USB Storage Data Playback API

Contents

1. Introduction

The playback library provides APIs for playback the stored data in the USB storage device via network or at local.

1.1. Provided Files

File	Description
StorageParser.dll/ StorageParser.h	Library for playback the stored files at local
RecorderTest	Sample program

2. Playback via network

2.1. Connections

For playback via network, two connections are required. One is for sending commands, the other is for retrieving data.

2.1.1. Connection for sending commands

The default port number of command is 2100. You can change the port number using property::port command or CGI command.

request

```
action=set&cmd=[cmd]&[ parameter]=[value]&[ parameter]=[value]…\r\n\r\n
action=get&cmd=[cmd]&[ parameter]&[ parameter] …\r\n\r\n
```

response

```
action=set&cmd=[cmd]&[parameter]=[ok|failed]&[ parameter]=[ok|failed]…\r\n\r\n
action=get&cmd=[cmd]&[parameter]=[value]&[ parameter]=[value]…\r\n\r\n
```

2.1.2. Connection for retrieving stream

The port number of stream is calculated as ‘command port number + 1’. In order to get streams, you have to set a session number retrieved from command port, ID and password that is encrypted by MD5 as follow after connecting to server with the port number for the stream..

```
session=[session number]&id=[id]&pwd=[password encrypted by MD5]
```

If the connection for command is disconnected, the session is disappeared and the connection for stream transfer also is disconnected. Only one client can connect to one session.

2.2. Command

login

Description

[set] This command sends ID and a password when the authorization is required.

Parameters

id
ID
pwd
Password. It must be encrypted byMD5.

Example

For example, supposed id is ‘root’ and password is ‘pass’.

Request

```
action=set&cmd=login&id=root&pwd=1A1DC91C907325C69271DDF0C944BC72\r\n\r\n
```

Response

```
action=set&cmd=login&id=ok&pwd=ok\r\n\r\n
```

search_data

Description

[set] This command searches the item matched with the conditions that parameter specifies.

Parameters

data_type

specifies the data type as follows.

- video : searching video data.
- audio : searching audio data.

ch

specifies the channel number for searching. All you have to do is left-shift the channels and then combine left-shift channels using the bitwise OR operator as follow.

ch = ((channel 3) <<3) | ((channel 2) <<2) | ((channel 1)<< 1) | ((channel 0)<<0)
where the value of (channel x) is 1 if you want to search, else 0.

start_time

specifies the start time for searching as YYYYMMDDHHMMSS

stop_time

specifies the end time for searching as YYYYMMDDHHMMSS

Example

Require

```
action=set&cmd=search_data&data_type=video&data_type=audio&ch=15&start_time=20080408010000&stop_time=20080408230000
```

Response
If the data corresponds with search conditions exists, a response is as follow

```
action=set&cmd=search_data&data_type=ok&data_type=ok&ch=ok&start_time=ok&end_time=ok
```

If the data does not exist, a response is as follow

```
action=set&cmd=search_data&search_data=failed
```

start_playing

Description
[set] This command starts playing data searched from search_data command.

Parameter

ch
specifies the channel for playback.

mode
specifies the playback mode as follows
- frame : gets the data in frame. To get data of next frame, use *next_frame* command.
- continuous : gets the data continuously.

direction
specifies the playback direction as follows
- forward
- backward : It is supported when the codec is only MJPEG.

speed
specifies the playback speed. Available value is from 0 to 3 down to one decimal place. ‘0’ means data can be retrieved without interval between frames.

Example

Request

```
action=set&cmd=start_playing&ch=15&mode=continuous&direction=forward&speed=0.5
```

Response

```
action=set&cmd=start_playing&ch=ok&mode=ok&direction=ok&speed=ok
```

stop_playing

Description

[set] This command stops playback of current searched data.

Parameters

ch
specifies the channel for stopping playback.

Example

Request

```
action=set&cmd=stop_playing&ch=15
```

Response

```
action=set&cmd=stop_playing&ch=ok
```

pause_playing

Description

[set] This command suspends playback at the current position.

Parameters

ch
specifies the channel for pausing playback.

Example

Request

```
action=set&cmd=pause_playing&ch=15
```

Response

```
action=set&cmd=pause_playing&ch=ok
```

next_frame

Description

[get] This command steps the play one frame forward or reverse when *start_laying::mode* is *frame*.

Parameters

Example

Request

```
action=get&cmd=next_frame
```

Response

```
action=get&cmd=next_frame=ok
```

property

Description

This command gets or sets the properties of recording function.

Parameters

recycle
[get/set] specifies or gets the operation as follows when the storage is full.
none : stop to record
rotate : delete the oldest file and record continuously

port
[get/set] specifies or gets the port number for connecting the searching server.

file_size
[get/set] specifies or gets the size of one chunk file for recording. It can be 1 to 128 as Mbyte unit.

record_channel
[get] Channel number that is combined with bitwise OR operation

num_of_list
[get] Number of the item searched

serach_list
[get] Lists of the item searched.
The format is CC:YYYYMMDDHHmmSS-YYYYMMDDHHmmss
where (channel number):(start time)-(end time)
CC : channel number,
YYYY : year
MM : month
DD : day
HH : hour, in twenty-four-hour system
mm : minute
ss : second

Example

- recycle
Request

```
action=get&cmd=property&recycle
```

Response

```
action=get&cmd=property&recycle=rotate
```

Request

```
action=set&cmd=property&recycle=none
```

Response

```
action=set&cmd=property&recycle=ok
```

- port
Request

```
action=get&cmd=property&port
```

Response

```
action=get&cmd=property&port=2100
```

Request

```
action=set&cmd=property&port=3100
```

Response

```
action=set&cmd=property&port=ok
```

- num_of_list
Request

```
action=get&cmd=property&num_of_list
```

Response

```
action=get&cmd=property&num_of_list=1
```

- search_list
Request

```
action=get&cmd=property&search_list
```

Response

```
action=get&cmd=property&search_list=09:20080410091852-20080410091932
07:200804100015503-20080410015545
```

mount

Description

[get/set] This command mount or unmount the USB device or it represents the mount status of device.

Parameters

device
: Device to mount or to unmount

enable
: If *enable* is 1, it requests the device to mount or it indicates the device as mounted. If *enable* is 0, it requests the device to unmount or it indicates the device as unmounted.

Example

request

```
action=get&cmd=mount&device&enable
```

response

```
action=get&cmd=mount&device=usb&enable=1
```

request

```
action=set&cmd=mount&device=usb&enable=1
```

response

```
action=set&cmd=mount&device=ok&enable=ok
```

delete

Description

[set] This command delete one selected file or all files.

Parameters

file

File names to delete or ‘all’ for deleting all files.
* Only ‘all” is supported now.

Example

Request

```
action=set&cmd=delete&file=all
```

Response

```
Action=set&cmd=delete&file=ok
```

2. 3. Data structure

Each media data transferred via stream port has the header which contains information about the media data as follow structure.

```
union INDEX_MEDIA_INFO {
        unsigned int uAll;
        struct {
                unsigned int    uVer            : 4;
                unsigned int    uWidth          : 10;
                unsigned int    uHeight         : 10;
                unsigned int    reserved : 8;
        };
        struct {
                unsigned int    uVer            : 4
                unsigned int    uSampleRate : 8; // kHz
                unsigned int    uDataBits       : 1;    // 0: 8, 1: 16
                unsigned int    uChannel        : 1;    // 0: mono, 1: stereo
                unsigned int    reserved : 18;
        };
};
```

uVer
Version of header

uWidth
Width of image

uHeight
Height of image

uSampleRate
Sampling frequency of audio data, in KHz

uDataBits
Bits per sample.
0 : 8 bits
1 : 16 bits

uChannel
Number of channels in audio data.
0 : mono
1 : stereo

```
typedef struct _REC_DATA_HEADER {
        unsigned int            uVer;
        unsigned int            uDataType;
        unsigned int            uFrameType;
        unsigned int            uChannelNum;
        unsigned int            uDataSize;
        struct timeval          TimeStamp;
```

```
        MEDIA_INDEX_INFO    MediaInfo;
} REC_DATA_HEADER;
```

uVer
Version of header

uDataType
Type of media data
1 : Only video
2 : Only audio
3 : Both video and audio

uFrameType
Type of each frame as follows
1 : I frame of MPEG-4
2 : P frame of MPEG-4
3 : B frame of MPEG-4
4 : I frame of MPEG-2
5 : P frame of MPEG-2
6 : B frame of MPEG-2
7 : MJPEG
8 : PCM
9 : uLaw PCM
10 : aLaw PCM

uChannelNum
Channel number of media data

uDataSize
Size of media data

TimeStamp
Timestamp of media data. It is defined in time.h or Winsock2.h.

MediaInfo
Information according to media type

3. Parsing the storage file for local playback

Suppose you want to bring the USB storage from IP encoders or cameras to your desktop PC and want to playback the data stored on the USB storage. APIs for local playback might be required. APIs for this purpose, however, are not supported directly. Instead, Parsing APIs are provided to be used for parsing the storage file data into a frame by frame structure. You can build your own file format (such as *.avi file) based on the files parsed from the original file format.

Below are APIs for parsing the original file format.

PasInit

```
BOOL PasInit(char *pszPath);
```

Description
This function initializes and open a path which contains the stored files.

Parameter
pszPath
Path of folder that contains the storage files

Return value
It is FALSE if the folder does not exist.

PasGetData

```
BOOL PasGetData(PAS_DATA_INFO *pInfo);
```

Description
This function gets the media data frame by frame from the stored file.

Parameter
pInfo
Information of the media data in frame. Refer to PAS_DATA_INFO structure

Return Value
It is FALSE if anymore media data to retrieve does not exist.

PasRelease

```
BOOL PasRelease();
```

Description
This function releases the resources.

Return Value
TRUE if successful; otherwise FALSE.

PAS_DATA_INFO

```
struct _PAS_DATA_INFO {
        union {
```

```
                ULONG                   uAll[16];
                struct {
                        ULONG           uDataType;
                        ULONG           uFrameType;
                        ULONG           uCh;
                        ULONG           uSize;
                        BYTE            *pBuffer;
                        LARGE_INTEGER   TimeStamp;
                        PAS_MEDIA_INFO  MediaInfo;// Additional information
                };
        };
} PAS_DATA_INFO;
```

Members

uDataType

Type of media data
1 : Only video
2 : Only audio
3 : Both video and audio

uFrameType

Type of each frame as follows
1 : I frame of MPEG-4
2 : P frame of MPEG-4
3 : B frame of MPEG-4
4 : I frame of MPEG-2
5 : P frame of MPEG-2
6 : B frame of MPEG-2
7 : MJPEG
8 : PCM
9 : uLaw PCM
10 : aLaw PCM

uChannelNum

Channel number of media data

uSize

Size of media data

pBuffer

Pointer of media data

TimeStamp

Timestamp of media data. It is defined in time.h or Winsock2.h.

MediaInfo

Information according to media type

PAS_MEDIA_INFO

```
struct _PAS_MEDIA_INFO {
        union {
```

```
                ULONG uAll[4];
                struct {
                        ULONG       uWidth;
                        ULONG       uHeight;
                };
                struct {
                        ULONG       uSampleRate;    // kHz
                        ULONG       uDataBits;      // 0: 8, 1: 16
                        ULONG       uChannel;       // 0: mono, 1: stereo
                };
        };
} PAS_MEDIA_INFO;
```

Members
uWidth
uWidth
Width of image

uHeight
Height of image

uSampleRate
Sampling frequency of audio data, in KHz

uDataBits
Bits per sample.
0 : 8 bits
1 : 16 bits

uChannel
Number of channels in audio data.
0 : mono
1 : stereo

3.1. Example

```
BOOL rs;
PAS_DATA_INFO data;

PasInit("e:\\data\");
do {
        rs =PasGetData(&data)
        if(rs) {
                // process the data
        }
}while(rs);
PasRelease();
```

Revision history

Revision	Date	Description
A	2008-04-10	Created