieve the associated encrypted title key from the AOBSA1.KEY file and appropriately decrypts the title key. The actual encrypted musical data is read from the related AOBnnn.SA1 file in the user area. Finally, based on the particular audio file format (AAC, MP3, or WMA) the data is decrypted and played for the user.

The above discussion demonstrated how API functions can be integrated into host software to perform the bulk of all SD-Audio tasks. The section below lists and describes each SD-Audio API function. Based on the previous code sample and the descriptions provided for each function, the user should have enough information to begin coding his/her music player application. To further assist with this process, the complete source code for a Sample Playback Program is provided with the SD-Audio SDK. This program demonstrates all aspects of using the API functions, such as the parameter passing and the order in which the functions should be used within the music player software. Chapter 5 provides a functional overview of the Sample Playback Program.

4.2 SDK APIs For Supporting Portable Devices

4.2.1 SdInit_AudioSystem

UINT SdInit_AudioSystem(int * ADrives)

Description:

- Initialize audio system
- Return number of drives available

Input:

Pointer to audio driver attribute

Output:

ADrives contain the number of available drives

0: // success

1: // No drive found

2: // initialization failed

4.2.2 *SdMountAudio*

UINT *SdMountAudio*(AD * ah, char * playlist_title)

Description:

Mount SD Audio System

Input:

Pointer to the Audio Device Structure

```
AD struct {  
    char    security;           // 0 no security  
                                // 1 CPRM  
    char    DriverNum;         // drive number  
};
```

Pointer to array to hold play list title

Char playlist_title[256];

Output:

ah (Audio Handler) structure stuffed with audio system information

0: // success
1: // fail in audio system
2: // file system fail

playlist_title filled in with first play list found on media

4.2.3 *SdUMountAudio*

*mediaHandle SdUMountAudio(AD * ah)*

Description:

Disconnect SD Audio System

Input:

Pointer to the Audio Device Structure

```
AD struct {  
    char    security;           // 0 no security  
                                // 1 CPRM  
    char    DriverNum;        // drive number  
};
```

Output:

mediaHandle returned

4.2.4 *SdEjectCard*

UINT SdEjectCard(AD * ah)

Description:

Remove the SD Card from the Portable Device

Input:

Pointer to the Audio Device Structure

```
AD struct {  
    char    security;           // 0 no security  
                                   // 1 CPRM  
    char    DriverNum;         // drive number  
};
```

Output:

```
call UnMountAudio  
0:    // done
```

4.2.5 *SdGetPlayListCount*

UINT *SdGetPlayListCount(mediaHandle m_mediaHandle, UINT8 * PListCount)*

Description:

Get number of play lists available in the SD Media

Input:

m_mediaHandle // handle
PlistCount // pointer to the number of play lists available on the SD Media

Output:

Number of play lists stored in PlistCount

Return status:

- 0: // success
- 1: // invalid audio handler
- 2: // can not get play list count

Details:

1. read SD_AUDIO.PLM file
2. find the number of play list from it
3. stuff the PlistCount with the number
4. if all passed return 0

4.2.6 *SdGetPlayLists*

*UINT SdGetPlayLists(mediaHandle m_mediaHandle, PList * pPList, UINT8 * PListCount)*

Description:

Return the play list names

Input:

Audio Handle

Pointer to list structure

Pointer to Number of play list

```
PList struct {
    char    index; // this number corresponds to the current play list name
    int     len;   // play list name's string length
    char    PListName[200]; // play list name maximum of 200 bytes
    char    TracksInPList; // number of tracks in the play list
}
```

Output:

pPList structure stuffed with play list text info

if the number of play list has been updated it will update the PListCount attribute

return value:

```
0: // success
1: // file does not exist
2: // file system error
3: // play list does not exist
ff: // play list count is changed
```

4.2.7 *SdGetTrackTitle*

*UINT SdGetTrackTitle(mediaHandle m_mediaHandle, UINT8 PlistIndex, PLTitle *pPLTitle)*

Description:

Retrieve Track Titles within the selected Play list

Input:

Audio Device Handle

Selected Play list number

Pointer to PLTitle (play list title) structure

```
struct track_title      {
TIMEMS_t               trackTime;    //Total track time in milli-seconds, includes related TKIs
UINT16                 tracknum;     //Track number
UINT16                 index;       //Index, corresponding to the current track
char                   trackName[TRACKNAME_MAXCHAR]; //Track name
};
```

Output:

pPLTitle stuffed with track information

Return value:

- 0: // success
- 1: // invalid audio handler
- 2: // invalid list
- 3: // Track title empty
- 4: // no track in the current play list

4.2.8 *SdGetTrackInfo*

*UINT SdGetTrackInfo(media_Handle m_media_Handle, UINT8 PlistIndex, Track *pTrackInfo)*

Description:

Retrieves complete track information that is in the selected play list. This includes: title, producer, author, track time, etc.

Input:

Audio Device Handle

Selected Play list index

Pointer to Track Information structure

```
struct track_info      {
TIMEMS_t              trackTime;    //Total track time in milli-seconds, includes related TKIs
SIZE_t                bytesize;    //Total track size in bytes, includes related TKIs
UINT16                tkisInTrack;  //Number of track unit (TKIs) in track
UINT16                tracknum;     //Track number
UINT16                index;       //Index, corresponding to the current track
char                  trkInformation[TRACKINFO_MAXCHAR]; //Track information
};
```

Output:

pTrack stuffed with track information

Return value:

```
0:    // success
1:    // invalid audio handle
2:    // invalid list
3:    // Track title empty
4:    // no track in the current play list
FF:   // new number of Tracks in the play list
```

4.2.9 SdOpenTrack

*UINT SdOpenTrack(media_Handle m_media_Handle, UINT8 PlistIndex, UINT16 Track, TrackGenInfo *pTrackGenInfo)*

Description:

Get track information from TKI manger, as its physical location in the SD Card, title key, current location pointer within the Track, etc.

Input:

Audio Device Handle
Selected Play list index
Track Number
Pointer to track general information structure

```
struct track_gen_info {
UINT32magicnum; //Magic number for validity
CMD_t cmd; //Operation command defined by CMD_t e.g. play
AUDIO_t audioformat; //Audio format defined by AUDIO_t e.g. MP3
FREQ_t sampfreq; //Codec sampling frequency defined by FREQ_t
APP_t appAttrib; //Application attribute e.g. music, book
SIZE_t sizeAOB; //Size of AOB in bytes
UINT16currentAOB; //Current AOB, being decrypted
UINT16firstAOB; //First AOB block for track
UINT16lastAOB; //Last AOB block for track
UINT16countAOB; //Total AOBs for track
UINT16syncword; //Current position of sync position in AOB block
SIZE_t seekposAOB; //Seek position in AOB
TIMEMS_t trkElapsedTime; //Elapsed time of track in milli second
TIMEMS_t trkTotalTime; //Total play time of track in milli second (all related TKIs)
SIZE_t bytesize; //Total track size in bytes, includes related TKIs
UINT16elementplaytime; //Each element play time in milli second
UINT16tracknum; //Track number to be played
INT16 fwTime; //0 - play normal, > 0 forward, < 0 rewind. Step of 2 sec.
INT16 fwNext; //0 - play current, > 0 play next, < 0 play previous
UINT16tracksInPlist; //Number of tracks in play list
TIMEMS_t pListTime; //Total Playback time in milliseconds
UINT16elementsize; //Size of current element
UINT16elementoffset; //Offset in current element
UINT8 currentelement; //Current element # in AOB
UINT8 totalelements; //Total element in AOB
UINT8 plistnum; //Current play list number
};
```

Output:

Track general information structure stuffed with information

Return value:

- 0: // success
- 1: // invalid audio handler
- 2: // invalid list
- 3: // Track title empty
- 4: // no track in the current play list

4.2.10 SdPlayTrack

UINT SdPlayTrack(media_Handle m_media_Handle, TrackGenInfo *pTrackGenInfo, char *AudioBuf, int SizeOfAudioBuf)

Description:

Decrypt the current buffer, using the information from the TrackHandle structure, and fill the AudioBuf with Decrypted content

Input:

Pointer to Audio Device Handle

Pointer to general track information structure

Pointer to buffer where the application wants to store the decrypted content

Output:

AudioBuf filled with decrypted data

Following attributes of the general track information structure need to be updated in here

```
pTrackGenInfo->status = 0;           //Play content
pTrackGenInfo->NextTrackNum = new value; //Track number that is going to be played next
pTrackGenInfo->sync_word = syncblock; //Last sync_word in AOB_BLOCK
pTrackGenInfo->FWTime = 0;           //Continuos playback
pTrackGenInfo->FWNext = 0;           //Default value
```

Return value:

```
0: // success
1: // invalid audio handler
2: // invalid list
3: // general track information structure empty
4: // unable to decrypt content
5: // unable to find Title Key
```

4.2.11 *SdForward*

UINT SdForward(media_Handle m_media_Handle, TrackGenInfo *pTrackGenInfo, char *AudioBuf, int SizeOfAudioBuf)

Description:

Move the current track pointer in the buffer to new position, by the value of FWTime;
If FWTime > 0 move track pointer forward
If FWTime < 0 move track pointer backward
Start decrypting the content from current position, and stuff the audioBuf with new data

Input:

Audio Device Handle
Pointer to general track information structure
Pointer to buffer where the application wants to store the decrypted content

Output:

AudioBuf filled with decrypted data

Following attributes of the Track Handle structure need to be updated in here

```
pTrackGenInfo->status = 1;           // Forward content
pTrackGenInfo->NextTrackNum = new value; //Track number that is going to be played next
pTrackGenInfo->sync_word = syncblock; //Next sync_word in AOB_BLOCK
pTrackGenInfo->FWTime = 0;           //Continuos playback
pTrackGenInfo->FWNext = 0;           //Default value
```

Return value:

```
0: // success
1: // invalid audio handler
2: // invalid list
3: // TrackHandle structure empty
4: // unable to decrypt content
5: // unable to find Title Key
```

4.2.12 *SdNextTrack*

UINT *SdNextTrack*(media_Handle m_media_Handle, TrackGenInfo *pTrackGenInfo, char *AudioBuf, int SizeOfAudioBuf)

Description:

Move to another track in the play list

If FWNext > 0 move to next track, from the current track position

If FWNext < 0 move to previous track, from the current track position

Start decrypting the content from new selected track, and decrypt and stuff the audiobuf with decrypted content

Input:

Audio Device Handle

Pointer to general track information structure

Pointer to buffer where the application wants to store the decrypted content

Output:

AudioBuf filled with decrypted data

Following attributes of the Track Handle structure need to be updated in here

```
pTrackGenInfo->status = 2;           // Next Track state
pTrackGenInfo->NextTrackNum = new value; //Track number that is going to be played next
pTrackGenInfo->sync_word = syncblock; //Next sync_word in AOB_BLOCK
pTrackGenInfo->FWTime = 0;           //Continuous playback
pTrackGenInfo->FWNext = 0;           //Default value
```

Return value:

```
0: // success
1: // invalid audio handler
2: // invalid list
3: // TrackHandle structure empty
4: // unable to decrypt content
5: // unable to find Title Key
```

4.2.13 *SdStopPlay*

UINT SdStopPlay(media_Handle m_media_Handle, TrackGenInfo *pTrackGenInfo)

Description:

Stop close AOBxxx.SA1 file;
Set Current Location to the beginning of the current Track

Input:

Pointer to Audio Device Handle
Pointer to general track information structure

Output:

Following attributes of the Track Handle structure need to be updated in here

```
pTrackGenInfo->status = 3;           //Stop
pTrackGenInfo->sync_word = 0;        //Next sync_word in AOB_BLOCK
pTrackGenInfo->FWTime = 0;           //Continuos playback
pTrackGenInfo->FWNext = 0;           //Default value
pTrackGenInfo->CurrentLocation = 0;  //Reset
```

Return value:

```
0:      // done
```

4.2.14 *SdPauseTrack*

UINT *SdPauseTrack*(media_Handle m_media_Handle, TrackGenInfo *pTrackGenInfo)

Description:

Set status field of the general track information structure to pause
Stop decrypting content

Input:

Audio Device Handle
Pointer to general track information structure

Output:

pTrackGenInfo->CurrentLocation //No change
pTrackGenInfo->status = 4; //Pause

Return value:

0: // done

4.2.15 *SdResetPlayList*

UINT SdResetPlayList(media_Handle m_media_Handle, TrackGenInfo *pTrackGenInfo)

Description:

Stop close AOBxxx.SA1 file;
Set general track information structure to the beginning of the current play list

Input:

Pointer to Audio Device Handle
Pointer to general track information structure

Output:

Following attributes of the general track information structure need to be updated in here

```
pTrackGenInfo->status = 5;           //Reset play list
pTrackGenInfo->NextTrackNum = 0;     //Track number that's played next
pTrackGenInfo->sync_word = 0;        //Next sync_word in AOB_BLOCK
pTrackGenInfo->FWTime = 0;           //Continuos playback
pTrackGenInfo->FWNext = 0;           //Default value
pTrackGenInfo->CurrentLocation = 0;  //Reset
```

Return value:

```
0: // done
```

5.0 Sample Playback Program

Complete source code for a Sample Playback Program is provided in the SanDisk SD-Audio SDK. This program demonstrates how the SD-Audio SDK API functions can be used to create all of the functionality needed by a music player.

The application, SDAPP.EXE, uses the Windows GUI interface to mimic the buttons that are typically provided on an actual hardware music player. The actions taken by the software when options such as PLAY or NEXT TRACK are selected form a great template that the user can follow when implementing host software in a music player.

Although the C++ source code itself documents how the API function calls are utilized, the following section describes each of the main areas of functionality within the code, and further elaborates on how the user's application can best make use of the APIs.

5.1 Overview

SDAPP.EXE was written in Visual C++ using the Microsoft Foundation Classes (MFC). It is a "dialog-based" application, which means that all processing within the application is initiated via the user interface elements contained in the dialog box that appears on the screen. In SDAPP.EXE, buttons within the dialog box can be clicked to select the following options:

PLAY, PAUSE, STOP, PREVIOUS_TRACK, REWIND, FORWARD, NEXT_TRACK, and EJECT.

In the `sdaudioldg.cpp` source file, the user will see the "message map" (look for "BEGIN_MESSAGE_MAP(CSAudioDlg, Cdialog)") that associates the clicking of the buttons with the corresponding action. Processing for each button is carried out by message handlers which are member functions within the `CsdAudioDlg` dialog class. For example, when the `IDC_OP_PLAY` message is generated by the user clicking the "PLAY" button, the `OnOperationPlay` function is called. Even if the user isn't familiar with Windows or MFC, by looking at the code within each of these handler functions, the user can see how SD-Audio SDK API functions are used to perform each task.

5.2 Initialization

The first step in using the SD-Audio SDK API functions is to initialize the system and ensure that an SD Memory Card is present on the hardware platform. This should be done at the beginning of the music player software. In SDAPP.EXE, this is conveniently handled within the `OnInitDialog()` function which is automatically called when the dialog box is being created. Within that code, the user should look at the calls to `SdInitAudioSystem` and `SdMountAudio`. `SdInitAudioSystem` interrogates the system, initializes each SD Memory Card that's found, and ensures that each card is accessible and formatted. The total number of cards that are successfully detected is returned. `SdMountAudio` "mounts" the prescribed card by locating and reading the play list manager file. If the operation is successful, a description of the located play list is placed in the buffer indicated by a passed parameter. The function returns with a handle that will be used in subsequent API calls.

5.3 Retrieval of Play Lists and Track Titles

The last step in the initialization code, found in `OnInitDialog()`, is to accumulate all of the play list descriptions and the titles for all the audio tracks contained in first play list. The `InitInfo()` function gets the number of play lists by calling `SDGetPlayListCount`. Then, for each play list, it calls `sdGetPlayLists`, specifying a play list index which causes a structure of play list information to be returned. A count of the number of tracks contained in the play list and a play list description is housed in each `play_list` structure (see `sdaudapi.h` for definitions of all data types). It then retrieves track titles for each track in the first play list by calling `AddTrackTitle` which in turn calls the `SdGetTrackTitle` API function.

5.4 Retrieval of Specific Track Information

Notice at the end of `InitInfo()`, it calls `InitTrackInfo()` passing a value of zero for the track number. This will utilize two important API functions: `SdGetTrackInfo` and `SdOpenTrack`, to retrieve all the information for the track and prepare it for playing. Initializing the index value for the desired track within the track information structure and passing it to `sdGetTrackInfo` results in the track information structure being filled in with all the pertinent track information such as the total size of the audio selection in bytes, the total time (in milliseconds) that the playing of the track consumes, number of track units (TKIs) in the track, and other data. The calculations performed within the `InitTrackInfo` function are a good example of how to initialize the display of elapsed playing time.

`SdOpenTrack` is called to prepare for playing the track. It completely initializes a `trackGenInfo` structure for the track. This is the structure that's involved in all aspects of controlling the playback of the track. A pointer to this structure will be passed to numerous other API functions (such as `SdPlayTrack`, `SdPauseTrack`, etc). Values housed in `trackGenInfo` include: the audio format, size of AOB, current AOB being decrypted during playback, current position in AOB, elapsed time, etc. (`SdOpenTrack` initializes the current AOB element to the first TKI which corresponds to the specified track.)

5.5 Selecting a Track

`SdNextTrack()` is used to move to the next or previous track. Before calling this function, which takes a pointer to a `trackGenInfo` structure as an argument, the track number within `trackGenInfo` is either incremented or decremented. See `OnOperationPrevTrack` and `OnOperationNextTrack` functions for examples of this. `SdNextTrack()` opens the newly selected track and automatically initiates playing.

5.6 Playing a Track

Playing of a track is initiated by calling the `SdPlayTrack` API function. A pointer to the `trackGenInfo` structure is passed as an argument. `SdPlayTrack()` opens the associated file by generating the appropriate file name, `\SD_AUDIO\AOBnnn.SA1`. The `nnn` portion of the file name is generated from the current AOB element (set in `SdOpenTrack()`) within the `trackGenInfo` structure. The data is decrypted and playing is initiated. AAC, MP3 and WMA formats are supported.