XOR Media MCC SDK User Guide

Version 3.3.2.1

Table of Contents

1 Revision History 5

2 Introduction 6

2.1 Purpose 6

2.2 Definitions, Acronyms, and Abbreviations 6

2.3 Preface 6

2.4 Overview 7

2.5 System Requirements 8

2.5.1 Minimum Hardware Requirements 8

2.5.2 Software Requirements 8

3 Getting Started 9

3.1 Introduction 9

3.2 Installing MCC SDK 9

3.3 Uninstall 10

3.4 SDK Folder Structure 10

4 Sample Applications 11

4.1 Creation of New Applications 12

5 C++ Components 13

5.1 The Component Class Structure Tree 13

5.1.1 Class View 15

5.2 Class CScAsDecoder 16

5.2.1 Construction and destruction 17

5.2.2 Member functions 17

5.3 Class CScAsRecorder 30

5.3.1 Construction and destruction 31

5.3.2 Member functions 31

5.4 Class CScAsPlaylistMgr 41

5.4.1 Construction and destruction 41

5.4.2 Member functions 43

5.5 Class CScAsRecordlistMgr 60

5.5.1 Construction and destruction 60

5.5.2 Member functions 61

5.6 Class CScAsSystemStatus 74
5.6.1 Construction and destruction 74
5.6.2 Member functions 75
5.7 Class CScAsCache 81
5.7.1 Construction and destruction 81
5.7.2 Member functions 82
5.8 Class CScAsArchive 95
5.8.1 Construction and destruction 95
5.8.2 Member functions 96

Appendix A: Include Files 103

Appendix B: Sample Applications 105

SampleForCache 106
SampleForArchive 109
SampleForDecoder 111
SampleForRecord 114
SampleForPlaylistMgr 117
SampleForRecordlistMgr 121

Appendix C: Auxiliary Class 125

Class CScAsRpcSessionManager 125
Construction and destruction 125
Member functions 125
Class CDeviceStatus 127
Construction and destruction 127
Member functions 127
Type definitions 131
Class CDeviceStatusInfo 132
Construction and destruction 132
Member functions 132
Type definitions 134
Class CRequestArchive 134
Construction and destruction 135
Member functions 135
Type definitions 137
Class CArchiveService 138
Construction and destruction 138
Member functions 139
Type definitions 142

Class CSCRPCException..... 142
Construction and destruction 142
Member functions 143
Class CTimeCode 143
Construction and destruction 143
Member Functions 145
Class CBriefPrivateData 153
Construction and destruction 153
Member Functions 154
Type definitions 170

Appendix D: Definition..... 175

Appendix E: The status map of play list / record list 191

Appendix F: RemoteVstrmAPI Calls vs. MCCAPI Calls..... 192

Appendix G: Q&A 198

1 Revision History

Date	Rev	Author	Description
2-12-2007	0.1	Echo Zhang	Initial Draft
4-3-2007	0.2	Echo Zhang	Modified some description in class CDeviceStatus, CScAsSystemStatus, CScAsCache,CscAsArchive , CScAsDecoder
3-14-2008	0.3	Joe Zou	1. Modify AFD & Scaled Aspect Ratio 2. No longer use namespace "MCCSDK" 3. Add sample code for Cache manager 4. Update the AFD on Page 52
4-7-2008	1.1.0.0	Joe Zou	1. Add new RPC interface of Sub-Clip 2. Modify SD_CACHE_ELEMENT structure (Page 167) and class CCacheElement (Page 151) 3. Modify class CScAsCache(Page 56)
4-23-2008	1.1.0.1	Joe Zou	1. Add new RPC interface of Decoder 2. Add new member functions "SetAudioMap()" and "SetAudioMapForSimulcast()" in class "CScAsDecoder" (Page 91)
7-17-2008	1.1.0.32	Joe Zou	Modify ScAsArchive::GetArchiveCopyStatus() In order to get actual finished transfer size and transferred file size, add two output pararmeters and one input parameter to this function. (Note: Get percent completed as finished transfer size outputs last version)
22-10-2008	1.0.0.36	Joe Zou	Modify ScAsArchive::Copy() Add a parameter for moving operation.
26-12-2008	2.1.0.36	Joe Zou	New feature - Support Playlist and Recordlist for Playout and record. 1. Add two new classes: CScAsPlaylistMgr & CScAsRecordlistMgr 2. Sample Applications
09-04-2009	2.1.0.40	Joe Zou	New function in CScAsRecorder ModifyRecordingVideoDuration()
27-04-2009	2.1.0.40A	Joe Zou	Modify description for CScAsCache::GetNext() & CScAsSystemStatus::GetLocalNodeVideoStandard()
22-09-2009	2.1.0.45	Joe Zou	Support Playlist event type which intergrates with MCT tool. Add playlist interface InsertClipEx,PlayEx,GetClipAttribute,SetClipAttribute,GetPlaylist Add recordlist interface InsertClipEx,GetClipAttribute,SetClipAttribute,GetRecordlist
26-11-2009		Joe Zou	Update some function descriptions of CScAsPlaylistMgr class.
31-12-2009	2.2.0.0	Joe Zou	Add new function to CScAsPlaylistMgr: GetTimeRemaining()
15-01-2010	2.2.0.0	Joe Zou	Modified the description of function CScAsCache::IdRequest()
25-01-2010	2.2.1.0	Joe Zou	Update the description of class CScAsDecoder and CScAsRecorder. Add play list and record list functions to these two classes.
04-02-2010	2.2.1.1	Joe Zou	Add new function "IsListEnd ()" to class CScAsPlaylistMgr, CScAsRecordlistMgr, CScAsDecoder and CScAsRecord. About the description of this function, the users can read CScAsPlaylistMgr chapter. Add the description of play / record list status flow to Appendix E.
11-11-2011	2.2.1.1 R1	Joe Zou	Update the description of sample code; add new samples, including cache sample code and archive sample code.
08-03-2012	2.2.1.1R2	Xiaoyan Zhou	Add some alternative MCCAPI calls for RemoteVstrmAPI calls in Appendix F.
03-20-2012	3.3.2.1	Michael	Changed the doc number.

2 Introduction

2.1 Purpose

This guide gives an overview of the installation, setup and use of the MCC SDK. Also included are the structure of the MCC SDK and detailed description of its components along with a few sample applications.

Topics Covered

Topics covered in this guide are:

· **Section 1: Introduction** – provides a general overview of the SDK and the purpose of the guide.

· **Section 2: Getting Started** – describes how to install, uninstall, and run the SDK. In addition, the MCC SDK folder structure is given in this section too.

· **Section 3: Sample Applications** – describes samples included in the MCC SDK.

· **Section 4: C++ Components** – describes the complete list of components included in MCC SDK with detailed description of class layout, public member functions, etc.

· **Section 5: Appendix** – describes include files, sample application, auxiliary class and structure and definitions.

2.2 Definitions, Acronyms, and Abbreviations

***MCC SDK*:** MediaClient Controller Software Development Kit.

MCL server: MediaClient server acts as an independent I/O that uses CIFS (Common Internet File System) and IP networking to record and play serial digital video.

Back to Back playout: While one clip is playing in the decoder, decoder is able to be cued on next clip and play the next clip when the prev clip is finished.The signal is seamless outputted with frame accuracy.

Back to Back record: While one clip is recording in the encoder, encoder is able to be initiated on next clip and recorded the next clip when the prev clip is finished.The inputted signal is seamless captured with frame accuracy.

2.3 Preface

The MCC SDK is a software development kit intended to enable programmers to develop broadcast system applications using the MCC components. It enables them to query XOR Media's system for a complete list of assets, perform encode, decode or archive

operations on those assets. Furthermore it enables them to poll the system to determine the status of the system as a whole and the status of each individual codec task.

The objective of this MCC API is to provide an easy interface for third-party user, such as automation system to work smoothly with XOR Media MediaClient System. The mechanism interacting with kernel service is RPC calls (Remote Procedure Call) on Win32 platform.

2.4 Overview

To help understand the XOR Media MCC API, high level architecture of the system is shown below.

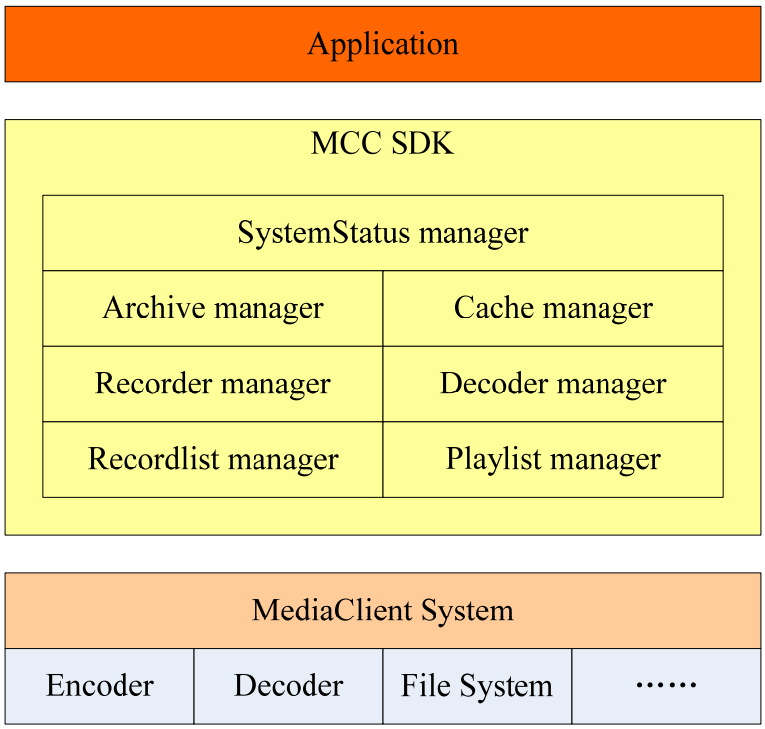


Figure 1- Block Diagram of the MCC SDK

In the block diagram of the **MCC SDK** above, MCC SDK includes Archive manager component, Cache manager component, Recorder manager component, Decoder manager component, Record list manager component, Play list manager component and System status manager component at the top.

MediaClient system executes the corresponding tasks on encoder, decoder and file system component and so forth after receiving calls from MCC SDK.

2.5 System Requirements

The MCC SDK has the following minimum hardware and software requirements:

2.5.1 Minimum Hardware Requirements

Minimum hardware requirements:
SSE-enhanced CPU (Intel® Pentium III, Celeron, AMD® Athlon, Opteron etc.)
128 MB RAM
Any VGA card

2.5.2 Software Requirements

Windows® ME/2000/XP OS

3 Getting Started

3.1 Introduction

The following section details the procedures for installing the MCC SDK. In addition, it provides the description of the MCC SDK folder structure.

Note: *The described installation/uninstallation process is relevant for all versions of the MCC SDK.*

3.2 Installing MCC SDK

Caution: *Installing newer version of MCC SDK without removing a previously installed old one will cause problems. ALWAYS UNINSTALL PREVIOUS RELEASES of the MCC SDK before installing newer one (see page 11 for UNINSTALL).*

Configuration:

MCC SDK provides remote support capability through Windows NT Remote Access Service (RAS). So it is crucial for end-users to adjust the security level accordingly to shield the workstation from any malicious remote intrusion.

1. Modified file "%System32%\drivers\etc\hosts" to add MediaClient servers' hostname and their IP address as below:

// IP address hostname

127.0.0.1 localhost

10.20.3.1 MCL4012A1

10.20.3.2 MCL4012A2

10.20.3.22 TYAN2

2. Accessing those servers by hostnames, which are listed in the hosts file. E.g. run \\MCL4012A2 to access this MediaClient server and input the administrator account name and password.

3. Running the sample application ExdApiSampleApp_d.exe, input one of the servers' hostname, then press the "Connect" button.

If connected successfully the configuration is finished.

Remark

Remember MCC SDK is able to work properly only if the MediaClient server starts up first.

3.3 Uninstall

To uninstall, click "Add/Remove Programs" from the Control Panel to uninstall the InstallShield installations.

3.4 SDK Folder Structure

\Bin

Contains XOR Media MCC SDK Dlls;

\Doc

Contains reference documentation for the XOR Media MCC APIs;

\Include

Contains all the headers files needed to compile applications that use the MCC SDK;

\Lib

Contains all required library files for compile applications that use the MCC SDK;

\Samples

Contains all sample code;

4 Sample Applications

The MCC SDK sample applications are grouped according to the components they use (e.g. Decoder, Encoder, Archive, etc.). Samples are written in C++. The following table provides a brief overview of each sample.

SDK Component	Sample Name	Description	Catalog
System Status	ExdApiSampleApp	This sample demonstrates the Usage of System Status manager component API (Class CScAsSystemStatus)	\MCCSDK\Samples\ ExdApiSampleApp
Cache Manager	SampleForCache	This sample demonstrates the usage of Cache Manager component API (Class CScAsCache)	\MCCSDK\Samples\ SampleForCache
Archive Manager	NULL	This sample demonstrates the usage of Archive Manager component API (Class CScAsArchive)	NULL
Record	SampleForRecord	This sample demonstrates the usage of Encoder component API (Class CScAsRecorder)	\MCCSDK\Samples\ SampleForRecord
Decoder	SampleForDecoder	This sample demonstrates the usage of Decoder component API (Class CScAsDecoder)	\MCCSDK\Samples\ SampleForDecoder
Playlist	SampleForPlaylist Mgr	This sample demonstrates the usage of Playlist component API (Class CScAsPlaylistMgr)	\MCCSDK\Samples\ SampleForPlaylistMgr
Recordlist	SampleForRecordlistMgr	This sample demonstrates the usage of Recordlist component API (Class CScAsRecordlistMgr)	\MCCSDK\Samples\ SampleForRecordlistMgr

4.1 Creation of New Applications

To create a new application, follow the guidelines below:

1. Add %SDK root%\include paths to Project, for Visual Studio 2003 input the path to Configuration Properties -> C/C++ -> General -> Additional Include Directories.

2. The paths to the libraries %SDK root%\Lib are also set in project files Additional library path. For Visual studio 2003, input this path to Configuration Properties -> Linker -> General -> Additional Library Directories;

3. Add the MCC SDK dependences libraries to Project, for Visual Studio 2003, input "ScAsRpc.lib" to Configuration Properties -> Linker -> Input -> Additional Dependencies; If you need to list all video in your UI(User Interface) MediaClient Server, you may add library "ScExdUtl.lib".

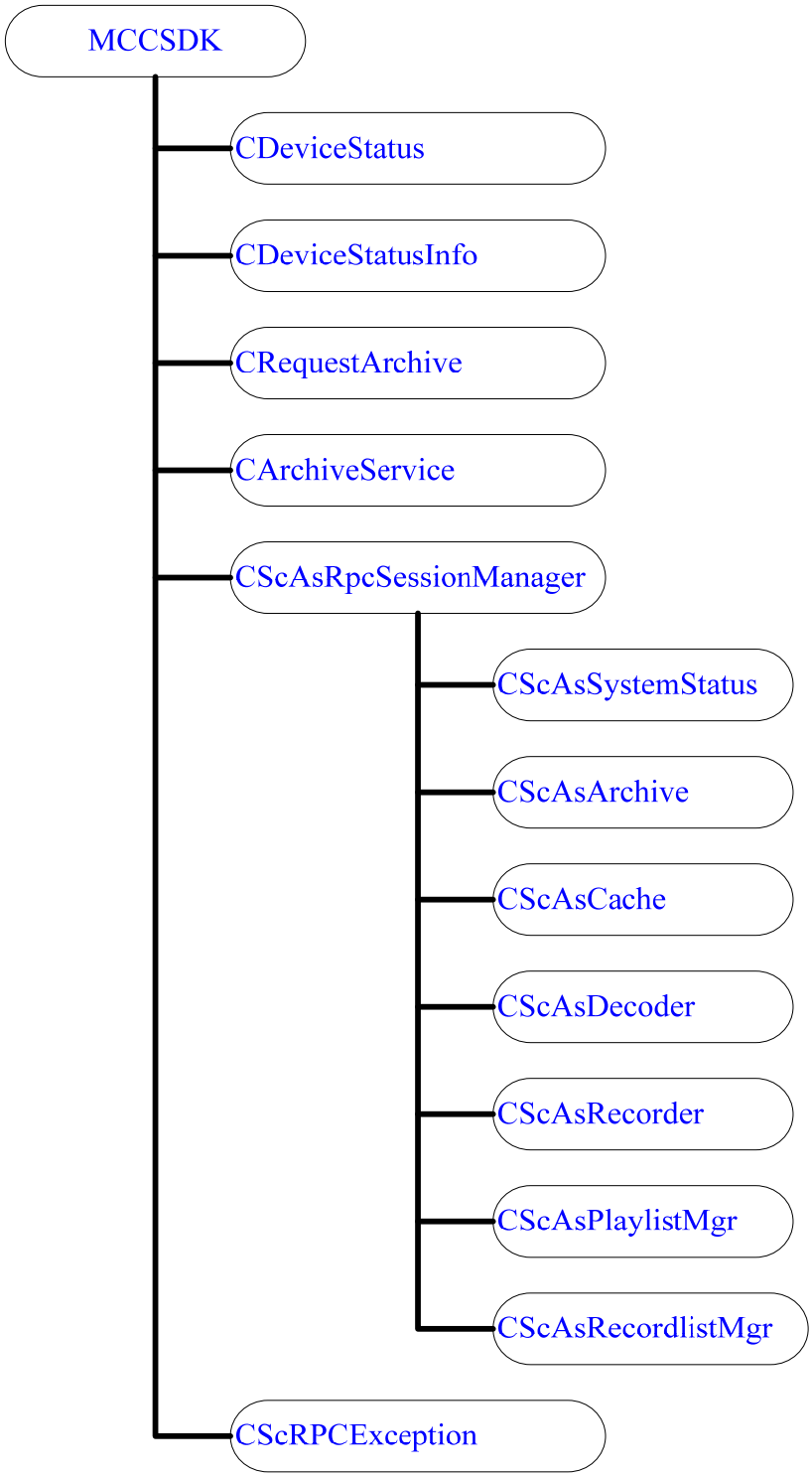
4. To use any of the MCC SDK components, add the following in your Application.

```
#include "ScAsUtility.h"
```

Note: Version a.b.c.d (b=2) supports Visual Studio 2005.

5 C++ Components

5.1 The Component Class Structure Tree



MCC SDK includes seven main components:

1. **Decoder Manager Component**: Provides control API for each decoder port, by this component the user can play or do jog/shuttle playback to some video file with one of the decoder ports. The corresponding class is class CScAsDecoder

2. **Encoder Manager Component**: Provides control API for every encoder port. The corresponding class is class CScAsRecorder.

3. **Playlist Manager Component**: Provides control API to playlist on a decoder port, by this component the user can create a play list, and control decoder to play. The corresponding class is class CScAsPlaylistMgr.

4. **Recordlist Manager Component**: Provides control API to recordlist on an encoder port, by this component the user can create a record list, and control encoder to record. The corresponding class is class CScAsRecordlistMgr.

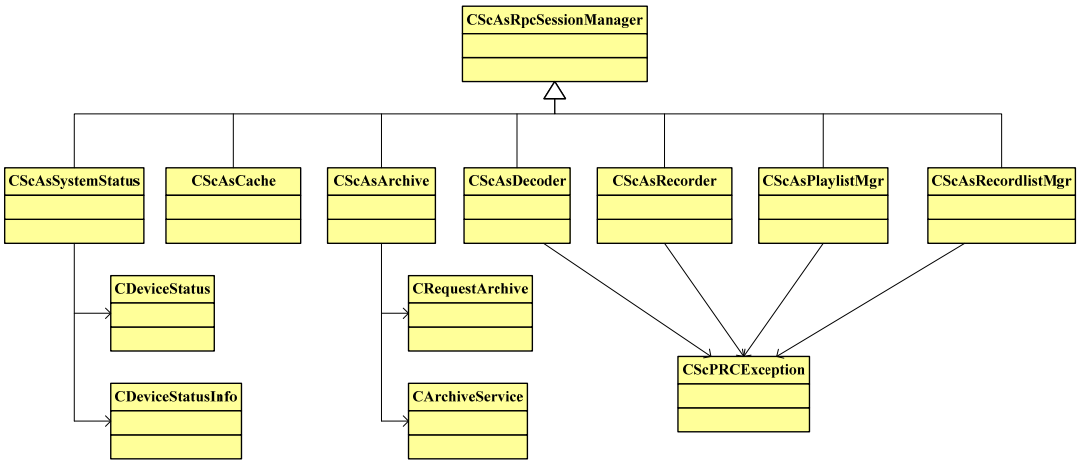
5. **System Status Manager Component**: Provides system status or version info manager API, which includes obtaining of all of the encoder and decoder ports, port status, system version info, codec video standard (PAL, NTSC), reset port, release port and so on. The corresponding class is class CScAsSystemStatus.

6. **Cache Manager Component**: Provides file manager API which includes obtaining of video file list, reading Metadata info of every video file, writing or modifying those Metadata info, deleting or creating video file and more. The corresponding class is class CScAsCache.

7. **Archive Manager Component**: Provides files control manager API among multiple video storage nodes, which includes copy, requeue copy tasks, abort or modify copy task and so on. The corresponding class is class CScAsArchive.

MCC SDK includes auxiliary components. Refer to Appendix C.

5.1.1 Class View



5.2 Class CScAsDecoder

***class** AFX_EXT_CLASS CScAsDecoder : **public** CScAsRpcSessionManager，*

***public** CScAsSingleLinkedBase*

Location: ScAsDecoder.h

Description

Class is for control decoder device in MediaClient server.

Derivation

CScAsRpcSessionManager - Base class for provided common functionality for connecting with MediaCient server.

CScAsSingleLinkedBase – Base class for provided common functionality for single linked.

Members

OpenDecoder(), CloseDecoder(), GetDeviceName(), Cue(), Pause(), Continue(), StepBack(), StepForward(), JogForward(), JogReverse(), ShuttleForward(), ShuttleReverse(), FastForward(), FastReverse(), Goto(), Play(), Stop(), Unload(), UnloadAll(), Reset(), IsTrickModeEnabled(), SetTrickModeEnabled(), IsCueInZeroBasedIndex(), SetCueInZeroBasedIndex(), GetPortStatus(), GetSignalStatus(), GetBulkDeviceStatus(), GetCurrentPosition(), GetTimeElapsed(), GetTimeRemaining(), GetCuedFirstFrame(), GetPlayingVideo(), GetCuedVideo(), GetLastCompletions(), SetAudioMap(), SetAudioMapForSimulcast()

Members for Play list function

Initialize(), Uninitialize(), SetPrePlayFrames(), IsNext(), GetNextClipIndex(), GetNextScheduleTime(),GetNextCommand(),GetStartTime(), GetTotalDuration(), GetCurrentClipIndex(),GetCurrentStatus(),IsIndexUsable(), CueEx(), IsCued(), Skip(), GetDecoderMode(), SetDecoderMode(), SetPrecueSeconds(), SetVideoFormat(), SetDefinition(), IsListEnd()

InsertClipEx(), RemoveClip(), RemoveAll(), GetFirstClip(), GetNextClip(), SetClipAttribute(), GetClipAttribute(), GetClipCount(),GetClipInfoByPos(), GetClipInfoByClipIndex ()

Note: The description of the above functions refers to class CScAsPlaylistMgr.

Sample Application

Path: %Install Root%\Sample\SampleForDecoder\

Refer to Appendix B.

5.2.1 Construction and destruction

- ***CScAsDecoder (const CString& csNodeName, const CString& csDecoderName)***

Description

It is constructor function. It sets the default value to every member variable.

Parameters

const CString& csNodeName	[in]The required connecting host name of MediaClient server like MCL4012.
const CString& csDecoderName	[in]Decoder device name like MC_SD_DECODER1.

Remark

To get all of decoder device name can call function CScAsSystemStatus::EnumerateDevices() first, then get entire device name by member variable CStringList m_decoderList.

- ***virtual ~CScAsDecoder()***

Description

Virtual destructor. The connected session and opened decoder port will be released in this function.

5.2.2 Member functions

- ***DWORD OpenDecoder(HANDLE hStatusNotifyEvent=INVALID_HANDLE_VALUE,BOOL bOpenShared=FALSE)***

Description

Open current device decoder and port, so that user can execute playing operations (e.g. cue, playing, pause, jog, and shuttle and so on)

Parameters

HANDLE hStatusNotifyEvent	*[in]Input parameter event handle.**Note: Current system does not support this parameter, set it as INVALID_HANDLE_VALUE by default.
BOOL bOpenShared	*[in]A Boolean value, which specified the open mode.*

	*Note: Current system does not support this parameter, set it as FALSE by default.

Return

If successful, return zero else return error code.

Remark

This function is valid after calling Connect() successfully.

- ***DWORD CloseDecoder()***

Description

Release the control resource from current decoder device.

Return

If successful, return zero else return error code.

- ***CString GetDeviceName()***

Description

Get the current decoder device name.

Return

The decoder name.

- ***DWORD Cue(LPCTSTR lpVideoId,const CTimeCode* ptcSegmentStartTime=0,const CTimeCode* ptcSegmentEndTime=0)***

Description

Cue to prepare with the specified video clip, start time code and end time code.

There are three cases:

1. Cue (lpVideoId)

Cue at the first frame of the clip.

2. Cue (lpVideoId,ptcSegmentStartTime)

Cue at the timecode in the stream.

3. Cue (lpVideoId, ptcSegmentStartTime ptcSegmentEndTime)

Cue at a segment of the clip with its markin timecode and markout timecode.

Parameters

LPCTSTR lpVideoId	[in]The requested video name.
const CTimeCode* ptcSegmentStartTime	[in]The playing start position.
const CTimeCode* ptcSegmentEndTime	[in]The playing session End position.

Return

If successful, return zero else return error code.

- ***DWORD Pause()***

Description

Pause on current playing session. After paused, port status is changed to PAUSE.

Return

If successful, return zero else return error code.

- ***DWORD Continue()***

Description

Resume on current playing session if port status is PAUSE.

Return

If successful, return zero else return error code.

- ***DWORD StepBack()***

Description

Step backward for current decoding position. The function is available if SetTrickMode TRUE.

Return

If successful, return zero else return error code.

- ***DWORD StepForward()***

Description

Step forward for current decoding position. The function is available if SetTrickMode TRUE.

Return

If successful, return zero else return error code.

- ***DWORD JogForward(const unsigned long ulFrames,const float fSpeed)***

Description

Jog forward jump to specified count frames on current decoding position. The function is available if SetTrickMode TRUE.

Parameters

const unsigned long ulFrames	[in]The jump frame counts
const float fSpeed	*[in]Jumping speed.* Note: current system does not support, so user should set it as "1".

Return

If successful, return zero else return error code.

- ***DWORD JogReverse(const unsigned long ulFrames,const float fSpeed)***

Description

Jog backward jump to specified count frames on current decoding position. The function is available if SetTrickMode TRUE.

Parameters

const unsigned long ulFrames	[in]The jump frame counts
const float fSpeed	*[in]Jumping speed,* Note: current system does not support, so user should set it as "1".

Return

If successful, return zero else return error code.

- ***DWORD ShuttleForward(const float fSpeed)***

Description

Shuttle forward with specified speed. The function is available if SetTrickMode TRUE.

Parameters

const float fSpeed	[in]playing speed

Return

If successful, return zero else return error code.

- ***DWORD ShuttleReverse(const float fSpeed)***

Description

Shuttle reverse with specified speed. The function is available if SetTrickMode TRUE.

Parameters

const float fSpeed	[in]playing speed

Return

If successful, return zero else return error code.

- ***DWORD FastForward()***

Description

Forward with specified large speed, the speed is configured by MediaClient server, and default value is 50. The function is available if SetTrickMode TRUE.

Return

If successful, return zero else return error code.

- ***DWORD FastReverse()***

Description

Reverse with specified large speed, the speed is configured by MediaClient server, and default value is 50. The function is available if SetTrickMode TRUE.

Return

If successful, return zero else return error code.

- ***DWORD Goto(const CTimeCode* ptcStartTime)***

Description

Go to the specified position of the stream. The function is available if SetTrickMode TRUE.

Parameters

const CTimeCode* ptcStartTime	[in]The specified time code which decoder will go to.

Return

If successful, return zero else return error code.

- ***DWORD Play(const CTimeCode* ptcStartTime=0,BOOL bAbsoluteTc=TRUE)***

Description

Playback on current decoder device, the playback session should be cued first with function Cue().

There are three cases:

1.Play ()
Play immediate
*2.*Play (ptcStartTime)*
Play at the schedule time
*3.*Play (ptcStartTime, FALSE)*
Play at a relative schedule time.

**Note:Current system does not support schedule time.*

Parameters

const CTimeCode* ptcStartTime	[in]The delay time code.
BOOL bAbsoluteTc	[in]TRUE if decoder is to play at an absolute time code. FALSE if decoder is to play at a relative time code from now.

Return

If successful, return zero else return error code.

- ***DWORD Stop()***

Description

Stop the playing session.

Return

If successful, return zero else return error code.

- ***DWORD Unload()***

Description

Unload current active playing session.

Return

If successful, return zero else return error code.

- ***DWORD UnloadAll()***

Description

Unload all of active or inactive sessions for current decoder device.

Return

If successful, return zero else return error code.

- ***DWORD Reset()***

Description

Reset the current decoder device, and stop all of decoding sessions.

Return

If successful, return zero else return error code.

- ***BOOL IsTrickModeEnabled()***

Description

Judge whether the current decoder device is Trick-Mode work mode. With trick-mode work mode the playback session could performance Goto, Jog, shuttle operations.

Return

If successful, return zero else return error code.

- ***DWORD SetTrickModeEnabled(const BOOL bEnabled=TRUE)***

Description

Set current decoder device Trick-Mode to enable or disable.

Parameters

const BOOL bEnabled	[in]TRUE if enable; FALSE if disable.

Return

If successful, return zero else return error code.

- ***BOOL IsCueInZeroBasedIndex()***

Description

Judge whether the current decoder device is enabled Cue in ZeroBaseIndex parameter. This parameter is setting by SetCueInZeroBasedIndex()

Return

If successful, return zero else return error code.

- ***DWORD SetCueInZeroBasedIndex(const BOOL bEnabled=TRUE)***

Description

Enable or disable CueInZeroBasedIndex function. This parameter is setting for cue operation.

If setting enable, the cue with start time command will start with SOM time code and add specified start time code(cue in time code = SOM + start time).

If setting disable, the cue with start time command will start from the specified start time absolutely.

Parameters

const BOOL bEnabled	[in]TRUE if enable; FALSE if disable.

Return

If successful, return zero else return error code.

- ***DWORD GetPortStatus(DWORD* pdwPortStatus)***

Description

Get decoder status.

Parameters

DWORD* pdwPortStatus	[out]The port status, which is defined as enum *Device States*.

Return

If successful, return zero else return error code.

- ***DWORD GetSignalStatus(DWORD* pdwSignalStatus)***

Description

Get the signal status.

Parameters

DWORD* pdwSignalStatus	[out]The signal status, which is defined by *Signal Status*.

Return

If successful, return zero else return error code.

- ***DWORD GetBulkDeviceStatus(CTimeCode* ptcCurrentTime,CTimeCode* ptcPosition,DWORD* pdwPortStatus,DWORD* pdwSignalStatus = NULL)***

Description

Get multiple statuses, including playback position, port status and signal status.

Parameters

CTimeCode* ptcCurrentTime	[out]The MediaClient server current clock time code.
CTimeCode* ptcPosition	[out]The session playback current position.
DWORD* pdwPortStatus	[out]Port status.
DWORD* pdwSignalStatus	[out]Signal status.

Return

If successful, return zero else return error code.

- ***DWORD GetCurrentPosition(CTimeCode* pCurrentPosition)***

Description

Get playback timecode position.

Parameters

CTimeCode* pCurrentPosition	[out]the session playback position.

Return

If successful, return zero else return error code.

- ***DWORD GetTimeElapsed(CTimeCode* pTimeElapsed)***

Description

Get playback elapsed time

Parameters

CTimeCode* pCurrentPosition	[out]The elapsed time code.

Return

If successful, return zero else return error code.

- ***DWORD GetTimeRemaining(CTimeCode* pTimeRemaining)***

Description

Get the remain time for the decoding session

Parameters

CTimeCode* pTimeRemaining	[out]The remaining time code.

Return

If successful, return zero else return error code.

- ***DWORD GetCuedFirstFrame(CTimeCode* ptcFirstFrame)***

Description

Get the cued start position, this function is called after cued the clip and before playing this session, with this function user can get the first frame position.

Parameters

CTimeCode* ptcFirstFrame	[out]The cued start frame position.

Return

If successful, return zero else return error code.

- ***DWORD GetPlayingVideo(CString* pcsVideoName)***

Description

Get the playing video name.

Parameters

CString* pcsVideoName	[out]The playing video name.

Return

If successful, return zero else return error code.

- ***DWORD GetCuedVideo(CString* pcsVideoName)***

Description

Get the cued video name.

Parameters

CString* pcsVideoName	[out]The cued video name.

Return

If successful, return zero else return error code.

- ***DWORD GetLastCompletions(DWORD& dwPlay, DWORD& dwPlayRequested, DWORD& dwCue)***

Description

Get the last completion status about the playback command, playback request command and cue command.

Parameters

DWORD& dwPlay	[out]The last playback status.
DWORD& dwPlayRequested	[out]The last playback request status.
DWORD& dwCue	[out]The last cue status.

Return

If successful, return zero else return error code.

- ***DWORD SetAudioMap(const CString& csAudioMap);***

Description

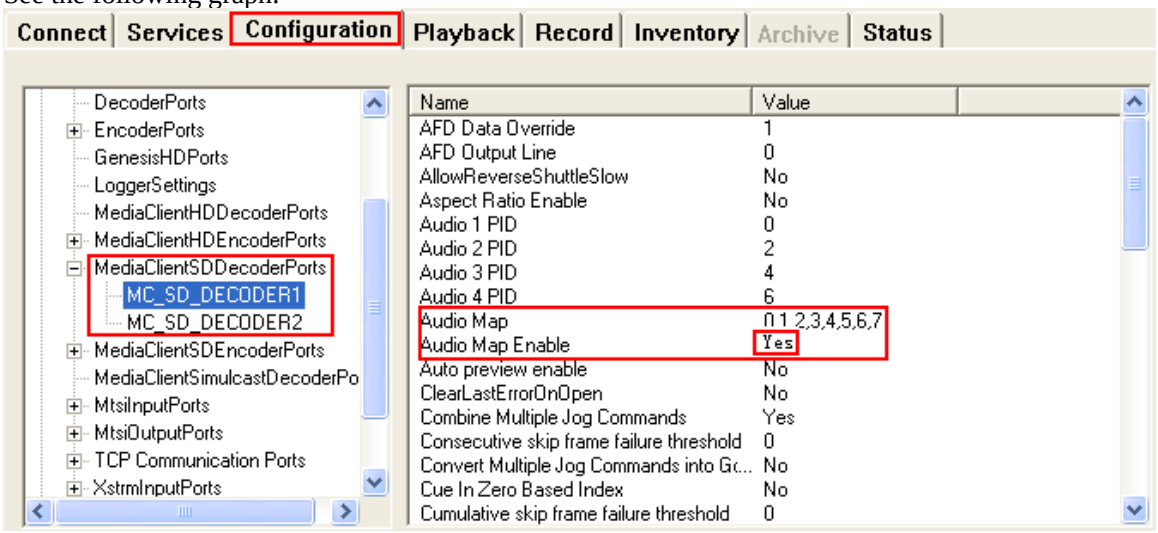
Set play sequence of audio for audio mapping.

The accessible audio channel Index value is: {0, 1, 2, 3, 4, 5, 6, 7};
0-------> [ch0/ch1];
1-------> [ch2/ch3]
2-------> [ch4/ch5];
3-------> [ch6/ch7];
4-------> [ch8/ch9];
5-------> [ch10/ch11];
6-------> [ch12/ch13];
7-------> [ch14/ch15];

For example:
a. Audio Remap sequence "0,0,2,6,4,0,6,3" , this channel number list means EXD will set below audio map to MediaClient:
Audio Port: AES_1 output audio 0 [ch0/ch1];
Audio Port: AES_2 output audio 0 [ch0/ch1];
Audio Port: AES_3 output audio 2 [ch4/ch5];
Audio Port: AES_4 output audio 6 [ch12/ch13];
Audio Port: AES_5output audio 4 [ch8/ch9];
Audio Port: AES_6 output audio 0 [ch0/ch1];
Audio Port: AES_7 output audio 6 [ch12/ch13];
Audio Port: AES_8 output audio 3 [ch6/ch7];
b. Audio Remap sequence "0, ,2,6, ,0, ,3" , this channel number list means EXD will set below audio map to MediaClient:

Audio Port: AES_1 output audio 0 [ch0/ch1];
Audio Port: AES_3 output audio 2 [ch4/ch5];
Audio Port: AES_4 output audio 6 [ch12/ch13];
Audio Port: AES_6 output audio 0 [ch0/ch1];
Audio Port: AES_8 output audio 3 [ch6/ch7];

Note:Audio Map value is available if Audio Map Enable is 'Yes'.
See the following graph:



Parameters

CString& csAudioMap	**[in]String of audio map**

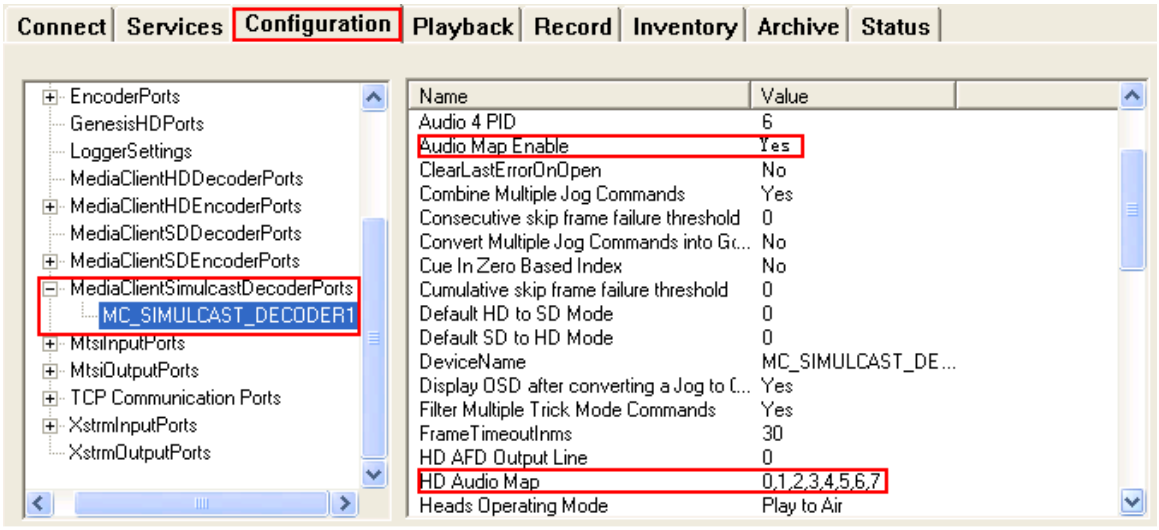
Return

If successful, return zero else return error code.

- ***DWORD SetAudioMapForSimulcast(const CString& csSdAudioMap, const CString& csHdAudioMap)***

Description

Set play sequence of audio for MCL Simulcast.
Note: Refer to SetAudioMap().
See the following graph:



Parameters

CString& csSdAudioMap	[in] a character string of audio map for SD.
CString& csHdAudioMap	[in] a character string of audio map for HD.

Return

If successful, return zero else return error code.

5.3 Class CScAsRecorder

***class** AFX_EXT_CLASS CScAsRecorder : **public** CScAsRpcSessionManager*

Location: ScAsRecorder.h

Description

Class for control encoder device in MediaClient server

Derivation

CScAsRpcSessionManager - Base class for provided common functionality for connecting with MediaCient server.

Members

ReadMediaClientSDProfiles(), ReadMediaClientHDProfiles(), GetLoadedProfile(), LoadProfile(), OpenRecorder(), CloseRecorder(), GetDeviceName(), IsPreviewEnabled(), EnablePreview(), RecordInit(), Record(), Stop(), Unload(), UnloadAll(), Reset(), SetMediaClientSDAttribute(), GetMediaClientSDAttribute(), GetPortStatus(), GetSignalStatus(), GetBulkDeviceStatus(), GetCurrentPosition(), GetTimeElapsed(), GetTimeRemaining(), GetRecordingVideo(), GetRecordInitVideo(), GetAllVideoInfo(), GetLastCompletions(), ModifyRecordingVedioDuration()

Members for Recorder list function

Initialize(), Uninitialize(), SetPrePlayFrames(), IsNext(), GetNextClipIndex(), GetNextScheduleTime(),GetNextCommand(),GetStartTime(), GetTotalDuration(), GetCurrentClipIndex(),GetCurrentStatus(),IsIndexUsable(), RecordInitEx(), IsRecordInit(), GetRecorderMode(), SetRecorderMode(), SetPreInitSeconds(), SetVideoFormat(), SetDefinition(), IsListEnd()

InsertClipEx(), RemoveClip(), RemoveAll(), GetFirstClip(), GetNextClip(), SetClipAttribute(), GetClipAttribute(), GetClipCount(),GetClipInfoByPos(), GetClipInfoByClipIndex ()

Note: The description of the above functions refers to class CScAsRecordlistMgr.

Sample Application

Path: %Install Root%\Sample\SampleForRecorder\

Refer to Appendix B.

Remark

Following preview functions are same as CScAsDecoder.

PreviewPrepareCue(), PreviewCuePrepared(), PreviewCue(), PreviewPlay (), PreviewPause(), PreviewContinue(), PreviewGoto(), PreviewStop(), PreviewUnload(), PreviewUnloadPrepared(), PreviewUnloadAllPrepared(), PreviewUnloadAll(), PreviewReset(), IsPreviewTrickModeEnabled(), SetPreviewTrickModeEnabled(), IsPreviewCueInZeroBasedIndex(), SetPreviewCueInZeroBasedIndex(), GetPreviewPortStatus(), GetPreviewPortSignalStatus(), GetPreviewBulkDeviceStatus(), GetPreviewCurrentPosition(), GetPreviewTimeElapsed(), GetPreviewTimeRemaining(), GetPreviewCurrentTime(), GetPreviewPlayingVideo(), GetPreviewCuedVideo(), GetPreviewLastCompletions(), StepBack(), StepForward(), JogForward(), JogReverse(), ShuttleForward(), ShuttleReverse(), FastForward(), FastReverse()

5.3.1 Construction and destruction

- ***CScAsRecorder (const CString& csNodeName, const CString& csRecorderName)***

Description

Constructor function. It sets the default value to every member variable.

Parameters

const CString& csNodeName	[in]The required connecting host name of MediaClient server like **MCL4012A1**.
const CString& csRecorderName	[in]Encoder device name like **MC_SD_ENCODER1.**

Remark

To get all of decoder device name should call function CScAsSystemStatus::EnumerateDevices() first, then get entire device name by member variable CStringList m_encoderList.

- ***virtual ~CScAsRecorder ()***

Description

Virtual destructor. The connected session and opened encoder port will be released in this function.

5.3.2 Member functions

- ***DWORD ReadMediaClientSDProfiles ()***

Description

Get the encoding profile name list for SD encoder device (e.g. USER1, USER2…).

Return

If successful, return zero else return error code.

- ***DWORD ReadMediaClientHDProfiles ()***

Description

Get the encoding profile name list for HD Encoder device (e.g. USER1, USER2…).

Return

If successful, return zero else return error code.

- ***CString GetLoadedProfile ()***

Description

Get the current encoding profile name. Only the user called function ReadMediaClientSDProfiles () or ReadMediaClientHDProfiles () first, based on the relative encoder type, this function could return the valid value.

Return

The encoding profile name.

- ***WORD LoadProfile (LPCTSTR lpProfileName) const;***

Description

Reload the encoding profile from the register.

Parameters

LPCTSTR lpProfileName	[in]The requested to reload encoding profile name. This profile name could be got from the member variable: CStringList m_profileList.

Return

If successful, return zero else return error code.

Remark

The profile list could be got by calling function ReadMediaClientSDProfiles () or ReadMediaClientHDProfiles () based on the relative encoder type.

- ***DWORD OpenRecorder (HANDLE hStatusNotifyEvent = INVALID_HANDLE_VALUE, BOOL bOpenShared = FALSE)***

Description

Open current encoder port, so that user can execute encoder operations (e.g. cue, recoding and so on)

Parameters

HANDLE hStatusNotifyEvent	[in]Input parameter event handle.**Note: Current system does not support the parameter. Set it as INVALID_HANDLE_VALUE by default.*
BOOL bOpenShared	[in]A Boolean value, which specified the open mode. **Note: Current system does not support the parameter. Set it as FALSE by default.*

Return

If successful, return zero else return error code.

Remark

This function is valid after calling Connect() successfully.

- ***DWORD CloseRecorder ()***

Description

Release the control power from current encoder device.

Return

If successful, return zero else return error code.

- ***CString GetDeviceName()***

Description

Get the current device name.

Return

The device name.

- ***BOOL IsPreviewEnabled()***

Description

Judge whether the preview function is enabled. If the preview function is enabled, user could play the video clip which is encoded by encoder in the same time.

Return

If successful, return zero else return error code.

- ***DWORD EnablePreview(BOOL bEnable)***

Description

Implement to enable the Preview function

Parameters

BOOL bEnable	[in]TRUE if enable the Preview function. FALSE if disable the preview function.

Return

If successful, return zero else return error code.

- ***DWORD RecordInit(LPCTSTR lpVideoId, const CTimeCode* ptcStartTime = 0, const CTimeCode* ptcDuration = 0)***

Description

Cue to prepare encoding initialization and the work mode could be as below:

1. RecordInit (lpVideoId)

Initial recording a clip without start-time and duration.

2.RecordInit (lpVideoId, ptcDuration)

Initial recording a clip with duration.

3.RecordInit (lpVideoId, ptcStartTime, ptcSegmentDuration)

Initial recording a clip with a duration and given start time.

Parameters

LPCTSTR lpVideoId	[in]The requested video name.

const CTimeCode* ptcStartTime	[in]The recording start position.
const CTimeCode* ptcDuration	[in]The recording duration. if duration is zero, recording will not stop until Stop is issued.

Return

If successful, return zero else return error code.

- ***DWORD Record(const CTimeCode* ptcStartTime=0, BOOL bAbsoluteTc=TRUE)***

Description

Start recording the clip.

Parameters

const CTimeCode* ptcStartTime	[in]The delay time code to start recording
BOOL bAbsoluteTc	[in]TRUE if start to record at an absolute time code. *FALSE if start to record at a relative time code from now **Note:Current system does not support FALSE.*

Return

If successful, return zero else return error code.

- ***DWORD Stop(const CTimeCode* ptcStartTime=0)***

Description

Stop recording session.

Parameters

const CTimeCode* ptcStartTime	[in]The delay time code to execute stop command. **Note:Current system does not support delay time. Set zero by default.*

Return

If successful, return zero else return error code.

- ***DWORD Unload()***

Description

Unload current encoding session.

Return

If successful, return zero else return error code.

- ***DWORD UnloadAll()***

Description

Unload all of the encoding sessions in this encoder device.

Return

If successful, return zero else return error code.

- ***DWORD Reset()***

Description

Reset current encoder device.

Return

If successful, return zero else return error code.

- ***DWORD SetMediaClientSDAttribute(int nIndex, DWORD dwValue)***

Description

Set the encoding profile attributes (e.g. video format, bit rate, mux type…).

Parameters

int nIndex	[in]Attribute index, which is defined in Encoder config Profile
DWORD dwValue	[in]The specified attribute value, refer to enum *MCL SD Encoder config Profile.*

Return

If successful, return zero else return error code.

- ***DWORD GetMediaClientSDAttribute(int nIndex, DWORD* pdwValue)***

Description

Get encoding SD profile attributes (e.g. video format, bit rate, mux type…).

Parameters

int nIndex	[in]Attribute index, which is defined in Encoder config Profile
DWORD dwValue	[out]The attribute value. Refer to enum MCL SD Encoder config Profile.

Return

If successful, return zero else return error code.

- ***DWORD GetPortStatus(DWORD* pdwPortStatus)***

Description

Get the port status.

Parameters

DWORD* pdwPortStatus	[out]The returned port status. The port statuses are defined as enum Device States.

Return

If successful, return zero else return error code.

- ***DWORD GetSignalStatus(DWORD* pdwSignalStatus)***

Description

Get the source signal status.

Parameters

DWORD* pdwSignalStatus	[out]The returned signal status. The signal statuses are defined by Signal Status.

Return

If successful, return zero else return error code.

- ***DWORD GetBulkDeviceStatus(CTimeCode* ptcCurrentTime,CTimeCode* ptcPosition,DWORD* pdwPortStatus,DWORD* pdwSignalStatus = NULL)***

Description

Get all of status in current encoder device, including current recording position, port status and signal status.

Parameters

CTimeCode* ptcCurrentTime	[out]The MediaClient server current clock time code

CTimeCode* ptcPosition	[out]The session playback current position
DWORD* pdwPortStatus	[out]The status
DWORD* pdwSignalStatus	[out]Signal status

Return

If successful, return zero else return error code.

- ***DWORD GetCurrentPosition(CTimeCode* pPosition)***

Description

Get current recording position.

Parameters

CTimeCode* pPosition	[out]The session recoring position.

Return

If successful, return zero else return error code.

- ***DWORD GetTimeElapsed(CTimeCode* pElapsed)***

Description

Get recording elapsed time.

Parameters

CTimeCode* pElapsed	[out]The session elapsed time.

Return

If successful, return zero else return error code.

- ***DWORD GetTimeRemaining(CTimeCode* pRemaining)***

Description

Get remain time for the encoding session.

Parameters

CTimeCode* pRemaining	[out]The remaining time code.

Return

If successful, return zero else return error code.

- ***DWORD GetRecordingVideo(CString* pcsVideoName)***

Description

Getting video name on the active recoding session.

Parameters

CString* pcsVideoName	[out]The recoding video name.

Return

If successful, return zero else return error code.

- ***DWORD GetRecordInitVideo(CString* pcsVideoName)***

Description

Get the video name which is initialized successfully.

Parameters

CString* pcsVideoName	[out]The initialized video name.

Return

If successful, return zero else return error code.

- ***DWORD GetAllVideoInfo(CTimeCode* ptcFirstFrame,CTimeCode* ptcDuration,CTimeCode* ptcContinuityEnd,CTimeCode* ptcContinuityResume)***

Description

Get the encoded video info.

Parameters

CTimeCode* ptcFirstFrame	[out]Start time code of stream time.
CTimeCode* ptcDuration	[out]Video duration.
CTimeCode* ptcContinuityEnd	*[out]Default is Zero,**Note:does not support.
CTimeCode* ptcContinuityResume	*[out]Default is Zero,**Note:does not support.

Return

If successful, return zero else return error code.

- ***DWORD GetLastCompletions(DWORD& dwRecord,DWORD& dwRecordRequested, DWORD& dwCue)***

Description

Get the last completion status on recording process status, record request process status and cue process status.

Parameters

DWORD& dwPlay	[out]The last recoding status.
DWORD& dwPlayRequested	[out]The last recoding request status.
DWORD& dwCue	[out]The last cue status.

Return

If successful, return zero else return error code.

- ***DWORD ModifyRecordingVideoDuration (const CTimeCode *ptcExtendedDuration);***

Description

When a video is being recorded, this function can modify its duration.

Parameters

Const CTimeCode *ptcExtendedDuration	[in] Video duration modified

Return

If successful, return zero else return error code.

5.4 Class CScAsPlaylistMgr

***class** AFX_EXT_CLASS CScAsPlaylistMgr : **public** CScAsRpcSessionManager*

Location: ScAsPlaylistMgr.h

Description

This class is for controlling play list to support Back to Back playout.

User can make up a playlist. With the playlist, control it as a clip to cue, play, pause, goto, stop, etc.

Derivation

CScAsRpcSessionManager - Base class for providing common functionality of connecting with MediaCient server.

Members

Initialize(), Uninitialize(), SetPrePlayFrames(), GetDeviceName(), GetNextClipIndex(), GetNextScheduleTime(), GetStartTime(), GetClipInfoByClipIndex(), GetClipInfoByPos(), GetClipCount(), GetTotalDuration(), GetCurrentClipIndex(), GetCurrentStatus(), GetPortStatus(), GetBulkDeviceStatus(), GetCurrentPosition(), GetTimeRemaining(), GetTimeElapsed(), GetPlayingVideo(), IsIndexUsable(), InsertClip(), InsertClipEx(), RemoveClip(), RemoveAll(), Cue(), CueEx(), IsCued(), Play(), Pause(), Continue(), Goto(), Stop(), Skip(), GetPlaylist(), GetFirstClip(), GetNextClip(), SetClipAttribute(), GetClipAttribute(), IsListEnd().

Sample Application

Path: %Install Root%\Sample\SampleForPlaylistMgr\

5.4.1 Construction and destruction

- ***CScAsPlaylistMgr (const CString& csNodeName, const CString& csDeviceName)***

Description

Constructor function. It sets the default value to every member variable.

Parameters

const CString& csNodeName	[in]The required connecting host name of MediaClient server like MCL4012A1.
const CString& csDeviceName	[in]Decoder device name like MC_SD_DECODER1.

- ***~CScAsPlaylistMgr ()***

Description

It is a virtual destructor. The connected session and opened decoder port will be released in this function.

5.4.2 Member functions

- ***DWORD Initialize ()***

Description

Initialize resources and open decoder.

Return

If successful, return zero else return error code.

- ***DWORD Uninitialize ( )***

Description

Uninitialize to free resources and close decoder.

Return

If successful, return zero else return error code.

- ***void SetPrePlayFrames (const DWORD iFrames)***

Description

Set pre-play frames to ensure playback frame accurracy.

Parameters

const DWORD iFrame	[in]Frame accuracy, its unit is frame. Default is 1.

- ***CString GetDeviceName ()***

Description

Get the decoder device name like MC_SD_DECODERx, MC_HD_DECODEx, MC_SIMULCAST_DECODERx. MediaClient device naming rule is easily distinguished device type like SD, HD and Simulcast. (x=1, 2… n)

Return

If successfully, return device name.

- ***DWORD GetNextClipIndex (DWORD& dwNextClipIndex)***

Description

Get the index number of next clip after play list is started up..

Parameters

DWORD& dwNextClipIndex	[out]Next clip index. Zero if no next clip.

Return

If successfully, return ERROR_SUCCESS.

Remark

1) Next clip is the clip to be scheduled playing. Same concept is applied to GetNextScheduleTime.

2) Index number is to identify the unique clip in the playlist. The value must be unique. Refer to InsertClip, InsertClipEx.

- ***DWORD GetNextScheduleTime (CTimeCode* ctNextScheduleTime)***

Description

Get next schedule system time.

Parameters

CTimeCode* ctNextScheduleTime	[out]Next schedule time.

Return

If successfully, return ERROR_SUCCESS. If not, return PL_ER_UNKNOWN_TIME, and output parameter is NULL.

Remark

1) If playlist doesn't start up, output parameter "ctNextScheduleTime" will be set NULL.

2) If the event type of next clip is PL_ET_MANUAL or no next clip, output parameter "ctNextScheduleTime" will be set NULL.

3) If the event type of next clip is PL_ET_HARD, output parameter "ctNextScheduleTime" will be set hard time.

- ***BOOL GetStartTime (CTimeCode* ctStartTime)***

Description

Get started time of play list.

Parameters

CTimeCode* ctStartTime	[out]Start time of play list.

Return

TRUE if it is being played, output actual started time. If playlist doesn't start up, return FALSE.

- ***PL_CLIPINFO*GetClipInfoByIndex (const DWORD dwIndex)***

Description

Get the clip information in the play list by clip index.

Parameters

const DWORD dwIndex	[in]The clip index which the user want to get the information.

Return

If the clip index is matched in the playlist, then return the pointer to clip structure.

Remark

1) Clip information does not include next clip pointor (PL_CLIPINFO::lpNext).

2) The user must free memory for PL_CLIPINFO. For example,

```
PL_CLIPINFO* lpNode = GetClipInfoByIndex (dwIndex);
…
if (lpNode)
{
    delete lpNode;
    lpNode = NULL;
}
```

- ***PL_CLIPINFO* GetClipInfoByPos (const CTimeCode* ctPosition, CTimeCode* ctMarkIn=0)***

Description

Get the clip information by time position in the playlist and the position in this clip..

Parameters

const CTimeCode* ctPosition	[in] The position in the play list.
CTimeCode* ctMarkIn	[out] It is the position in the clip

Return

If the time position is in the range, then return the pointer to clip structure.

Remark

1) Clip information does not include next clip pointor (PL_CLIPINFO::lpNext).

2) The user must free memory for PL_CLIPINFO. For example,

```
PL_CLIPINFO* lpNode = GetClipInfoByIndex (dwIndex);
…
if (lpNode)
{
    delete lpNode;
    lpNode = NULL;
}
```

- ***DWORD GetClipCount (DWORD& dwCount)***

Description

Get the number of play list.

Parameters

DWORD& dwCount	[out]The clip counts.

Return

Return the pointer to clip info; if failed, return NULL.

- ***DWORD GetTotalDuration (CTimeCode* ctTotalDuration)***

Description

Get the total duration of the playlist.

Parameters

CTimeCode* ctTotalDuration	[out]The total duration of clips.

Return

If successful, return zero and output total duration. If not, return error code.

Remark

If there is infinite loop in the playlist, return error code..

- ***DWORD GetCurrentClipIndex (DWORD& dwClipIndex)***

Description

Get the current clip index.

Parameters

DWORD& dwClipIndex	[out]The current clip index.

Return

If successful, return zero and output current clip index. If not, return error code.

- ***DWORD GetCurrentStatus (DWORD& dwStatus)***

Description

Get the current status of the play list.

Parameters

DWORD& dwStatus	[out]Current playout status.

Return

If successful, return zero and output current status. If not, return error code.

- ***DWORD GetPortStatus (DWORD& dwPortStatus)***

Description

Get the port status.

Parameters

DWORD& dwPortStatus	[out]The port status.

Return

If successful, return zero and output port status. If not, return error code.

- ***DWORD GetBulkDeviceStatus (CTimeCode* ptcCurrentTime, CTimeCode* ptcPosition, DWORD* pdwPortStatus, DWORD* pdwSignalStatus = NULL)***

Description

Get multiple statuses, including playback position, port status and signal status.

Parameters

CTimeCode* ptcCurrentTime	[out]The MediaClient server current clock time code.
CTimeCode* ptcPosition	[out]The session playback current position.
DWORD* pdwPortStatus	[out]Port status.
DWORD* pdwSignalStatus	[out]Signal status.

Return

If successful, return zero else return error code.

- ***DWORD GetCurrentPosition (CTimeCode* dwPosition, const BOOL bIsClip=TRUE)***

Description

Get the current position in the clip or in the play list.

Parameters

CTimeCode* dwPosition	[out]The current position.
const BOOL bIsClip	[in]If TRUE, return position in clip;else return position in list.

Return

If successful, return zero else return error code.

- ***DWORD GetTimeRemaining(CTimeCode* pTimeRemaining)***

Description

Get the remain time for the decoding session

Parameters

CTimeCode* pTimeRemaining	[out]The remaining time code.

Return

If successful, return zero else return error code.

- ***DWORD GetTimeElapsed (CTimeCode* pTimeElapsed)***

Description

Get playback elapsed time of the clip.

Parameters

CTimeCode* pCurrentPosition	[out]The elapsed time code.

Return

If successful, return zero else return error code.

ReMark

This function doesn't support sub-clip now.

- ***DWORD GetPlayingVideo (CString& pcsVideoName)***

Description

Get the playing clip name.

Parameters

CString& pcsVideoName	[out]The playing video name.

Return

If successful, return zero else return error code.

- ***BOOL IsIndexUsable (DWORD dwIndex)***

Description

To check clip index user inputs. If it isn't usable or reduplicative, the function "InsertClipEx ()" will return error code.

Parameters

DWORD dwIndex	[in] the specified clip index, which needs to be checked.

Return

If successful, return TRUE else return FALSE.

- ***DWORD InsertClip (const LPCTSTR lpClipId, const CTimeCode* ctMarkIn, const CTimeCode* ctDuration, const DWORD dwBeforeClipIndex=0, const DWORD dwShiftMode=PL_ST_TAIL, const CTimeCode* ctReserved=0)***

Description

Insert an individual clip into play list.

Parameters

const LPCTSTR lpClipId	[in]Video name inserted.
const CTimeCode* ctMarkIn	[in]Markin time point to be cued. **Note: the user must input this parameter. If the user doesn't need it, please use NULL.**
const CTimeCode* ctDuration	[in]Duration to play. **Note: the user must input this parameter and it must be actual and effective. If the user doesn't need it, please use NULL.**
const DWORD dwBeforeClipIndex	[in]The index number immediately before which clip is inserted.
const DWORD dwShiftMode	[in]Shift mode.
const CTimeCode* ctReserved	[in]Reserved.

Return

If successful, return zero else return error code.

Remark

Support v45 above.

To support event type above v45, MCCAPI enhance to keep logic:

1) Clip index is created by MCCAPI automatically.

2) Event type is AUTO by MCCAPI automatically.

3) User must call GetPlaylist to query clip info within the playlist like Index, EventType, etc.

- ***DWORD InsertClipEx (const DWORD dwIndex, const LPCTSTR lpClipId, const CTimeCode* ctStartTime, const CTimeCode* ctMarkIn, const CTimeCode* ctDuration, const DWORD dwBeforeClipIndex=0, const DWORD dwShiftMode=PL_ST_TAIL, const DWORD dwEventType=PL_ET_AUTO, const DWORD dwLoopCount=0)***

Description

Insert an individual clip into play list.

It is an extended function above v45. User can set clip index, event type, and loop count.

Parameters

const DWORD dwIndex	[in]Clip index, the program will check this index whether it is usable, if it is reduplicative or not usable, then this index user inputs is wrong**.** **Note: index must be greater than zero.**
const LPCTSTR lpClipId	[in]Video name inserted.
const CTimeCode* ctStartTime	[in] start time for playing **Note: if event type is HARD, this parameter must be set. If not, please set NULL.**
const CTimeCode* ctMarkIn	[in]Markin time point to be cued. **Note: the user must input this parameter. If the user doesn't need it, please use NULL.**
const CTimeCode* ctDuration	[in]Duration to play. **Note: the user must input this parameter and it must be actual and effective. If the user doesn't need it, please use NULL.**
const DWORD dwBeforeClipIndex	[in]The index number immediately before which clip is inserted.
const DWORD dwShiftMode	[in]Shift mode. Refer to PL_SHIFT_MODE
const DWORD dwEventType	[in] user event, there are three type in play list, they are PL_ET_AUTO, PL_ET_HARD ,PL_ET_MANUAL, PL_ET_LOOPIN, PL_ET_LOOPIN_HARD, PL_ET_LOOPIN_MANUAL and PL_ET_LOOPOUT. Default value is **PL_ET_AUTO**.Refer to PL_EVENT_TYPE
const DWORD dwLoopCount	[In] valid if dwEventType= LOOPOUT, dwLoopCount must be greater than 0, 0 – infinite loop, >0 - loop count.

Return

If successful, return zero else return error code.

Remark

A playlist is a group of subset lists with event type. The subset is headed with event type of MANUAL,HARD,LOOP. The default event type is AUTO which MCCAPI will count start time according to actual play status.

For a LOOP event list, no sub loop is permitted to enter into a LOOP list. No HARD/MANUAL event in loop.

Steps to insert a LOOP list,

1. InsertClipEx a record with index, PL_ET_LOOPIN,

2. InsertClipEx a record with index, ClipName, AUTO, Duration

3. …

4. InsertClipEx a record with index, PL_EX_LOOPOUT, nLoopCount

- ***DWORD RemoveClip (const DWORD dwClipIndex)***

Description

Delete an individual clip from play list.

Clip can not be removed if the clip is playing or already played.

Parameters

const DWORD dwClipIndex	[in]The clip index.

Return

If successful, return zero else return error code.

Remark

If the clip is cued, API will cancel the cued clip, and try to find next clip. If next clip is found, API will cue next clip. If next clip is NULL, no clip will be cued. Playlist will end with current clip.

But the command must be earier than cue time for next clip, otherwise, playback will BTA.

- ***DWORD RemoveAll ()***

Description

Delete all clips from record list.

Playlist can not be removed if playlist is playing. User must call Stop before RemoveAll.

Return

If successful, return zero else return error code.

- ***DWORD Cue (CString csClipId="", const CTimeCode* ctMarkIn=0, const CTimeCode* ctMarkOut=0, const BOOL bPlaylistTime=TRUE);***

Description

Cue to prepare the clip in the playlist.

After playlist is created successfully, user must call cue to have playlist prepared.

Parameters

const CString csClipId	[in]Clip ID in play list.
const CTimeCode ctMarkIn	[in]Start time code.
const CTimeCode ctMarkOut	[in]End time code.It could be equal to=MarkIn+Duration if not zero.
const BOOL bPlaylistTime	[in]If TRUE, ctMarkIn and ctMarkOut is play list time; If FALSE, ctMarkIn and ctMarkOut is clip time.

Return

If successful, return zero else return error code.

Remark

1. Cue()

Cue the playlist at the first frame;

2. Cue("", ctMarkIn)

Cue the play list at this start time code

3. Cue("", NULL, ctMarkOut)

Cue the play list at the first frame and its end frame is ctMarkOut

4. Cue("", ctMarkIn, ctMarkOut)

Cue the play list at this start time code, and its end frame is ctMarkOut

5. Cue(csClipId)

Cue the play list at the first frame of specified clip id(csClipId)

6. Cue(ClipId, ctMarkIn)

Cue the playlist at the clipid with its duration code.

7. Cue(ClipId, ctMarkIn,ctMarkOut,FALSE)

Cue the playlist at the clipid with its MarkIn and MarkOut point. Here bPlaylistTime must be FALSE.

- ***DWORD CueEx(const DWORD dwClipIndex, const CTimeCode* ctMarkIn=0, const CTimeCode* ctMarkOut=0, const BOOL bPlaylistTime=TRUE);***

Description

Cue to prepare the index-matched clip in the playlist.MarkIn and Duration info must be defined when user InsertClipEx.

After playlist is created successfully, user must call cue to have playlist prepared.

Parameters

const CString dwClipIndex	[in]Clip index in play list.

Return

If successful, return zero else return error code.

- ***BOOL IsCued()***

Description

Determine whether it is cued ready for playing.

Return

If it is ready, return TRUE; otherwise return FALSE.

- ***DWORD Play ()***

Description

Trigger to start playing the playlist from cued clip.

Return

If successful, return zero else return error code.

- ***DWORD Pause ()***

Description

Pause current playing session.

Return

If successful, return zero else return error code.

- ***DWORD Continue ()***

Description

Resume current playing session.

Return

If successful, return zero else return error code.

- ***DWORD Goto (const CTimeCode* ptcStartTime, BOOL bIsPlayTime)*****

Description

Go to specified position.

Parameters

const CTimeCode* ptcStartTime	[in]The specified goto time code.
BOOL bIsPlayTime	[in]If TRUE, goto the play list timeline position; If FALSE, goto current clip stream timeline position.

Return

If successful, return zero else return error code.

Remark

If goto operation is issued, playout will pause until the time position is cued. So pause time is depended on cuing time.

If the time position is out of range, return error.

After going to the time position, user should call Play() to go on.

Note: The paused time position to next clip should be longer than precue time with the exception that the next clip is the last clip. Otherwise, system will not achieve back to back play when Play() is called next time.

*Goto () function is not supported in this version.

- ***DWORD Stop ()***

Description

Stop playing session.

Return

If successful, return zero else return error code.

- ***DWORD Skip ()***

Description

Skip and stop playing current clip, and execute operation on next clip. This operation will be made according to the event type of next clip.

If the event type of next clip is AUTO, system can play immediately.

If the event type of next clip is MANUAL or LOOP_MANUAL, system can't play until user call PlayEx.

If the event type of next clip is HARD or LOOP_HARD, system can't play until hard time is coming.

If there is no next clip, the system will end playback.

If there is next clip in a loop, the system will skip out current looping list.

Return

If successful, return zero else return error code.

Remark

Support above v45.

- ***DWORD SetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, DWORD dwValue)***
- ***DWORD SetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, CString csValue)***
- ***DWORD SetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, CTimeCode* ctValue)***

Description

Set clip attribute according to clip index and attribute type.

Parameter

DWORD dwIndex	[in] clip index specified
DWORD dwAttributeType	[in] attribute type specified, it is included PL_ATTRIBUTE_TYPE_INDEX, PL_ATTRIBUTE_TYPE_ID, PL_ATTRIBUTE_TYPE_PATH, PL_ATTRIBUTE_TYPE_MARKIN, PL_ATTRIBUTE_TYPE_DURATION, PL_ATTRIBUTE_TYPE_START_TIME, PL_ATTRIBUTE_TYPE_EVENT_TYPE, PL_ATTRIBUTE_TYPE_STATUS, PL_ATTRIBUTE_TYPE_LOOP_COUNT
DWORD dwValue	[in] attribute value, field type is DWORD
CString csValue	[in] attribute value, field type is CString

CTimeCode* ctValue	[in] attribute value, field type is CTimeCode

Return

If successful, return zero else return error code.

Remark

Support above v45.

- ***DWORD GetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, DWORD& dwValue)***

- ***DWORD GetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, CString& csValue)***

- ***DWORD GetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, CTimeCode* ctValue)***

Description

Get clip attribute according to clip index and attribute type.

Parameter

DWORD dwIndex	[in] clip index specified
DWORD dwAttributeType	[in] attribute type specified, it is included PL_ATTRIBUTE_TYPE_INDEX, PL_ATTRIBUTE_TYPE_ID, PL_ATTRIBUTE_TYPE_PATH, PL_ATTRIBUTE_TYPE_MARKIN, PL_ATTRIBUTE_TYPE_DURATION, PL_ATTRIBUTE_TYPE_START_TIME, PL_ATTRIBUTE_TYPE_EVENT_TYPE, PL_ATTRIBUTE_TYPE_STATUS, PL_ATTRIBUTE_TYPE_LOOP_COUNT
DWORD& dwValue	[out] attribute value, field type is DWORD
CString& csValue	[out] attribute value, field type is CString
CTimeCode* ctValue	[out] attribute value, field type is CTimeCode

Return

If successful, return zero else return error code.

Remark

Support above v45.

- ***DWORD GetPlaylist (PList* pPlaylist)***

Description

Get the whole play list.

Parameter

PList* pPlaylist	[out] play list

Definition

```
typedef std::list<PL_CLIPINFO> PList;
```

Return

If successful, return zero else return error code.

Remark

Support above v45.

- ***PL_CLIPINFO* GetFirstClip ()***

Description

Get the first clip in play list.

Return

If successful, returns the first clip else return NULL.

Remark

Support above v45.

1) Clip information does not include next clip pointor (PL_CLIPINFO::lpNext).

2) The user must free memory for PL_CLIPINFO. For example,

PL_CLIPINFO lpNode = GetClipInfoByIndex (dwIndex);*

…

```
if (lpNode)
{
    delete lpNode;
    lpNode = NULL;
}
```

- ***PL_CLIPINFO* GetNextClip ()***

Description

Get the next clip in play list.

Return

If successful, returns the next clip else return NULL.

Remark

Support above v45.

1) Clip information does not include next clip pointor (PL_CLIPINFO::lpNext).
2) The user must free memory for PL_CLIPINFO. For example,

```
PL_CLIPINFO* lpNode = GetClipInfoByIndex (dwIndex);
…
if (lpNode)
{
    delete lpNode;
    lpNode = NULL;
}
```

- ***BOOL IsListEnd ()***

Description

Judge whether play list is finished.

Return

If successful, returns TRUE.

5.5 Class CScAsRecordlistMgr

***class** AFX_EXT_CLASS CScAsRecordlistMgr: **public** CScAsRpcSessionManager*

Location: ScAsRecordlistMgr.h

Description

It is class for control recording list to support back to back record. To support back to back record, MCL encoder must set "Enable B2B encoding" to TRUE. Thus user can encode the recordlist like a clip by RecordInit, Record, Stop, etc.

Derivation

CScAsRpcSessionManager - Base class for providing common functionality of connecting with MediaCient server.

Members

Initialize(), Uninitialize(), SetPrePlayFrames(), GetDeviceName(), GetNextClipIndex(), GetNextScheduleTime(), GetStartTime(), GetClipInfoByClipIndex(), GetClipInfoByPos(), GetClipCount(), GetTotalDuration(), GetCurrentClipIndex(), GetCurrentStatus(), GetPortStatus(), GetBulkDeviceStatus(), GetCurrentPosition(), GetTimeElapsed(),InsertClip(), InsertClipEx(), RemoveClip(), RemoveAll(), RecordInit(), Record(), Stop(), Abort(), SetClipAttribute(), GetClipAttribute(), GetFirstClip(), GetNextClip(), IsListEnd().

5.5.1 Construction and destruction

- ***CScAsRecordlistMgr (const CString& csNodeName, const CString& csDeviceName)***

Description

It is constructor function. It sets the default value to every member variable.

Parameters

const CString& csNodeName	[in]The required connecting host name of MediaClient server.
const CString& csDeviceName	[in]Encoder device name.

- ***~CScAsRecordlistMgr ()***

Description

Virtual destructor. The connected session and opened encoder port will be released in this function.

5.5.2 Member functions

- ***DWORD Initialize()***

Description

Initialize resource and encoder for record list.

Return

If successful, return zero else return error code.

- ***DWORD Uninitialize( )***

Description

Uninitialize play list.

Return

If successful, return zero else return error code.

- ***void SetPreRecorderFrames(const DWORD dwFrames)***

Description

Set the frame accuracy.

Parameters

const int iFrame	[in]Set preplay frames to ensure frame accuracy. Its unit is frame.

- ***CString GetDeviceName ()***

Description

Get the decoder device name like MC_SD_DECODERx, MC_HD_DECODEx, MC_SIMULCAST_DECODERx. Due to the MediaClient device naming rule, it is easy to distinguish device type like SD, HD and Simulcast from each other. (x=1, 2… n)

Return

If successfully, return device name.

- ***DWORD GetNextClipIndex(DWORD& dwNextClipIndex)***

Description

Get the index of next clip.

Parameters

DWORD& dwNextClipIndex	[out]Next clip index.

Return

If successful, return zero else return error code.

- ***DWORD GetNextScheduleTime(CTimeCode& ctNextScheduleTime)***

Description

Get next schedule time.

Parameters

CTimeCode& ctNextScheduleTime	[out]Next schedule time.

Return

If successful, return zero else return error code.

- ***DWORD GetStartTime(CTimeCode* ctStartTime)***

Description

Get the start time of play list.

Parameters

CTimeCode* ctStartTime	[out]Start time of play list.

Return

If successful, return zero else return error code.

- ***PL_CLIPINFO*GetClipInfoByClipIndex(const DWORD dwClipIndex)***

Description

Get the clip information in the play list by clip index. Refer to CLIPINFO.

Parameters

const DWORD dwClipIndex	[in]The clip index, which the user want to get the information.

Return

If the clip index is within this list, then return the pointer to clip info.

- ***PL_CLIPINFO* GetClipInfoByPos(const CTimeCode* ctPosition)***

Description

Get the clip information by list position. Refer to CLIPINFO.

Parameters

const CTimeCode *ctPosition	[in]The position in the play list.

Return

If the position is in the range, then return the pointer to clip info.

- ***DWORD GetClipCount(DWORD& dwCount)***

Description

Get clip count in the list.

Parameters

DWORD& dwCount	[out]The number of clips.

Return

If successful, return zero else return error code.

- ***DWORD GetTotalDuration(CTimeCode* ctTotalDuration)***

Description

Get the total duration of clips.

Parameters

CTimeCode* ctTotalDuration	[out]The total duration of clips.

Return

If successful, return zero else return error code.

- ***DWORD GetCurrentClipIndex(DWORD& dwClipIndex)***

Description

Get the index of current clip.

Return

If successful, return zero else return error code.

Parameters

DWORD& dwClipIndex	[out]The current clip index.

- ***DWORD GetCurrentStatus(DWORD& dwStatus)***

Description

Get the current status of list.

Return

If successful, return zero else return error code.

Parameters

DWORD& dwStatus	[out]The current status.

- ***DWORD GetPortStatus(DWORD& dwPortStatus)***

Description

Get the port status.

Return

If successful, return zero else return error code.

Parameters

DWORD& dwPortStatus	[out]The port status.

- ***DWORD GetBulkDeviceStatus(CTimeCode* ptcCurrentTime,CTimeCode* ptcPosition,DWORD* pdwPortStatus,DWORD* pdwSignalStatus = NULL)***

Description

Get all of status in current encoder device, including current recording position, port status and signal status.

Parameters

CTimeCode* ptcCurrentTime	[out]The MediaClient server current clock time code
CTimeCode* ptcPosition	[out]The session playback current position
DWORD* pdwPortStatus	[out]The status

DWORD* pdwSignalStatus	[out]Signal status

Return

If successful, return zero else return error code.

- ***DWORD GetCurrentPosition(CTimeCode* dwPosition,const BOOL bIsClip=TRUE)***

Description

Get the current position in the clip or list.

Parameters

CTimeCode* dwPosition	[out]The current position.
const BOOL bIsClip	[in]If TRUE, return position in clip; If FALSE, return position in list.

Return

If successful, return zero else return error code.

- ***DWORD GetTimeElapsed(CTimeCode* pElapsed)***

Description

Get recording elapsed time.

Parameters

CTimeCode* pElapsed	[out]The session elapsed time.

Return

If successful, return zero else return error code.

- ***DWORD InsertClip(const LPCTSTR lpClipId,const CTimeCode* ctMarkIn=0,const CTimeCode* ctDuration=0,const DWORD dwBeforeClipIndex=0,const DWORD dwShiftMode=PL_ST_TAIL,const CTimeCode* ctReserved=0)***

Description

Add a clip to record list.

Parameters

const LPCTSTR lpClipId	[in]The video name inserted.
const CTimeCode* ctMarkIn	[in]SOM. If ctMarkIn == 0, record will not set clip SOM; It depends on EXD user profile setting as “Use Start TC

	value for SOM".
const CTimeCode* ctDuration	[in]Clip duration. If duration is zero, recording will be stopped until Stop command gets called. Otherwise it will not stop.
const DWORD dwBeforeClipIndex	[in]The index number immediately before which clip is inserted.
const DWORD dwShiftMode	[in]Shift mode.
const CTimeCode* ctReserved	[in]Reserved.

Return

If successful, return zero else return error code.

Remark

Refer to Playlist.

- ***DWORD InsertClipEx (const DWORD dwIndex, const LPCTSTR lpClipId, const CTimeCode* ctStartTime=0, const CTimeCode* ctMarkIn=0, const CTimeCode* ctDuration=0, const DWORD dwBeforeClipIndex=0, const DWORD dwShiftMode=PL_ST_TAIL, const DWORD dwEventType=PL_ET_AUTO, const DWORD dwLoopCount=0)***

Description

Insert an individual clip to record list.

It is an extended function. User can set clip index, event type, and loop count.

Parameters

const DWORD dwIndex	[in]Clip index
const LPCTSTR lpClipId	[in]Video name inserted.
const CTimeCode* ctStartTime	[in] start time for playing
const CTimeCode* ctMarkIn	[in]Markin time point to be cued.
const CTimeCode* ctDuration	[in]Duration to play.
const DWORD dwBeforeClipIndex	[in]The index number immediately before which clip is inserted.
const DWORD dwShiftMode	[in]Shift mode. Refer to PL_SHIFT_MODE
const DWORD dwEventType	[in] user event, there are three type in play list, they are PL_ET_AUTO, PL_ET_HARD and PL_ET_MANUAL. Default value is **PL_ET_AUTO**.Refer to

	PL_EVENT_TYPE
const DWORD dwLoopCount	[In] valid if dwEventType= LOOPOUT, dwLoopCount must be greater than 0, 0 – infinite loop, >0 - loop count.

Return

If successful, return zero else return error code.

Remark

Support above v45. Refer to Playlist.

- ***DWORD RemoveClip(const DWORD dwClipIndex)***

Description

Delete an individual clip from play list.

Parameters

const LPCTSTR *lpClipId*	[in]The first clip name in the list.
const DWORD dwClipIndex	[in]The clip index.

Return

If successful, return zero else return error code.

- ***DWORD RemoveAll()***

Description

Delete all clips from record list.

Return

If successful, return zero else return error code.

- ***DWORD RecordInit()***

Description

Prepare clip for record list.

Return

If successful, return zero else return error code.

User can set precue time before recording. It is to ensure clip is ready for recording.

Remark

1) RecordInit()

RecordInit recordlist at the first frame

2) RecordInit("", duration)

RecordInit recordlist at the first frame with a duration

3) RecordInit("", markin, duration)

RecordInit recordlist at the first frame with a duration and start time code

4) RecordInit(ClipId)

RecordInit recordlist at the first frame of specified clip ID

5) RecordInit(ClipId, duration)

RecordInit recordlist at the first frame of specified clip ID with a duration

6) RecordInit(ClipId, markin, duration)

RecordInit recordlist at the first frame of specified clip ID with a duration and start time code

- ***DWORD Record()***

Description

Start Recording.

Return

If successful, return zero else return error code.

- ***DWORD Stop(const CTimeCode *ptcStopTime=0)***

Description

Stop recording.

Parameters

const CTimeCode *ptcStopTime	[in]The delay time code to execute stop command.

Return

If successful, return zero else return error code.

- ***DWORD Abort()****

Description

Abort recording.

Return

If successful, return zero else return error code.

Remark

If Abort command gets called, recording will be stopped and recorded clip will be deleted from storage.

*Abort() function is not supported in this version.

- ***DWORD SetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, DWORD dwValue)***

- ***DWORD SetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, CString csValue)***

- ***DWORD SetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, CTimeCode* ctValue)***

- ***DWORD SetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, SD_BRIEF_PRIVATE_DATA strValue)***

Description

Set clip attribute specified.

Parameter

DWORD dwIndex	[in] clip index specified
DWORD dwAttributeType	[in] attribute type specified, it is included PL_ATTRIBUTE_TYPE_INDEX, PL_ATTRIBUTE_TYPE_ID, PL_ATTRIBUTE_TYPE_PATH, PL_ATTRIBUTE_TYPE_MARKIN, PL_ATTRIBUTE_TYPE_DURATION, PL_ATTRIBUTE_TYPE_START_TIME, PL_ATTRIBUTE_TYPE_EVENT_TYPE, PL_ATTRIBUTE_TYPE_STATUS, PL_ATTRIBUTE_TYPE_PD, PL_ATTRIBUTE_TYPE_LOOP_COUNT
DWORD dwValue	[in] attribute value, field type is DWORD
CString csValue	[in] attribute value, field type is CString

CTimeCode* ctValue	[in] attribute value, field type is CTimeCode
SD_BRIEF_PRIVATE_DATA* strValue	[in] attribute value, field type is SD_BRIEF_PRIVATE_DATA

Return

If successful, return zero else return error code.

Remark

Support above v45.

- ***DWORD GetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, DWORD& dwValue)***

- ***DWORD GetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, CString& csValue)***

- ***DWORD GetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, CTimeCode* ctValue)***

- ***DWORD GetClipAttribute (DWORD dwIndex, DWORD dwAttributeType, SD_BRIEF_PRIVATE_DATA* strValue)***

Description

Get clip attribute specified.

Parameter

DWORD dwIndex	[in] clip index specified
DWORD dwAttributeType	[in] attribute type specified, it is included PL_ATTRIBUTE_TYPE_INDEX, PL_ATTRIBUTE_TYPE_ID, PL_ATTRIBUTE_TYPE_PATH, PL_ATTRIBUTE_TYPE_MARKIN, PL_ATTRIBUTE_TYPE_DURATION, PL_ATTRIBUTE_TYPE_START_TIME, PL_ATTRIBUTE_TYPE_EVENT_TYPE, PL_ATTRIBUTE_TYPE_STATUS, PL_ATTRIBUTE_TYPE_PD,

	PL_ATTRIBUTE_TYPE_LOOP_COUNT
DWORD& dwValue	[out] attribute value, field type is DWORD
CString& csValue	[out] attribute value, field type is CString
CTimeCode* ctValue	[out] attribute value, field type is CTimeCode
SD_BRIEF_PRIVATE_DATA* strValue	[out] attribute value, field type is SD_BRIEF_PRIVATE_DATA

Return

If successful, return zero else return error code.

Remark

Support above v45.

- ***DWORD GetRecordlist (`PList* pRecordlist`)***

Description

Get the whole record list.

Parameter

PList* pRecordlist	[out] record list

Definition

```
typedef std::list<PL_CLIPINFO> PList;
```

Return

If successful, return zero else return error code.

- ***PL_CLIPINFO* GetFirstClip ()***

Description

Get the first clip in play list.

Return

If successful, returns the first clip else return NULL.

Remark

Support above v45.

1) Clip information does not include next clip pointor (PL_CLIPINFO::lpNext).

2) The user must free memory for PL_CLIPINFO. For example,

```
PL_CLIPINFO* lpNode = GetClipInfoByIndex (dwIndex);
…
if (lpNode)
{
    delete lpNode;
    lpNode = NULL;
}
```

- ***PL_CLIPINFO* GetNextClip ()***

Description

Get the next clip in play list.

Return

If successful, returns the next clip else return NULL.

Remark

Support above v45.

1) Clip information does not include next clip pointor (PL_CLIPINFO::lpNext).

2) The user must free memory for PL_CLIPINFO. For example,

```
PL_CLIPINFO* lpNode = GetClipInfoByIndex (dwIndex);
…
if (lpNode)
{
    delete lpNode;
    lpNode = NULL;
}
```

- ***BOOL IsListEnd ()***

Description

Judge whether record list is finished.

Return

If successful, returns TRUE.

5.6 Class CScAsSystemStatus

***class** AFX_EXT_CLASS CScAsSystemStatus: **public** CScAsRpcSessionManager*

Location: ScAsSystemStatus.h

Description

Class for getting system info, version info and the status of each device from MediaClient server;

Derivation

CScAsRpcSessionManager - Base class for provided common functionality for connecting with MediaCient server.

Members

GetStatusHeadPosition(), GetStatusNext(), GetVersionHeadPosition(), GetVersionNext(), GetStatusInfoHeadPosition(), GetStatusInfoNext(), GetStatusInfoByType(), IsArchiveEnabled(), IsAutoArchive(), EnumerateDevices(), GetPeerNodeNames(), ForceDecoderReset(), ForceDecoderRelease(), ForceEncoderReset(), ForceEncoderRelease(), SignalEventReset(), SignalEventRelease(), SignalEventForceRelease(), SignalEventAquire(), ResetAllCounters(), GetDecoderDeviceCount(), GetEncoderDeviceCount(), GetBulkDeviceStatus(), GetDeviceStatusLabels(), GetDeviceVersionInfoList(), GetDiskStatus(), GetSubsystemStatus(), GetSignalFullStatus(), GetLocalNodeVideoStandard()

Sample Application

Path: %Install Root%\Sample\ExdApiSampleApp\

5.6.1 Construction and destruction

- ***CScAsSystemStatus(const CString& csNodeName)***

Description

Constructor function. It sets the default value to every member variable.

Parameters

const CString& csNodeName	[in]The required connecting host name of MediaClient server.

- ***virtual ~CScAsSystemStatus()***

Description

Virtual destructor, release connection session and other resources.

5.6.2 Member functions

- ***POSITION GetStatusHeadPosition()***

Description

Getting the head position of the device status array, the device status array is created by function GetBulkDeviceStatus(), and it should be called first.

Return

The position of the device status array head.

Remark

POSITION A value used to denote the position of an element in a collection; used by MFC collection classes.

- ***CDeviceStatus* GetStatusNext(POSITION& pos)***

Description

Get the next element status from the device status list, mostly making use of this function is combine with GetStatusHeadPosition().

Parameters

POSITION& pos	[in]Next position.

Return

A point to CDeviceStatus object

Remark

POSITION A value used to denote the position of an element in a collection; used by MFC collection classes.

- ***POSITION GetVersionHeadPosition()***

Description

Getting the head position of the version info array, the version info array is created by function GetDeviceVersionInfoList(), and preserve all of hardware and software version info in current connected MeidaClient server.

Return

The position of the version info array head.

Remark

POSITION A value used to denote the position of an element in a collection; used by MFC collection classes.

- ***CDeviceVersion* GetVersionNext(POSITION& pos)***

Description

Call this method to return the next element from the Version info list, mostly making use of this function is combine with GetVersionHeadPosition().

Parameters

POSITION& pos	[in]Next position.

Return

The point of required Device Status, the point data type is class CDeviceVersion.

Remark

POSITION A value used to denote the position of an element in a collection; used by MFC collection classes.

- ***POSITION GetStatusInfoHeadPosition()***

Description

Getting the head position from the Status label Info list, the Status label Info list is created by function GetDeviceStatusLabels(), and it should be called first.

Return

The position of the status info label's array head.

- ***CDeviceStatusInfo* GetStatusInfoNext(POSITION& pos)***

Description

Get next device status by position, mostly making use of this function is combine with GetStatusInfoHeadPosition().

Parameters

POSITION& pos	[out]Next position.

Return

The point of required device status, the point data type is class CDeviceStatusInfo.

Remark

POSITION A value used to denote the position of an element in a collection; used by MFC collection classes.

- ***CDeviceStatusInfo* GetStatusInfoByType(const DWORD dwInfoType)***

Description

Get device status by specified device type. The status label info list is created by function GetDeviceStatusLabels(), and it should be called first.

Parameters

const DWORD dwInfoType	[in]The device type,defined by enum RPC_DEVICE_TYPES.

Return

The point of required device status label info, the point data type is class CDeviceStatusInfo.

- ***BOOL IsArchiveEnabled()***

Description

Judge whether the archive function is enabled in MediaClient server, by this function user can check whether the Archive Component is available, the Archive Component should be create Archive service and enable first in the MediaCleint configuration table.

Return

True if the archive function is enabled. False if the archive function is disabled.

Remark

This function is only valid after calling Connect() successfully.

- ***BOOL IsAutoArchive()***

Description

Judge whether the auto archive function is enabled in MediaClient server.

Return

True if the archive function is enabled. False if the archive function is disabled.

- ***DWORD EnumerateDevices()***

Description

Get device list.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling Connect() successfully.

- ***DWORD SignalEventReset(const CString& csDeviceName,const DWORD dwDeviceType)***

Description

Reset signal event of device.

Parameters

const CString& csDeviceName	[in]The device name. Refer to GetStatusHeadPosition() & GetStatusNext().
const DWORD dwDeviceType	[in]The device type,Refer to enum RPC_DEVICE_TYPES.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling Connect() successfully.

- ***DWORD SignalEventRelease(const CString& csDeviceName,const DWORD dwDeviceType)***

Description

This operation is only supported for encoder device and decoder device. The function is to close codec device connecting session and releases the controlling power from the current master, and set the “Lock Holder” value to “unlocked”.

Parameters

const CString& csDeviceName	[in]The device name.Refer to GetStatusHeadPosition() & GetStatusNext().
const DWORD dwDeviceType	[in]The device type,Refer to RPC_DEVICE_TYPES.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling Connect() successfully.

- ***DWORD SignalEventAquire(const CString& csDeviceName,const DWORD dwDeviceType)***

Description

Send the signal to event for notify MediaClient server, the specified device request to acquire some essential resource (for example, VDCP automation need to acquire the com port if there are no COM Port resource assigned for it.)

Parameters

const CString& csDeviceName	[in]The device name.Refer to GetStatusHeadPosition() & GetStatusNext().
const DWORD dwDeviceType	[in]The device type. Refer to enum RPC_DEVICE_TYPES.

Return

If successful, return zero else return error code.

- ***DWORD ResetAllCounters()***

Description

Reset all of devices counters, by this function will reset every status counter to zero in each device.

Return

If successful, return zero else return error code.

- ***DWORD GetDecoderDeviceCount()***

Description

Get decoder device counts, This function is only valid after calling Connect()& EnumerateDevices() successfully.

Return

If successful, return zero else return error code.

- ***DWORD GetEncoderDeviceCount()***

Description

Get encoder device counts, same as decoder.

Return

If successful, return zero else return error code.

- ***DWORD GetBulkDeviceStatus()***

Description

Get all of devices status. This function is only valid after calling Connect() successfully.

Return

If successful, return zero else return error code.

- ***DWORD GetDeviceStatusLabels()***

Description

Get the status label text for every type device. This function is only valid after calling Connect() successfully.

Return

If successful, return zero else return error code.

- ***DWORD GetDeviceVersionInfoList()***

Description

Get the system version info list, including the relative HW & SW. This function is only valid after calling Connect() successfully.

Return

If successful, return zero else return error code.

- ***DWORD GetLocalNodeVideoStandard()***

Description

Get the current system video standard (NTSC or PAL).This function is only valid after calling Connect() successfully.

Return

If failed, return RPC_SERVICE_ERROR(0xFFFF). If successfully, return 0-3.
0: PL_VF_NTSC_DF
1: PL_VF_NTSC_NDF
2: PL_VF_PAL
3: PL_VF_PAL_NDF.

5.7 Class CScAsCache

class AFX_EXT_CLASS CScAsCache: public CScAsRpcSessionManager

Location: ScAsCache.h

Description

Class for manager cache elements,to get media file list and metadata info for every media file from the MediaClient server.

Main functions are OpenCache()& GetIdList(),so that the user can get the Media file info by function GetNext().

Derivation

CScAsRpcSessionManager - Base class for provided common functionality for connecting with MediaCient server.

Members

OpenCache(), GetIdList(), IdRequest(), GetStorageTimeRemaining(), GetIdCount(), GetIdsRemaining(), GetNext(), RenameId(), GetFileDeleteProtect(), GetFileWriteProtect(), GetFileReadOnly(), GetFileType(), SetFileDuration(), GetFileDuration(), CreateFile(), OpenFile(), WriteFile(), ReadFile(), InitFilePrivateData(), GetFilePrivateData(), GetFilePrivateData(), CloseFile(), DeleteFile(), GetFileSize(), SetFilePrivateData(), SetFileDeleteProtect(), SetFileWriteProtect(), SetFileReadOnly(), SetFileType(), SetAFDAspectRatio(), CreateSubClip()

Sample Application

Path: %Install Root%\Sample\SampleForCache\

5.7.1 Construction and destruction

- ***CScAsCache(const CString& csNodeName)***

Description

Construct CScAsCache object.

Parameters

const CString&csNodeName	[in]The required connecting host name of MediaClient server.

- ***virtual ~CScAsCache()***

Description

Virtual destructor. The connected session and opened Cache Manager Device will be released in this function.

5.7.2 Member functions

- ***DWORD OpenCache (HANDLEhStatusNotifyEvent=INVALID_HANDLE_VALUE)***

Description

Open cache manager device.

Parameters

HANDLE hStatusNotifyEvent	[in]Input parameter event handle.Set it as INVALID_HANDLE_VALUE by default. *Note: does not support.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling Connect() successfully.

- **GetIdList()**

- ***DWORD GetIdList (void)***

Description

Get media file list from video storage. All of the clip operations should be done after this function calling successfully.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling OpenCache() successfully.

- ***DWORD IdRequest (const CString& csVideoId)***

Description

Find a cache element in the cache list by specified video name.

Parameters

const CString& csVideoId	[in]The requested clip name.

Return

ERROR_SUCCESS if the clip name is existed in cache file list, else ERROR_FILE_NOT_FOUND.

- ***DWORD GetStorageTimeRemaining(CTimeCode* ptcTimeRemaining)***

Description

Get the video storage remaining free time.

Parameters

CTimeCode* ptcTimeRemaining	[out]A point for CTimeCode object to preserve remaining time.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

- ***DWORD GetStorageTimeRemaining(DWORD* pdwMinutes)***

Description

Get the video storage remaining free time.

Parameters

DWORD* pdwMinutes	[out]A point of DWROD type value to preserve remaining minutes.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

- ***DWORD GetIdCount(DWORD* pdwIdCount)***

Description

Get the media file counts in video storage.

Parameters

DWORD* pdwIdCount	[out]A point for DWORD value to preserve media file counts.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

- ***DWORD GetIdsRemaining(DWORD* pdwIdsRemainingCount)***

Description

Get the number of media file remaining

Parameters

DWORD* pdwIdsRemainingCount	[out]A point for DWORD value to preserve media file counts.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

- ***DWORD GetNext (CCacheElement*& pVid)***

Description

Get the next element CCacheElement object in Cache element list.

Parameters

CCacheElement *& pVid	[out]A pointer of CCacheElement object.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

Before calling next GetNext(), user must free CCacheElement class pointer. For example,

```
CCacheElement* pVid = NULL;
while(m_pScAsCache->GetNext(pVid))
{
```

```
        m_sVideoName = pVid->GetId(); // Free CCacheElement* class pointer
        if(pVid)
        {
                delete pVid;
                pVid = NULL;
        }

}
```

- ***DWORD RenameId (const CString& csVideoIdOld, const CString& csVideoIdNew)***

Description

Rename clip name in video storage.

Parameters

const CString& csVideoIdOld	[in]Old file name.
const CString& csVideoIdNew	[in]New file name.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

- ***DWORD GetFileDeleteProtect(const CString & csFileName, LPBOOL lpbDeleteProtect)***

Description

Indicates whether deleting protection is set.

Parameters

const CString & csFileName	[in]Requested file name.
LPBOOL lpbDeleteProtect	[out]A point of BOOL type value which indicated the media file whether set as delete-protected.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

- ***DWORD SetFileDeleteProtect(const CString & csFileName,BOOL bDeleteProtect)***

Description

Set delete protection flag for the specified media file, if the media file deleting protection attribute is true, means that the media file could not to be deleted. This media file attribute is preserved in .pd file which is the attachment file relative with the media file

Parameters

const CString & csFileName	[in]A CString type modified file name.
BOOL bDeleteProtect	[in]A BOOL type value which indicated the media file whether set as Delete-protect.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

- ***DWORD GetFileWriteProtect(const CString & csFileName, LPBOOL lpbWriteProtect)***

Description

Indicates whether the write protection is set

Parameters

const CString & csFileName	[in]A CString type requested file name.
LPBOOL lpbWriteProtect	[out]A point of BOOL type value, indicated the media file whether set as write only.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

- ***DWORD SetFileWriteProtect(const CString & csFileName, BOOL bWriteProtect)***

Description

Set write protection attribute to the Media file.

Parameters

const CString &csFileName	[in]A CString type modified file name.
BOOL bWriteProtect	[in]A BOOL type value which indicated the media file whether set as write-only.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

- ***DWORD GetFileReadOnly(const CString & csFileName,LPBOOL lpbReadOnly)***

Description

Indicates whether read only protection is set.

Parameters

const CString &csFileName	[in]A CString type requested file name
LPBOOL lpbReadOnly	[out]A point of BOOL type value which indicated the media file whether set as read-only.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

- ***DWORD SetFileReadOnly(const CString & csFileName,BOOL bReadOnly)***

Description

Set read only protection attribute for the media file.

Parameters

const CString csFileName	[in]A CString type modified file name.
BOOL bReadOnly	[in]A BOOL type value, indicated the media file whether set as read-only.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling GetIdList() successfully.

- ***DWORD GetFileType(const CString& csFileName, DWORD* pdwVideoType)***

Description

Get media file video standard (PAL, NTSC etc.).

Parameters

const CString & csFileName	[in]A CString type requested file name.
DWORD* pdwVideoType	[out]A point of DWORD type value, indicated the media file video standard.

Return

If successful, return zero else return error code.

- ***DWORD SetFileType(const CString & csFileName,DWORD dwVideoType)***

Description

Set the media file video standard (PAL, NTSC,etc.).

Parameters

const CString &csFileName	[in]A CString type requested file name.
DWORD dwVideoType	[in]A DWORD type value, indicated the media file video standard.

Return

If successful, return zero else return error code.

- ***DWORD SetFileDuration(const CString & csFileName,CTimeCode & tcDuration)***

Description

Modify the media file duration, this value is preserved in PD file which is relative with the media file.

Parameters

const CString &csFileName	[in]A CString type requested file name.
CTimeCode &tcDuration	[in]The media file duration.

Return

If successful, return zero else return error code.

- ***DWORD GetFileDuration(const CString& csFileName,CTimeCode* ptcDuration)***

Description

Get the media file duration, this value is preserved in pd file which is relative with the media file.

Parameters

const CString& csFileName	[in]A CString type requested file name.
CTimeCode* tcDuration	[out]A CTimeCode object, preserved the media file duration.

Return

If successful, return zero else return error code.

- ***DWORD CreateFile(unsigned long* pulFileHandle,const CString& csFileName,DWORD dwFlags,DWORD dwFork,CCacheElement* poCacheElement=NULL)***

Description

Create a new file to video storage. The file data is specified by parameter poCacheElement, if NULL, create a empty file.

Parameters

unsigned long* pulFileHandle	[out]The file handle.
const CString& csFileName	[in]A CString type requested file name.
DWORD dwFlags	[in]The file creating attribute.Refer to WIN32_FIND_DATA structure.
DWORD dwFork	[in]File type specified. Refer to enum file fork.
CCacheElement* poCacheElement	[in]A CCacheElement object,preserved the file data. If the value is NULL, creating a empty file.

Return

If successful, return zero else return error code.

- ***DWORD OpenFile (unsigned long* pulFileHandle, const CString &csFileName, DWORD dwAccessMode, DWORD dwFileFork)***

Description

Open a file from video storage.

Parameters

unsigned long* pulFileHandle	[out]The file handle.
const CString csFileName	[in]A CString type requested file name.
DWORD dwAccessMode	[in]File access mode. Refer to File Access Mode.

DWORD dwFork	[in]File type specified.Refer to enum file fork.

Return

If successful, return zero else return error code.

- ***DWORD WriteFile(unsigned long ulFileHandle,unsigned char* pBuffer,DWORD dwBytesToWrite,DWORD* pdwBytesWritten)***

Description

Write data to an opened or created file in Video storage.

Parameters

unsigned long ulFileHandle	[in]File handle, obtained by function OpenFile() or CreateFile().
unsigned char* pBuffer	[in]Buffer of writing data.
DWORD dwBytesToWrite	[in]Byte counts for the buffer size.
DWORD* pdwBytesWritten	[out]Written data size.

Return

If successful, return zero else return error code.

- ***DWORD ReadFile(unsigned long ulFileHandle,unsigned char* pBuffer,DWORD dwBytesToRead,DWORD* pdwBytesRead)***

Description

Read data from an opened or created file in Video storage.

Parameters

unsigned long ulFileHandle	[in]The file handle , obtained by function OpenFile() or CreateFile().
unsigned char*pBuffer	[out]Buffer of reading data.
DWORD dwBytesToRead	[in]Byte counts for the buffer size.
DWORD* pdwBytesRead	[out]Data size which have read.

Return

If successful, return zero else return error code.

- ***DWORD InitFilePrivateData(unsigned long ulFileHandle,unsigned long ulType,const CTimeCode* poDuration,BOOL bReadOnly)***

- ***DWORD InitFilePrivateData(LPCTSTR lpszFileName,unsigned long ulType,const TimeCode* poDuration,BOOL bReadOnly)***

Description

Initial private data (.pd) file with specified parameters.

Parameters

unsigned long ulFileHandle	[in]The file handle, obtained by function OpenFile() or CreateFile().
unsigned long ulType	[in]A ULONG type value, indicated the video standard.
const CTimeCode* poDuration	[in]A CTimeCode object, indicated the duration.
BOOL bReadOnly	[in]A BOOL type value, set the Read only attribute.
LPCTSTR lpszFileName	[in]A LPCSTR type text, set the PD file name.

Return

If successful, return zero else return error code.

Description

Initial Private Data (.pd) file with specified parameters.

- ***DWORD GetFilePrivateData (unsigned long ulFileHandle, unsigned long* pulType, CTimeCode* poDuration, LPBOOL lpbReadOnly)***

- ***DWORD GetFilePrivateData (LPCTSTR lpszFileName, unsigned long* pulType, CTimeCode* poDuration, LPBOOL lpbReadOnly)***

- ***DWORD GetFilePrivateData (unsigned long ulFileHandle, CBriefPrivateData& rBriefPrivateData)***

- ***DWORD GetFilePrivateData (LPCTSTR lpszFileName, CBriefPrivateData& rBriefPrivateData)***

Description

Get the Private Data information.

Parameters

unsigned long ulFileHandle	[in]The file handle, obtained by function OpenFile() or CreateFile().
unsigned long* pulType	[out]A ULONG type value which indicated the video

	standard.
CTimeCode* poDuration	[out]A CTimeCode object which indicated the duration.
LPBOOL bReadOnly	[out]A BOOL type value to set the read only attribute.
LPCTSTR lpszFileName	[in]A LPCSTR type text to set the pd file name.
CBriefPrivateData& rBriefPrivateData	[out]A CBriefPrivateData object to preserve the private data information.

Return

If successful, return zero else return error code.

- ***DWORD CloseFile(unsigned long ulFileHandle)***

Description

Close the opened or created file handle.

Parameters

unsigned long ulFileHandle	[in]The file handle, obtained by function OpenFile() or CreateFile().

Return

If successful, return zero else return error code.

- ***DWORD DeleteFile (const CString& csVideoId)***

Description

Delete specified media file.

Parameters

const CString& csVideoId	[in]A CString type value which indicates the file name to be deleted.

Return

If successful, return zero else return error code.

- ***DWORD GetFileSize(const CString &csVideoId,DWORD* pdwFileSizeHi,DWORD* pdwFileSizeLow)***

Description

Retrieve the size of a specified file.

Parameters

const CString csVideoId	[in]A CString type value which indicates the file name.

DWORD* pdwFileSizeHi	[out]Pointer to the file size high-order 32 bits.
DWORD* pdwFileSizeLow	[out]Pointer to the file size low-order 32 bits.

Return

If successful, return zero else return error code.

- ***DWORD SetFilePrivateData (unsigned long ulFileHandle, CBriefPrivateData& rBriefPrivateData)***

- ***DWORD SetFilePrivateData (LPCTSTR lpszFileName, CBriefPrivateData& rBriefPrivateData)***

Description

Set the Private Data information.

Parameters

unsigned long ulFileHandle	[in]The file handle, obtained by function OpenFile() or CreateFile().
LPCTSTR lpszFileName	[in]A LPCSTR type text to set the pd file name.
CBriefPrivateData& rBriefPrivateData	[in]A CBriefPrivateData object to preserve the private data information.

Return

If successful, return zero else return error code.

- ***DWORD SetAFDAspectRatio(const CString& csFileName,BOOL bRatioValid,BYTE bySourceAspectRatio,BYTE byScaleAspectRatio)***

Description

AFD is abbreviation of “Active Format Description”. This function is to insert AFD information into PD file.

On encoder device, When encoder to load and create to record a new clip,writing the clip native Aspect Ratio info to PD file, and record the source signal AFD data to the destination clip as user specified VBI line number. User also could specify the simulcast decoder playback scaling method and EXD will record this value to PD file.

On decoder device, user could specify the AFD data outputting VBI line, for simulcast, decoder should specified both SD and HD AFD outputting line. The scaling aspect ratio info could be read from PD file and passed this info to simulcast decoder, so that the simulcast decoder could scale the displaying mode for SD->HD up convert or HD->SD down convert.

Parameters

const CString& csFileName	[in]A string type, it specifies video file name.
BOOL bRatioValid	[in]A Boolean type, if TURE, it means this operation is valid; if FALSE, this function is unavailable.

BYTE bySourceAspectRatio	[in]A BYTE type, it specifies aspect ratio of source data. It is included 4:3, 14:9, 16:9 and asymmetrical. Their values is 0, 1, 2, 3 in turn.
BYTE byScaleAspectRatio	[in]A BYTE type, it specifies aspect ratio of scale data. About its value, reference to bySourceAspectRatio.

Return

If successful, return zero else return error code.

- ***DWORD CreateSubClip(const CString& csMasterClip,const CString& csSubClip,const CTimeCode* poStart,const CTimeCode* poDuration)***

Description

Create virtual subclip based on master clip.

Parameters

const CString& csMasterClip	[in]Master clip name.
const CString& csSubClip	[in]Subclip name.
const CTimeCode* poStart	[in]SOM of subclip.
const CTimeCode* poDuration	[in]Duration of subclip.

Return

If successful, return zero else return error code.

5.8 Class CScAsArchive

***class** AFX_EXT_CLASS CScAsArchive : **public** CScAsRpcSessionManager*

Location: ScAsArchive.h

Description

The Archive Manager component is associated with input and output file to and from the archive service.

Remark

The archive function is available if archive function is disabled in MediaClient server or there is no archive service available.

Derivation

CScAsRpcSessionManager - Base class for provided common functionality for connecting with MediaCient server.

Members

OpenArchive(), GetIdList(), GetRequestList(), GetNextRequest(), IsInArchive(), GetArchiveCopyStatus(), GetStorageTimeRemaining(), Copy(), Abort(), ReCue(), Modify(), ReadArchiveServices(), GetArchiveServicesList(), GetDefaultService(), DeleteId(), RenameId(), GetFileDuration()

Sample Application

Path: %Install Root%\Sample\ExdApiSampleApp\

5.8.1 Construction and destruction

- ***CScAsArchive(const CString& csNodeName)***

Description

Construct CScAsArchive object.

Parameters

const CString& csNodeName	[in]The required connecting host name of MediaClient server.

- ***virtual ~CScAsArchive()***

Description

Virtual destructor. The connected session and opened Archive Manager Device will be released in this function.

5.8.2 Member functions

- ***DWORD OpenArchive(HANDLE hStatusNotifyEvent=INVALID_HANDLE_VALUE)***

Description

Send command to MediaClient server for open Archive Manager Device, so that user can control resource to the CScAsArchive object.

Parameters

HANDLE hStatusNotifyEvent	[in]Input parameter event handle,by this event handle MediaCient server could be notify application when the device status is change. *Note:current system does not support this parameter.set it INVALID_HANDLE_VALUE by default.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling Connect() successfully.

- ***DWORD GetRequestList()***

Description

This function is just implement get the CRequestArchive object list.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling OpenArchive() successfully.

- ***BOOL GetNextRequest(CRequestArchive*& pRequestArchive)***

Description

Get next CRequestArchive object from the CRequestArchive object list.

Parameters

CRequestArchive*& pRequestArchive	[out]The requested pointer to the CRequestArchive object.

Return

If have next request info, return TRUE, otherwise return FALSE.

Remark

This function is only valid after calling OpenArchive() successfully.

- ***DWORD IsInArchive(const CString& csService,const CString& csVideoId)***

Description

Indicates whether the specified Media file is in archive request queue

Parameters

const CString& csService	[in]The archive service name which is preserved in CArchiveService object.
const CString& csVideoId	[in]A CString value to specified the searching media file name.

Return

The archive request status, this value is defined in enum Archive Request Status

Remark

This function is only valid after calling OpenArchive() successfully.

- ***DWORD GetArchiveCopyStatus(const unsigned long ulCopyHandle,const SYSTEMTIME stStartTime,unsigned long* pulState,LARGE_INTEGER* lpliLastCheckpointBytes,LARGE_INTEGER* lpliTotalBytes,LARGE_INTEGER* lpliPercentCompleted,double* dBandwidth)***

Description

Get the copy request status by specified copy handle.

Parameters

const unsigned long ulCopyHandle	[in]A handle of copy session, returned by a previous Copy().
const SYSTEMTIME stStartTime	[in]Start time when the file begins to copy.
unsigned long* pulState	[out]The request state defined as enum Archive Request

	states.
LARGE_INTEGER* lpliLastCheckpointBytes	[out]A point to LARGE_INTERGER to preserve the current copy bytes.
LARGE_INTEGER* lpliTotalBytes	[out]A point to LARGE_INTERGER to preserve the total copy bytes.
LARGE_INTEGER*lpliPercentCompleted	[out]A point to LARGE_INTERGER to preserver the current copy progress.
double* dBandwidth	[out]A point to double to preserve the average bandwidth.

Return

The archive request status, this value is defined in enum Archive Request Status

Remark

This function is only valid after calling OpenArchive() successfully.

- ***DWORD Copy (unsigned long* pulCopyHandle, const CString & csSrcServiceName, const CString& csDestServiceName, const CString & csSrcFileName, const CString& csDestFileName, const BOOL bIsMoveOperation)***

Description

Create a copy session to copy the file to destination archive services.

Parameters

unsigned long* ulCopyHandle	[out]A handle of copy session.
const CString& csSrcServiceName	[in]The source archive service name.
const CString& csDestServiceName	[in]The destination archive service name.
const CString& csSrcFileName	[in]The source file name.
const CString& csDestFileName	[in]The destination file name.
const BOOL bIsMoveOperation	[in]If TRUE, it executes move operation. If FALSE, it executes copy operation.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling OpenArchive() successfully.

- ***DWORD Abort(unsigned long ulCopyHandle)***

Description

Abort the specified copy sessions

Parameters

unsigned long ulCopyHandle	[in]A handle of copy session, returned by a previous Copy().

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling OpenArchive() successfully.

- ***DWORD ReCue(unsigned long ulCopyHandle)***

Description

Recue the copy session, which maybe aborted in previous.

Parameters

unsigned long ulCopyHandle	[in]A handle of copy session, returned by a previous Copy()

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling OpenArchive() successfully.

- ***DWORD Modify(unsigned long ulCopyHandle,DWORD dwPriority)***

Description

Modify the copy session priority

Parameters

unsigned long ulCopyHandle	[in]A handle of copy session, returned by a previous Copy().
DWORD dwPriority	[in]Priorify value.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling OpenArchive() successfully.

- ***DWORD ReadArchiveServices()***

Description

Get the archive services list from MediaClient server.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling OpenArchive() successfully.

- ***CArchiveServiceList& GetArchiveServicesList()***

Description

Get the archive services list, this services list is returned by a previous ReadArchiveServices().

Return

A CArchiveServiceList type reference;

Remark

This function is only valid after calling OpenArchive() successfully.

- ***DWORD GetDefaultService(CString* pcsServiceName,CString* pcsServicePath,DWORD* pdwServiceType)***

Description

Get the default archive service from the service list, returned by a previous GetArchiveServicesList()

Parameters

CString* pcsServiceName	[out]A CString value to preserve the default archive service name.
CString* pcsServicePath	[out]The default archive service path list.
DWORD* pdwServiceType	[out]The default archive service type.

Return

If successful, return zero else return error code.

Remark

This function is only valid after calling ReadArchiveServices() successfully.

- ***DWORD DeleteId(const CString& csService,const CString& csVideoId)***

Description

Delete a media file from the specified archive service.

Parameters

const CString&csService	[in]The specified archive service name, the archive service name is returned by a previous GetArchiveServicesList().
const CString&csVideoId	[in]The requested media file name.

Return

If successful, return zero else return error code.

- ***DWORD RenameId(const CString& csService,const CString& csVideoIdOld,const CString& csVideoIdNew)***

Description

Rename a media file from the specified Archive service.

Parameters

const CString&csService	[in]The specified archive service name
const CString&csVideoIdOld	[in]The source Media file name
const CString&csVideoIdNew	[in]The destination Media file name

Return

If successful, return zero else return error code.

- ***DWORD GetFileDuration(const CString& csService,const CString& csFileName,CTimeCode* ptcDuration)***

Description

Get Media file duration from the specified Archive service.

Parameters

const CString& csService	[in]The specified archive service name.
const CString& csFileName	[in]The requested media file name.

CTimeCode* ptcDuration	[out]A pointer to the CTimeCode object, to preserve the duration.

Return

If successful, return zero else return error code.

Appendix A: Include Files

The following “Include” files are provided in the SDK package:

CacheElement.h

CachePrivateData.h

DeviceState.h

DeviceVersion.h

PrivateData.h

Resource.h

ScAsArchive.h

ScAsCache.h

ScAsDecoder.h

ScAsRecorder.h

ScAsPlaylistMgr.h

ScAsRecordlistMgr.h

ScAsRpc.h

ScAsRpcIntf.h

ScAsRpcNodeConnectionThread.h

ScAsRpcSessionMgr.h

ScAsRpcShared.h

ScAsRpc_C_Wrapper.h

ScAsSystemStatus.h

ScAsUtility.h

StreamApi.h

TimeCode.h

VstrmConstants.h

VstrmTypes.h

ScAsCommon.h

ScAsSingleLinkedBase.h

Appendix B: Sample Applications

The following sample applications are provided in the SDK package:

ExdApiSampleApp

SampleForCache

SampleForArchive

SampleForDecoder

SampleForRecord

SampleForPlaylist

SampleForRecordlist

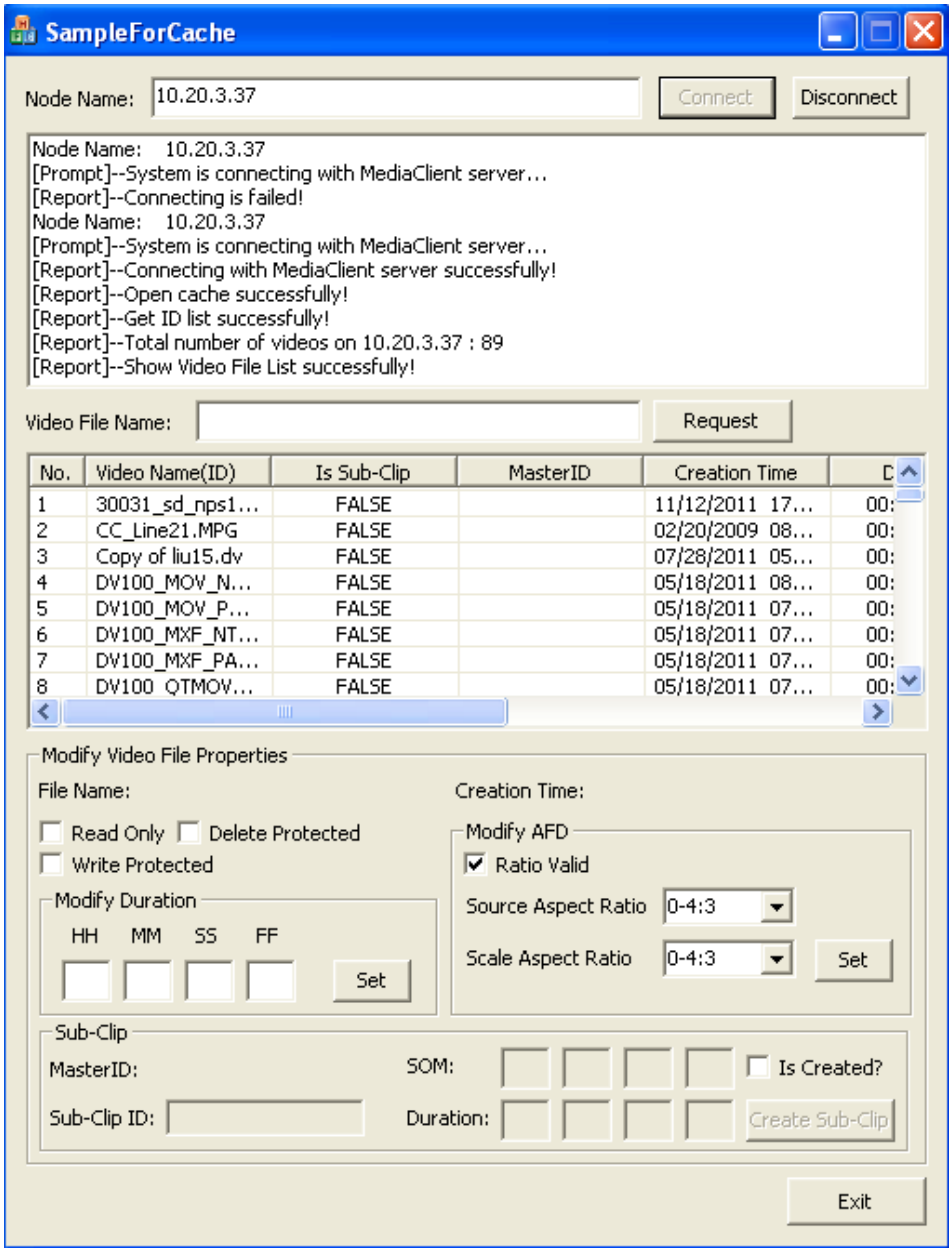
SampleForCache

Introduction

This **SampleForCache** application is used for decoder functions by MCC SDK. The application is a simple UI based program.

Main components are CScAsCache.

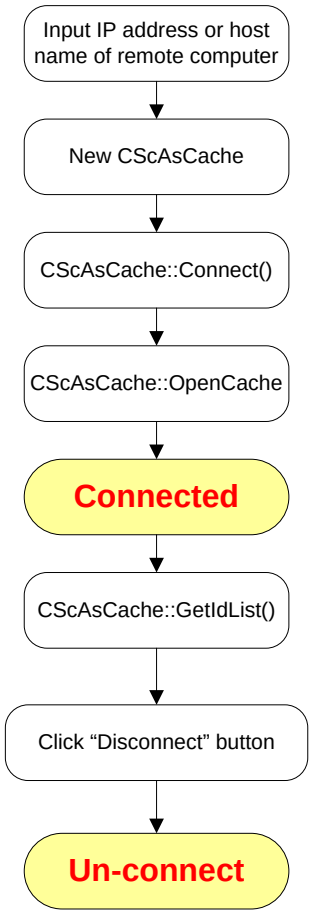
UI and Workflow



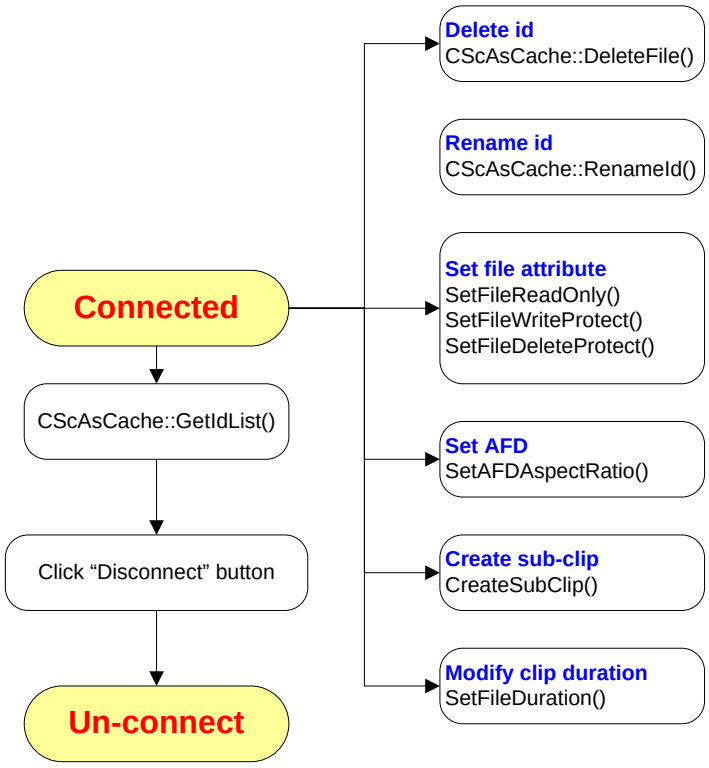
Step 1: Connect Cahce

Input node name or IP address, and click “Connect” button, user will connect node and get id list which are shown in list control box.

User clicks “Disconnect” button before user exits application, which application disconnects with EXD service.



Step 2: Clip operation



After application connects with EXD service, user can delete id, rename id, modify id attributes, set AFD value, create sub-clip and modify clip duration.

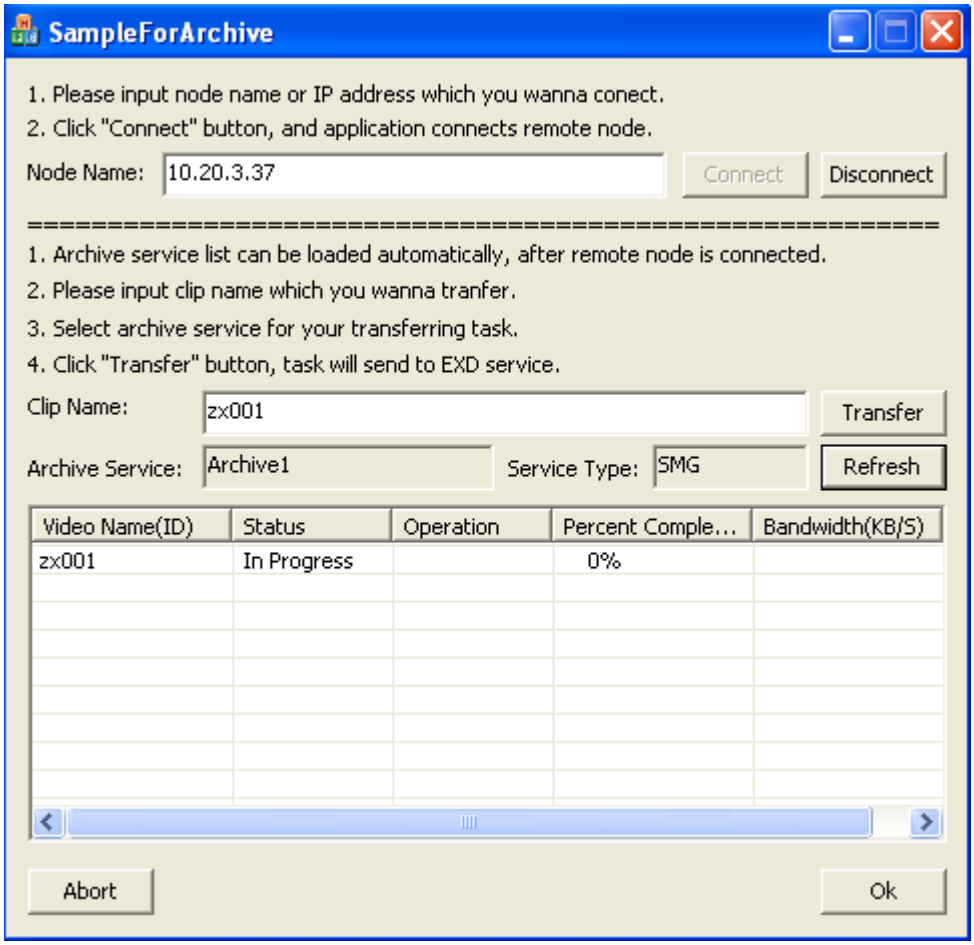
SampleForArchive

Introduction

This **SampleForArchive** application is used for decoder functions by MCC SDK. The application is a simple UI based program.

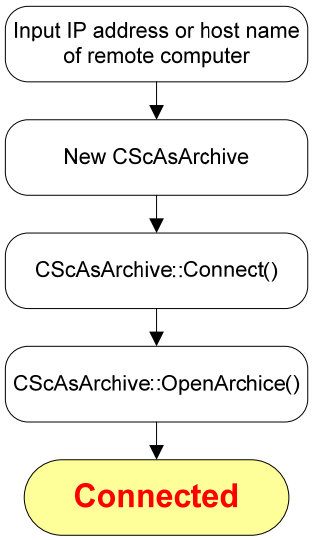
Main components are CScAsArchive.

UI and Workflow

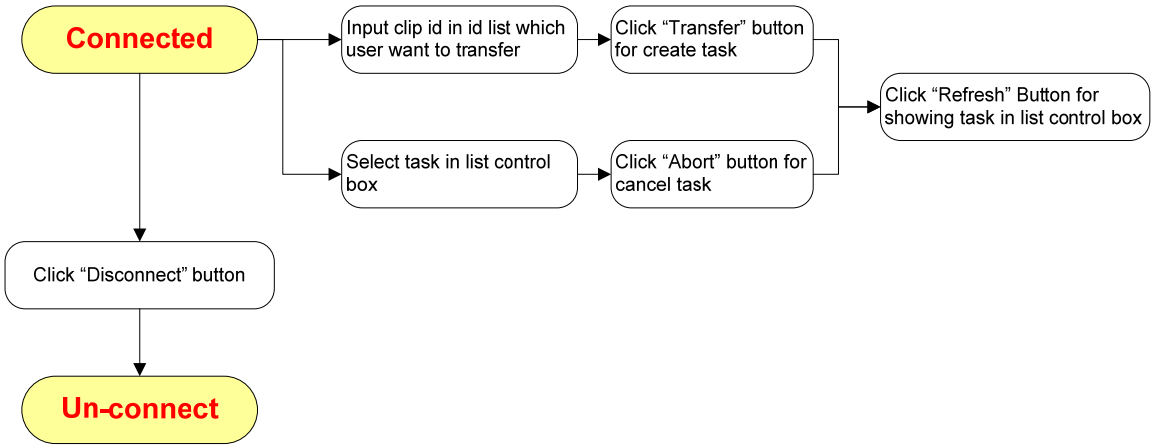


Step 1: Connect Archive

Input node name or IP address, and click “Connect” button, user will connect with EXD service.



Step 2: Archive operation



Clip id user wants to transfer must exist in the id list of source targer. If clip id doesn’t exist, application creating task will not success.

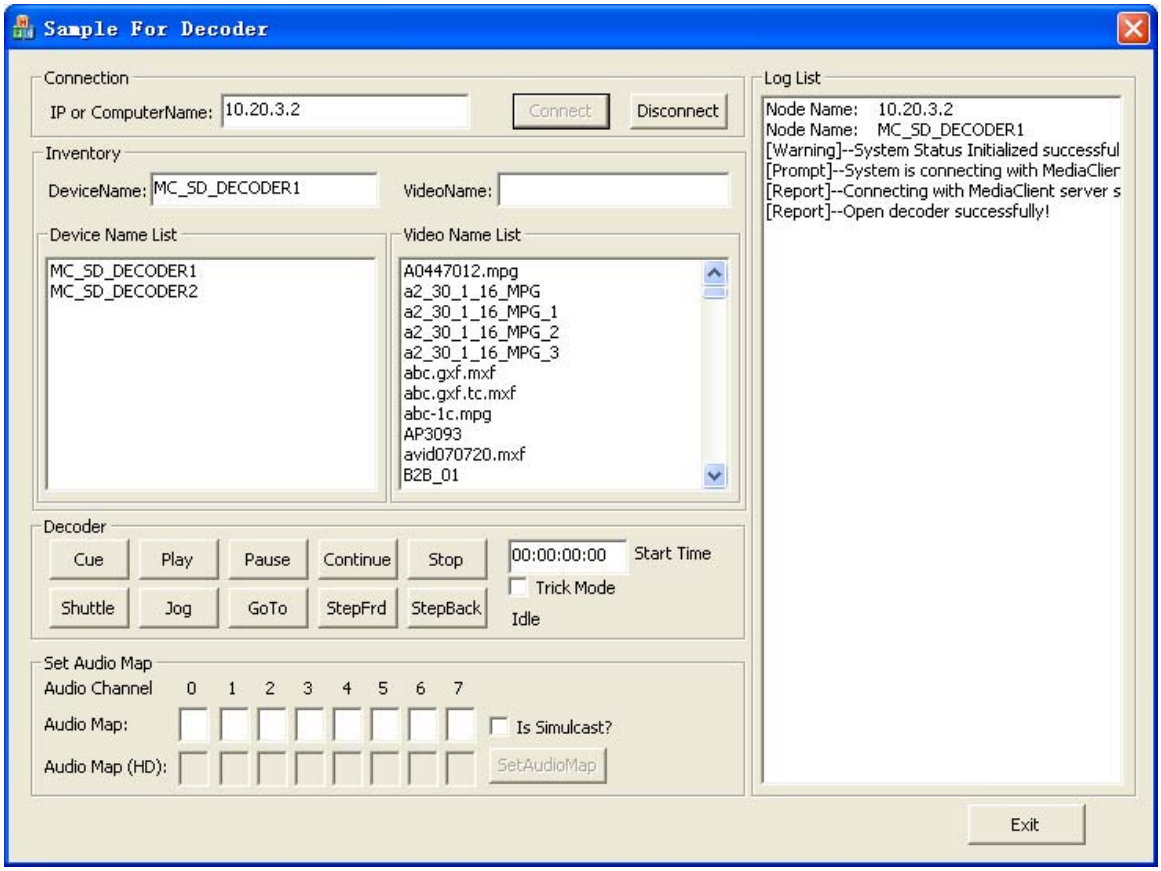
SampleForDecoder

Introduction

This **SampleForDecoder** application is used for decoder functions by MCC SDK. The application is a simple UI based program.

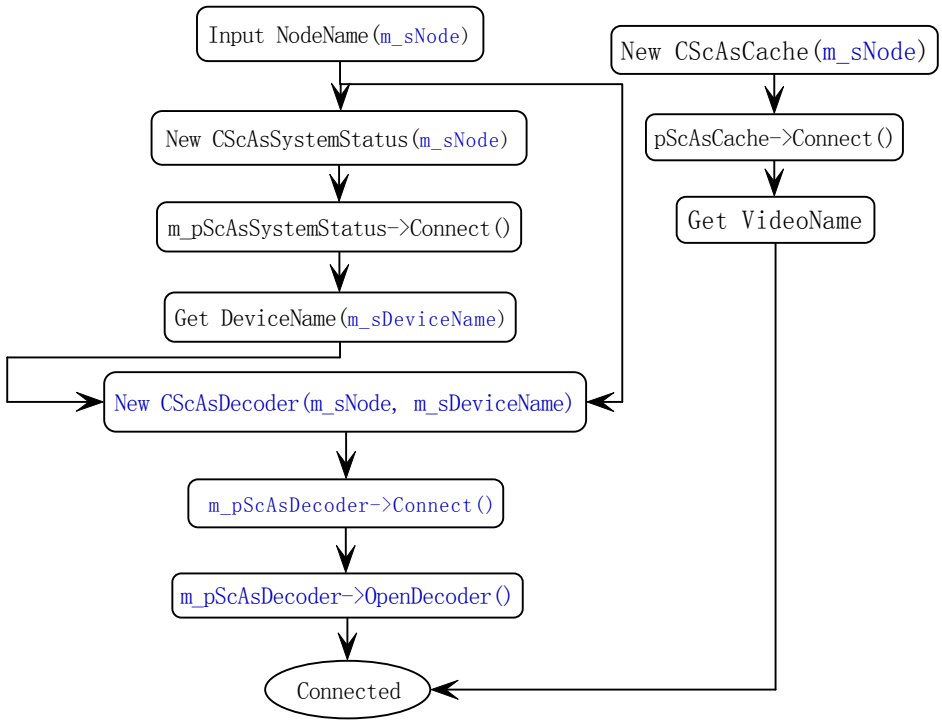
Main components are CScAsSystemStatus, CScAsCache, CScAsDecoder.

UI and Workflow



Step1: Connect Device Decoder

Input MCL Node Name or IP Address, and Click ‘Connect’, you will get Decoder Device Name List and Video Name List.

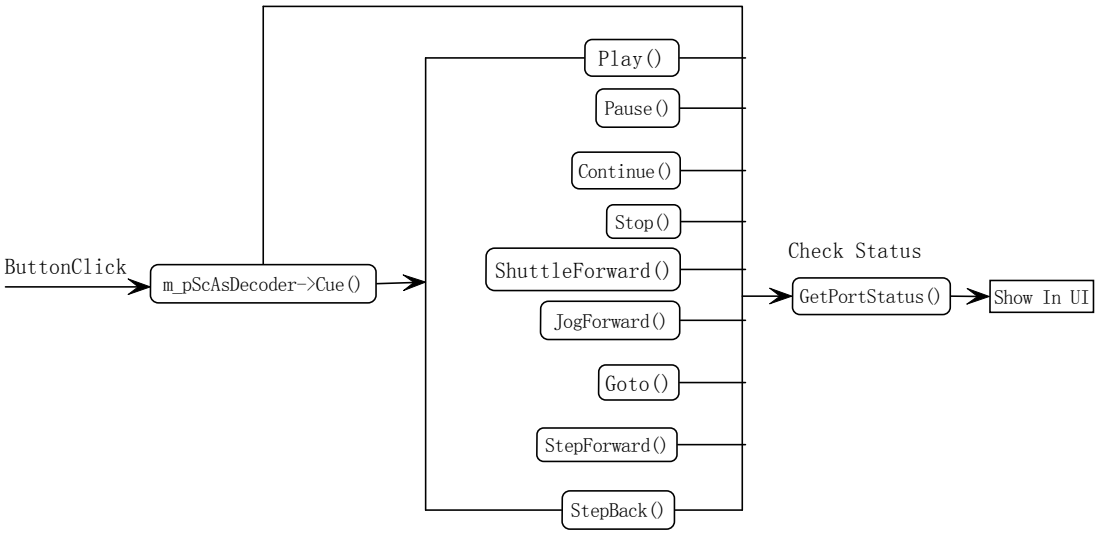


Step2: Control Decoder

Select a device name and video name, you can control decoder for Cue,Play,Pause,…,etc.

UI will display device status by calling GetPortStatus().

Set Trick Mode TRUE or FALSE by clicking checkbox “Trick Mode”.



*Note: *Set Audio Map before ‘Cue’, Refer to SetAudioMap.*

Step3: Disconnect Decoder

Before exiting, you must click “Disconnect” by calling CloseDecoder() and release resources.

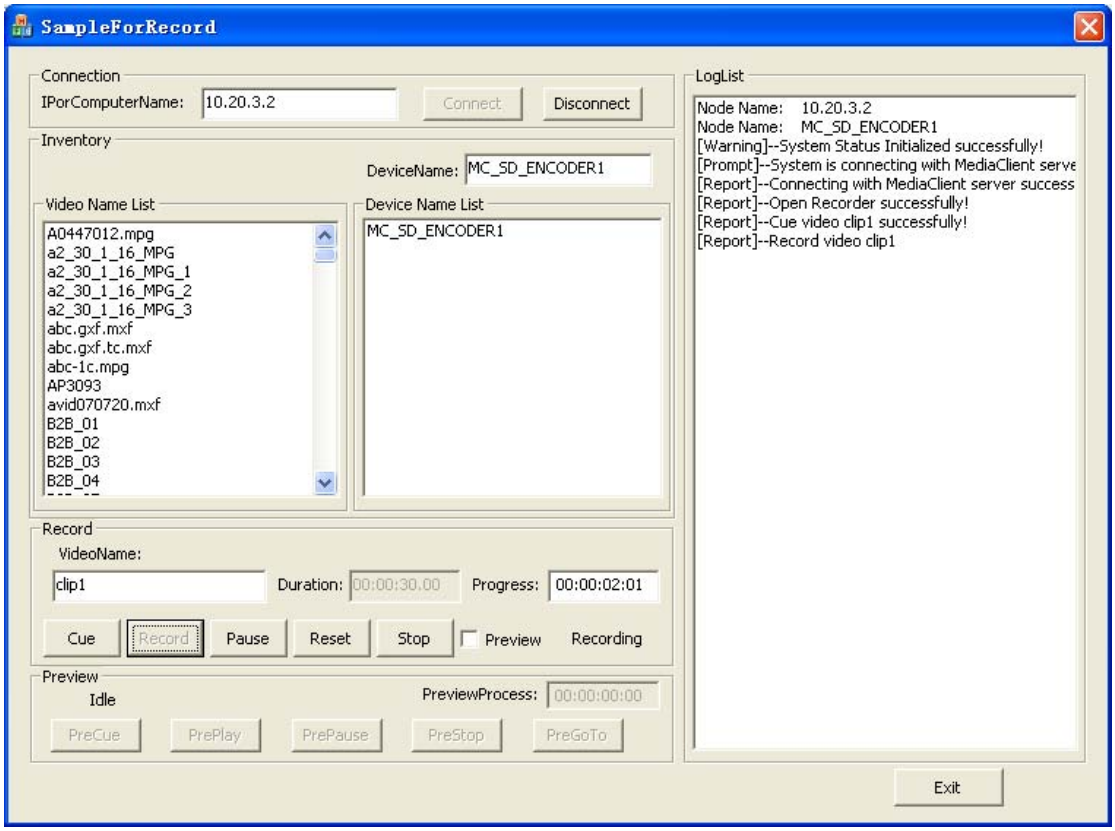
SampleForRecord

Introduction

This **SampleForRecord** application is used for encoder functions by MCC SDK. The application is a simple UI based program.

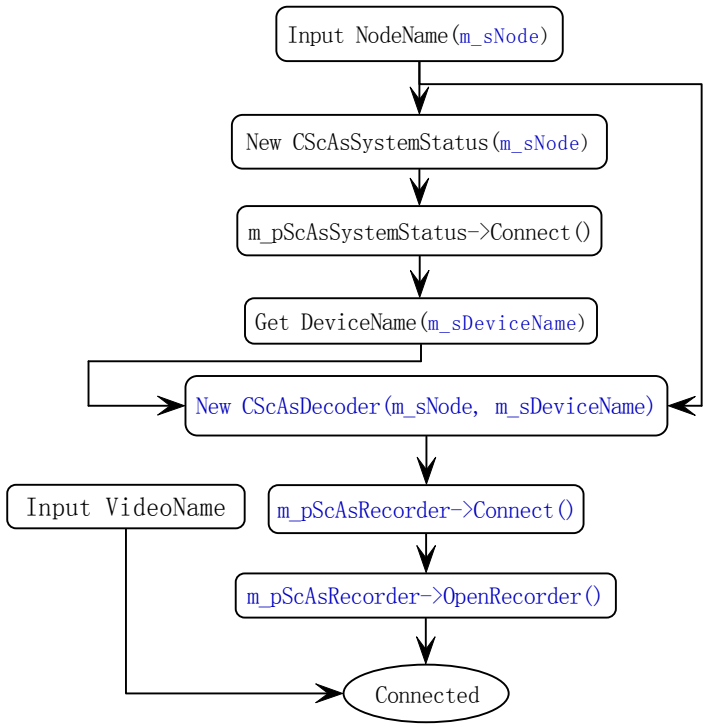
Main components are CScAsSystemStatus, CScAsCache, CScAsRecorder.

UI and Workflow



Step1: Connect Device Encoder

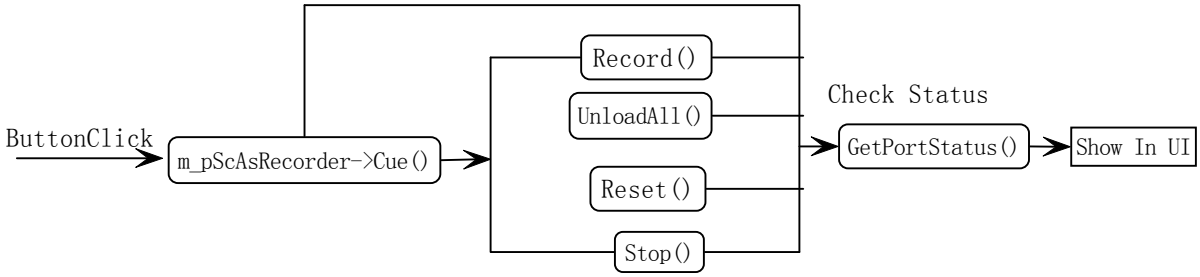
Input MCL Node Name or IP Address, and Click ‘Connect’, you will get Encoder Device Name List and Video Name List.



Step2: Control Encoder

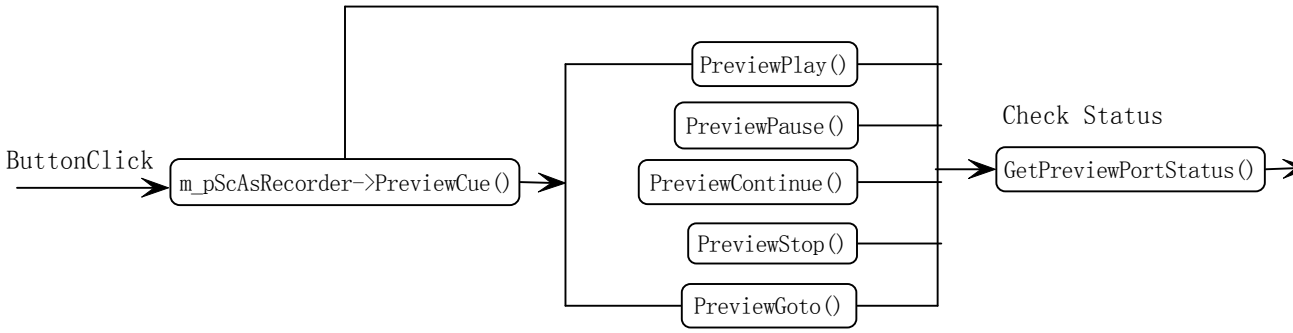
Select a device name and input video name, duration, you can control decoder for Cue, Record, Pause, etc.

UI will display device status by calling GetPortStatus()



Step3: Preview While Recoding

You must select “Preview” to enable the function.



Step4: Disconnect Encoder

Before exiting, you must click “Disconnect” by calling CloseRecorder() and release resources.

SampleForPlaylistMgr

Introduction

This **SampleForPlaylistMgr** application is used for playlist functions by MCC SDK. The application is a simple UI based program.

Main components are CScAsSystemStatus, CScAsCache, CScAsPlaylistMgr.

Playlist calls in sequence should be:

pScAsPlaylistMgr->Connect() // Connect decoder for preparing

pScAsPlaylistMgr->Initialize() //Initialize resource after new instance

pScAsPlaylistMgr->SetPrePlayFrames() //Set pre frames to control playing frame accuracy

pScAsPlaylistMgr-> InsertClip()//Insert clips into playlist, also must insert one by one

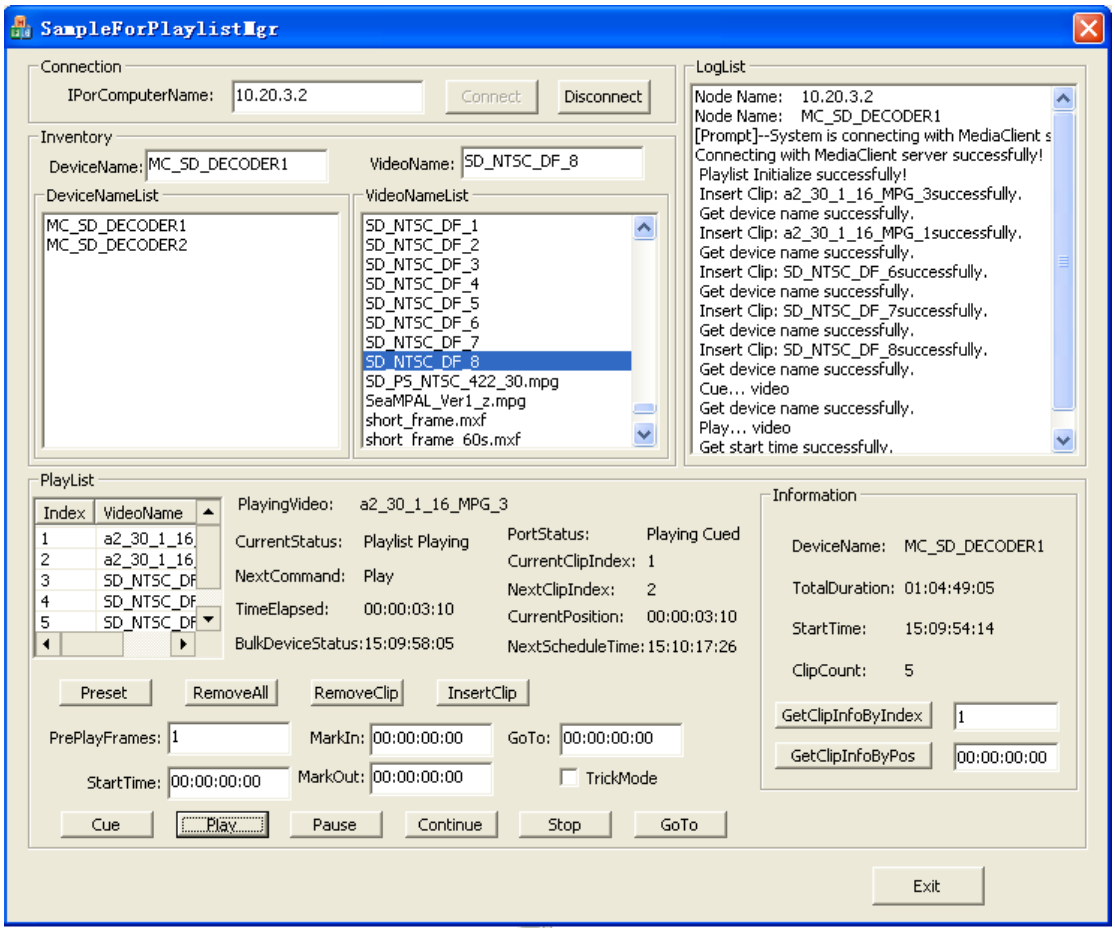
pScAsPlaylistMgr->Cue() //Cue the playlist and preparing play

pScAsPlaylistMgr->Play() //Play clips one by one to the end

pScAsPlaylistMgr->Stop() //Stop playing

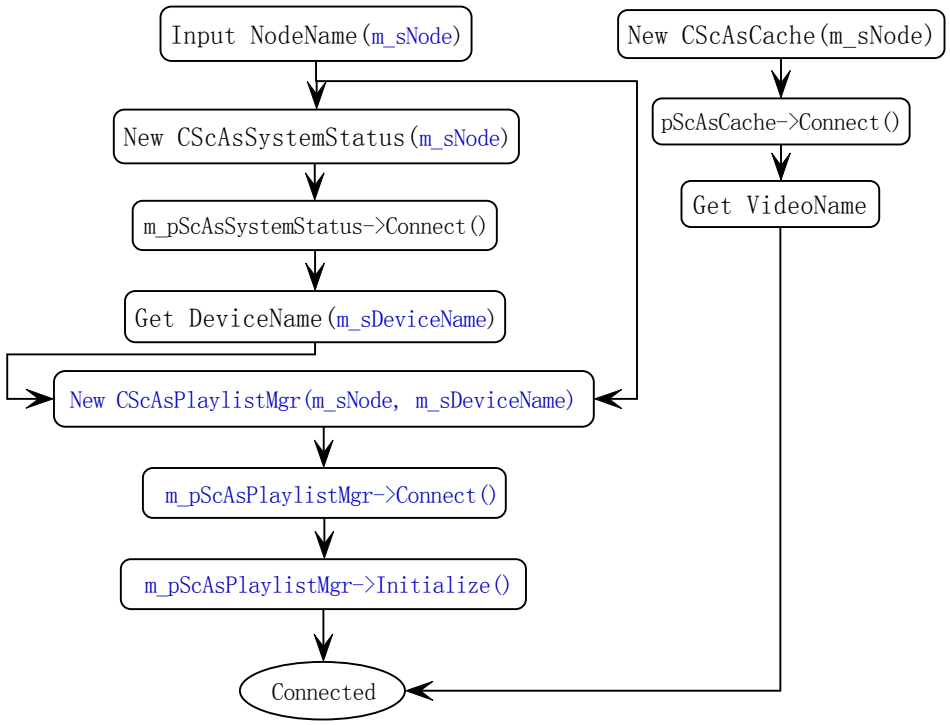
pScAsPlaylistMgr->UnInitialize() //Uninitialize all resources

UI and Workflow



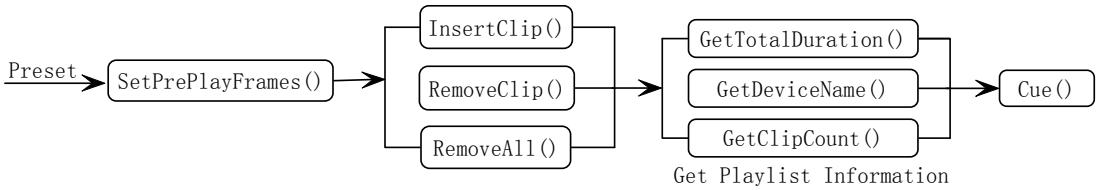
Step1: Connect Device Playlist

Input MCL Node Name or IP Address, and Click ‘Connect’, you will get Decoder Device Name List and Video Name List.



Step2: Operate Playlist

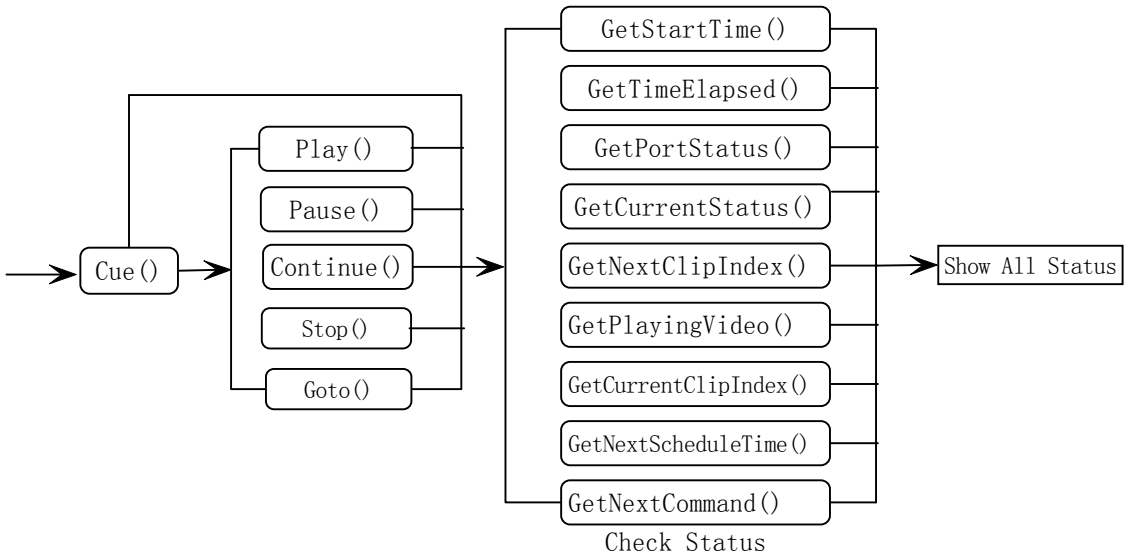
After connected, user can click “Preset” to set preplay frames by SetPrePlayFrames(). And user must click “InsertClip” to insert a clip into playlist, click “RemoveClip” to remove a clip from clip, click ‘RemoveAll’ to remove all clips from list.



User also can get playlist info by GetTotalDuration(), GetDeviceName(), GetClipCount().

Step3: Control Playback

After playlist is made up, you can control decoder for Cue, Play, Pause, etc.



UI will display device status by calling GetPortStatus().

Step4: Query playback status

UI will display playback status by calling GetCurrentStatus(), GetCurrentPos(), GetCurrentClipIndex(), GetStartTime().

Step5: Query playlist and clip info

UI will display playlist info by calling GetTotalDuration(), GetClipCount(), GetClipInfoByClipIndex().

Step6: Disconnect Playlist

Before exiting, you must click “Disconnect” by calling Uninitialize() and release resources.

SampleForRecordlistMgr

Introduction

This **SampleForRecordlistMgr** application is used for recordlist functions by MCC SDK. The application is a simple UI based program.

Main components are CScAsSystemStatus, CScAsCache, CScAsRecordlistMgr.

Recordlist calls in sequence should be:

pScAsRecordlistMgr->Connect() // Connect encoder for preparing

pScAsRecordlistMgr->Initialize() //Initialize resource after new instance

pScAsRecordlistMgr->SetPreRecorderFrames() //Set pre frames to control recording frame accuracy

pScAsRecordlistMgr->InsertClip()//Insert clips into recordlist,also must insert one by one

Or

pScAsRecordlistMgr->CreateList()//Create a list with segment clips into recordlist

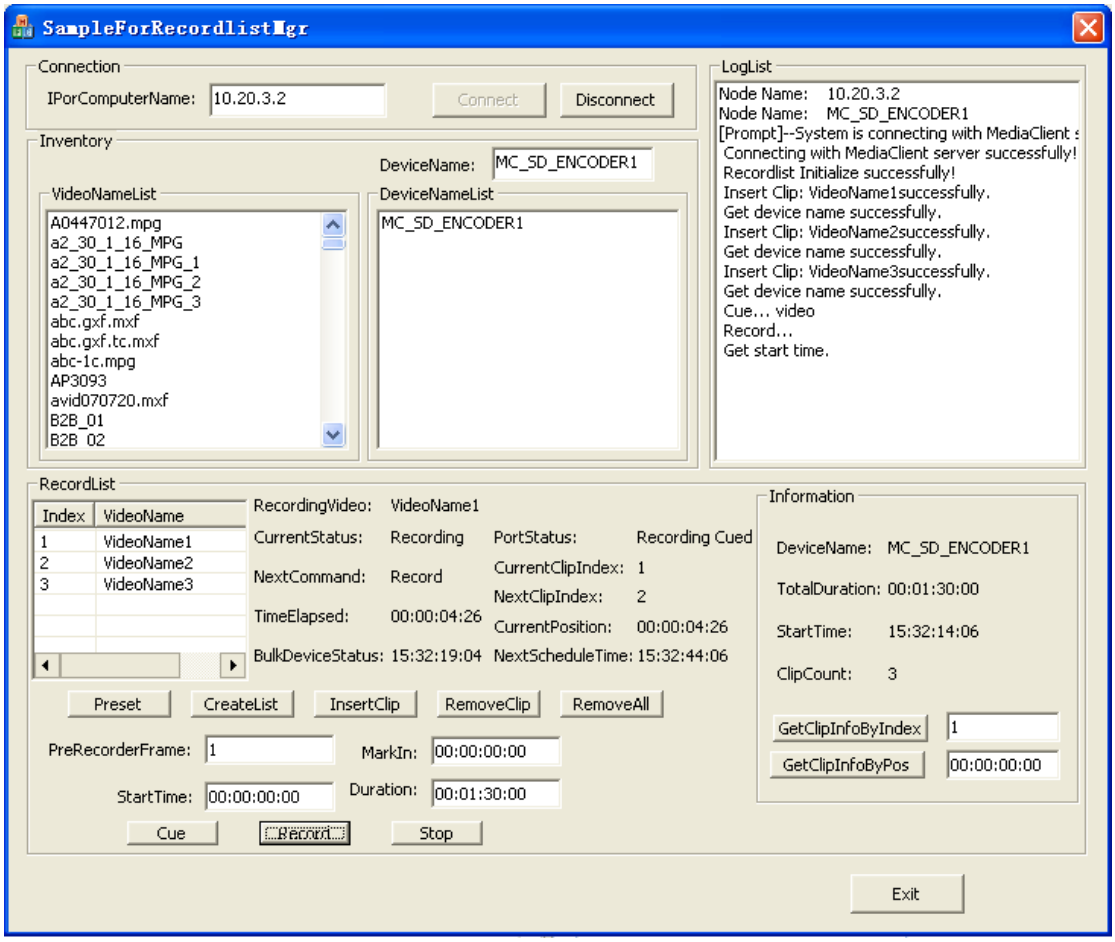
pScAsRecordlistMgr->RecordInit() //Record init the list and preparing record

pScAsRecordlistMgr->Record() //Record the clips one by one

pScAsRecordlistMgr->Stop() //Stop recording

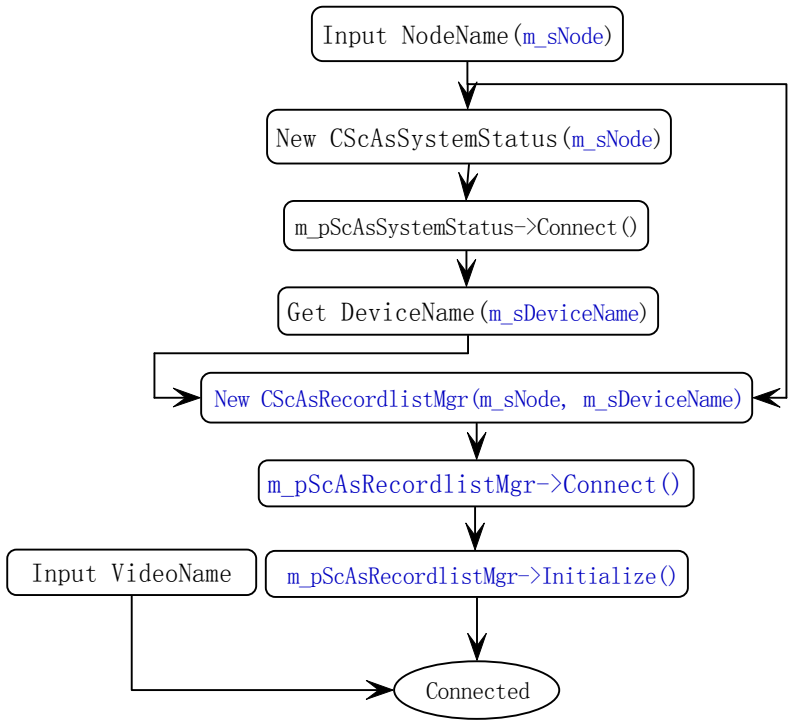
pScAsRecordlistMgr->UnInitialize() //Uninitialize all resources

UI and Workflow



Step1: Connect Device Recordlist

Input MCL Node Name or IP Address, and Click ‘Connect’, you will get Encoder Device Name List and Video Name List.

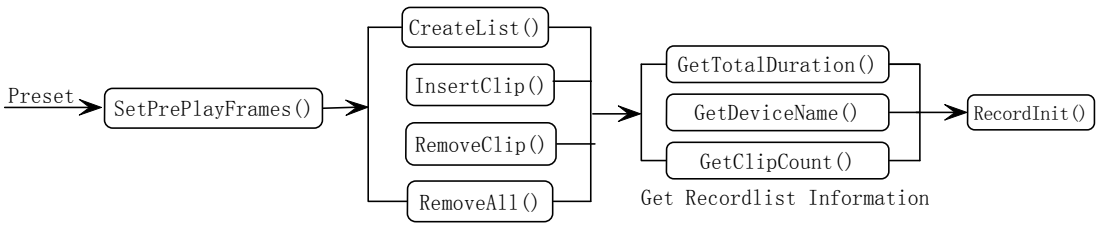


Step2: Operate Recordlist

After connected, user can click “Preset” to set pre record frames by SetPrePlayFrames(). And user must click “InsertClip” to insert a clip into recordlist, click “RemoveClip” to remove a clip from clip, click ‘RemoveAll’ to remove all clips from list.

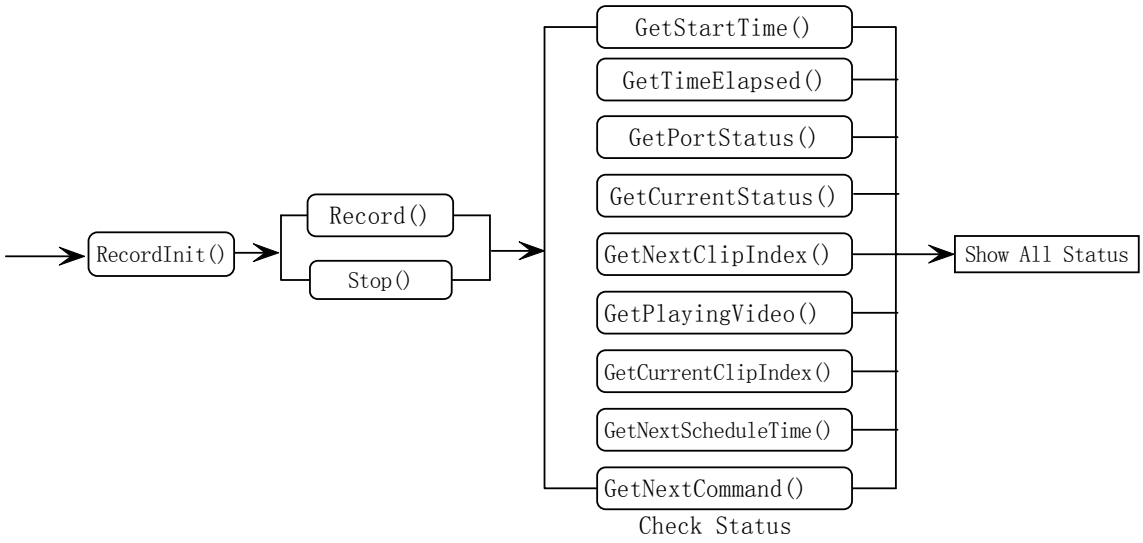
For recordlist, user can click “CreateList” to call CreateList() to make a quick list for a segment recording list. CreateList will remove all clips from current list, then make up a new list for segments.

User also can get recordlist info by GetTotalDuration(), GetDeviceName(), GetClipCount().



Step3: Control Recording

After recordlist is made up, you can control encoder for RecordInit, Record, Stop, etc.



UI will display device status by calling GetPortStatus ().

Step4: Query recording status

UI will display recording status by calling GetCurrentStatus(), GetCurrentPos(), GetCurrentClipIndex(), GetStartTime().

Step5: Query recordlist and clip info

UI will display playlist info by calling GetTotalDuration(), GetClipCount(), GetClipInfoByClipIndex().

Step6: Disconnect Recordlist

Before exiting, you must click “Disconnect” by calling Uninitialize() and release resources.

Appendix C: Auxiliary Class

Class CScAsRpcSessionManager

***class** AFX_EXT_CLASS CScAsRpcSessionManager*

Location: ScAsRPCSessionMgr.h

Description

The class provides common functionality for connecting with MediaClient server. So every component in MCC SDK inherited this class could utilize this function. **Only when service is successfully connected the other APIs in every component could take effect.**

Member

GetNodeName(), Connect(), OpenSession(), CloseSession(), GetSessionHandle(), GetConnectionStatus(), RegisterEvent(), UnregisterEvent().

Construction and destruction

- ***CScAsRpcSessionManager(const CString& csNodeName)***

Description

Default constructor

Parameters

const CString& csNodeName	[in]The required connecting host name of MediaClient server

- ***virtual ~CScAsRpcSessionManager()***

Description

Virtual destructor. The connected session will be released in this function.

Member functions

- ***CString GetNodeName()***

Description

Return current connected node name.

- ***DWORD Connect()***

Description

Connect to specified MediaClient server thru RPC. Every component should individually call this function first.

This function will get session handle for function GetSessionHandle().

Return

The connection status. Zero if the function is successful; otherwise nonzero

- ***OpenSession()***

Description

A private member function

- ***CloseSession()***

Description

A private member function

- ***unsigned long GetSessionHandle()***

Description

Get the connected session handle, only valid when previous calling Connect()

Return

Nonzero if it is valid session handle.

- ***DWORD GetConnectionStatus()***

Description

Get the connected status.

Return

Refer to RPC_SESSION_STATUS.

Class CDeviceStatus

***class** AFX_EXT_CLASS CDeviceStatus*

Location: ScAsSystemStatus.h

Description

Class for preserve device status, device type, device name and other device info. The device list including COM Ports, decoder Ports, Encoder Ports, LOUTH Ports.

Members

CDeviceStatus(), SetUpdated(), IsUpdated(), GetDeviceName(), GetDeviceType(), GetValue(), GetValue(), GetSize(), IsDeviceReleaseEnabled(), IsDeviceForceReleaseEnabled(), IsDeviceResetEnabled(), IsDeviceAcquireEnabled(), SetDeviceName(), SetDeviceType(), SetValue(), SetStateMask(), GetStateMask()

Construction and destruction

- ***CDeviceStatus(DWORD dwDeviceType=0xFFFFFFFF)***

Description

Constructor to set the default value to every element;

Parameters

DWORD dwDeviceType	[in]Device type is defined as RPC_DEVICES_TYPE.

Member functions

- ***Void SetUpdated (BOOL bUpdated)***

Description

Set the device status is updated or not updated; **this function is not useful for SDK User.**

Parameters

BOOL bUpdated	[in]TRUE if the device status is updated. FALSE if the device status is not updated.

- ***BOOL IsUpdated()***

Description

Determine whether the device status is updated.

Return

TRUE if the device status is updated.

FALSE if the device status is never updated.

- ***CString GetDeviceName(void) const;***

Description

Get the device name.

Return

The device name.

- ***DWORD GetDeviceType() const;***

Description

Get the device type

Return

The device type, which is defined as enum RPC_DEVICE_TYPES.

- ***void GetValue(int nIndex, CString& csValue)***

- ***CString GetValue(int nIndex)***

Description

Get the specified device status info.

Parameters

int nIndex	[in] The index of status info in the status list.
CString& csValue	[out]The Return about the requested status info.

Remark

Status list sequence is based on following rules:

For Archive device:

Archive device name, IDs added count, IDs deleted count, Delete requests, Delete error, Change priority requests, Change priority errors, Send to requests, Send to errors, Get from

requests, Get from errors, Pending requests, Archive requests, Completed requests, Abort requests, Abort error.

For COM Ports:

COM port device name, port status, lock holder

For Cache Manager:

Cache device name, ID counts, IDs added counts, IDs deleted, Create requests, Create errors, Open requests, Open errors, Close Requests, Close errors, Read Requests, Read errors, Write requests, Write errors, Delete requests, Delete errors, Rename requests, Rename errors, Modify attributes requests, Modify attributes errors.

For Louth Port device:

Louth ports name, Connected COM port device, Device status, Preview device, Preview device status, Communication port status, Communication port purged, Complete message, Incomplete message, Message replies, Open port successes, Open port errors, Select port successes, De-select port successes, Select port errors, Port closed by controller, Port closed by another port, Port closed by ExdUtil, Port reopened for change in Setting, Port reopen for change in settings failed.

For Codec device:

Device name, Port status, Lock holder, Active ID, Position, cued ID, Cue requests, Cue errors, Play requests, Play errors, Stop requests, Stop error.

- ***int GetSize()***

Description

Get the size of the status info list.

Return

The number of device status info (for examples HD Decoder port 1, the device status info including Device name, status, Lock Holder, active ID, Position, Cued Id and so on).

- ***BOOL IsDeviceReleaseEnabled()***

Description

Determine whether current device is releasable.

Return

TRUE if current device is releasable;

FALSE if current device is not releasable.

- ***BOOL IsDeviceForceReleaseEnabled()***

Description

The function is same as IsDeviceReleaseEnabled ().

- ***BOOL IsDeviceResetEnabled ()***

Description

Determine whether current device is able to reset.

Return

TRUE if current device can be reset;

FALSE if current device can not be reset.

- ***BOOL IsDeviceAcquireEnabled()***

Description

Determine whether current device is able to acquire resource like COM port.

Return

TRUE if current device can acquire resource;

FALSE if current device can not acquire resource.

- ***void SetDeviceName(const CString& csDeviceName)***

Description

Set device name.

Parameters

const CString& csDeviceName	[in]The device name.

- ***void SetDeviceType(DWORD dwDeviceType)***

Description

Set device type.

Parameters

DWORD dwDeviceType	[in]The device type.Refer to enum RPC_DEVICE_TYPES.

- ***void SetValue(int nIndex,CString& csValue)***

Description

Set the device status info.

Parameters

int nIndex	[in]The index of status info in the status list.
CString& csValue	[in]The specified device info value.

- ***void SetStateMask(ULONG dwStateMask)***

Description

Set the device state.

Parameters

ULONG dwStateMask	[in]The combination value of multiple device status, the status is defined as Device States.

- ***ULONG GetStateMask()***

Description

Get device state.

Return

The combination value of multiple device status, the status is defined as Device States.

Type definitions

- ***typedef CTypedPtrList<CPtrList, CDeviceStatus*> CDeviceStatusList;***

- ***typedef CTypedPtrArray<CPtrArray, CDeviceStatus*> CDeviceStatusArray;***

Class CDeviceStatusInfo

***class** AFX_EXT_CLASS CDeviceStatusInfo*

Location: ScAsSystemStatus.h

Description

Class for preserve device status info label, this class mainly provided static label info in every device status and the status info level

Members

CDeviceStatusInfo(), GetInfoType(), SetInfoType(), SetLabel(), GetLabel(), SetLevel(), GetLevel(), GetSize()

Construction and destruction

- ***CDeviceStatusInfo (DWORD dwInfoType=0xFFFFFFFF)***

Description

Constructor function to set the default value to every member variable.

Parameters

DWORD dwDeviceType	[in]The device type.Refer to enum RPC_DEVICE_TYPES.

Member functions

- ***DWORD GetInfoType()***

Description

Get label Info sequence type. It is same as device type.

Return

The label Info type, defined by enum RPC_DEVICE_TYPES.

- ***void SetInfoType(DWORD dwInfoType)***

Description

Set label Info sequence type. It is same as device type.

Parameters

DWORD dwDeviceType	[in]The label Info type.Refer to enum RPC_DEVICE_TYPES.

- ***void SetLabel(int nIndex,CString& csLabel)***

Description

Set the label text which is specified by the index

Parameters

int nIndex	[in]The label index in the status label list.
CString& csLabel	[in]Specified label text.

- ***void GetLabel(int nIndex,CString& csLabel)***

Description

Get the label text which is specified by the index

Parameters

int nIndex	[in]The label index in the status label list, the status label list is correspondence with the status list of GetValue.
CString& csLabel	[out]Returned label text.

- ***void SetLevel(int nIndex,const int& nLevel)***

Description

Set some of device status display level which is specified by the index

Parameters

int nIndex	[in]The index in device status info list.
const int& nLevel	[in]Specified level, it's defined in Label Level.

- ***void GetLevel(int nIndex,int& nLevel)***

Description

Get some of device status display level which is specified by the index

Parameters

int nIndex	[in]The index in device status info sequence.

int& nLevel	[out]Returned level, it's defined in Label Level.

- ***int GetSize()***

Description

Get the count of device status info label

Return

The size of status label array

Type definitions

- ***typedef CTypedPtrList<CPtrList, CDeviceStatusInfo*> CDeviceStatusInfoList;***

- ***typedef CTypedPtrArray<CPtrArray, CDeviceStatusInfo*> CDeviceStatusInfoArray;***

Class CRequestArchive

class AFX_EXT_CLASS CRequestArchive

Location: ScAsArchive.h

Description

This class is implemented to reserve archive request status.

Friend class

CScAsArchive - provided common functionality for Archive Media file.

Members

GetRequestHandle(), GetState(), GetOperation(), GetSourceId(), GetDestinationId(), GetSourceServiceName(), GetDestinationServiceName(), GetSourcePath(), GetDestinationPath(), GetErrorCode(), SetLastError()

Construction and destruction

- ***CRequestArchive ()***

Description

Constructor CScAsCache object.

Member functions

- ***unsigned long GetRequestHandle()***

Description

Retrieves request handle.

Return

The request handle.

- ***WORD GetState()***

Description

Retrieves request state

Return

The request state

- ***DWORD GetOperation()***

Description

Retrieves request operation.

Return

The request operation.

- ***const CString& GetSourceId()***

Description

Retrieves request Source clip name.

Return

A CString type value to preserve source clip name.

- ***const CString& GetDestinationId()***

Description

Retrieves the Archive request destination clip name.

Return

A CString type value to preserve destination clip name.

- ***const CString& GetSourceServiceName()***

Description

Retrieves the Archive request source video storage server name.

Return

A CString type value to preserve source video storage server name.

- ***const CString& GetDestinationServiceName()***

Description

Retrieves the Archive request destination video storage server name.

Return

A CString type value to preserve destination video storage server name.

- ***const CString& GetSourcePath()***

Description

Retrieves the Archive request source store path.

Return

A CString type value to preserve source store path.

- ***const CString& GetDestinationPath()***

Description

Retrieves the Archive request destination store path.

Return

A CString type value to preserve destination store path.

- ***DWORD GetErrorCode()***

Description

Retrieves the Archive request error code.

Return

A DWORD type value to preserve last error code.

- ***void SetLastError (DWORD dwErrCode=ERROR_SUCCESS)***

Description

Set the Archive request last error code.

Parameters

DWORD dwErrCode	A DWORD type value to preserve last error code.

Type definitions

- ***typedef CTypedPtrList<CPtrList, CRequestArchive*> CRequestArchiveList;***

Class CArchiveService

***class** AFX_EXT_CLASS CArchiveService*

Location: ScAsArchive.h

Description

This class is implemented to preserve archive service information, the archive function is copy or move a media clip from one archive service to another.

Members

GetServiceName(), GetServiceType(), SetServiceName(), SetServiceType(), SetDefaultService(), SetServicePath(), GetServicePathList(), GetServicePathHeadPosition(), GetServicePathTailPosition(), GetServicePathNext(), GetServicePathPrev(), GetServicePathAt(), IsDefaultService()

Construction and destruction

- ***CArchiveService()***

  Constructs an uninitialized CArchiveService object.

- ***CArchiveService(const CString& csServiceName,const CStringList& clServicePath,DWORD dwServiceType,BOOL bDefaultService=FALSE)***

  Description

  Constructor CArchiveService object with variable mode.

  Parameters

const CString& csServiceName	[in]A CStirng type value, which indicated the Archive service name, this name is just a symbol for the created archive service
const CStringList& clServicePath	[in]A CStringList type value, to preserve the Video storage access path (e.g. IP address, Host name , absolute path)
DWOR DdwServiceType	[in]A DWORD value ,to indicated the Archive service accessing type.Refer to enum Archive Service Type
BOOL bDefaultService	[in]A BOOL value , to indicated the current archive whether is the default archive service , the default archive service should be single

- ***virtual ~CArchiveService()***

Description

Destructor CArchiveService object, and release the assigned resource.

Member functions

- ***CString GetServiceName()***

Description

Retrieves the Archive service name.

Return

A CString type value to preserve the service name.

- ***DWORD GetServiceType()***

Description

Retrieves Archive service type.

Return

A DWORD type value to preserve the service type.Refer to enum Archive Service Type.

- ***void SetServiceName (const CString& csServiceName)***

Description

Set the Archive service name.

Parameters

A CString type value to preserve the service name.

- ***void SetServiceType(const DWORD dwServiceType)***

Description

Set the Archive service type.

Parameters

A DWORD type value to preserve the service type.Refer to enum Archive Service Type.

- ***void SetDefaultService(BOOL bDefaultService)***

Description

Set the current CArchiveService object as default archive service, there are only one destination default archive service.

Parameters

BOOL bDefaultService	[in]A BOOL type value, which indicates the current CArchiveServices object whether is default one. True, this service is default, otherwise not.

- ***LONG SetServicePath (const CStringList& clServicePath)***

Description

Set the archive service access path, for BMLE is the Data access IP address or host name, for local cache is the access path.

Parameters

const CStringList& clServicePath	[in]A CStringList value, to preserve the access path list, this value is relative with service type.

- ***LONG GetServicePathList (CStringList& csList)***

Description

Retrieves the Archive service accessing path list.

Parameters

CStringList& csList	[out]The path list

Return

A LONG type value to preserve the path list size.

- ***POSITION GetServicePathHeadPosition()***

Description

Returns the head element position of the path list (cannot be empty).

Return

A POSITION object, to indicate the head position of the path list.

Remark

You must ensure that the list is not empty before calling GetServicePahtHeadPosition.

- ***POSITION GetServicePathTailPosition()***

Description

Return the tail element position of the path list (cannot be empty).

Return

A POSITION object, to indicate the tail position of the path list.

- ***CString GetServicePathNext(POSITION& pos)***

Description

Gets the path list element as the parameters rPosition specified, and then sets rPosition to the POSITION value of the next entry in the list.

Parameters

POSITION pos	[in]A reference to a POSITION value returned by a previous GetServicePathNext, GetServicePathHeadPosition, or other member function call

Return

A CStirng type value to preserve the requested Path value.

- ***CString GetServicePathPrev (POSITION& pos)***

Description

Gets the list element identified by rPosition, and then sets rPosition to the POSITION value of the previous entry in the list.

Parameters

POSITION& pos	[in]A reference to a POSITION value returned by a previous GetServicePathNext, GetHeadServicePathPosition, or other member function call.

Return

A CStirng type value to preserve the requested Path value.

- ***CString GetServicePathAt (POSITION pos)***

Description

Gets the element at a given position.

Parameters

POSITION pos	[in]A reference to a POSITION value returned by a previous

	GetServicePathNext, GetHeadServicePathPosition, or other member function call.

Return

A CStirng type value to preserve the requested Path value.

- ***BOOL IsDefaultService()***

Description

Indicates whether the current CArchiveService object is the default archive service.

Return

A BOOL type value, which indicates the current CArchiveServices object whether is default one. True, this service is default, otherwise not.

Type definitions

- ***typedef CTypedPtrList<CPtrList, CArchiveService*> CArchiveServiceList;***

Class CSCRPCException

class AFX_EXT_CLASS CSCRPCException

Location: ScAsUtility.h

Description

A CScAsRPCException object represents an exception that is the result of RPC calling failed. This Exception is only valid for CScAsRecorder and CScAsDecoder objects.

Members

CSCRPCException(), GetErrorCode()

Construction and destruction

- ***explicit CSCRPCException(DWORD m_dwErrorCode)***

Description

Constructor function to set the default value to every member variable.

Parameters

DWORD m_dwErrorCode	[in]the error code

Member functions

- ***DWORD GetErrorCode()***

Description

Get the exception error code.

Return

The DWORD type error code.

Class CTimeCode

class AFX_EXT_CLASS CTimeCode

Location: TimeCode.h

Description

This class is implemented to reserve and convert time code from one mode to another, the supported time code tranform format including BCD time code, Frame count and CString type time code and so on.

Members

operator=(), operator+=(), operator-=(), operator+(), operator-(), operator<(), operator<=(), operator==(), operator!=(), operator>(), operator>=().

IsForwardCorrection(), IsBackwardCorrection(), SetForwardCorrection(), SetBackwardCorrection(), IsTimeCodeNow(), SetTimeCodeNow(), ClearTimeCodeNow(), SetVideoType(), SetFrameRateStandard(), GetFrameRateStandard(), GetFrameRate(), GetFrameRate(), GetBCD(), GetHour(), GetMinute(), GetSecond(), GetFrame(), IsOddField(), SetOddField(), ClearOddField(), Reset(), GetTotalFrames(), GetFrameCount(), ConvertToBCD(), GetTimeCodeString(), BCD2String(), GetTimeString(), GetVstrmTimeCode()

Construction and destruction

- ***CTimeCode(const FrameRateStandard standard= CTimeCode::NTSC_NDF,const BOOL bForwardCorrection=TRUE)***

Constructs an uninitialized CTimeCode object.

- ***CTimeCode(const BYTE hours,const BYTE minutes,const BYTE seconds,const BYTE frames,const FrameRateStandard standard=CTimeCode::NTSC_NDF,const BOOL bForwardCorrection=TRUE)***

  Constructs a CTimeCode object from time components, including hour, minute, second, frame.

- ***CTimeCode(const long nFrames,const FrameRateStandard standard=CTimeCode::NTSC_NDF,const BOOL bForwardCorrection=TRUE)***

  Constructs a CTimeCode object from frame count.

- ***CTimeCode(const BCD_TIMECODE& rBcd,const FrameRateStandard standard=CTimeCode::NTSC_NDF,const BOOL bForwardCorrection=TRUE)***

  Constructs a CTimeCode object from a structure BCD_TIMECODE

- ***CTimeCode(const LPCBCD_TIMECODE pBcd,const FrameRateStandard standard=CTimeCode::NTSC_NDF,const BOOL bForwardCorrection=TRUE)***

  Constructs a CTimeCode object from a structure LPCBCD_TIMECODE

- ***CTimeCode(const CString& sTimeString,const FrameRateStandard standard=CTimeCode::NTSC_NDF,const BOOL bForwardCorrection=TRUE)***

  Constructs a CTimeCode object from a value of CString type, This CString type text should format as “00:00:00.00”

- ***CTimeCode(const ULONG ulVstrmTc,const FrameRateStandard standard=CTimeCode::NTSC_NDF,const BOOL bForwardCorrection=TRUE)***

  Constructs a CTimeCode object from a VSTRM TimeCode structure

- ***CTimeCode(const CTimeCode& rThat,const FrameRateStandard standard,const BOOL bForwardCorrection)***

  Constructs a CTimeCode object from another CTimeCode value with specified Frame Rate and Field count convert method.

- ***CTimeCode(const CTimeCode& rThat)***

  Constructs a CTimeCode object from another CTimeCode value, copy all of the default values to the new object.

Description

Constructs CTimeCode objects in various ways.

Parameters

const FrameRateStandard standard	[in]The time code video standard.
const BOOL bForwardCorrection	[in]For the field counts convert to Frame counts mode: True, the odd field will take as one frame. The frame count = field count /2 +1 //if there are an odd field exist. False, the odd field will not take as a frame. The frame count = field count /2 //if there are an odd field exist.
const BYTE Hours,Minutes,Seconds,Frames	[in]Indicates the time values to be copied into the new CTimeCode object.
const BCD_TIMECODE& rBcd	[in]A BCD_TIMECCODE structure to be converted to a time value and copied into the new CTimeCode object.
const LPCBCD_TIMECODE pBcd	[in]A LPBCD_TIMECODE structure to be converted to a time value and copied into the new CTimeCode object.
const CString& sTimeString	[in]A CString value to be copy into the new CTimeCode object.
const ULONG ulVstrmTc	[in]A VSTRM_TIMECODE structure to be converted to a time value and copied into the new CTimeCode object.
const CTimeCode& rThat	Indicates a CTimeCode object that already exists.

Member Functions

- ***CTimeCode& operator=(const long nFrames)***
- ***CTimeCode& operator=(const BCD_TIMECODE& rBcd)***
- ***CTimeCode& operator=(const ULONG ulVstrmTc)***
- ***CTimeCode& operator=(const CTimeCode &rThat)***

Description

The assignment operator.

Parameters

const long nFrames	The new time value with Frame count format.

const BCD_TIMECODE&rBcd	The new time value with BCD_TIMECODE structure format.
const ULONG ulVstrmTc	The new time value with VSTRM_TIMECODE format.
const CTimeCode & rThat	The new time value.

Return

The updated CTimeCode object.

- ***CTimeCode& operator+=(const long nFrames)***
- ***CTimeCode& operator+=(const CTimeCode &rThat)***
- ***CTimeCode& operator-=(const long nFrames)***
- ***CTimeCode& operator-=(const CTimeCode &rThat)***

Description

These operators add and subtract some frame count to and from this CTimeCode object.

Parameters

const long nFrames	the frame count to be added or subtracted
const CTimeCode &rThat	the CTimeCode object to be added or subtracted

Return

The updated CTimeCode object

- ***AFX_EXT_CLASS friend CTimeCode operator+(const CTimeCode& rTimeCode,const long nFrames)***
- ***AFX_EXT_CLASS friend CTimeCode operator+(const CTimeCode& a,const CTimeCode& b)***
- ***AFX_EXT_CLASS friend CTimeCode operator-(const CTimeCode& rTimeCode,const long nFrames)***
- ***AFX_EXT_CLASS friend CTimeCode operator-(const CTimeCode& a,const CTimeCode& b)***

Description

These operators add and subtract Frame Count and CTimeCode objects.

Parameters

const CTimeCode& rTimeCode,a,b	A CTimeCode object to be added or subtracted.

const long nFrames	A frame count value to be added or subtracted.

Return

The updated CTimeCode object.

- ***AFX_EXT_CLASS friend BOOL operator<(const CTimeCode& a,const CTimeCode& b)***
- ***AFX_EXT_CLASS friend BOOL operator<=(const CTimeCode& a,const CTimeCode& b)***
- ***AFX_EXT_CLASS friend BOOL operator==(const CTimeCode& a,const CTimeCode& b)***
- ***AFX_EXT_CLASS friend BOOL operator!=(const CTimeCode& a,const CTimeCode& b)***
- ***AFX_EXT_CLASS friend BOOL operator>(const CTimeCode& a,const CTimeCode& b)***
- ***AFX_EXT_CLASS friend BOOL operator>=(const CTimeCode& a,const CTimeCode& b)***

Description

Comparison operators.

Parameters

The CTimeCode object to be compared.

Return

These operators compare two time code and return true if the condition is TRUE; otherwise FALSE.

- ***BOOL IsForwardCorrection()***

Description

Indicates whether the field to frame calculate method is forward correction.

Return

TRUE if the calculate method is forward correction.

- ***BOOL IsBackwardCorrection()***

Description

Indicates whether the field to frame calculate method is backward correction.

Return

TRUE if the calculate method is backward correction.

- ***void SetForwardCorrection()***

Description

Set the field count calculate method to forward correction.

- ***void SetBackwardCorrection()***

Description

Set the field count calculate method to forward correction.

- ***BOOL IsTimeCodeNow ()***

Description

Indicates whether the time code is now.

Return

True if the time code is now.

- ***void SetTimeCodeNow()***

Description

Set the time code to now.

- ***void ClearTimeCodeNow ()***

Description

Clear the time now flag.

- ***void SetVideoType(const FrameRateStandard standard)***

Description

Set Frame rate to specified standard.

Parameters

const FrameRateStandard standard	[in]A FramRateStandard type value

- ***void SetFrameRateStandard (const FrameRateStandard standard)***

Description

Set Frame rate to specified standard.

Return

const FrameRateStandard standard	[in]A FramRateStandard type value

- ***FrameRateStandard GetFrameRateStandard()***

Description

Get Frame rate standard.

Return

A FramRateStandard type value.

- ***FrameRateStandard GetType()***

Description

Get Frame rate standard.

Return

A FramRateStandard type value.

- ***int GetFrameRate ()***

Get current CTimeCode object frame rate.

- ***static UINT GetFrameRate (const FrameRateStandard standard)***

Get the frame rate count based on specified Frame Rate Standard.

Description

Get the Frame rate base on variable request.

Parameters

const FrameRateStandard standard	[in]A FramRateStandard type value

- ***BCD_TIMECODE GetBCD()***

Description

Get BCD_TIMECODE type value.

Return

The BCD_TIMECODE type value.

- ***BYTE GetHour()***

- ***BYTE GetMinute()***

- ***BYTE GetSecond()***

- ***BYTE GetFrame()***

Description

Get the time code components value.

Return

The requested time code component value.

- ***BOOL IsOddField()***

Description

Indicates whether the field count is odd.

Return

TRUE if the field count is odd.

- ***void SetOddField()***

Description

Set the field count conversion to odd.

- ***void ClearOddField()***

Description

Clear odd field flag, so that the field counts conversion not to odd.

- ***void Reset (void)***

Description

Reset the CTimeCode object.

- ***long GetTotalFrames()***

Description

Get the total frame count which contained in this CTimeCode object.

Return

The frame count value.

- ***long GetFrameCount ()***

Description

Get the frame counts which contained in this CTimeCode object.

Return

The frame count value.

- ***BOOL ConvertToBCD(const LPBCD_TIMECODE pBcd)***

Description

Gets a struct LPBCD_TIMECODE containing a decomposition of the time contained in this CTimeCode object.

Return

A BCD_TIMECODE structure value.

- ***CString GetTimeCodeString()***

Description

Gets a CString time code value containing a decomposition of the time contained in this CTimeCode object.

Return

A CString time code value

- ***static void BCD2String(const BCD_TIMECODE& rBcd, CString& rsBcd)***

Description

Converts a struct BCD_TIMECODE value to a CString time code value.

- ***CString GetTimeString()***

Description

This function is the same as GetTimeCodeString.

- ***ULONG GetVstrmTimeCode()***

Description

Gets a struct VSTRM Time Code value containing a decomposition of the time contained in this CTimeCode object.

Return

A VSTRM time code value.

Class CBriefPrivateData

***class** AFX_EXT_CLASS CBriefPrivateData : **public** CObject*

Location: PrivateData.h

Description

This class is implemented to read and set the metadata information in PD file. For the PD file definition please reference document MCC_Privatedata_SP.doc.

Derivation

CObject is the principal base class for the Microsoft Foundation Class Library.

Members

Construction: CBriefPrivateData()

operator=()

GetBriefPrivateData(), GetMajorVersion(), GetMinorVersion()GetLength(), GetType(), GetVideoStandard(), GetFrameRateStandard(), GetFileAttributes(), GetVideoAttributesLegacy(), GetVideoAttributes(), GetDuration(), GetFirstFrame(), ClearReadOnly(), GetReadOnly(), GetDeleteProtected(), GetWriteProtected(), GetVideoResolutionLegacy(), GetStreamTypeLegacy(), GetChromaFormatLegacy(), GetMpegTypeLegacy(), GetAspectRatioLegacy(), GetVideoResolution(), GetStreamMultiplex(), GetChromaFormat(), GetVideoElementryStream(), GetVerticalLines(), GetHorizontalLines(), GetFrameRate(), GetFrameFormat(), GetDefinition(), GetProxy(), Reset(), SetDefaults(), SetMajorVersion(), SetMinorVersion(), SetLength(), SetType(), SetVideoStandard(), SetFrameRateStandard(), SetFileAttributes(), SetVideoAttributesLegacy(), SetVideoAttributes(), SetDuration(), SetFirstFrame(), SetFirstFrame(), SetReadOnly(), SetReadOnly(), SetDeleteProtected(), ClearDeleteProtected(), SetBriefPrivateData(), SetWriteProtected(), SetVideoResolutionLegacy(), SetStreamTypeLegacy(), SetChromaFormatLegacy(), SetMpegTypeLegacy(), SetAspectRatioLegacy(), SetVideoResolution(), SetStreamMultiplex(), SetChromaFormat(), SetVideoElementryStream(), SetVerticalLines(), SetHorizontalLines(), SetFrameRate(), SetFrameFormat(), SetDefinition(), SetProxy()

Construction and destruction

- ***CBriefPrivateData(const VideoStandard eVideoStandard=K_TYPE_NTSC_NDF)***
  Initial CBriefPrivateData object with specified Video standard.

- ***CBriefPrivateData(const CTimeCode::FrameRateStandard eFrameRateStandard)***
  Initial CBriefPrivateData object with specified Frame Rate standard.

- ***CBriefPrivateData(const CTimeCode& rDuration)***
  Initial CBriefPrivateData object with specified Duration.

- ***CBriefPrivateData(const CBriefPrivateData* pBriefPrivateData)***
  Initial CBriefPrivateData object with another CBriefPrivateData object pointer.

- ***CBriefPrivateData(const CBriefPrivateData& rBriefPrivateData)***
  Initial CBriefPrivateData object with another CBriefPrivateData object reference.

- ***CBriefPrivateData(const SD_BRIEF_PRIVATE_DATA* pPrivateData,const ULONG ulPrivateData)***
  Initial CBriefPrivateData object with struct SD_BRIEF_PRIVATE_DATA.

  Description

  Constructs CBriefPrivateData objects in various ways.

  Parameters

const VideoStandard eVideoStandard	[in]the video standard defined by VideoStandard.
const CTimeCode::FrameRateStandard eFrameRateStandard	[in]a CTimeCode::FrameRateStandard type value to preserve the frame rate.
const CTimeCode& rDuration	[in]a CTimeCode object to preserve Duration time code.
const CBriefPrivateData* pBriefPrivateData	the pointer to CBriefPrivateData object, preserved the Private Data info.
const CBriefPrivateData & rBriefPrivateData	[in]A reference to CBriefPrivateData object, preserved the Private Data info.
const SD_BRIEF_PRIVATE_DATA* pPrivateData	[in]the point to struct SD_BRIEF_PRIVATE_DATA, preserved the Private Data info.
const ULONG ulPrivateData	[in]The struct SD_BRIEF_PRIVATE_DATA data size.

- ***virtual ~CBriefPrivateData()***

  Description

  Virtual destructor, to release the allocated resource.

Member Functions

- ***CBriefPrivateData& operator=(const CBriefPrivateData& rElement)***

- ***CBriefPrivateData& operator=(const SD_BRIEF_PRIVATE_DATA* pPrivateData)***

Description

The assignment operator.

Parameters

const CBriefPrivateData& rElement	The new CBriefPrivateData object
const SD_BRIEF_PRIVATE_DATA* pPrivateData	The new Private Data value with SD_BRIEF_PRIVATE_DATA structure format

Return

The updated CBriefPrivateData object.

- ***HRESULT GetBriefPrivateData(PBYTE pPrivateData, ULONG &ulPrivateData***
- ***HRESULT GetBriefPrivateDataRaw(PBYTE&pPrivateData,ULONG &ulPrivateData)***
- ***HRESULT GetBriefPrivateDataVXX(BYTE* pPrivateData, ULONG &ulPrivateData)***
- ***HRESULT GetBriefPrivateData(SD_BRIEF_PRIVATE_DATA* pV12)***
- ***HRESULT GetBriefPrivateData(SD_BRIEF_PRIVATE_DATA_V11* pV11)***
- ***HRESULT GetBriefPrivateData(SD_BRIEF_PRIVATE_DATA_V01* pV01)***

Description

Get Private Data in various ways.

Parameters

PBYTE pPrivateData	[out]A BYTE array to preserve the Private Data info
ULONG & ulPrivateData	[out]The Byte array size
SD_BRIEF_PRIVATE_DATA* pV12	[out]A SD_BRIEF_PRIVATE_DATA structure to preserve the Private Data info with V1.2 format
SD_BRIEF_PRIVATE_DATA_V11* pV11	[out]A SD_BRIEF_PRIVATE_DATA_V11 structure to preserve the Private Data info with V1.1 format
SD_BRIEF_PRIVATE_DATA_V01* pV01	[out]A SD_BRIEF_PRIVATE_DATA_V01 structure to preserve the Private Data info with V1.0 format

Return

If successful, return zero else return error code.

- ***BYTE GetMajorVersion()***

Description

Get Private Data major version.

Return

A BYTE type value to preserve major version.

- ***BYTE GetMinorVersion()***

Description

Get Private Data minor version.

Return

A BYTE type value to preserve minor version

- ***ULONG GetLength()***

Description

Get Private Data size.

Return

A ULONG type value to preserve Private Data size

- ***inline BYTE GetType()***

Description

Get the video standard from the CBriefPrivateData object.

Return

A BYTE type value to preserve video standard.

- ***inline VideoStandard GetVideoStandard()***

Description

Get the Video Standard from the CBriefPrivateData object.

Return

A VideoStandard type value to preserve video standard.

- ***inline CTimeCode::FrameRateStandard GetFrameRateStandard()***

Description

Get the frame rate from the CBriefPrivateData object.

Return

A CTimeCode::FrameRateStandard type value to preserve frame rate

- ***BYTE GetFileAttributes()***

Description

Get the file attributes from the CBriefPrivateData object.

Return

A BYTE type value to preserve file attributes, refer to FILE_ATTRIBUTES.

- ***BYTE GetVideoAttributesLegacy()***

Description

Get the video attributes legacy from the CBriefPrivateData object. This value belongs to Private Data V1.1

Return

A BYTE type value to preserve file attributes, the value format is defined as VIDEO_ATTRIBUTES_LEGACY.

- ***ULONGLONG GetVideoAttributes(void) const***

Description

Get the Video Attributes from the CBriefPrivateData object.

Return

A BYTE type value to preserve file attributes, the value format is defined as VIDEO_ATTRIBUTES.

- ***SD_BCD_TIME GetDurationBCD()***

Get SD_BCD_TIME format duration value.

- ***void GetDuration(SD_BCD_TIME& rDuration)***

Get SD_BCD_TIME format duration value.

- ***CTimeCode GetDuration()***

Get a CTimeCode object to preserve duration value.

- ***void GetDuration(CTimeCode& rDuration)***

  Get a CTimeCode object to preserve duration value.

  Description

  Get the duration from the CBriefPrivateData object with various ways.

- ***SD_BCD_TIME GetFirstFrameBCD()***

- ***void GetFirstFrame(SD_BCD_TIME &rFirstFrame)***

- ***CTimeCode GetFirstFrame()***

- ***void GetFirstFrame(CTimeCode& rFirstFrame)***

  Description

  Get the start time code from the CBriefPrivateData object with various ways.

- ***BOOL GetReadOnly()***

  Description

  Get the Read only file attributes from the CBriefPrivateData object.

  Return

  A BOOL type value to preserve Read only file attributes.

- ***BOOL GetDeleteProtected()***

  Description

  Get the Delete protect file attributes from the CBriefPrivateData object.

  Return

  A BOOL type value to preserve Delete protect file attributes.

- ***BOOL GetWriteProtected()***

  Description

  Get the Write protect file attributes from the CBriefPrivateData object.

  Return

  A BOOL type value to preserve Write protect file attributes.

- ***BYTE GetVideoResolutionLegacy()***

Description

Get the Legacy Video Resolution from the CBriefPrivateData object. This value belongs to Private Data V1.1.

Return

A BYTE type value to preserve the legacy video resolution, and the value defined in VIDEO_RESOLUTION.

- ***BYTE GetStreamTypeLegacy()***

Description

Get the Legacy Stream type from the CBriefPrivateData object. This value belongs to Private Data V1.1.

Return

A BYTE type value to preserve the legacy stream type, and the value defined in STREAM_TYPE.

- ***BYTE GetChromaFormatLegacy()***

Description

Get the Legacy Chroma format from the CBriefPrivateData object. This value belongs to Private Data V1.1.

Return

A BYTE type value to preserve the legacy chroma format, and the value defined in Chroma Format.

- ***BYTE GetMpegTypeLegacy()***

Description

Get the legacy mpeg type from the CBriefPrivateData object. This value belongs to Private Data V1.1.

Return

A BYTE type value to preserve the legacy mpeg type, and the value defined in MPEG_TYPE.

- ***BYTE GetAspectRatioLegacy()***

Description

Get the legacy aspect ratio from the CBriefPrivateData object. This value belongs to Private Data V1.1.

Return

A BYTE type value to preserve the legacy aspect ratio, and the value defined in Aspect_Ratio.

- ***BYTE GetVideoResolution()***

Description

Get the video resolution from the CBriefPrivateData object. This value belongs to Private Data V1.2.

Return

A BYTE type value to preserve the Video Resolution, and the value defined in Video _esolution.

- ***BYTE GetStreamMultiplex()***

Description

Get the Video Stream Multiplex type from the CBriefPrivateData object. This value belongs to Private Data V1.2.

Return

A BYTE type value to preserve the video stream multiplex type, and the value defined in Video_Stream_Mutiplex.

- ***BYTE GetChromaFormat()***

Description

Get the video chroma format from the CBriefPrivateData object. This value belongs to Private Data V1.2.

Return

A BYTE type value to preserve the chroma format, and the value defined in Chroma_Format.

- ***BYTE GetVideoElementryStream()***

Description

Get the video elementary stream type from the CBriefPrivateData object. This value belongs to Private Data V1.2.

Return

A BYTE type value to preserve the video elementary Stream type, and the value defined in Elementry_Stream.

- ***DWORD GetVerticalLines()***

Description

Get the video vertical lines from the CBriefPrivateData object. This value belongs to Private Data V1.2.

Return

A BYTE type value to preserve the video vertical lines.

- ***DWORD GetHorizontalLines()***

Description

Get the video horizontal lines from the CBriefPrivateData object. This value belongs to Private Data V1.2

Return

A BYTE type value to preserve the video horizontal lines.

- ***BYTE GetFrameRate()***

Description

Get the frame rate from the CBriefPrivateData object. This value belongs to Private Data V1.2.

Return

A BYTE type value to preserve the frame rate.

- ***BYTE GetFrameFormat()***

Description

Get the frame format from the CBriefPrivateData object. This value belongs to Private Data V1.2.

Return

A BYTE type value to preserve the Frame Format and the value defined in Frame_Format.

- ***BYTE GetDefinition()***

Description

Get the video definition from the CBriefPrivateData object. This value belongs to Private Data V1.2.

Return

A BYTE type value to preserve the video definition and the value defined in VideoDefinition.

- ***BOOL GetProxy()***

Description

Check the Video Proxy flag whether enabled in the CBriefPrivateData object. This value belongs to Private Data V1.2.

Return

A BOOL Type value to indicate the video proxy whether enabled.

- ***void SetDefaults(const VideoStandard eVideoStandard=K_TYPE_NTSC_NDF)***

Description

Set the CBriefPrivateData object to default value or the specified video standard.

Parameters

const VideoStandard eVideoStandard	[in]A VideoStandard type value to indicate the video standard.

- ***void ClearDeleteProtected()***

Description

Clear the delete protect flag to FALSE.

- ***void SetMajorVersion(const BYTE bMajorVersion)***

Description

Set the major version to CBriefPrivateData object with specified value.

Parameters

const BYTE bMajorVersion	[in]A BYTE type value to preserve the major version

- ***void SetMinorVersion(const BYTE bMinorVersion)***

Description

Set the minor version to CBriefPrivateData object with specified value.

Parameters

const BYTE bMinorVersion	[in]A BYTE type value to preserve the minor version.

- ***void SetLength(const ULONG ulPrivateData)***

Description

Set the private data buffer size to CBriefPrivateData object with specified value.

Parameters

const ULONG ulPrivateData	[in]A ULONG type value to preserve Private Data buffer size.

- ***inline void SetType(const BYTE bType)***

Description

Set the video standard to CBriefPrivateData object with specified value.

Parameters

const BYTE bType	[in]A BYTE type value to preserve the video standard

- ***inline void SetVideoStandard(const VideoStandard eVideoStandard)***

Description

Set the Video standard to CBriefPrivateData object with specified value.

Parameters

const VideoStandard eVideoStandard	[in]A VideoStandard type value to preserve the video standard

- ***inline void SetFrameRateStandard(const CTimeCode::FrameRateStandard eFrameRateStandard)***

Description

Set the frame rate standard to CBriefPrivateData object with specified value.

Parameters

const CTimeCode::FrameRateStandard eFrameRateStandard	[in] A CTimeCode::FrameRateStandard type value to preserve Frame rate

- ***void SetFileAttributes(const BYTE bAttributes)***

Description

Set the file attribute to CBriefPrivateData object with specified value.

Parameters

const BYTE bAttributes	[in] A BYTE type value to preserve File attribute. Refer

	to FILE_ATTRIBUTES

- ***void SetVideoAttributesLegacy(const BYTE bAttributes)***

Description

Set the legacy video attributes to CBriefPrivateData object with specified value.

Parameters

const BYTE bAttributes	[in] A BYTE Type value to preserve File Attributes. Refer to VIDEO_ATTRIBUTES_LEGACY

- ***void SetVideoAttributes(const ULONGLONG ullAttributes)***

Description

Set the video attributes to CBriefPrivateData object with specified value.

Parameters

const ULONGLONG ullAttributes	[in] A ULONGLONG Type value to preserve file attributes, the value format is defined as VIDEO_ATTRIBUTES

- ***void SetDuration(const SD_BCD_TIME &rDuration)***

Description

Set the duration to CBriefPrivateData object with specified value.

Parameters

const SD_BCD_TIME & rDuration	[in] the duration value with SD_BCD_TIME format or CTimeCode class

- ***void SetDuration(const CTimeCode &rDuration)***

Description

Set the duration to CBriefPrivateData object with specified value.

Parameters

const CTimeCode & rDuration	[in] the duration value with SD_BCD_TIME format or CTimeCode class

- ***void SetFirstFrame(const SD_BCD_TIME &rFirstFrame)***

- ***void SetFirstFrame(const CTimeCode &rFirstFrame)***

Description

Set start time code to CBriefPrivateData object with specified value.

Parameters

const SD_BCD_TIME & rFirstFrame	[in] the SD_BCD_TIME struct format start time code
const CTimeCode & rFirstFrame	[in]a CTimeCode object to preserve the start time code

- ***void SetReadOnly()***

Set the read only flag to TRUE.

- ***void SetReadOnly(const BOOL bValue)***

Set the read only flag to Parameter bValue specified value.

Description

Set the read only file attribute to CBriefPrivateData object with specified value.

- ***void ClearReadOnly()***

Description

Set the read only flag to False in the CBriefPrivateData object.

- ***void SetDeleteProtected(void)***

Set the Delete protect flag to TRUE.

- ***void SetDeleteProtected(const BOOL bValue)***

Set the Delete protect flag to Parameter bValue specified value.

Description

Set the Delete protect file attribute to CBriefPrivateData object with specified value.

- ***void ClearDeleteProtected()***

Description

Set the Delete protect flag to False in the CBriefPrivateData object.

- ***HRESULT SetBriefPrivateData(const CBriefPrivateData &rElement)***

- ***HRESULT SetBriefPrivateData(const PBYTE pPrivateData,const ULONG ulPrivateData,const SD_BCD_TIME *pTimeCode=NULL)***

- ***HRESULT SetBriefPrivateDataVXX(const BYTE *pPrivateData,const ULONG ulPrivateData,const SD_BCD_TIME *pTimeCode)***

- ***HRESULT SetBriefPrivateData(const SD_BRIEF_PRIVATE_DATA*pV12,const SD_BCD_TIME *pTimeCode = NULL)***

- ***HRESULT SetBriefPrivateData(const SD_BRIEF_PRIVATE_DATA_V11 *pV11,const SD_BCD_TIME *pTimeCode = NULL)***

- ***HRESULT SetBriefPrivateData(const SD_BRIEF_PRIVATE_DATA_V01 *pV01,const SD_BCD_TIME *pTimeCode = NULL)***

Description

Set Private Data in various ways.

Parameters

const PBYTE pPrivateData	[in]A BYTE array to preserve the Private Data info.
const ULONG ulPrivateData	[in]The BYTE array size.
const SD_BRIEF_PRIVATE_DATA* pV12	[in]A SD_BRIEF_PRIVATE_DATA structure to preserve the Private Data info with V1.2 format.
const SD_BRIEF_PRIVATE_DATA_V11* pV11	[in]A SD_BRIEF_PRIVATE_DATA_V11 structure to preserve the Private Data info with V1.1 format.
const SD_BRIEF_PRIVATE_DATA_V01* pV01	[in]A SD_BRIEF_PRIVATE_DATA_V01 structure to preserve the Private Data info with V1.0 format.
const CBriefPrivateData & rElement	[in]A reference to new CBriefPrivateData object.
const SD_BCD_TIME * pTimeCode	[in]A pointer to CTimeCode object to preserve the Start Time code.

Return

If successful, return zero else return error code. It could be check from error code list.

- ***void Reset()***

Description

Reset the private data all of component value to zero.

- ***void SetWriteProtected(const BOOL bValue)***

Description

Set the Delete protect file attribute to CBriefPrivateData object with specified value.

Parameters

const BOOL bValue	[in]A BOOL type value to preserve the specified write protect.

- ***void SetVideoResolutionLegacy(const BYTE byVideoResolution)***

Description

Set the Legacy Video resolution to CBriefPrivateData object with specified value.

Parameters

const BYTE byVideoResolution	[in]A BYTE type value to preserve the Legacy Video Resolution, and the value defined in VIDEO_RESOLUTION.

- ***void SetStreamTypeLegacy(const BYTE byStreamType)***

Description

Set the legacy stream type to CBriefPrivateData object with specified value.

Parameters

const BYTE byStreamType	[in]A BYTE type value to preserve the Legacy Stream type, and the value defined in Stream Type.

- ***void SetChromaFormatLegacy(const BYTE bChromaFormat)***

Description

Set the legacy chroma format to CBriefPrivateData object with specified value.

Parameters

const BYTE bChromaFormat	[in]A BYTE type value to preserve the Legacy Chroma format, and the value defined in ChromaFormat.

- ***void SetMpegTypeLegacy(const BYTE byMpegType)***

Description

Set the legacy mpeg type to CBriefPrivateData object with specified value.

Parameters

const BYTE byMpegType	[in]A BYTE type value to preserve the legacy mpeg type, and the value defined in MPEG_TYPE.

- ***void SetAspectRatioLegacy(const BYTE byAspectRatio)***

Description

Set the legacy aspect ratio value to CBriefPrivateData object with specified value.

Parameters

const BYTE byAspectRatio	[in]A BYTE type value to preserve the aspect ratio value, and the value defined in ASPECT_RATIO.

- ***void SetVideoResolution(const BYTE byVideoResolution)***

Description

Set the Video Resolution to CBriefPrivateData object with specified value.

Parameters

const BYTE byVideoResolution	[in] A BYTE type value to preserve the video resolution, and the value defined in Resolution

- ***void SetStreamMultiplex(const BYTE byStreamMultiplex)***

Description

Set the stream multiplex type to CBriefPrivateData object with specified value.

Parameters

const BYTE byStreamMultiplex	[in] A BYTE type value to preserve the stream multiplex type, and the value defined in STREAM_MULTIPLEX

- ***void SetChromaFormat(const BYTE byChromaFormat)***

Description

Set the video chroma format to CBriefPrivateData object with specified value.

Parameters

const BYTE byChromaFormat	[in] A BYTE type value to preserve the Chroma format, and the value defined in CHROMA_FORMAT

- ***void SetVideoElementryStream(const BYTE byVideoElementryStream)***

Description

Set the Video elementary stream type to CBriefPrivateData object with specified value.

Parameters

const BYTE byVideoElementryStream	[in] A BYTE type value to preserve the Video elementary stream type, and the value defined in VIDEO_ELEMENTRY_STREAM_TYPE

- ***void SetVerticalLines(const DWORD dwVerticalLines)***

Description

Set the Video Vertical lines to CBriefPrivateData object with specified value.

Parameters

const DWORD dwVerticalLines	[in] A DWORD Type value to preserve the counts of Video Vertical lines

- ***void SetHorizontalLines(const DWORD dwHorizontalLines)***

Description

Set the Video Horizontal lines to CBriefPrivateData object with specified value.

Parameters

const DWORD dwHorizontalLines	[in] A DWORD Type value to preserve the counts of Video Horizontal lines

- ***void SetFrameRate(const BYTE byFrameRate)***

Description

Set the Video Frame rate to CBriefPrivateData object with specified value.

Parameters

const BYTE byFrameRate	[in] A BYTE type value to preserve the Frame rate, and the value defined in Frame Rate

- ***void SetFrameFormat(const BYTE byFrameFormat)***

Description

Set the video frame format to CBriefPrivateData object with specified value.

Parameters

const BYTE byFrameFormat	[in] A BYTE type value to preserve the Frame Format, and the value defined in Frame Format

- ***void SetDefinition(const BYTE byDefinition)***

Description

Set the video definition to CBriefPrivateData object with specified value.

Parameters

const BYTE byDefinition	[in] A BYTE type value to preserve the Definition, and the value defined in DEFINITION

- ***void SetProxy(const BOOL bProxy)***

Description

Set the proxy flag to CBriefPrivateData object with specified value.

Parameters

const BOOL byDefinition	[in] A BOOL Type value to indicate whether enable the proxy flag. TRUE if the low bit rate proxy is created, otherwise FALSE.

Type definitions

- ***FILE_ATTRIBUTES***

```
typedef union
{
        struct
        {
        BYTE   fReadOnly: 1;
        BYTE   fDeleteProtected: 1;
        BYTE   fWriteProtected: 1;
        } flags;
        BYTE   bAttributes;
} FILE_ATTRIBUTES;
```

- ***VIDEO_ATTRIBUTES_LEGACY***

```
typedef union
{
        struct
        {
                BYTE   fResolution: 3;
```

```
            BYTE   fStreamtype: 2;
            BYTE   fChromaFormat: 1;
            BYTE   fMpegType: 1;
            BYTE   fAspectRatio: 1;
        } flags;
        BYTE   bAttributes;
    } VIDEO_ATTRTIBUTES_LEGACY;
```

- ***VIDEO_ATTRIBUTES***

```
    typedef union
    {
        struct
        {
            BYTE   ucVideoResolution;
            BYTE   ucStreamMultiplex;
            BYTE   ucChromaFormat;
            BYTE   ucVideoElementryStream;
            BYTE   ucFrameRate;
            BYTE   ucFrameFormat;
            BYTE   ucDefinition;
            BYTE   fProxy : 1;
        } flags;
        ULONGLONG           ullAttributes;
    } VIDEO_ATTRIBUTES;
```

- ***SD_BRIEF_PRIVATE_DATA***

```
typedef struct _sd_brief_private_data {
    unsigned char   majorVersion:4;
    unsigned char   minorVersion:4;
    unsigned char   bType;
    SD_BCD_TIME     rDuration;
    unsigned char   fileAttributeReadOnly:1;            // BOOL
    unsigned char   fileAttributeDeleteProtect:1;       // BOOL
    unsigned char   fileAttributeWriteProtect:1;        // BOOL
    unsigned char   fileAttributeReserved:5;            // <reserved>
    unsigned char   videoAttributeResolutionLegacy:3;// PRIVATE_DATA_VIDEO_RESOLUTION_
    unsigned char   videoAttributeStreamTypeLegacy:2;// PRIVATE_DATA_STREAM_TYPE_
(PROG, TRANS, SYS, OTHER only)
    unsigned char   videoAttributeChromaFormatLegacy:1;//
PRIVATE_DATA_CHROMA_FORMAT_ (422 and 420 only)
    unsigned char   videoAttributeMpegTypeLegacy:1;// PRIVATE_DATA_MPEG_TYPE_ (MPEG2
and MPEG1 only)
    unsigned char   videoAttributeAspectRatioLegacy:1;// PRIVATE_DATA_ASPECT_RATIO_ (4_3
and 16_9 only)
    SD_BCD_TIME     rFirstFrame;                    // SOM
        unsigned char   videoAttributeResolution;  // PRIVATE_DATA_VIDEO_RESOLUTION_
        unsigned char   videoAttributeStreamMultiplex;// PRIVATE_DATA_STRM_MULTIPLEX_
        unsigned char   videoAttributeChromaFormat;// PRIVATE_DATA_CHROMA_FORMAT_
        unsigned char   videoAttributeVideoElementryStream;// PRIVATE_DATA_VIDEO_ES_
        DWORD           videoAttributeVerticalLines;//
        DWORD          videoAttributeHorizontalLines;//
```

```
        unsigned char   videoAttributeFrameRate; // PRIVATE_DATA_FRAME_RATE_
        unsigned char   videoAttributeFrameFormat;// PRIVATE_DATA_FRAME_FORAMT_
        unsigned char   videoAttributeDefinition;// PRIVATE_DATA_DEFINITION_
        unsigned char   videoAttributeProxy;     // BOOL
   } SD_BRIEF_PRIVATE_DATA, *PSD_BRIEF_PRIVATE_DATA;

   typedef SD_BRIEF_PRIVATE_DATA           SD_BRIEF_PRIVATE_DATA_V12;
   typedef SD_BRIEF_PRIVATE_DATA_V12  *PSD_BRIEF_PRIVATE_DATA_V12;
```

- ***SD_BRIEF_PRIVATE_DATA_V01***

```
//  mjm - this data structure is the version used for eXd V1.0 and V1.2
//  and is associated with major_version=0 and minor_version=1
//
typedef struct _sd_brief_private_data_v01 {
   unsigned char   majorVersion:4;
   unsigned char   minorVersion:4;
   unsigned char   bType;
   SD_BCD_TIME     rDuration;
   unsigned char   fileAttributeReadOnly:1;
   unsigned char   fileAttributeReserved:7;
} SD_BRIEF_PRIVATE_DATA_V01, *PSD_BRIEF_PRIVATE_DATA_V01;
```

- ***SD_BRIEF_PRIVATE_DATA_V11***

```
//
// - this data structure, the "current" one, is the version used for eXd V1.4 and above
//  and is associated with major_version=1 and minor_version=0 or 1
//
typedef struct _sd_brief_private_data_v11 {
   unsigned char   majorVersion:4;
   unsigned char   minorVersion:4;
   unsigned char   bType;
   SD_BCD_TIME     rDuration;
   unsigned char   fileAttributeReadOnly:1;
   unsigned char   fileAttributeDeleteProtect:1;
   unsigned char   fileAttributeWriteProtect:1;
   unsigned char   fileAttributeReserved:5;
   unsigned char   videoAttributeResolution:3;
   unsigned char   videoAttributeStreamType:2;
   unsigned char   videoAttributeChromaFormat:1;
   unsigned char   videoAttributeMpegType:1;
   unsigned char   videoAttributeAspectRatio:1;
   SD_BCD_TIME     rFirstFrame;                                    // SOM
   } SD_BRIEF_PRIVATE_DATA_V11, *PSD_BRIEF_PRIVATE_DATA_V11;
```

- ***Video Resolution***

```
#define VIDEO_RESOLUTION_FULL                0x00
#define VIDEO_RESOLUTION_HALF                0x01
#define VIDEO_RESOLUTION_SIF                 0x02
#define VIDEO_RESOLUTION_DVD                 0x03
#define VIDEO_RESOLUTION_OTHER               0x04
#define VIDEO_RESOLUTION_UNKNOWN             0x05
```

- ***STREAM_TYPE***

```
#define STREAM_TYPE_PROGRAM          0x00
#define STREAM_TYPE_TRANSPORT        0x01
#define STREAM_TYPE_SYSTEM           0x02
#define STREAM_TYPE_OTHER            0x03
#define STREAM_TYPE_UNKNOWN          0x04
```

- ***CHROMA_FORMAT***

```
#define CHROMA_FORMAT_422            0x00
#define CHROMA_FORMAT_420            0x01
#define CHROMA_FORMAT_411            0x02
#define CHROMA_FORMAT_UNKNOWN        0x03
```

- ***MPEG_TYPE***

```
#define MPEG_TYPE_MPEG2              0x00
#define MPEG_TYPE_MPEG1              0x01
#define MPEG_TYPE_UNKNOWN            0x02
```

- ***ASPECT_RATIO***

```
#define ASPECT_RATIO_4_3             0x00
#define ASPECT_RATIO_16_9            0x01
#define ASPECT_RATIO_UNKNOWN         0x02
```

- ***STREAM_MULTIPLEX***

```
#define STRM_MULTIPLEX_NONE          0x00
#define STRM_MULTIPLEX_UNKNOWN       0x01
#define STRM_MULTIPLEX_MPEG1_SYSTEM  0x02
#define STRM_MULTIPLEX_MPEG2_PROGRAM 0x03
#define STRM_MULTIPLEX_MPEG2_TRANSPORT 0x04
#define STRM_MULTIPLEX_MPEG2_DVB     0x05
#define STRM_MULTIPLEX_ELEMENTRY     0x06
#define STRM_MULTIPLEX_MXF           0x07
#define STRM_MULTIPLEX_QT            0x08
#define STRM_MULTIPLEX_AVI           0x09
```

- ***Video Element Stream Type***

```
#define VIDEO_ES_UNKNOWN             0x00
#define VIDEO_ES_MPEG1               0x01
#define VIDEO_ES_MPEG2               0x02
#define VIDEO_ES_DV25                0x03
#define VIDEO_ES_DV50                0x04
#define VIDEO_ES_DV100               0x05
#define VIDEO_ES_DVSD                0x06
```

- ***FRAME_RATE***

```
#define FRAME_RATE_UNKNOWN           0x00
#define FRAME_RATE_25                0x01
#define FRAME_RATE_29_97             0x02
```

```
#define FRAME_RATE_30                0x03
#define FRAME_RATE_50                0x04
#define FRAME_RATE_60                0x05
#define FRAME_RATE_59_94             0x06
```

- ***FRAME_FORMAT***

```
#define FRAME_FORMAT_UNKNOWN         0x00
#define FRAME_FORMAT_INTERLACED      0x01
#define FRAME_FORMAT_PROGRESSIVE     0x02
```

- ***DEFINITION***

```
#define DEFINITION_UNKNOWN           0x00
#define DEFINITION_SD                0x01
#define DEFINITION_HD                0x02
```

Appendix D: Definition

- ***Enum RPC_SESSION_STATUS***

```
enum RPC_SESSION_STATUS {
    RPC_SESSION_OPEN = 0, // Connection is successful
    RPC_SESSION_CLOSED, // Connection status is closed
    RPC_SESSION_NEVER_OPENED,// connection is never opened
    RPC_SESSION_OPEN_FAILED, // connection is failed to open
    RPC_SESSION_SECURITY_CHECK_FAILED,// Security check is failed
    RPC_SESSION_DISCONECTED,//connection is lost
    RPC_SESSION_MAX
};
```

- ***Enum RPC_DEVICE_TYPES***

```
enum RPC_DEVICE_TYPES{
 RPC_IO_PORT_MIN = 100,
        RPC_IO_PORT_DECODER /* = 101 */,
        RPC_IO_PORT_ENCODER,
        RPC_IO_PORT_COM_PORT,
        RPC_IO_PORT_CACHE,
        RPC_IO_PORT_ARCHIVE,
        RPC_LOUTH_PORT,
        RPC_AVS_PORT,
        RPC_ODETICS,
        RPC_SEACHANGE,
        RPC_SONY,
        RPC_IO_PORT_TCP_PORT,
      RPC_IO_PORT_MAX};
```

- ***MCL SD Encoder Config Profile***

```
enum LIGOS_MC_SD_ENCODER_CONFIG_PROFILE {
/*0*/   MCSDE_CFG_PRO_NAME = 0,   //profile name
/*1*/   MCSDE_CFG_PRO_VIDEO_SOURCE//Video source, defined in enum
ENC_VIDEO_SOURCE,
/*2*/   MCSDE_CFG_PRO_VIDEO_FORMAT,//Defined in VIDEO FORMAT
/*3*/   MCSDE_CFG_PRO_VIDEO_DROP_FRAME,//BOOL value type
/*4*/   MCSDE_CFG_PRO_VIDEO_IMAGE_WIDTH,//Defined with Marco defination Image
Resolution
/*5*/   MCSDE_CFG_PRO_VIDEO_IMAGE_HEIGHT_NTSC,
/*6*/   MCSDE_CFG_PRO_VIDEO_IMAGE_HEIGHT_PAL,
/*7*/   MCSDE_CFG_PRO_AUDIO1_SOURCE,          //Defined in enum AUDIO SOURCE
/*8*/   MCSDE_CFG_PRO_AUDIO2_SOURCE,
/*9*/   MCSDE_CFG_PRO_AUDIO1_SOURCE_EXTRA,//Defined with Marco Audio Source Extra
/*10*/  MCSDE_CFG_PRO_AUDIO2_SOURCE_EXTRA,
/*11*/  MCSDE_CFG_PRO_AUDIO1_FORMAT,          //Defined with enum Audio Format
/*12*/  MCSDE_CFG_PRO_AUDIO2_FORMAT,
/*13*/  MCSDE_CFG_PRO_AUDIO1_MODE,            //Defined with enum Audio Mode
```

```
/*14*/  MCSDE_CFG_PRO_AUDIO2_MODE,
/*15*/  MCSDE_CFG_PRO_STREAM_TYPE,                  //Defined with enum Stream type
/*16*/  MCSDE_CFG_PRO_VIDEO_STREAM_ID,
/*17*/  MCSDE_CFG_PRO_AUDIO1_STREAM_ID,
/*18*/  MCSDE_CFG_PRO_AUDIO2_STREAM_ID,
/*19*/  MCSDE_CFG_PRO_PMT_PID,
/*20*/  MCSDE_CFG_PRO_VIDEO_PID,
/*21*/  MCSDE_CFG_PRO_AUDIO1_PID,
/*22*/  MCSDE_CFG_PRO_AUDIO2_PID,
/*23*/  MCSDE_CFG_PRO_VIDEO_ENCODE_RATE,
/*24*/  MCSDE_CFG_PRO_VIDEO_FIELD_PICTURES, //Defined with enum Field Frame Pictures
/*25*/  MCSDE_CFG_PRO_GOP_STRUCTURE,//Defined with enum GOP Structure
/*26*/  MCSDE_CFG_PRO_GOP_PERIOD,
/*27*/  MCSDE_CFG_PRO_SEQUENCE_PERIOD,
/*28*/  MCSDE_CFG_PRO_CHROMA_FORMAT,
/*29*/  MCSDE_CFG_PRO_32_PULLDOWN_INVERSION,//BOOL value
/*30*/  MCSDE_CFG_PRO_AUDIO1_RATE,
/*31*/  MCSDE_CFG_PRO_AUDIO2_RATE,
/*32*/  MCSDE_CFG_PRO_VIDEO_BIT_RATE_CHAR,//Defined with enum Bit Rate Character
/*33*/  MCSDE_CFG_PRO_MPEG_VERSION,   //Defined with enum MPEG Vesion
/*34*/  MCSDE_CFG_PRO_AUDIO3_SOURCE,
/*35*/  MCSDE_CFG_PRO_AUDIO4_SOURCE,
/*36*/  MCSDE_CFG_PRO_AUDIO3_SOURCE_EXTRA,
/*37*/  MCSDE_CFG_PRO_AUDIO4_SOURCE_EXTRA,
/*38*/  MCSDE_CFG_PRO_AUDIO3_FORMAT,
/*39*/  MCSDE_CFG_PRO_AUDIO4_FORMAT,
/*40*/  MCSDE_CFG_PRO_AUDIO3_MODE,
/*41*/  MCSDE_CFG_PRO_AUDIO4_MODE,
/*42*/  MCSDE_CFG_PRO_AUDIO3_STREAM_ID,
/*43*/  MCSDE_CFG_PRO_AUDIO4_STREAM_ID,
/*44*/  MCSDE_CFG_PRO_AUDIO3_PID,
/*45*/  MCSDE_CFG_PRO_AUDIO4_PID,
/*46*/  MCSDE_CFG_PRO_AUDIO3_RATE,
/*47*/  MCSDE_CFG_PRO_AUDIO4_RATE,
/*48*/  MCSDE_CFG_PRO_DESCRIPTION,
/*49*/   MCSDE_CFG_PRO_D10_FLAG,
/*50*/  MCSDE_CFG_PRO_AUDIO5_SOURCE,
/*51*/  MCSDE_CFG_PRO_AUDIO5_SOURCE_EXTRA,
/*52*/  MCSDE_CFG_PRO_AUDIO5_FORMAT,
/*53*/  MCSDE_CFG_PRO_AUDIO5_MODE,
/*54*/  MCSDE_CFG_PRO_AUDIO5_STREAM_ID,
/*55*/  MCSDE_CFG_PRO_AUDIO5_PID,
/*56*/  MCSDE_CFG_PRO_AUDIO5_RATE,
/*57*/  MCSDE_CFG_PRO_AUDIO6_SOURCE,
/*58*/  MCSDE_CFG_PRO_AUDIO6_SOURCE_EXTRA,
/*59*/  MCSDE_CFG_PRO_AUDIO6_FORMAT,
/*60*/  MCSDE_CFG_PRO_AUDIO6_MODE,
/*61*/  MCSDE_CFG_PRO_AUDIO6_STREAM_ID,
/*62*/  MCSDE_CFG_PRO_AUDIO6_PID,
/*63*/  MCSDE_CFG_PRO_AUDIO6_RATE,
/*64*/  MCSDE_CFG_PRO_AUDIO7_SOURCE,
```

```
/*65*/  MCSDE_CFG_PRO_AUDIO7_SOURCE_EXTRA,
/*66*/  MCSDE_CFG_PRO_AUDIO7_FORMAT,
/*67*/  MCSDE_CFG_PRO_AUDIO7_MODE,
/*68*/  MCSDE_CFG_PRO_AUDIO7_STREAM_ID,
/*69*/  MCSDE_CFG_PRO_AUDIO7_PID,
/*70*/  MCSDE_CFG_PRO_AUDIO7_RATE,
/*71*/  MCSDE_CFG_PRO_AUDIO8_SOURCE,
/*72*/  MCSDE_CFG_PRO_AUDIO8_SOURCE_EXTRA,
/*73*/  MCSDE_CFG_PRO_AUDIO8_FORMAT,
/*74*/  MCSDE_CFG_PRO_AUDIO8_MODE,
/*75*/  MCSDE_CFG_PRO_AUDIO8_STREAM_ID,
/*76*/  MCSDE_CFG_PRO_AUDIO8_PID,
/*77*/  MCSDE_CFG_PRO_AUDIO8_RATE,
/*78*/  MCSDE_CFG_PRO_ASPECT_RATIO,
/*79*/  MCSDE_CFG_PRO_VBR_AVG_ENCODE_RATE,
/*80*/  MCSDE_CFG_PRO_VBR_MAX_ENCODE_RATE,
/*81*/  MCSDE_CFG_PRO_TRANSPORT_CAPTURE_RATE,
/*82*/  MCSDE_CFG_PRO_CONFIG_SOM,//the time code format is “00:00:00.00”
/*83*/  MCSDE_CFG_PRO_USE_CONFIG_SOM_FOR_ENCODES,//BOOL value
/*84*/  MCSDE_CFG_PRO_OVERRIDE_AUTOMATION_SOM_WITH_CONFIG_SOM,
/*85*/  MCSDE_CFG_PRO_PROFILE_COMPRESION_TYPE,//Can’t be setting
/*86*/  MCSDE_CFG_PRO_DROP_FRAME_THRESHOLD,
/*87*/  MCSDE_CFG_PRO_MUX_TYPE,//Defined with enum Video Multiplex type
/*88*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL1,//Defined with enum VBI TYPE
/*89*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL1,
/*90*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL2,
/*91*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL2,
/*92*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL3,
/*93*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL3,
/*94*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL4,
/*95*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL4,
/*96*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL5,
/*97*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL5,
/*98*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL6,
/*99*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL6,
/*100*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL7,
/*101*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL7,
/*102*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL8,
/*103*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL8,
/*104*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL9,
/*105*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL9,
/*106*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL10,
/*107*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL10,
/*108*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL11,
/*109*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL11,
/*110*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL12,
/*111*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL12,
/*112*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL13,
/*113*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL13,
/*114*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL14,
/*115*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL14,
```

```
/*116*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL15,
/*117*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL15,
/*118*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL16,
/*119*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL16,
/*120*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL17,
/*121*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL17,
/*122*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL18,
/*123*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL18,
/*124*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL19,
/*125*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL19,
/*126*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL20,
/*127*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL20,
/*128*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL21,
/*129*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL21,
/*130*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL22,
/*131*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL22,
/*132*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL23,
/*133*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL23,
/*134*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL24,
/*135*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL24,
/*136*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL25,
/*137*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL25,
/*138*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL26,
/*139*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL26,
/*140*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL27,
/*141*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL27,
/*142*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL28,
/*143*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL28,
/*144*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL29,
/*145*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL29,
/*146*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL30,
/*147*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL30,
/*148*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL31,
/*149*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL31,
/*150*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL32,
/*151*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL32,
/*152*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL33,
/*153*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL33,
/*154*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL34,
/*155*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL34,
/*156*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL35,
/*157*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL35,
/*158*/  MCSDE_CFG_PRO_VBI_TYPE_CHANNEL36,
/*159*/  MCSDE_CFG_PRO_VBI_LINE_CHANNEL36,
   MCSDE_CFG_PRO_MAX
        };
```

- ***Video Source***

```
enum ENC_VIDEO_SOURCE {
   ENC_VIDEO_SOURCE_NONE,
```

```
   ENC_VIDEO_SOURCE_D1,
   ENC_VIDEO_SOURCE_RESERVED,
   ENC_VIDEO_SOURCE_MOVIE2_I1,
   ENC_VIDEO_SOURCE_MOVIE2_I2,
   ENC_VIDEO_SOURCE_MOVIE2_I3,
   ENC_VIDEO_SOURCE_HOST,
   ENC_VIDEO_SOURCE_TEST
        };
```

- ***Video Format***

```
enum CDC_VIDEO_FORMAT {
   CDC_VIDEO_NTSC,
   CDC_VIDEO_PAL,
   CDC_VIDEO_NTSC_AND_16VBI,
   CDC_VIDEO_PAL_AND_16VBI,
   CDC_VIDEO_NTSC_AND_32VBI,
   CDC_VIDEO_PAL_AND_32VBI,
   CDC_VIDEO_PAL_B,
   CDC_VIDEO_PAL_D,
   CDC_VIDEO_PAL_G,
   CDC_VIDEO_PAL_H,
   CDC_VIDEO_PAL_I,
   CDC_VIDEO_PAL_M,
   CDC_VIDEO_PAL_N
        };
```

- ***Audio Source***

```
enum ENC_AUDIO_SOURCE {
   ENC_AUDIO_SOURCE_NONE,
   ENC_AUDIO_SOURCE_AES_EBU_AUDIO1,
   ENC_AUDIO_SOURCE_AES_EBU_AUDIO2,
   ENC_AUDIO_SOURCE_MOVIE2_AUDIO1,
   ENC_AUDIO_SOURCE_MOVIE2_AUDIO2,
   ENC_AUDIO_SOURCE_RESERVED,
   ENC_AUDIO_SOURCE_HOST,
   ENC_AUDIO_SOURCE_EMBEDDED,
   ENC_AUDIO_SOURCE_AES_EBU_AUDIO3,
   ENC_AUDIO_SOURCE_AES_EBU_AUDIO4,
   ENC_AUDIO_SOURCE_TEST
        };
```

- ***Audio Mode***

```
enum CDC_AUDIO_MODE {
   CDC_AUDIO_MODE_JOINT_STEREO_BANDS431_PROT,
   CDC_AUDIO_MODE_JOINT_STEREO_BANDS431_NOPROT,
   CDC_AUDIO_MODE_JOINT_STEREO_BANDS831_PROT,
   CDC_AUDIO_MODE_JOINT_STEREO_BANDS831_NOPROT,
   CDC_AUDIO_MODE_JOINT_STEREO_BANDS1231_PROT,
```

```
   CDC_AUDIO_MODE_JOINT_STEREO_BANDS1231_NOPROT,
   CDC_AUDIO_MODE_JOINT_STEREO_BANDS1631_PROT,
   CDC_AUDIO_MODE_JOINT_STEREO_BANDS1631_NOPROT,
   CDC_AUDIO_MODE_STEREO_PROTECTED,
   CDC_AUDIO_MODE_STEREO_NOT_PROTECTED,
   CDC_AUDIO_MODE_DUAL_CHANNEL_MONO_PROTECTED,
   CDC_AUDIO_MODE_DUAL_CHANNEL_MONO_NOT_PROTECTED,
   CDC_AUDIO_MODE_SINGLE_CHANNEL_MONO_PROT,
   CDC_AUDIO_MODE_SINGLE_CHANNEL_MONO_NOT_PROT
        };
```

- ***Element Stream Type***

```
typedef enum
{
   CDC_VIDEO_ES_RESERVED = 0,
   CDC_VIDEO_ES_MPEG1,
   CDC_VIDEO_ES_MPEG2,
   CDC_VIDEO_ES_DV25,
   CDC_VIDEO_ES_DV50,
   CDC_VIDEO_ES_DVSD,
   CDC_VIDEO_ES_DV100,
   CDC_VIDEO_ES_HDV
}CDC_VIDEO_ES;
```

- ***Video Multiplex Type***

```
typedef enum
{
   CDC_STREAM_MPLEX_RESERVED = 0,
   CDC_STREAM_MPLEX_RAW,
   CDC_STREAM_MPLEX_ELEMENTARY,
   CDC_STREAM_MPLEX_PROGRAM,
   CDC_STREAM_MPLEX_TRANSPORT,
   CDC_STREAM_MPLEX_MXF,
   CDC_STREAM_MPLEX_QUICKTIME,
   CDC_STREAM_MPLEX_AVI,
   CDC_STREAM_MPLEX_DVB_TRANSPORT,
   CDC_STREAM_MPLEX_ATSC_TRANSPORT
} CDC_STREAM_MULTIPLEX;
```

- ***Bit Rate Character***

```
enum CDC_VIDEO_BIT_RATE_CHAR {
   CDC_CONSTANT_BIT_RATE_ENCODING = 0,
   CDC_VARIABLE_BIT_RATE_ENCODING = 1
};
```

- ***Field Frame Pictures***

```
enum CDC_FIELD_FRAME_PICTURES {
```

```
   CDC_FRAME_PICTURES=0,
   CDC_FIELD_PICTURES
};
```

♦ ***GOP Structure***

```
enum CDC_GOP_STRUCTURE {
   CDC_GOP_I_FRAME,
   CDC_GOP_IP_FRAME,
   CDC_GOP_IBP_FRAME,
   CDC_GOP_IBP_FRAME_CLOSED,
   CDC_GOP_IBBP_FRAME,
   CDC_GOP_IBBP_FRAME_CLOSED
};
```

♦ ***Chroma Format***

```
enum  CDC_CHROMA_FORMAT {
   CDC_CHROMA_FORMAT_420,
   CDC_CHROMA_FORMAT_422,
   CDC_CHROMA_FORMAT_411
};
```

♦ ***MPEG Version***

```
enum CDC_MPEG_VERSION {
   CDC_MPEG_VERSION_MPEG1 = 1,
   CDC_MPEG_VERSION_MPEG2 = 2
};
```

♦ ***Audio Format***

```
enum CDC_AUDIO_FORMAT {
   CDC_AUDIO_FORMAT_32K_LAYER1=0,
   CDC_AUDIO_FORMAT_48K_LAYER1=2,
   CDC_AUDIO_FORMAT_RESERVED=3,
   CDC_AUDIO_FORMAT_32K_LAYER2=4,
   CDC_AUDIO_FORMAT_48K_LAYER2=6,
   CDC_AUDIO_FORMAT_16UNCOMPRESSED=7,
   CDC_AUDIO_FORMAT_20UNCOMPRESSED=8,
   CDC_AUDIO_FORMAT_24UNCOMPRESSED=9,
   CDC_AUDIO_FORMAT_32UNCOMPRESSED=10,
   CDC_DATA_FORMAT_16UNCOMPRESSED=11,
   CDC_DATA_FORMAT_20UNCOMPRESSED=12,
   CDC_DATA_FORMAT_24UNCOMPRESSED=13,
   CDC_DATA_FORMAT_32UNCOMPRESSED=14,
   CDC_AUDIO_FORMAT_SMPTE302_16,
   CDC_AUDIO_FORMAT_SMPTE302_20,
   CDC_AUDIO_FORMAT_SMPTE302_24,
   CDC_AUDIO_FORMAT_SMPTE331,
   CDC_DATA_FORMAT_SMPTE302_16,
```

```
   CDC_DATA_FORMAT_SMPTE302_20,
   CDC_DATA_FORMAT_SMPTE302_24,
   CDC_DATA_FORMAT_SMPTE331,
};
```

- ***VBI Source***

```
enum ENC_VBI_SOURCE {
   ENC_VBI_SOURCE_NONE = 0,
   ENC_VBI_SOURCE_16LINES_COMP,
   ENC_VBI_SOURCE_32LINES_COMP,
   ENC_VBI_SOURCE_UNCOMP
};
```

- ***VBI Type***

```
enum ENC_VIDEO_VBI_TYPE {
   ENC_VBI_CC = 0,
         ENC_VBI_TELETEXT,
         ENC_VBI_VITC
};
```

- ***File Fork***

```
enum FILE_FORK {
   FILE_FORK_MPEG_DATA = 0,
   FILE_FORK_INDEX_DATA,
   FILE_FORK_PRIVATE_DATA
};
```

- ***Archive Service Type***

```
enum ARCHIVE_SERVICE_TYPE {
ARCHIVE_SERVICE_TYPE_OTG = 0, //Local cache type, the service path could tack as
"C:\cache"
ARCHIVE_SERVICE_TYPE_WIN32 = 1, //same as ARCHIVE_SERVICE_TYPE_OTG
ARCHIVE_SERVICE_TYPE_VSTRM = 2, //LocalVSTRM type, this type archive service is
only valid on BML //server, MediaClient system is not support this type archive
ARCHIVE_SERVICE_TYPE_CLUSTER = 3, //this type is same as
ARCHIVE_SERVICE_TYPE_REMOTE_VSTRM
ARCHIVE_SERVICE_TYPE_REMOTE_VSTRM = 3, //Implement archive with VSTRMAPI,
                                        the service path //should be setting as the
                                        IP address or hostname for //Video
                                        storage(e.g.BMLE)
ARCHIVE_SERVICE_TYPE_VIP_STREAMING = 4}  //Implement archive with RPCAPI, the
                                          service path //should be setting as the IP
                                          address or hostname for //Video
                                          storage(e.g.BMLE)
```

- ***Archive Request Status***

```
enum IdRequestStatus
```

```
{
        ID_REQUEST_UNKNOWN = -2,        // never requested
        ID_REQUEST_OUT_OF_DATE,                 // out of date
        ID_REQUEST_PENDING,                     // pending request
        ID_REQUEST_FOUND,               // found
        ID_REQUEST_NOT_FOUND            // not found
    };
```

- ***Archive Request State***

```
enum ARCHIVE_REQUEST_STATE {
  ARCHIVE_REQUEST_NEW,
  ARCHIVE_REQUEST_PENDING,
  ARCHIVE_REQUEST_PROCESSING,
  ARCHIVE_REQUEST_COMPLETE,
  ARCHIVE_REQUEST_FAILED_ACTIVE,
  ARCHIVE_REQUEST_FAILED_INACTIVE,
  ARCHIVE_REQUEST_ABORTING
};
```

- ***VideoStandard***

```
enum VideoStandard
{
  K_TYPE_NTSC_DF = 0,
  K_TYPE_NTSC_NDF = 1,
  K_TYPE_PAL = 2,
  K_TYPE_PAL_M = 3
};
```

- ***generic device states***

```
#define DEVICE_STATE_IDLE              (1 <<  0)
#define DEVICE_STATE_BUSY              (1 <<  6)
#define DEVICE_STATE_RESET             (1 << 29)
#define DEVICE_STATE_INOP              (1 << 30)
#define DEVICE_STATE_MORE_STATES       (1 << 31)
```

- ***device states for codec***

```
#define DEVICE_STATE_CUEING            (1 <<  1)
#define DEVICE_STATE_INITIALIZING       DEVICE_STATE_CUEING
#define DEVICE_STATE_PLAYING           (1 <<  2)
#define DEVICE_STATE_RECORDING          DEVICE_STATE_PLAYING
#define DEVICE_STATE_PAUSED            (1 <<  3)
#define DEVICE_STATE_JOGGING           (1 <<  4)
#define DEVICE_STATE_PLAY_REQUESTED    (1 <<  5)
#define DEVICE_STATE_RECORD_REQUESTED   DEVICE_STATE_PLAY_REQUESTED
#define DEVICE_STATE_SHUTTLING         (1 << 10)
#define DEVICE_STATE_CUED              (1 <<  7)
```

```
#define DEVICE_STATE_INITIALIZED          DEVICE_STATE_CUED
#define DEVICE_STATE_FAST_FORWARD         (1 << 11)
#define DEVICE_STATE_FAST_REWIND          (1 << 12)
#define DEVICE_STATE_STOPPING             (1 << 13)
#define DEVICE_STATE_LOADED               (1 << 14)
#define DEVICE_STATE_PRIMING              (1 << 15)
#define DEVICE_STATE_PRIMED               (1 << 16)
```

- ***Label Level***

```
#define LEVEL_PRIVATE     0
#define LEVEL_PUBLIC      1
#define LEVEL_ENGINEERING 2
```

- ***Signal status***

```
#define DEVICE_STATUS_GENLOCK_SIGNAL_PRESENT          (1 <<  0)
#define DEVICE_STATUS_LTC_SIGNAL_PRESENT              (1 <<  1)
#define DEVICE_STATUS_HOUSE_CLOCK_SIGNAL_PRESENT      (1 <<  2)
#define DEVICE_STATUS_VIDEO_SIGNAL_PRESENT            (1 <<  3)
#define DEVICE_STATUS_AUDIO_CHANNEL1_SIGNAL_PRESENT           (1 <<  4)
#define DEVICE_STATUS_AUDIO_CHANNEL2_SIGNAL_PRESENT           (1 <<  5)
#define DEVICE_STATUS_AUDIO_CHANNEL3_SIGNAL_PRESENT           (1 <<  6)
#define DEVICE_STATUS_AUDIO_CHANNEL4_SIGNAL_PRESENT           (1 <<  7)
#define DEVICE_STATUS_EMBEDDED_AUDIO_CHANNEL1_SIGNAL_PRESENT  (1 <<  8)
#define DEVICE_STATUS_EMBEDDED_AUDIO_CHANNEL2_SIGNAL_PRESENT  (1 <<  9)
#define DEVICE_STATUS_EMBEDDED_AUDIO_CHANNEL3_SIGNAL_PRESENT  (1 <<  10)
#define DEVICE_STATUS_EMBEDDED_AUDIO_CHANNEL4_SIGNAL_PRESENT  (1 <<  11)
#define DEVICE_STATUS_AUDIO_CHANNEL5_SIGNAL_PRESENT           (1 <<  12)
#define DEVICE_STATUS_AUDIO_CHANNEL6_SIGNAL_PRESENT           (1 <<  13)
#define DEVICE_STATUS_AUDIO_CHANNEL7_SIGNAL_PRESENT           (1 <<  14)
#define DEVICE_STATUS_AUDIO_CHANNEL8_SIGNAL_PRESENT           (1 <<  15)
#define DEVICE_STATUS_EMBEDDED_AUDIO_CHANNEL5_SIGNAL_PRESENT  (1 <<  16)
#define DEVICE_STATUS_EMBEDDED_AUDIO_CHANNEL6_SIGNAL_PRESENT  (1 <<  17)
#define DEVICE_STATUS_EMBEDDED_AUDIO_CHANNEL7_SIGNAL_PRESENT  (1 <<  18)
#define DEVICE_STATUS_EMBEDDED_AUDIO_CHANNEL8_SIGNAL_PRESENT  (1 <<  19)
```

- ***Image Resolution***

```
#define CDC_IMAGE_WIDTH_MIN             352/*360*/
#define CDC_IMAGE_WIDTH_FULL_RES        720
#define CDC_IMAGE_WIDTH_HALF_RES        352/*360*/
#define CDC_IMAGE_WIDTH_MAX             720
#define CDC_DEF_IMAGE_WIDTH             720
#define CDC_IMAGE_HD_WIDTH_1280         1280
#define CDC_IMAGE_HD_WIDTH_1440         1440
#define CDC_IMAGE_HD_WIDTH_1920         1920
#define CDC_IMAGE_HEIGHT_MIN            240
#define CDC_IMAGE_HEIGHT_FULL_NTSC      480
#define CDC_IMAGE_HEIGHT_HALF_NTSC      240
```

```
#define CDC_IMAGE_HEIGHT_FULL_PAL        576
#define CDC_IMAGE_HEIGHT_HALF_PAL        288
#define CDC_IMAGE_HEIGHT_MAX             576
#define CDC_DEF_IMAGE_HEIGHT_NTSC        480
#define CDC_DEF_IMAGE_HEIGHT_PAL         576
#define CDC_IMAGE_HD_HEIGHT_720          720
#define CDC_IMAGE_HD_HEIGHT_1080         1080
```

- ***Audio Source Extra***

```
#define MC_NO_AUDIO1_SOURCE_EXTRA        0x00
#define MC_DEF_AUDIO1_SOURCE_EXTRA       0x01
#define MC_NO_AUDIO2_SOURCE_EXTRA        0x00
#define MC_DEF_AUDIO2_SOURCE_EXTRA       0x23
#define MC_NO_AUDIO3_SOURCE_EXTRA        0x00
#define MC_DEF_AUDIO3_SOURCE_EXTRA       0x45
#define MC_NO_AUDIO4_SOURCE_EXTRA        0x00
#define MC_DEF_AUDIO4_SOURCE_EXTRA       0x67
```

- ***File Access Mode***

```
#define GENERIC_READ                (0x80000000L)
#define GENERIC_WRITE               (0x40000000L)
#define GENERIC_EXECUTE             (0x20000000L)
#define GENERIC_ALL                 (0x10000000L)
```

- ***VSTRM TIME CODE***

```
typedef ULONG TIME_CODE
```

Description:

TIME_CODE is a 32-bit value, encoded as follows:

	30 24 Hours	23 16 Minutes	15 8 Seconds	7 0 Frames

Each of the Hours, Minutes, Seconds, and Fields is encoded in separate binary fields in 24 hour format. For example, frame 6 of 1:23:45 PM is encoded as 0x0D172D06. Although 7 or 8 bits are allocated for each field, values are restricted as follows: hours 0-23; minutes 0-59; seconds 0-59; video fields 0-59 (NTSC) or 0-49 (PAL).

The “mode” bit (M) indicates whether the time specified is relative to the system time clock (value = 0), or relative to the stream time code (value = 1).

A value of “99:99:99:99” indicates “now”

- ***SD_CACHE_ELEMENT_V00***

```
typedef struct SD_CACHE_ELEMENT_V00
{
   TCHAR          szId[K_MAX_CACHE_ID_SIZE];
   LARGE_INTEGER   liFileSize;
   FILETIME        ftCreation;
   SD_BCD_TIME     rDuration;
   union
   {
      struct {
         unsigned char readonly : 1;
         unsigned char deleteprotected : 1;
         unsigned char writeprotected : 1;
      } fileattr;
      DWORD dwFileAttributes;
   } uFileAttr;
   BYTE           bType;
   DWORD          dwFileFlags;
   SD_BCD_TIME     rFirstFrame;
   union
   {
      struct {
         unsigned char resolution : 3;
         unsigned char streamtype : 2;
         unsigned char chromaformat : 1;
         unsigned char mpegtype: 1;
         unsigned char aspectratio : 1;
      } vidattr;
      BYTE bVideoAttributes;
     } uVidAttr;
 } SD_CACHE_ELEMENT_V00, *PSD_CACHE_ELEMENT_V00;
```

- ***SD_CACHE_ELEMENT***

```
typedef struct SD_CACHE_ELEMENT
   : public SD_CACHE_ELEMENT_V00
{
   union
   {
            struct
            {
                     BYTE   ucVideoResolution;
                     BYTE   ucStreamMultiplex;
                     BYTE   ucChromaFormat;
                     BYTE   ucVideoElementryStream;
                     BYTE   ucFrameRate;
                     BYTE   ucFrameFormat;
                     BYTE   ucDefinition;
```

```
                BYTE    fProxy : 1;
            } vidattr;
            ULONGLONG           ullVideoAttributes;
    } uVidAttrEx;
    DWORD       dwHorizontalLines;
    DWORD       dwVerticalLines;
    DWORD       dwOrigLength;
    bool    bIsSubClip;// Added on Apr.7th
    TCHARszMasterId[K_MAX_CACHE_ID_SIZE]; // Added on Apr.7th
} SD_CACHE_ELEMENT, *PSD_CACHE_ELEMENT;
```

- ***VSTRM_FIND_DATA***

```
typedef unsigned long       VFORK;
typedef struct _VSTRM_FIND_DATA {
    //
    // Returned information on a successful find. Add new fields by deducting from
    // the Reserved array so the structure size does not change!
    //
        WIN32_FIND_DATAA w;
        //
        // Fields not in base structure
        //
            ULONG UserFullPrivateDataLength;  // So the caller can allocate the right size
buffer to call GET_PRIVATE_DATA
            UCHAR UserBriefPrivateDataLength;// What is actually valid in the next array
            UCHAR
UserBriefPrivateData[MAX_USER_BRIEF_PRIVATE_DATA_LENGTH];
            UCHAR SearchFileName[MAX_SYSTEM_NAME_LENGTH];
            UCHAR SearchFilePath[MAX_NAME_LENGTH];
            VSTRM_FILE_VERSION VstrmFileVersion;
            USHORT ActiveReaderCount;
            USHORT ActiveWriterCount;
            //
            // Fields used internally to perform the next search
            //
            VHANDLE_TABLE   CurrentPortHandle;      // .type is the current file system
            BOOLEAN             UserSpecifiedPort;
            ULONG               CurrentFileIndex;
            PVOID           CallersContext;
            BOOLEAN             RaidRequest;
            ULONG               PortSpecificContext;
            PVOID           ClusterStateContext;
            ULONG               (*PortCallbackRoutine)();
            PVOID           RemoteCallersContext;
            VFORK           ForkNumber;

    } VSTRM_FIND_DATA, *PVSTRM_FIND_DATA, DLL_FIND_DATA,
*LPDLL_FIND_DATA, VSTRM_FIND_FORK_DATA, *PVSTRM_FIND_FORK_DATA,
DLL_FIND_FORK_DATA, *LPDLL_FIND_FORK_DATA;
```

Description

Short form of the VSTRM_FIND_DATA_LONG structure. The SearchFile... fields are set to the old size of ~65 bytes. Any addition of fields to this structure will require changes to the VSTRM_FIND_DATA_LONG structure as well.

♦ VSTRM_FIND_DATA_LONG

```
typedef struct _VSTRM_FIND_DATA_LONG {
        //
        // Returned information on a successful find. Add new fields by deducting from
        // the Reserved array so the structure size does not change!
        //
        WIN32_FIND_DATAA w;
          //
          // Fields not in base structure
          //
        ULONG UserFullPrivateDataLength;// So the caller can allocate the right size buffer to call
GET_PRIVATE_DATA
        UCHAR UserBriefPrivateDataLength;      // What is actually valid in the next array
        UCHAR UserBriefPrivateData[MAX_USER_BRIEF_PRIVATE_DATA_LENGTH];
        UCHAR SearchFileName[MAX_SYSTEM_NAME_LENGTH_LONG];
        UCHAR SearchFilePath[MAX_NAME_LENGTH_LONG];
        VSTRM_FILE_VERSION VstrmFileVersion;
        USHORT ActiveReaderCount;
        USHORT ActiveWriterCount;
        //
        // Fields used internally to perform the next search
        //
        VHANDLE_TABLE     CurrentPortHandle;      // .type is the current file system
        BOOLEAN           UserSpecifiedPort       : 1;
        BOOLEAN           RemoteRedispatch        : 1;
        ULONG             CurrentFileIndex;
        PVOID             CallersContext;
        BOOLEAN           RaidRequest;
        ULONG             PortSpecificContext;
        PVOID             ClusterStateContext;
        ULONG             (*PortCallbackRoutine)();
        PVOID             RemoteCallersContext;
        VFORK             ForkNumber;

        } VSTRM_FIND_DATA_LONG, *PVSTRM_FIND_DATA_LONG,
DLL_FIND_DATA_LONG, *LPDLL_FIND_DATA_LONG,
VSTRM_FIND_FORK_DATA_LONG, *PVSTRM_FIND_FORK_DATA_LONG,
DLL_FIND_FORK_DATA_LONG, *LPDLL_FIND_FORK_DATA_LONG;
```

Description

Long form of the VSTRM_FIND_DATA structure—The SearchFilefield—are set to the new size of ~260 bytes. Any addition of fields to this structure will require changes to the VSTRM_FIND_DATA

structure as well.

- ***WIN32_FIND_DATA***

The WIN32_FIND_DATA structure describes a file found by the FindFirstFile, FindFirstFileEx, or FindNextFile.

Declared in Winbase.h;

```
typedef struct _WIN32_FIND_DATA{ DWORD dwFileAttributes;
  FILETIME ftCreationTime;
  FILETIME ftLastAccessTime;
  FILETIME ftLastWriteTime;
  DWORD nFileSizeHigh;
  DWORD nFileSizeLow;
  DWORD dwReserved0;
  DWORD dwReserved1;
  TCHAR cFileName [MAX_PATH];
  TCHAR cAlternateFileName [14];
}
WIN32_FIND_DATA, *PWIN32_FIND_DATA;
```

- ***FILETIME***

The **FILETIME** structure is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC).Declared in Winbase.h;

```
typedef struct _FILETIME {
   DWORD dwLowDateTime;
   DWORD dwHighDateTime;
      }
FILETIME, *PFILETIME, *LPFILETIME;
```

- ***CLIPINFO***

```
typedef struct tagPlaylistClipInfo
{
      DWORD       dwIndex; /*!< Clip index in play list, this value
must be large than 0. 0 is invalid value. */

      unsigned char lpClipId[MAX_PATH];/*!< Clip name in play list. */
      unsigned char lpClipPath[MAX_PATH];/*!< Clip path in play list.
*/

      CTimeCode   ctMarkIn;               /*!< SOM. */
      CTimeCode   ctDuration;             /*!< Duration. */
      CTimeCode   ctStartTime;            /*!< Clip start time. */

      DWORD       dwEventType;            /*!< Event type. */
```

```
    DWORD       dwStatus;               /*!< Clip status. */

    SD_BRIEF_PRIVATE_DATA sPD;          /*!< Clip private data. */

    DWORD       dwLoopCount;

    tagPlaylistClipInfo *lpNext;        /*!< node pointer. */

} PL_CLIPINFO;
```

- ***Event type for playlist clip: PL_EVENT_TYPE***

```
enum PL_EVENT_TYPE
{
    PL_ET_AUTO = 0,
    PL_ET_HARD,
    PL_ET_MANUAL,
    PL_ET_LOOPIN,
    PL_ET_LOOPOUT
};
```

- ***Playlist clip attribute type for user query***

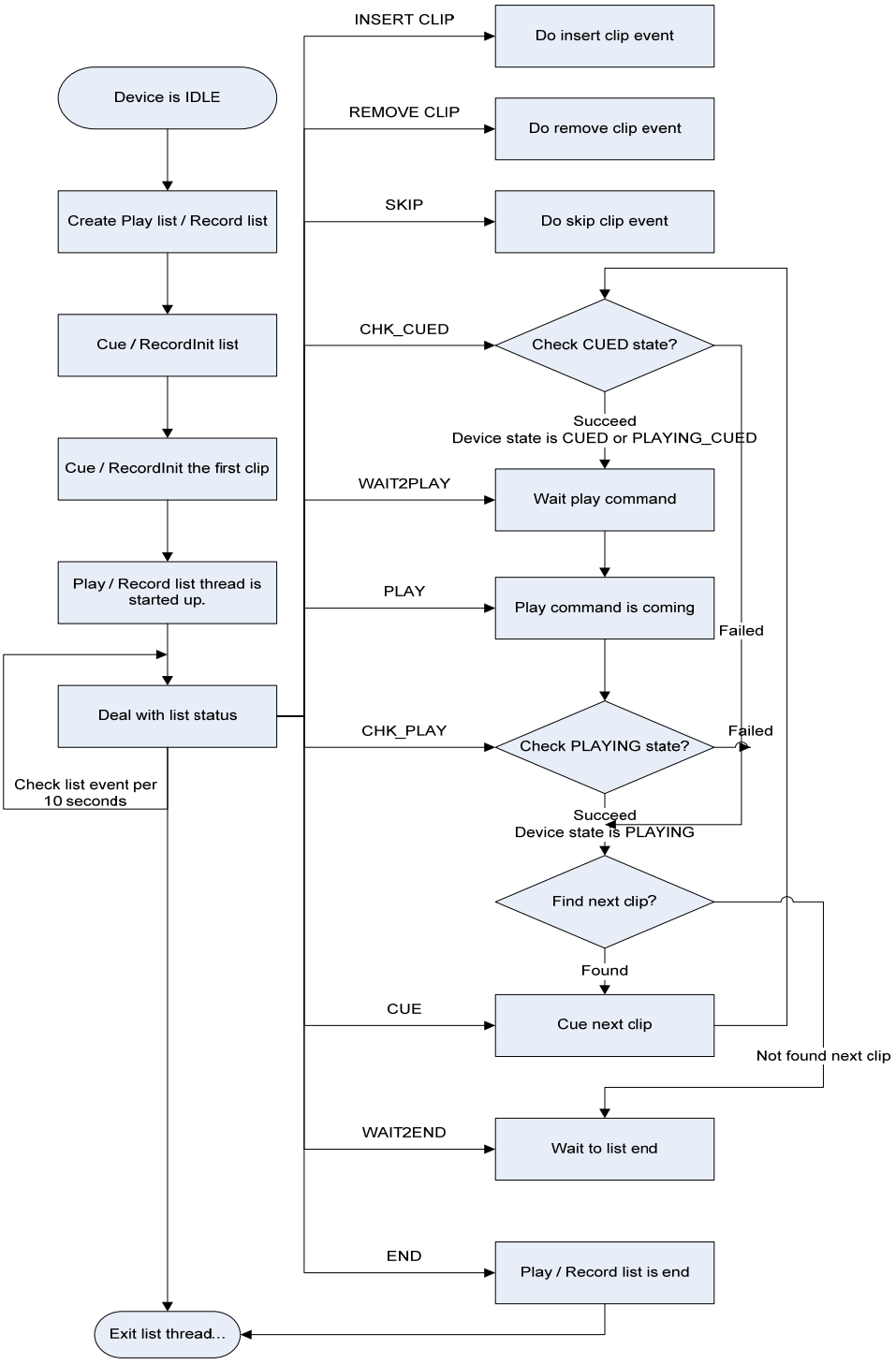
```
enum
{
    PL_ATTRIBUTE_TYPE_INDEX=0,
    PL_ATTRIBUTE_TYPE_ID,
    PL_ATTRIBUTE_TYPE_PATH,
    PL_ATTRIBUTE_TYPE_MARKIN,
    PL_ATTRIBUTE_TYPE_DURATION,
    PL_ATTRIBUTE_TYPE_START_TIME,
    PL_ATTRIBUTE_TYPE_EVENT_TYPE,
    PL_ATTRIBUTE_TYPE_STATUS,
    PL_ATTRIBUTE_TYPE_PD,
    PL_ATTRIBUTE_TYPE_LOOP_COUNT,
    PL_ATTRIBUTE_TYPE_MAX

};
```

- ***Error code for RPC***

```
#define RPC_SERVICE_INVALID_PARAMETER        0xFFFA
#define RPC_SERVICE_FILE_SESSION_NOT_FOUND   0xFFFB
#define RPC_SERVICE_FILE_CREATE_FAILED       0xFFFC
#define RPC_SERVICE_OPEN_SESSION_FAILED      0xFFFD
#define RPC_SERVICE_SESSION_NOT_FOUND        0xFFFE
#define RPC_SERVICE_ERROR                    0xFFFF
```

Appendix E: The status map of play list / record list



Appendix F: RemoteVstrmAPI Calls vs. MCCAPI Calls

Currently, some third-party systems are using Remote Vstrm API for getting asset information from BMLex storage. Now they also require getting asset information from UML storage or MediaServer. To meet the requirement, it is suggested to adopt MCC API instead of Remote Vstrm API to achieve the same functions.

Here below provides some alternative MCC API calls used to substitute corresponding RemoteVstrmAPI calls and gives some remarks about the difference between them for your reference.

- **VstrmFindFirstFile (IN HANDLE *classHandle,* IN LPCTSTR *fileName,* IN LPDLL_FIND_DATA *findData*)**

- **VstrmFindNextFile (IN HANDLE *classHandle,* IN VHANDLE *findfileHandle,* IN LPDLL_FIND_DATA *findData*)**

- **VstrmFindClose (IN HANDLE *classHandle,* IN LPCTSTR *fileHandle*)**

  Description

  They are used together to return information about the clip files matching the search criteria.

  Alternative MCCAPI Call

  CScAsCache::OpenCache ();

  CScAsCache::GetIdList ();

  CScAsCache::GetNext ();

  Refer to the detailed functions in

  DWORD OpenCache (HANDLEhStatusNotifyEvent=INVALID_HANDLE_VALUE)

  DWORD GetIdList (void)

  DWORD GetNext (CCacheElement*& pVid)

Remark

1) Same calls also can be supported by Windows API calls combination: FindFirstFile (), FindNextFile (), FindClose ().

2) This RemoteVstrmAPI call provides the function of getting file name instead of clip ID. User application must filter .pd/.vix file to get the clip files that user needs.

3) Using this MCCAPI call, it can get clipID directly without filter and also support reference formats.

- **VstrmCreateVixNameEx (IN HANDLE *vstrmClassHandle,* IN char **infile,* IN char **outfile*)**

Description

It is called by user application to generate the correct filename to be used in searching for a VIX file.

Alternative MCCAPI Call

Not support yet in MCC API.

- **VstrmGetFilePrivateData (IN HANDLE *classHandle,* IN VHANDLE *fileHandle,* IN PVOID *privateDataBuffer,* IN ULONG *privateDataLen,* OUT PULONG *retPrivateDataLen,* IN PVOID *briefDataBuf,* IN ULONG *briefDataBufLen,* OUT PULONG *retBrfDataLen*)**

Description

It is called by user application to retrieve the user private data and brief private data associated with the clip file.

Alternative MCCAPI Call

CScAsCache::GetFilePrivateData ();

Refer to the detailed function in

DWORD GetFilePrivateData (unsigned long ulFileHandle, unsigned long* pulType, CTimeCode* poDuration, LPBOOL lpbReadOnly) or

DWORD GetFilePrivateData (LPCTSTR lpszFileName, unsigned long* pulType, CTimeCode* poDuration, LPBOOL lpbReadOnly) or

DWORD GetFilePrivateData (unsigned long ulFileHandle, CBriefPrivateData& rBriefPrivateData) or

DWORD GetFilePrivateData (LPCTSTR lpszFileName, CBriefPrivateData& rBriefPrivateData)

Remark

1) Any one of the above four member functions can be used, depending on parameter type.

2) Only 12-bytes of brief private data can be retrieved by CScAsCache::GetFilePrivateData (); user private data can't be supported yet by MCC API.

3) Calls combination of CScAsCache::GetIdList () and CScAsCache::GetNext (CCacheElement* pVid) can get all of clip information.

- **VstrmGetVideoData (HANDLE *classHandle,* LPCTSTR *fileName,* VSTRM_VIDEO_DATA **buffer,* DWORD *bufferSize,* DWORD **returnSize*)**

Description

It is called by user application to retrieve information about the specified clip file located in the system.

Alternative MCCAPI Call

CScAsCache::GetFilePrivateData ();

Refer to the detailed function in

DWORD GetFilePrivateData (unsigned long ulFileHandle, unsigned long* pulType, CTimeCode* poDuration, LPBOOL lpbReadOnly) or

DWORD GetFilePrivateData (LPCTSTR lpszFileName, unsigned long* pulType, CTimeCode* poDuration, LPBOOL lpbReadOnly) or

DWORD GetFilePrivateData (unsigned long ulFileHandle, CBriefPrivateData& rBriefPrivateData) or

DWORD GetFilePrivateData (LPCTSTR lpszFileName, CBriefPrivateData& rBriefPrivateData)

Remark

1) Only FirstFrameLTC (SOM), VideoFormat and Duration of the specified clip file can be retrieved by CScAsCache::GetFilePrivateData ().

2) A calls combination of CScAsCache::GetIdList () and CScAsCache::GetNext (CCacheElement* pVid) can get all of clip information.

- **VstrmDeleteFile (IN HANDLE *classHandle,* IN LPCTSTR *fileName*)**

Description

It is called by user application to delete the specified clip file.

Alternative MCCAPI Call

CScAsCache::DeleteFile ();

Refer to the detailed function in DWORD DeleteFile (const CString& csVideoId).

Remark

MCCAPI call will delete specified clip file with related vix/pd files.

- **VstrmMoveFile (IN HANDLE *classHandle,* IN LPCTSTR *oldFileName,* IN LPCTSTR *newFileName*)**

Description

It is called by user application to rename the specified clip file.

Alternative MCCAPI Call

CScAsCache::RenameId ();

Refer to the detailed function in DWORD RenameId (const CString& csVideoIdOld, const CString& csVideoIdNew).

Remark

MCCAPI call will rename specified clip file with related vix/pd files.

- **VstrmSetFilePrivateData (IN HANDLE *classHandle,* IN VHANDLE *fileHandle,* IN PVOID *privateDataBuffer,* IN ULONG *privateDataLen,* IN PVOID *briefDataBuffer,* IN ULONG *briefBufferLen*)**

Description

It is called by user application to write private data for the clip file associated with the specified file handle.

Alternative MCCAPI Call

CScAsCache::SetFilePrivateData ();

Refer to the detailed function in

DWORD SetFilePrivateData (unsigned long ulFileHandle, CBriefPrivateData& rBriefPrivateData) or

DWORD SetFilePrivateData (LPCTSTR lpszFileName, CBriefPrivateData& rBriefPrivateData)

Remark

1) Either of the above two member functions can be used, depending on parameter type.

2) Only 12-byte of brief private data can be set by CScAsCache::SetFilePrivateData (); user private data can't be supported yet by MCC API.

- **VstrmFindFirstFIleNotification ()**

- **VstrmFindNextFileNotfification ()**

- **VstrmFindCloseFileNotification ()**

Description

They are used together to track changes to files stored in system. Once a file change occurs in SeaFile, it returns the change type and changed filename.

Alternative MCCAPI Call

Not support yet in MCCAPI.

Remark

Same call can also be supported by windows API calls combination of FindFirstFIleNotification (), FindNextFIleNotification (), FindCloseFIleNotification ().

- **VstrmClassCopyObject (IN HANDLE *classHandle,* IN IOCTL_CONTROL_PARMS **paramBufP,* IN ULONG *paramBufLength,* IN PVOID *(*cbP)(HANDLE classHandle, PVOID PIOCTL_STATUS_BUFFER, PVOID bufP, ULONG bufLen),* IN ULONG *CallBackParam)***

Description

It is called by user application to copy a file. The file can be copied to local storage with a new name (rename), or to another cluster.

Alternative MCCAPI Call

CScAsArchive::Copy ();

Refer to the detailed function function in

DWORD Copy (unsigned long* pulCopyHandle, const CString & csSrcServiceName, const CString& csDestServiceName, const CString & csSrcFileName, const CString& csDestFileName, const BOOL bIsMoveOperation)

Remark

1) Copy function is realized through EXD archive function in MCC API. Using CScAsArchive::Copy () has no callback function and parameters are also different from VstrmClassCopyObject ().

2) User can use MCC API calls combination of CScAsArchive::GetRequestList (), GetNextRequest (), GetArchiveCopyStatus () to get all the archive status.

Appendix G: Q&A

Q: Are the classes CScAsPlaylistMgr and CScAsDecoder supposed to be used simultaneously on the same connection, or using one of them excludes the other?
A: No, they cannot be used on same connection. They are exclusive.
So are CScAsRecordlistMgr and CScAsRecorder.

Q: Is trick mode supported on CScAsPlaylistMgr?
A: No. It does not support in this phase because we can not ensure B2B shuttle which means between two clips there should have idle screen or freeze frames while shuttling the playlist.

Q: How to create back to back playout/recording?
A: Refer to sample application SampleForPlaylistMgr or SampleForRecordlistMgr.

Q: Suppose that somehow I succeeded to start playing a back to back session. Now I want to add more clips to the session, and they too should play back to back with the existing ones (without stopping). How to insert clip while playing in CScAsPlaylistMgr?
A: The rule of inserting clip is that the clip must be inserted after next-to-play clip.Otherwise it will return FALSE. Thus it can ensure play back to back.

Q: I presume I can InsertClip (), but can I update my Cue() while playing?
A: No. you don’t need to send Cue again. MCL will automatically cue next clip in the playlist.
If you want to recue the clip, you must stop the playlist first as CScAsDecoder did.