ACTi Linux SDK

C Edition

V2.0

Programming Guide

www.acti.com

ACTi Linux SDK

V2.1 Edition Dec. 2007

Table of Contents

1 Overview

Introduction

This SDK can help with application go beyond passive viewing to interact with the application developed by system integrator. This SDK provides a real time streaming to deliver live video and other surveillance functions controlling.

Start Up with Streaming Client Library

Streaming client Library is developed for ACTi Video Streaming camera & server.

It contains following abilities:

- Multicast and Unicast Streaming
- Embedded Time Code
- IO Controlling
- Event Notify from Server

In general, the SDK handles three types of traffics, Audio/Video (A/V) stream, Control Message session and URL which are treated independently. In Audio stream, there are two types of audio traffics, AudioIn and AudioOut. The AudioIn traffic is the audio data sampled by server and streamed to SDK. The AudioOut traffic is the audio data sampled by host (SDK was implemented on it) and streamed to server. The AudioOut data is treated as one of Control Message carried by Control Message session.

The AudioIn and AudioOut data was sampled at 8KHz/16bit format.

Decoding A/V data and sampling AudioIn are not included in the SDK.

A scenario of an application

Create a Connect Object

There is one structure need to be prepared for the connection.

It's:

```
CONECT_OBJ *connect_1;
connect_1=util_creat_connect_obj();
```

Connect Object "CONECT_OBJ" contains the configuration, private data and thread control variables (see "Data Structures- CONECT_OBJ"). Function "util_creat_connect_obj" can allocate memory for this Connect Object.

Setup callback functions.

Audio/Video data and control message can be accessed in the correspondent callback functions listed below.

```
util_set_error_callback(connect_1, &error_callback);
stream_set_video_callback(connect_1, &stream_video_callback);
stream_set_audio_callback(connect_1, &stream_audio_callback);
control_set_callback(connect_1, &control_callback);
```

Configure Connect Object

Some of the configuration must be set correctly before init the connect object. Initiation process needs these configurations to proceed to the preparation. We can get most of the configuration automatically by function “util_get_server_info”, but there are still four configurations must be setup before using “util_get_server_info”. These four configurations are “WAN_IP”, “PORT_HTTP”, “user_name” and “password”. For example

```
strcpy(connect_1->WAN_IP, "172.16.3.59");
connect_1->PORT_HTTP=80;
strcpy(connect_1->user_name, "Admin");
strcpy(connect_1->password, "123456");
util_get_server_info(connect_1);
```

“util_get_server_info” can get “V2_MULTICAST_IP”, “PORT_VIDEO”, “PORT_CONTROL”, “PORT_MULTICAST”, “V2_PORT_RTSP”, “ V2_STREAMING_METHOD” and “ V2_AUDIO_ENABLED” from server through URL Command. Users can set these configure values by themselves instead of using “util_get_server_info”.

Init Connect Object

After the configuration, “util_init_connect_obj” will use the connection object setting to prepare thread (Stream thread, Control thread and Audio_Out thread) control variables in Connect Object. For example

```
util_init_connect_obj(connect_1);
```

Start threads

Threads can connect to the server, receive data from the server and handle the data between the server and SDK.

Stream thread is used to receive Audio/Video stream. In the beginning of the thread start, the handshake of the streaming protocol (ACTi TCP2.0, ACTi Muticast, RTP) will be proceeded. If the handshake is success, “callback_stream_connected” in the Connect Object will be launched. After “callback_stream_connected” return, Stream Thread begins to receive Audio/Video stream and call “callback_video”, and “callback_audio”. For example:

```
stream_start(connect_1);
```

Control thread is used to receive events or serial stream. In the beginning of the thread, the

handshake of the control session protocol (ACTi TCP2.0) will be proceeded. If the handshake is success, “callback_ callback_control_connected _connected” in the Connect Object will be launched. After “callback_ callback_control_connected _connected” return, Control Thread begins to receive control message and call “callback_control”. For example:

```
control_start(connect_1);
```

Stop threads

“stream_stop”, “control_stop” and “audioout_stop” can stop the Stream Thread, Control Thread and Audio_Out Thread. These three functions only change the flags in Thread Control Variable to terminate the threads, the threads won’t stop immediately. Function “util_destroy_connect_obj” can check all threads been closed and release the Connect Object.

In this case, we only have to stop stream and control thread. For example:

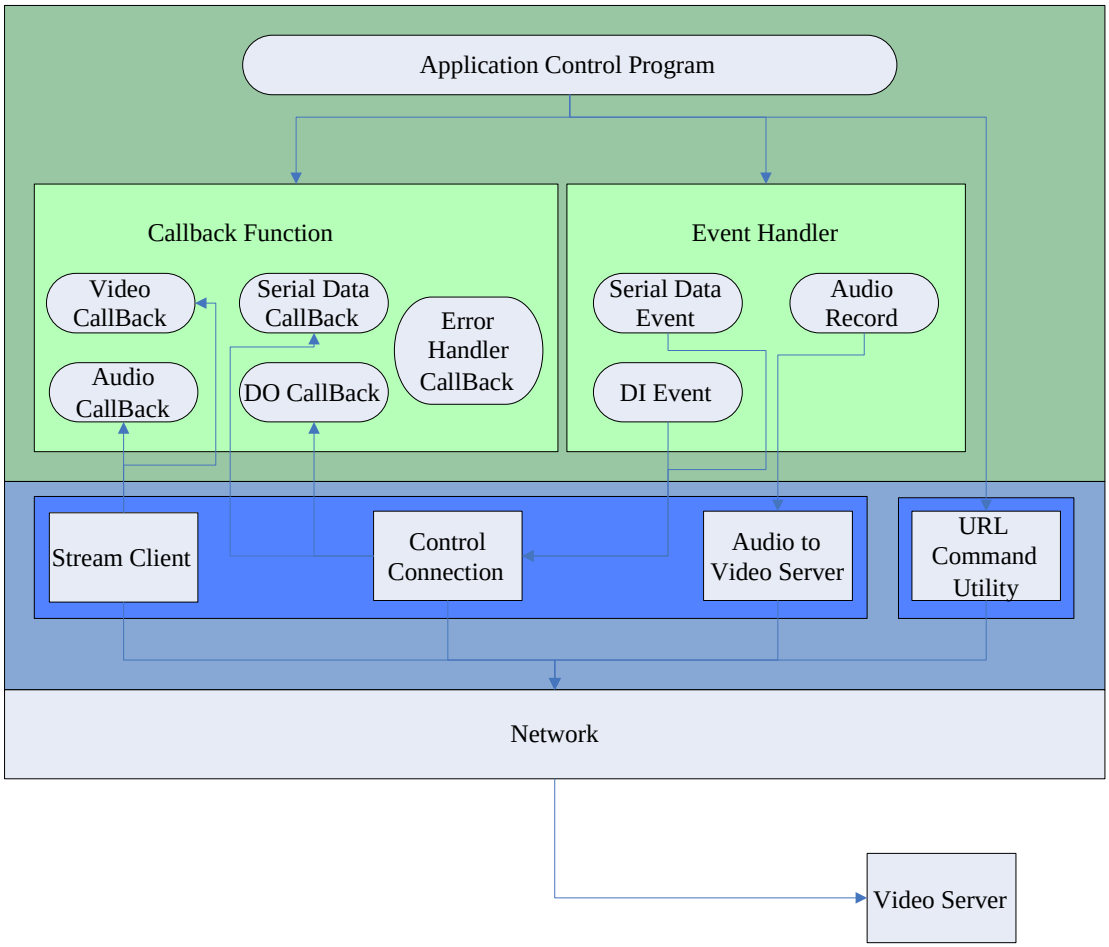
```
stream_stop(connect_1);
control_stop(connect_1);
```

Destroy Connect Object

When the Connect Object is not needed anymore, “util_destroy_connect_obj” must be used to release all the resource. Before “util_destroy_connect_obj” release resource, it will wait for all threads stop.

```
util_destroy_connect_obj(connect_1);
```

ACTi Linux Streaming SDK Architecture



2 DATA STRUCTURE

- CONECT_OBJ
- B2_HEADER
- VIDEO_PRIVATE_DATA
- AUDIO_PRIVATE_DATA
- VIDEO_B2_FRAME
- AUDIO_B2_FRAME
- HANDSHAKE_HEADER

CONECT_OBJ

```
typedef struct
{
    unsigned int seq;

    /* Server Configuration */
    char WAN_IP[16];           /*IP of video server*/
    char V2_MULTICAST_IP[16];  /*Muticast IP of video server*/
    int PORT_HTTP;             /*HTTP Port of video server*/
    int PORT_VIDEO;            /*Video Stream Port of video server*/
    int PORT_CONTROL;          /*Control Connection Port of video server*/
    int PORT_MULTICAST;        /*Multicast Stream Port of video server*/
    int V2_PORT_RTSP;          /*RTP Port of video server*/
    char user_name[32];        /*User Name of video server*/
    char password[64];         /*Password of video server*/
    int V2_STREAMING_METHOD;   /*(0:TCP Only,(1:Multicast Only,
                                 (2:TCP&Multicast,(3:RTP Over UDP,
                                 (4:RTP Over Multicast,
                                 (5:RTP Over UDP & Multicast*/
    int V2_AUDIO_ENABLED;      /*0)Disable, 1)Enable*/
    int VIDEO_VARFPS;          /*Variable frame rate*/
    int SOURCE_TYPE;           /*0)Video Server 1)Quad Video Server
                                 2)Multi-Channel Video Server*/
    int QUAD_CHANNEL;          /*Channel of Quad Video Server*/
    int multi_channel;         /*Channel of Multi-Channel Video Server*/

    /* Local Configuration */
    char Local_WAN_IP[16];         /*Local WAN ip address*/
    char Local_LAN_IP[16];         /*Local LAN ip address*/

    /* User Private Data */
    void *priv;   /* User can define there own private data structure and register
```

```
                    to connection object by using function
                    “util_set_priv_data”. */

    /* SDK private data : has not be modified by user. */
    char camera_name[32]; /* This is the camera name of the server. This
                             information will be get after stream
                             connection is ready. */

    /* Thread control */
    THREAD_T stream_thread;
    THREAD_T control_thread;
    THREAD_T audio_out_thread;

    /* Call Backs */
    void (* callback_video)(VIDEO_B2_FRAME *video_b2, void *arg);
    void (* callback_audio)(AUDIO_B2_FRAME *audio_b2, void *arg);
    void (* callback_control)(HANDSHAKE_HEADER *handshake_header, void *arg);
    void (* callback_stream_connected)(void *arg);
    void (* callback_audioout_connected)(void *arg);
    void (* callback_control_connected)(void *arg);
    void (* callback_err)(int err_maj, int err_min, void *arg);
}CONECT_OBJ;
```

B2_HEADER

```
typedef struct {
    unsigned char head[4];  /* 00 00 01 B2 */
    unsigned char msg_type;  /*    0x01) B2_VIDEO_MPEG4
                                   0x02) B2_AUDIO_8KPCM
                                   0x03) B2_AUDIO_TS_8KPCM
                                   0x04) B2_VIDEO_MJPEG
                                   0x05) B2_VIDEO_H264 */
    unsigned char stream_id;
    unsigned char ext_b2_len;
    unsigned char rsvd;
    unsigned int  len; /* The length of data that behinds B2 header include B2
                          private data and the media raw data. */
} B2_HEADER;
```

VIDEO_PRIVATE_DATA

```
typedef struct {
time_t       date_time; /* Timestamp in sec of encoded this video frame */
    unsigned char  time_zone; /* Mapping the TIME_ZONE in conf file. 0:-12, ...,
                                 24:+13 */

    unsigned char  video_loss;  /* 0) B2_VIDEO_LOSS_NOT_FOUND(video ok)
                                   1) B2_VIDEO_LOSS_FOUND(video loss) */

    unsigned char  motion;     /* 0) Motion not trigger 1) Motion trigger
                                  bit1) motion region1
                                  bit2) motion region2
                                  bit3) motion region3  */

    unsigned char  dio; /* bit0) DI1  pins level bit1) DI2  pins level */
    unsigned int    count; /* Video frame counter */
    unsigned char  resolution; /* Mapping the VIDEO_RESOLUTION in cond file.
                                  0:N720x480, ...*/
    unsigned char  bitrate; /* Mapping the VIDEO_BITRATE in cond file.
                               0:28K, ...*/
    unsigned char  fps_mode; /* Mapping the VIDEO_FPS in cond file.
                                0:MODE1(constant), 1:0:MODE2. */
    unsigned char  fps_number; /* In constant FPS mode, it indicates the video
                                  server's constant FPS number,
                                  i.e.atoi(VIDEO_FPS_NUM). in variable FPS
                                  mode, in indicates the variable FPS number
                                  which was requested by the TCP host. If it is
                                  not in TCP, it indicates the variable FPS
                                  number, i.e. atoi(VIDEO_VARIABLE_FPS) */
    struct timeval timestamp; /* Timestamp in usec of encoded this video frame*/
    unsigned char  reserved[8]; /* not used */
} VIDEO_PRIVATE_DATA;
```

AUDIO_PRIVATE_DATA

```
typedef struct {
    struct timeval timestamp; /* Timestamp in usec of encoded this video frame*/
    unsigned char  reserved[8];
} AUDIO_PRIVATE_DATA;
```

VIDEO_B2_FRAME

```
typedef struct {
    B2_HEADER        header;
    VIDEO_PRIVATE_DATA prdata;
} VIDEO_B2_FRAME;
```

AUDIO_B2_FRAME

```
typedef struct {
    B2_HEADER        header;
    AUDIO_PRIVATE_DATA prdata;
} AUDIO_B2_FRAME;
```

HANDSHAKE_HEADER

```
typedef struct {
    unsigned char head[4];  /* A C T i */
    unsigned int  msg_type;
/*
* RS485 used in the control session
*    0x00010035    CTRL_SERIAL_RECV_RSP
*    0x00000036    CTRL_SERIAL_SEND_REQ
* AUDIO_IN used in the control session
*    0x00000037    CTRL_AUDIO_PLAY_REQ
* VIDEO LOSS used in the control session
*    0x00010038    CTRL_VIDEO_LOSS_RSP
* MOTION used in the control session
*    0x00010039    CTRL_MOTION_DETECT_RSP
* CAMERA NAME in the control session
*    0x00000040    CTRL_CAMERA_NAME_REQ
*    0x00010040    CTRL_CAMERA_NAME_RSP
*/
    unsigned int  len; /* The length of payload data. */
} HANDSHAKE_HEADER;
```

3 PROGRAMMING GUIDE

Utility

- util_url_command
- util_url_get_setting
- util_get_server_info
- util_creat_connect_obj
- util_init_connect_obj
- util_destroy_connect_obj
- util_set_error_callback
- util_set_priv_data

util_url_command

Description

Send a URL Command to the server and get the return message. The return message will be put to return_value.

Syntax

```
int util_url_command(CONECT_OBJ *con, char *cmd_maj, char *cmd_min, char
*return_value, int return_value_len);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.
cmd_maj	char *	Video Server's CGI name like "system"or "mpeg4" etc.
cmd_min	char *	URL COMMAND of CGI like "SYSTEM_INFO", "ACCOUNT" etc.
return_value	char *	The return message will be putted here.
return_value_len	int	The maximum length of "return_value"

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

-3	LS_BUFFER_FULL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

Example

```
Unsigned char buf[256];
/*Creat connect obj*/
     connect_1=util_creat_connect_obj();

/*Config connect obj*/
     strcpy(connect_1->WAN_IP, "172.16.3.59");
     connect_1->PORT_HTTP=80;
     strcpy(connect_1->user_name, "Admin");
     strcpy(connect_1->password, "123456");

/*Get system info*/
     util_url_command(connect_1, "system", "SYSTEM_INFO", buf, 256);
     printf("%s\n", buf);
```

util_url_get_setting

Description

This function can get the value from the message return by server.

Syntax

```
int util_url_get_setting(CONECT_OBJ *con, char *cmd_maj, char *cmd_min, char
*return_value, int return_value_len);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.
cmd_maj	char *	Video Server's CGI name like "system"or "mpeg4" etc.
cmd_min	char *	URL COMMAND of CGI like "SYSTEM_INFO", "ACCOUNT" etc.
return_value	char *	The return value will be putted here.
return_value_len	int	The maximum length of "return_value"

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

Example

```
/*Creat connect obj*/
    connect_1=util_creat_connect_obj();

/*Config connect obj*/
    strcpy(connect_1->WAN_IP, "172.16.3.59");
    connect_1->PORT_HTTP=80;
    strcpy(connect_1->user_name, "Admin");
    strcpy(connect_1->password, "123456");

/*Get system info*/
```

```
util_url_command(connect_1, "system", " WAN_IP ", buf, 256);
printf("%s\n", buf);
```

util_get_server_info

Description

This function fills some settings in connection object database by reading configurations form video server through URLs. Several URLs listed below will be sent in the function.

V2_MULTICAST_IP,

PORT_HTTP,

PORT_VIDEO,

PORT_CONTROL,

PORT_MULTICAST,

V2_PORT_RTSP,

V2_STREAMING_METHOD,

V2_AUDIO_ENABLED.

Connection object's settings updated in this function are listed below

listed the variable in connection object

This function blocked for waiting server's reply.

Syntax

```
int util_get_server_info(CONECT_OBJ *con);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Remarks

There are four configurations MUST be set before using "util_get_server_info". The fore information are "WAN_IP", "PORT_HTTP", "user_name" and "password".

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

util_creat_connect_obj

Description

Connect Object “CONECT_OBJ” contains the configuration, private data and thread control variables (see “Data Structures- CONECT_OBJ”). Function “util_creat_connect_obj” can allocate memory for this Connect Object.

Syntax

```
CONECT_OBJ* util_creat_connect_obj(void);
```

Returns

A pointer indict to a CONECT_OBJ structure.

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

Example

```
CONECT_OBJ *connect_1;

    /*Creat connect obj*/
    connect_1=util_creat_connect_obj();

    /*Config connect obj*/
    strcpy(connect_1->WAN_IP, "172.16.3.59");
    connect_1->PORT_HTTP=80;
    strcpy(connect_1->user_name, "Admin");
    strcpy(connect_1->password, "123456");

    if(util_get_server_info(connect_1)==LS_SUCCESS)
    {
        printf("Multicast IP : %s\n", connect_1->V2_MULTICAST_IP);
        printf("Port of Video : %d\n", connect_1->PORT_VIDEO);
        printf("Port of Control : %d\n", connect_1->PORT_CONTROL);
        printf("Port of Muticast : %d\n", connect_1->PORT_MULTICAST);
        printf("Port of RTSP : %d\n", connect_1->V2_PORT_RTSP);
        printf("Streaming Method : %d\n", connect_1->V2_STREAMING_METHOD);
        printf("Audio Enabled : %d\n", connect_1->V2_AUDIO_ENABLED);
}
```

util_init_connect_obj

Description

Allocate memory and initialize the Thread Control Variables in Connect Object.

Syntax

```
int util_init_connect_obj(CONECT_OBJ* con);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

util_destroy_connect_obj

Description

When the Connect Object is not needed anymore, “util_destroy_connect_obj” must be used to release all the resource. Before “util_destroy_connect_obj” releases resource, it will be blocked until all threads stop.

Syntax

```
int util_destroy_connect_obj(CONECT_OBJ* con);
```

Parameters

Name	Type	Description
Con	CONECT_OBJ *	Connection Object of this connection.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

util_set_error_callback

Description

Set error callback function to a Connection Object.

Syntax

```
int util_set_error_callback(CONECT_OBJ* con, void *callback);
```

Parameters

Name	Type	Description
Con	CONECT_OBJ *	Connection Object of this connection.
Callback	void *(int err_maj, int err_min, void *arg)	Function pointer of the error callback function.

Call convention of call back

callback(int err_maj, int err_min, void *arg)

where

err_maj Major error code

err_min Miner error code

arg Pointer indicates to the Connect Object which have this error.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

util_set_priv_data

Description

Users can define their own data structure and attaches to the Connection Object by function “util_set_priv_data”.

Syntax

```
int util_set_priv_data(CONECT_OBJ* con, void *priv);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.
priv	void *	Function pointer of the data structure.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Remarks

Users have to allocate/release their own data structure.

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

Example

```
Typedef struct{
    Int seq;
    Char note[128];
}MY_DATA;

Main(void){
CONECT_OBJ *connect_1;
MY_DATA *my_data;

connect_1=util_creat_connect_obj();
my_data=malloc(sizeof(MY_DATA));
my_data->seq=1;
sprintf(my_data->note, “Data in connect object!\n”);
```

```
util_set_priv_data(connect_1, my_data);

……}
Video_callback (VIDEO_B2_FRAME* video_b2_frame, void *arg)
{
     CONECT_OBJ *con=(CONECT_OBJ *)arg;
     MY_DATA *my_data;
     my_data = con->priv;
     printf("Seq=%d, note: %s\n", my_data->seq, my_data->note);
     ……
}
```

4 Stream connection

Utility

- stream_set_video_callback
- stream_set_audio_callback
- stream_set _connected_callback
- stream_start
- stream_stop

stream_set_video_callback

Description

Set video callback function to the Connection Object.

Syntax

```
int stream_set_video_callback(CONECT_OBJ* con, void *callback);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.
callback	void *	Function pointer of the video stream callback function.

Call convension of call back

callback(VIDEO_B2_FRAME* video_b2_frame, void *arg)

Value	Description
video_b2_frame	Pointer indicates to the Video B2 Frame which have B2 Header, Video private data and the video stream RAW data.and the audio stream RAW data.
arg	Pointer indicates to the Connect Object which receives this video.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

stream_set_audio_callback

Description

Set audio callback function to the Connection Object.

Syntax

```
int stream_set_audio_callback(CONECT_OBJ* con, void *callback);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.
callback	void *	Function pointer of the video stream callback function.

Call convention of call back

callback(AUDIO_B2_FRAME* video_b2_frame, void *arg)

Value	Description
audio_b2_frame	Pointer indicates to the Audio B2 Frame which have B2 Header, Audio private data and the audio stream RAW data.
arg	Pointer indicates to the Connect Object which receives this audio.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

stream_set_connected_callback

Description

Set stream connection check callback function to the Connection Object.

Syntax

```
int stream_set_connected_callback (CONECT_OBJ* con, void *callback);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.
callback	void *	Function pointer of the stream connected callback function.

Call convention of call back

callback*(void *arg)

Value	Description
arg	Pointer indicates to the Connect Object which has the connection.

This callback function will be called right after the connection handshake success.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

stream_start

Description

Start the streaming thread to connect to server and receive Audio/Video stream.

This function returns right after the creation of the streaming thread. A/V Streaming Data won't be received immediately, but after the streaming protocol handshake proceed by streaming thread.

Syntax

```
int stream_start(CONECT_OBJ* con);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

stream_stop

Description

Stop the stream thread. This function only change the flags in Thread Control Variable to terminate the threads, the threads won't stop immediately.

Syntax

```
int stream_stop(CONECT_OBJ* con);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

5 Control connection

Utility

- control_set_callback
- contro_connected_set_callback
- control_start
- control_stop
- inline control_is_connected
- control_serial_send
- conrol_DIO_send

control_set_callback

Description

Set control callback function to the Connection Object.

Syntax

```
int control_set_callback(CONECT_OBJ* con, void *callback);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.
callback	void *	Function pointer of the control callback function.

Call convention of call back

callback(HANDSHAKE_HEADER *handshake_header, void *arg)

Value	Description
audio_b2_frame	Pointer indicates to the Audio B2 Frame which have B2 Header, Audio private data and the audio stream RAW data.
arg	Pointer indicates to the Connect Object which receives this audio.

Returns

Value	Description

0	LS_SUCCESS
-1	LS_FAIL

Remarks

The control callback function must have two arguments including a pointer indicates to the HANDSHAKE_HEADER and a pointer indicates to a CONECT_OBJ.

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

contro_set _connected _callback

Description

Set control connected callback function to the Connection Object.

Syntax

```
int contro_set _connected _callback(CONECT_OBJ* con, void *callback);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.
callback	void *	Function pointer of the control connected callback function.

Call convention of call back

callback(void *arg)

Value	Description
arg	Pointer indicates to the Connect Object .

This callback function will be called right after the handshake of control connection success.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

control_start

Description

Start the control thread to connect to server and exchange control message between SDK and the server.

This function returns right after the creation of control thread. Control message won't be received immediately until success in the handshake of the control session.

Syntax

```
int control_start(CONECT_OBJ* con);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

control_stop

Description

Stop the control thread. This function only change the flags in Thread Control Variable to terminate the thread, the threads won't stop immediately.

Syntax

```
int control_stop(CONECT_OBJ* con);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

control_is_connected

Description

Check if the control connection is alive or not. This function checks status flag in Thread Control Variable of Control connection and returns the connection status.

Syntax

```
int inline control_is_connected(CONECT_OBJ* con);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.

Returns

Value	Description
0	Not Connected
1	Connected

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

control_serial_send

Description

Send serial data to the server through control message session. It will be returned after the serial data has been send.

Syntax

```
int control_serial_send(CONECT_OBJ* con, char *buf, int len);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.
bud	char *	Serial data.
len	int	Length of the serial data.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

conrol_DIO_send

Description

Send DIO status message to the server through control message session. It will be returned after the DIO status message has been send.

Syntax

```
int conrol_DIO_send(CONECT_OBJ* con, int di1, int di2);
```

Parameters

Name	Type	Description
Con	CONECT_OBJ *	Connection Object of this connection.
di1	int	Status of DI1. 1:short 0:open
di2	int	Status of DI2. 1:short 0:open

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

6 Audio out connection

Utility

- audioout_connected_set_callback
- audioout_start
- audioout_stop
- audioout_send
- inline audioout_is_connected

audioout_set_connected_callback

Description

Set AudioOut Connected Callback function to the Connection Object.

Syntax

```
int audioout_set_connected_callback(CONECT_OBJ* con, void *callback);
```

Parameters

Name	Type	Description
Con	CONECT_OBJ *	Connection Object of this connection.
Callback	void *	Function pointer of the Audio Out Connected Callback function.

Call convention of call back

callback(void *arg)

Value	Description
arg	Pointer indicates to the Connect Object .

This callback function will be called right after the success in the handshake of AudioOut connection

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

audioout_start

Description

Start the AudioOut thread to connect to server. This function returns right after the creation of control thread. Before send audio data to server, users have to check the connection by using function “audioout_is_connected” to check the connection status.

Syntax

```
int audioout_start(CONECT_OBJ* con);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

audioout_stop

Description

Stop the AudioOut thread. This function only changes flags in Thread Control Variable to terminate the thread, the threads won't stop immediately.

Syntax

```
int audioout_stop(CONECT_OBJ* con);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

audioout_send

Description

Send AudioOut data to the server through AudioOut connection.

Syntax

```
int audioout_send(CONECT_OBJ* con, char *buf, int len);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.
buf	Char *	Audio data.
len	int	Length of the audio data.

Returns

Value	Description
0	LS_SUCCESS
-1	LS_FAIL

Remarks

Audio data MUST in 8K PCM MONO format and length MUST be equal to 4096 byte.

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

audioout_is_connected

Description

Check if the AudioOut connection is alive or not. This function checks status flag in Thread Control Variable of AudioOut thread and returns the connection status.

Syntax

```
int inline audioout_is_connected(CONECT_OBJ* con);
```

Parameters

Name	Type	Description
con	CONECT_OBJ *	Connection Object of this connection.

Returns

Value	Description
0	Not Connected
1	Connected

Requirements

Header files	linux_sdk.h
Link Object	linux_sdk_util.o

7 ERROR CODE

Major Error Code

```
LS_ERR_TCP3          -1
LS_ERR_MUL           -2
LS_ERR_RTP           -3
LS_ERR_CTRL          -4
LS_ERR_URL           -5
LS_ERR_AUDIOOUT      -6
```

Minor Error Code

Minor error code of Audio Out

Name	Error Code	Description
LS_ERR_AUDIOOUT_CONNECT	-1	<__audioout_thread>__audioout_connect FAIL!
LS_ERR_AUDIOOUT_LEN	-2	<audioout_send>Audio out data length MUST be 4096 bytes!
LS_ERR_AUDIOOUT_NOCONNECTION	-3	<audioout_send>Audio out didn't connect.
LS_ERR_AUDIOOUT_SEND1	-4	<audioout_send>Audio out header send FAIL!
LS_ERR_AUDIOOUT_SEND2	-5	<audioout_send>Audio out data send FAIL!

Minor error code of Control thread

Name	Error Code	Description
LS_ERR_CTRL_SOCKET	-1	<__control_connect>create_tcp_socket FAIL
LS_ERR_CTRL_CONNECT	-2	<__control_connect> Can't connect
LS_ERR_CTRL_AUTHEN_REQ	-3	<__control_connect> Can't send CTRL_AUTHEN_REQ
LS_ERR_CTRL_AUTHEN_RSP	-4	<__control_connect> Can't read CTRL_AUTHEN_RSP
LS_ERR_CTRL_RSP_ERR	-5	<__control_connect>CTRL_RSP_ERR
LS_ERR_CTRL_LIVE_REQ	-6	<__control_live_check> Can't send CTRL_LIVE_REQ
LS_ERR_CTRL_EXIT_REQ	-7	<__control_exit> Can't send CTRL_EXIT_REQ

LS_ERR_CTRL_READ_HEADER	-8	<__control_thread>read HANDSHAKE_HEADER FAIL!
LS_ERR_CTRL_READ_DATA	-9	<__control_thread>read DATA FAIL!

Minor error code of Unicast Streaming

Name	Error Code	Description
LS_ERR_TCP3_SOCKET	-1	<__tcp3_connect>create_tcp_socket FAIL
LS_ERR_TCP3_CONNECT	-2	<__tcp3_connect> Can't connect
LS_ERR_TCP3_AUTHEN_REQ	-3	<__tcp3_connect> Can't send CTRL_AUTHEN_REQ
LS_ERR_TCP3_AUTHEN_RSP	-4	<__tcp3_connect> Can't read CTRL_AUTHEN_RSP
LS_ERR_TCP3_RSP_ERR	-5	<__tcp3_connect>TCP3_RSP_ERR
LS_ERR_TCP3_READ_B2	-6	<__tcp3_thread>read b2_header FAIL!
LS_ERR_TCP3_CHECK_B2	-7	<__tcp3_thread>check b2_header FAIL!
LS_ERR_TCP3_READ_DATA	-8	<__tcp3_thread>read b2 payload FAIL!

Minor error code of RTP Streaming

Name	Error Code	Description
LS_ERR_RTP_CONNECT	-1	<__rtp_start> __rtp_connect FAIL
LS_ERR_RTP_RTCP_THREAD	-2	<__rtp_start>Creat __rtp_rtcp_thread FAIL
LS_ERR_RTP_VIDEO_THREAD	-3	<__rtp_start>Creat __rtp_video_thread FAIL
LS_ERR_RTP_AUDIO_THREAD	-4	<__rtp_start>Creat __rtp_audio_thread FAIL

Minor error code of Multicast Streaming

Name	Error Code	Description
LS_ERR_MUL_SOCKET	-1	mul_connect>create_tcp_socket FAIL
LS_ERR_MUL_CONNECT	-2	<__mul_connect>bind_socket FAIL
LS_ERR_MUL_INTERFACE	-3	<__mul_connect>set_mcast_interface FAIL
LS_ERR_MUL_SET_BUF	-4	<__mul_connect>set_rcvbuf FAIL
LS_ERR_MUL_RECEIVE	-5	<__mul_get_frame>read_udp_socket FAIL
LS_ERR_MUL_DATA_LEN_1	-6	<__mul_get_frame>data length error-1!
LS_ERR_MUL_DATA_LEN_2	-7	<__mul_get_frame>data length error-2!

8 SAMPLE CODES

- Get information from Video Server by URL commands
- Signal Camera
- Multi-Channel Video Server
- Quad Video Server
- Control Events(send & receive)
- Audio to Video Server

Get information from Video Server by URL commands

```
#include "common.h"
#include "header.h"
#include "linux_sdk.h"
#include "netlib.h"
#include "msgQ.h"

int main(void)
{
    CONECT_OBJ *connect_1;

    char buf[256];

    printf("Linux SDk %s Test AP - URL Command\n", VERSION);

    messageQ_create(&id_msg2mainloop, key_msg2mainloop);

    /*Creat connect obj*/
    connect_1=util_creat_connect_obj();

    /*Config connect obj*/
    strcpy(connect_1->WAN_IP, "172.16.3.59");
    connect_1->PORT_HTTP=80;
    strcpy(connect_1->user_name, "Admin");
    strcpy(connect_1->password, "123456");

    util_url_command(connect_1, "system", "SYSTEM_INFO", buf, 256);
    printf("%s\n", buf);

    util_url_get_setting(connect_1, "system", "V2_MULTICAST_IP", buf, 256);
    printf("%s\n", buf);

    if(util_get_server_info(connect_1)==LS_SUCCESS)
    {
```

```
        printf("Multicast IP : %s\n", connect_1->V2_MULTICAST_IP);
        printf("Port of Video : %d\n", connect_1->PORT_VIDEO);
        printf("Port of Control : %d\n", connect_1->PORT_CONTROL);
        printf("Port of Muticast : %d\n", connect_1->PORT_MULTICAST);
        printf("Port of RTSP : %d\n", connect_1->V2_PORT_RTSP);
        printf("Streaming Method : %d\n", connect_1->V2_STREAMING_METHOD);
        printf("Audio Enabled : %d\n", connect_1->V2_AUDIO_ENABLED);
    }

    /*Destroy connect obj*/
    if(util_destroy_connect_obj(connect_1)==LS_FAIL)
        printf("<main>util_destroy_connect_obj FAIL!");

    printf("Linux SDk %s Test AP EXIT-URL Command\n", VERSION);
    return 0;
}
```

Signal Camera

```
#include "common.h"
#include "header.h"
#include "linux_sdk.h"
#include "netlib.h"
#include "msgQ.h"

#define SUCCESS   0
#define FAIL      -1

int server_status=FAIL;

void stream_video_callback(VIDEO_B2_FRAME* video_b2_frame, void *arg)
{
     CONECT_OBJ *con=(CONECT_OBJ *)arg;
     char *raw_data;
     int raw_data_len;

     raw_data=(char *)video_b2_frame+VIDEO_B2_FRAME_LEN;
     raw_data_len=video_b2_frame->header.len-VIDEO_PRIVATE_DATA_LEN;

     printf("<stream_video_callback>con->seq=%d, MPEG4--0x%x, len=%d\n", con->seq,
raw_data[3], raw_data_len);
}

void stream_audio_callback(AUDIO_B2_FRAME* audio_b2_frame, void *arg)
{
     CONECT_OBJ *con=(CONECT_OBJ *)arg;
     char *raw_data;
     int raw_data_len;

     raw_data=(char *)audio_b2_frame+AUDIO_B2_FRAME_LEN;
     raw_data_len=audio_b2_frame->header.len-AUDIO_PRIVATE_DATA_LEN;

     printf("<stream_audio_callback>con->seq=%d, len=%d\n", con->seq, raw_data_len);
}

void control_callback(HANDSHAKE_HEADER *handshake_header, void *arg)
{
     struct timeval now;
     gettimeofday(&now, NULL);
     printf("%d-%d-%c%c%c%c ", (int)now.tv_sec, (int)now.tv_usec,
handshake_header->head[0], handshake_header->head[1], handshake_header->head[2],
handshake_header->head[3]);
     if(handshake_header->msg_type==CTRL_MOTION_DETECT_RSP)
```

```
        printf("<control_callback>CTRL_MOTION_DETECT_RSP\n");
    else if(handshake_header->msg_type==CTRL_DIO_INPUT_RSP)
        printf("<control_callback>CTRL_DIO_INPUT_RSP\n");
    else
    {
        printf("<control_callback>handshake_header->msg_type=0x%x\n",
handshake_header->msg_type);
    }
}

void error_callback(int err_maj, int err_min, void *arg)
{
    printf("<error_callback>err_maj-%d, err_min-%d\n", err_maj, err_min);
}

int main(void)
{
    CONECT_OBJ *connect_1;
    s_message message;

    printf("Linux SDk %s Test AP - Video Camera\n", VERSION);

    messageQ_create(&id_msg2mainloop, key_msg2mainloop);

    /*Creat connect obj*/
    connect_1=util_creat_connect_obj();


    /*Setup callback functions.*/
    util_set_error_callback(connect_1, &error_callback);
    stream_set_video_callback(connect_1, &stream_video_callback);
    stream_set_audio_callback(connect_1, &stream_audio_callback);
    control_set_callback(connect_1, &control_callback);

    /*Config connect obj*/
    strcpy(connect_1->WAN_IP, "172.16.3.59");
    connect_1->PORT_HTTP=80;
    strcpy(connect_1->user_name, "Admin");
    strcpy(connect_1->password, "123456");
    if(util_get_server_info(connect_1)==LS_SUCCESS)
    {
        printf("Multicast IP : %s\n", connect_1->V2_MULTICAST_IP);
        printf("Port of Video : %d\n", connect_1->PORT_VIDEO);
        printf("Port of Control : %d\n", connect_1->PORT_CONTROL);
        printf("Port of Muticast : %d\n", connect_1->PORT_MULTICAST);
        printf("Port of RTSP : %d\n", connect_1->V2_PORT_RTSP);
```

```
        printf("Streaming Method : %d\n", connect_1->V2_STREAMING_METHOD);
        printf("Audio Enabled : %d\n", connect_1->V2_AUDIO_ENABLED);


        /*Init connect obj*/
        if(util_init_connect_obj(connect_1)==LS_FAIL)
        {
            printf("<main>util_init_connect_obj FAIL!\n");
            return 0;
        }

        /*Start threads*/
        stream_start(connect_1);
        control_start(connect_1);

        /*Main loop*/
        printf("<Main Loop>start!\n");
        while(1)
        {
            if(messageQ_receve(id_msg2mainloop, &message, NONBOLCK)==MSG_SUCCESS)
            {
                printf("<Main Loop>Receive message, %s\n", message.name);
                break;
            }
            usleep(500*1000);
        }
        printf("<Main Loop>stop!\n");

        /*Stop threads*/
        stream_stop(connect_1);
        control_stop(connect_1);

        /*Destroy connect obj*/
        if(util_destroy_connect_obj(connect_1)==LS_FAIL)
            printf("<main>util_destroy_connect_obj FAIL!");
    }
    else
    {
        printf("<main>util_get_server_info FAIL!\n");
    }

    printf("Linux SDk %s Test AP EXIT\n", VERSION);
    return 0;
}
```

Multi-Channel Video Server

```
#include "common.h"
#include "header.h"
#include "linux_sdk.h"
#include "netlib.h"
#include "msgQ.h"

#define SUCCESS   0
#define FAIL      -1

void stream_video_callback(VIDEO_B2_FRAME* video_b2_frame, void *arg)
{
     CONECT_OBJ *con=(CONECT_OBJ *)arg;
     char *raw_data;
     int raw_data_len;

     raw_data=(char *)video_b2_frame+VIDEO_B2_FRAME_LEN;
     raw_data_len=video_b2_frame->header.len-VIDEO_PRIVATE_DATA_LEN;

     printf("<stream_video_callback>con->seq=%d, MPEG4--0x%x, len=%d\n", con->seq,
raw_data[3], raw_data_len);
}

void stream_audio_callback(AUDIO_B2_FRAME* audio_b2_frame, void *arg)
{
     CONECT_OBJ *con=(CONECT_OBJ *)arg;
     char *raw_data;
     int raw_data_len;

     raw_data=(char *)audio_b2_frame+AUDIO_B2_FRAME_LEN;
     raw_data_len=audio_b2_frame->header.len-AUDIO_PRIVATE_DATA_LEN;

     printf("<stream_audio_callback>con->seq=%d, len=%d\n", con->seq, raw_data_len);
}

void control_callback(HANDSHAKE_HEADER *handshake_header, void *arg)
{
     struct timeval now;
     gettimeofday(&now, NULL);
     printf("%d-%d-%c%c%c%c ", (int)now.tv_sec, (int)now.tv_usec,
handshake_header->head[0], handshake_header->head[1], handshake_header->head[2],
handshake_header->head[3]);
     if(handshake_header->msg_type==CTRL_MOTION_DETECT_RSP)
          printf("<control_callback>CTRL_MOTION_DETECT_RSP\n");
     else if(handshake_header->msg_type==CTRL_DIO_INPUT_RSP)
```

```
        printf("<control_callback>CTRL_DIO_INPUT_RSP\n");
    else
    {
        printf("<control_callback>handshake_header->msg_type=0x%x\n",
handshake_header->msg_type);
    }
}

void error_callback(int err_maj, int err_min, void *arg)
{
    printf("<error_callback>err_maj-%d, err_min-%d\n", err_maj, err_min);
}

int main(void)
{
    CONECT_OBJ *connect_1;
    s_message message;
    char buf[256];
    printf("Linux SDk %s Test AP - Muti-Channel Video Server\n", VERSION);

    messageQ_create(&id_msg2mainloop, key_msg2mainloop);

    /*Creat connect obj*/
    connect_1=util_creat_connect_obj();


    /*Setup callback functions.*/
    util_set_error_callback(connect_1, &error_callback);
    stream_set_video_callback(connect_1, &stream_video_callback);
    stream_set_audio_callback(connect_1, &stream_audio_callback);
    control_set_callback(connect_1, &control_callback);

    /*Config connect obj*/
    strcpy(connect_1->WAN_IP, "172.16.3.50");
    connect_1->PORT_HTTP=80;
    strcpy(connect_1->user_name, "Admin");
    strcpy(connect_1->password, "123456");

    util_url_command(connect_1, "system", "SYSTEM_INFO", buf, 256);
    printf("%s\n", buf);

    connect_1->multi_channel=4;/*Setup channel number*/

    if(util_get_server_info(connect_1)==LS_SUCCESS)
    {
        printf("Multicast IP : %s\n", connect_1->V2_MULTICAST_IP);
```

```
        printf("Port of Video : %d\n", connect_1->PORT_VIDEO);
        printf("Port of Control : %d\n", connect_1->PORT_CONTROL);
        printf("Port of Muticast : %d\n", connect_1->PORT_MULTICAST);
        printf("Port of RTSP : %d\n", connect_1->V2_PORT_RTSP);
        printf("Streaming Method : %d\n", connect_1->V2_STREAMING_METHOD);
        printf("Audio Enabled : %d\n", connect_1->V2_AUDIO_ENABLED);


        /*Init connect obj*/
        if(util_init_connect_obj(connect_1)==LS_FAIL)
        {
            printf("<main>util_init_connect_obj FAIL!\n");
            return 0;
        }

        /*Start threads*/
        stream_start(connect_1);
        control_start(connect_1);

        /*Main loop*/
        printf("<Main Loop>start!\n");
        while(1)
        {
            if(messageQ_receve(id_msg2mainloop, &message, NONBOLCK)==MSG_SUCCESS)
            {
                printf("<Main Loop>Receive message, %s\n", message.name);
                break;
            }
            usleep(500*1000);
        }
        printf("<Main Loop>stop!\n");

        /*Stop threads*/
        stream_stop(connect_1);
        control_stop(connect_1);

        /*Destroy connect obj*/
        if(util_destroy_connect_obj(connect_1)==LS_FAIL)
            printf("<main>util_destroy_connect_obj FAIL!");
    }
    else
    {
        printf("<main>util_get_server_info FAIL!\n");
    }
```

```
    printf("Linux SDk %s Test AP EXIT\n", VERSION);
    return 0;
}
```

Quad Video Server

```
#include "common.h"
#include "header.h"
#include "linux_sdk.h"
#include "netlib.h"
#include "msgQ.h"

#define SUCCESS   0
#define FAIL      -1

void stream_video_callback(VIDEO_B2_FRAME* video_b2_frame, void *arg)
{
     CONECT_OBJ *con=(CONECT_OBJ *)arg;
     char *raw_data;
     int raw_data_len;

     raw_data=(char *)video_b2_frame+VIDEO_B2_FRAME_LEN;
     raw_data_len=video_b2_frame->header.len-VIDEO_PRIVATE_DATA_LEN;

     printf("<stream_video_callback>con->seq=%d, MPEG4--0x%x, len=%d\n", con->seq,
raw_data[3], raw_data_len);
}

void stream_audio_callback(AUDIO_B2_FRAME* audio_b2_frame, void *arg)
{
     CONECT_OBJ *con=(CONECT_OBJ *)arg;
     char *raw_data;
     int raw_data_len;

     raw_data=(char *)audio_b2_frame+AUDIO_B2_FRAME_LEN;
     raw_data_len=audio_b2_frame->header.len-AUDIO_PRIVATE_DATA_LEN;

     printf("<stream_audio_callback>con->seq=%d, len=%d\n", con->seq, raw_data_len);
}


/*After stream connection is down, send DISPLAY command to set quad channel.*/
void stream_connected_callback(void *arg)
{
     char command[128];
     CONECT_OBJ * con=(CONECT_OBJ *)arg;
     printf("<stream_connected_callback-%d>\n", con->seq);

     sprintf(command, "DISPLAY=%d", con->QUAD_CHANNEL);
     util_url_command(con, "quad", command, NULL, 0);
```

```
}

void control_callback(HANDSHAKE_HEADER *handshake_header, void *arg)
{
#if 1
     struct timeval now;
     gettimeofday(&now, NULL);
     printf("%d-%d-%c%c%c%c ", (int)now.tv_sec, (int)now.tv_usec,
handshake_header->head[0], handshake_header->head[1], handshake_header->head[2],
handshake_header->head[3]);
     if(handshake_header->msg_type==CTRL_MOTION_DETECT_RSP)
          printf("<control_callback>CTRL_MOTION_DETECT_RSP\n");
     else if(handshake_header->msg_type==CTRL_DIO_INPUT_RSP)
          printf("<control_callback>CTRL_DIO_INPUT_RSP\n");
     else
     {
          printf("<control_callback>handshake_header->msg_type=0x%x\n",
handshake_header->msg_type);
     }
#endif
}

void error_callback(int err_maj, int err_min, void *arg)
{
     printf("<error_callback>err_maj-%d, err_min-%d\n", err_maj, err_min);
}

int main(void)
{
     CONECT_OBJ *connect_1;
     s_message message;

     printf(“Linux SDk %s Test AP - Quad Video Server\n", VERSION);

     messageQ_create(&id_msg2mainloop, key_msg2mainloop);

     /*Creat connect obj*/
     connect_1=util_creat_connect_obj();


     /*Setup callback functions.*/
     util_set_error_callback(connect_1, &error_callback);
     stream_set_video_callback(connect_1, &stream_video_callback);
     stream_set_audio_callback(connect_1, &stream_audio_callback);
     control_set_callback(connect_1, &control_callback);
```

```
    stream_set_stream_connected_callback(connect_1,
&stream_connected_callback);/*After stream connection is down, send DISPLAY command to
set quad channel.*/

    /*Config connect obj*/
    strcpy(connect_1->WAN_IP, "172.16.3.34");
    connect_1->PORT_HTTP=80;
    strcpy(connect_1->user_name, "Admin");
    strcpy(connect_1->password, "123456");

    connect_1->QUAD_CHANNEL=0;/*Setup quad channel*/

    if(util_get_server_info(connect_1)==LS_SUCCESS)
    {
        printf("Multicast IP : %s\n", connect_1->V2_MULTICAST_IP);
        printf("Port of Video : %d\n", connect_1->PORT_VIDEO);
        printf("Port of Control : %d\n", connect_1->PORT_CONTROL);
        printf("Port of Muticast : %d\n", connect_1->PORT_MULTICAST);
        printf("Port of RTSP : %d\n", connect_1->V2_PORT_RTSP);
        printf("Streaming Method : %d\n", connect_1->V2_STREAMING_METHOD);
        printf("Audio Enabled : %d\n", connect_1->V2_AUDIO_ENABLED);


        /*Init connect obj*/
        if(util_init_connect_obj(connect_1)==LS_FAIL)
        {
            printf("<main>util_init_connect_obj FAIL!\n");
            return 0;
        }

        /*Start threads*/
        stream_start(connect_1);
        control_start(connect_1);

        /*Main loop*/
        printf("<Main Loop>start!\n");
        while(1)
        {
            if(messageQ_receve(id_msg2mainloop, &message, NONBOLCK)==MSG_SUCCESS)
            {
                printf("<Main Loop>Receive message, %s\n", message.name);
                break;
            }
            usleep(500*1000);
        }
        printf("<Main Loop>stop!\n");
```

```
        /*Stop threads*/
        stream_stop(connect_1);
        control_stop(connect_1);

        /*Destroy connect obj*/
        if(util_destroy_connect_obj(connect_1)==LS_FAIL)
            printf("<main>util_destroy_connect_obj FAIL!");
    }
    else
    {
        printf("<main>util_get_server_info FAIL!\n");
    }



    printf(“Linux SDk %s Test AP EXIT\n”, VERSION);
    return 0;
}
```

Control Events (send & receive)

```
#include "common.h"
#include "header.h"
#include "linux_sdk.h"
#include "netlib.h"
#include "msgQ.h"

#define SUCCESS   0
#define FAIL      -1

void *control_thread(void *arg)
{
     CONECT_OBJ *con=(CONECT_OBJ *)arg;
     int flip=0;

     printf("<control_thread-%d>\n", con->seq);
     while(1)
     {
          usleep(1000*1000);
          printf("<control_thread>send serial data: ABCDEFG\n");
          control_serial_send(con, "ABCDEFG", 7);

          if(flip==0)
          {
               conrol_DIO_send(con, LS_DIO_SHORT, LS_DIO_OPEN);
               flip=1;
          }
          else
          {
               conrol_DIO_send(con, LS_DIO_OPEN, LS_DIO_SHORT);
               flip=0;
          }
     }

     pthread_detach(pthread_self());
     pthread_exit(NULL);
}

void control_callback(HANDSHAKE_HEADER *handshake_header, void *arg)
{
     char buf[128];

     if(handshake_header->msg_type==CTRL_MOTION_DETECT_RSP)
     {
          printf("<control_callback>CTRL_MOTION_DETECT_RSP\n");
```

```
    }
    else if(handshake_header->msg_type==CTRL_DIO_INPUT_RSP)
    {
        printf("<control_callback>CTRL_DIO_INPUT_RSP\n");
    }
    else if(handshake_header->msg_type==CTRL_SERIAL_RECV_RSP)
    {
        memcpy(buf, (char *)handshake_header+HANDSHAKE_HEADER_LEN,
handshake_header->len);
        buf[handshake_header->len]='\0';
        printf("<control_callback>CTRL_SERIAL_RECV_RSP\n");
        printf("<control_callback>Serial data: %s\n", buf);
    }
    else
    {
        printf("<control_callback>handshake_header->msg_type=0x%x\n",
handshake_header->msg_type);
    }
}

void error_callback(int err_maj, int err_min, void *arg)
{
    CONECT_OBJ *con=(CONECT_OBJ *)arg;
    printf("<error_callback-%d>err_maj-%d, err_min-%d\n", con->seq, err_maj,
err_min);
}

void control_connected_callback(void *arg)
{
    pthread_t thread_id;

    CONECT_OBJ *con=(CONECT_OBJ *)arg;
    printf("<control_connected_callback-%d>\n", con->seq);
    pthread_create(&thread_id, NULL, control_thread, (void *)con);
}

int main(void)
{
    CONECT_OBJ *connect_1;
    s_message message;
    char buf[256];
    printf("Linux SDk %s Test AP - Control\n", VERSION);

    messageQ_create(&id_msg2mainloop, key_msg2mainloop);

    /*Creat connect obj*/
```

```
connect_1=util_creat_connect_obj();


/*Setup callback functions.*/
util_set_error_callback(connect_1, &error_callback);
control_set_callback(connect_1, &control_callback);
contro_connected_set_callback(connect_1, &control_connected_callback);

/*Config connect obj*/
strcpy(connect_1->WAN_IP, "172.16.3.59");
connect_1->PORT_HTTP=80;
strcpy(connect_1->user_name, "Admin");
strcpy(connect_1->password, "123456");

util_url_command(connect_1, "system", "SYSTEM_INFO", buf, 256);
printf("%s\n", buf);

connect_1->multi_channel=4;/*Setup quad channel*/

if(util_get_server_info(connect_1)==LS_SUCCESS)
{
    printf("Multicast IP : %s\n", connect_1->V2_MULTICAST_IP);
    printf("Port of Video : %d\n", connect_1->PORT_VIDEO);
    printf("Port of Control : %d\n", connect_1->PORT_CONTROL);
    printf("Port of Muticast : %d\n", connect_1->PORT_MULTICAST);
    printf("Port of RTSP : %d\n", connect_1->V2_PORT_RTSP);
    printf("Streaming Method : %d\n", connect_1->V2_STREAMING_METHOD);
    printf("Audio Enabled : %d\n", connect_1->V2_AUDIO_ENABLED);


    /*Init connect obj*/
    if(util_init_connect_obj(connect_1)==LS_FAIL)
    {
        printf("<main>util_init_connect_obj FAIL!\n");
        return 0;
    }

    /*Start threads*/
    control_start(connect_1);

    /*Main loop*/
    printf("<Main Loop>start!\n");
    while(1)
    {
        if(messageQ_receve(id_msg2mainloop, &message, NONBOLCK)==MSG_SUCCESS)
        {
```

```
                printf("<Main Loop>Receive message, %s\n", message.name);
                break;
            }
            usleep(500*1000);
        }
        printf("<Main Loop>stop!\n");

        /*Stop threads*/
        control_stop(connect_1);

        /*Destroy connect obj*/
        if(util_destroy_connect_obj(connect_1)==LS_FAIL)
            printf("<main>util_destroy_connect_obj FAIL!");
    }
    else
    {
        printf("<main>util_get_server_info FAIL!\n");
    }



    printf("Linux SDk %s Test AP EXIT\n", VERSION);
    return 0;
}
```

Audio to Video Server

```
#include "common.h"
#include "header.h"
#include "linux_sdk.h"
#include "netlib.h"
#include "msgQ.h"

#define SUCCESS   0
#define FAIL      -1

#define AUDIO_BLOCK_SIZE   4096
#define AUDIO_BUF_SIZE AUDIO_BLOCK_SIZE *2
char audio_buf[AUDIO_BUF_SIZE];
int audio_buf_index=0;/*Which block*/
int audio_buf_head=0;/*Head pointer of audio data.*/

void stream_audio_callback(AUDIO_B2_FRAME* audio_b2_frame, void *arg)
{
     CONECT_OBJ *con=(CONECT_OBJ *)arg;
     char *raw_data;
     int raw_data_len;

     raw_data=(char *)audio_b2_frame+AUDIO_B2_FRAME_LEN;
     raw_data_len=audio_b2_frame->header.len-AUDIO_PRIVATE_DATA_LEN;

     if(audio_buf_head+raw_data_len>AUDIO_BUF_SIZE)
     {
          memcpy(audio_buf+audio_buf_head, raw_data, AUDIO_BUF_SIZE-audio_buf_head);
          memcpy(audio_buf, raw_data+(AUDIO_BUF_SIZE-audio_buf_head),
raw_data_len-(AUDIO_BUF_SIZE-audio_buf_head));
          audio_buf_head=raw_data_len-(AUDIO_BUF_SIZE-audio_buf_head);
          audioout_send(con, audio_buf+AUDIO_BLOCK_SIZE, AUDIO_BLOCK_SIZE);
          audio_buf_index=0;
     }
     else
     {
          memcpy(audio_buf+audio_buf_head, raw_data, raw_data_len);
          audio_buf_head+=raw_data_len;
          if(audio_buf_index==0 && audio_buf_head>AUDIO_BLOCK_SIZE)
          {
               audioout_send(con, audio_buf, AUDIO_BLOCK_SIZE);
               audio_buf_index=1;
          }
     }
```

```
    //printf("<stream_audio_callback>con->seq=%d,len=%d\n",con->seq,raw_data_len);
}

void error_callback(int err_maj, int err_min, void *arg)
{
    printf("<error_callback>err_maj-%d, err_min-%d\n", err_maj, err_min);
}

int main(void)
{
    CONECT_OBJ *connect_1;
    s_message message;

    printf("Linux SDk %s Test AP - Audio to Video Server\n", VERSION);

    messageQ_create(&id_msg2mainloop, key_msg2mainloop);

    /*Creat connect obj*/
    connect_1=util_creat_connect_obj();


    /*Setup callback functions.*/
    util_set_error_callback(connect_1, &error_callback);
    stream_set_audio_callback(connect_1, &stream_audio_callback);

    /*Config connect obj*/
    strcpy(connect_1->WAN_IP, "172.16.3.11");
    connect_1->PORT_HTTP=80;
    strcpy(connect_1->user_name, "Admin");
    strcpy(connect_1->password, "123456");
    if(util_get_server_info(connect_1)==LS_SUCCESS)
    {
        printf("Multicast IP : %s\n", connect_1->V2_MULTICAST_IP);
        printf("Port of Video : %d\n", connect_1->PORT_VIDEO);
        printf("Port of Control : %d\n", connect_1->PORT_CONTROL);
        printf("Port of Muticast : %d\n", connect_1->PORT_MULTICAST);
        printf("Port of RTSP : %d\n", connect_1->V2_PORT_RTSP);
        printf("Streaming Method : %d\n", connect_1->V2_STREAMING_METHOD);
        printf("Audio Enabled : %d\n", connect_1->V2_AUDIO_ENABLED);


        /*Init connect obj*/
        if(util_init_connect_obj(connect_1)==LS_FAIL)
        {
            printf("<main>util_init_connect_obj FAIL!\n");
```

```
            return 0;
        }

        /*Start threads*/
        audioout_start(connect_1);
        stream_start(connect_1);

        /*Main loop*/
        printf("<Main Loop>start!\n");
        while(1)
        {
            if(messageQ_receve(id_msg2mainloop, &message, NONBOLCK)==MSG_SUCCESS)
            {
                printf("<Main Loop>Receive message, %s\n", message.name);
                break;
            }
            usleep(500*1000);
        }
        printf("<Main Loop>stop!\n");

        /*Stop threads*/
        stream_stop(connect_1);
        audioout_stop(connect_1);

        /*Destroy connect obj*/
        if(util_destroy_connect_obj(connect_1)==LS_FAIL)
            printf("<main>util_destroy_connect_obj FAIL!");
    }
    else
    {
        printf("<main>util_get_server_info FAIL!\n");
    }



    printf("Linux SDk %s Test AP EXIT\n", VERSION);
    return 0;
}
```