# 云存储产品线接口文档
# V02.00.00



公司：深圳市泽宝网络科技有限公司

2019 年 4 月 29

| | | | | | |
|---|---|---|---|---|---|
| 文档版本号 | V02.00.00 | | | 保密等级 | 机密 |
| 概要需求文档版本号 | V02.00.00 | | | | |
| 产品文档版本号 | V02.00.00 | | | | |
| 测试用例文档版本号 | | | | | |
| 代码版本号 | V2.1.0.0 | | | | |
| 产品线 | VAVA 云存储 | | 归属部门/项目 | | IOT |
| 编　　制 | dawson | 审　　核 | | 批　　准 | |
| 日　　期 | 19-04-29 | 日　　期 | | 日　　期 | |

# 修改历史

| 序号 | 日期 | 3 说明 | 责任人 |
|---|---|---|---|
| 1 | 2019-04-29 | 创建初稿 | dawson |
| 2 | 2019-05-20 | 评审修改 | dawson |
| 3 | 2020-10-12 | V2.0.0 版本接口变更 | Dawson |

# 目录

1. 概述 ..... 5
2. 接口 ..... 5
2.1. SDK 初始化接口 ..... 5
2.2. 日志初始化接口 ..... 5
2.2.1. 日志等级枚举(E_SV_LOG_LEVEL)类型定义 ..... 5
2.2.2. 日志输出方式 ..... 6
2.3. 设置 TOKEN 获取接口 ..... 6
2.4. SDK 反初始化接口 ..... 6
2.5. 连接推流服务器 ..... 6
2.5.1. 消息回调接口 ..... 7
2.5.2. 回调通知事件枚举(E_SV_EVENT_TYPE)类型定义 ..... 7
2.6. 连接推流服务器 ..... 8
2.7. 连接推流服务器 ..... 8
2.8. 直播连接推流服务器 ..... 9
2.9. 发送音视频数据 ..... 9
2.9.1. 媒体数据编码类型枚举(E_SV_MEDIA_TYPE)类型定义 ..... 9
2.10. 发送元数据 ..... 10
2.10.1. 元数据类型枚举(E_SV_METADATA_TYPE)类型定义 ..... 10
2.10.2. 媒体数据加密类型枚举 ..... 10
2.11. 推流连接是否正常连接 ..... 11
2.12. 关闭推流连接 ..... 12
2.13. 获取推流 SDK 版本信息 ..... 12
3. 接入演示代码 ..... 12

# 1. 概述

Cloud Push SDK 是泽宝网络科技有限公司(Sunvalley) IOT 事业部云存储项目组开发的 IOT 嵌入式设备推流 sdk, 该 sdk 旨在为能够采集音视频媒体数据的联网设备提供高效稳定地将媒体数据推送到 Sunvalley 云端，并为用户提供便捷的保存和点播，直播等服务。此 SDK 将对外开放，以方便其他 IOT 终端快速便捷地接入 Sunvalley 并提供的云服务功能。

# 2. 接口

## 2.1. SDK 初始化接口

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVCloudPush_API_Initialize |
| 2 | 接口说明 | 初始化 SDK 全局变量，在 sdk 加载后调用 |
| 3 | 参数 1[input] | int memory_pool_size 内存缓冲池大小 |
| 4 | 参数 2[input] | int Max_conn_cont 最大连接数 |
| 4 | 返回值(int) | 成功返回 0，失败返回非 0 值(-1) |

## 2.2. 日志初始化接口

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVCloudPush_API_Init_log |
| 2 | 接口说明 | 初始化日志输出方式、输出等级以及文件日志输出目录 |
| 3 | 参数 1 [input] | E_SV_LOG_LEVEL eLogLevel:日志等级，详细定义见 2.2.1 |
| 4 | 参数 2 [input] | unsigned int nLogFlag：日志输出方式详见 2.2.2 |
| 5 | 参数 3 [input] | const char* pLogPath:文件日志输出目录，输出文件日志时此参数不能为 NULL |
| 6 | 返回值(int) | 成功返回 0，失败返回非 0 值(-1) |

### 2.2.1. 日志等级枚举(E_SV_LOG_LEVEL)类型定义

| No | 枚举定义 | 枚举值 | 描述 |
|---|---|---|---|
| 1 | E_LOG_LEVEL_VERB | 0x01 | 输出最详细日志(级别>= 0x01) |
| 2 | E_LOG_LEVEL_INFO | 0x02 | 输出信息以上级别的日志(级别>= 0x02) |
| 3 | E_LOG_LEVEL_MAIN | 0x03 | 输出主要信息以上级别日志(级别>= 0x03 则输出) |
| 4 | E_LOG_LEVEL_WARN | 0x04 | 输出警告以上级别日志(级别>= 0x04) |
| 5 | E_LOG_LEVEL_ERROR | 0x5 | 输出错误级别日志(级别>= 0x05s)s |
| 6 | E_LOG_LEVEL_DISABLE | 0x06 | 不输出日志 |

2.2.2. 日志输出方式

| No | 宏定义 | 枚举值 | 描述 |
|---|---|---|---|
| 1 | SV_LOG_OUTPUT_MODE_NONE | 0x00 | 不输出日志 |
| 2 | SV_LOG_OUTPUT_MODE_CONSOLE | 0x01 | 输出控制台日志 |
| 3 | SV_LOG_OUTPUT_MODE_FILE | 0x02 | 输出文件日志 |

eg:日志支持同时输出到控制台和文件,设置方式为：
int nLogFlag = SV_LOG_OUTPUT_MODE_CONSOLE | SV_LOG_OUTPUT_MODE_FILE;

## 2.3. 设置 TOKEN 获取接口

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVPush_API_Set_Token_Serve_Interface |
| 2 | 接口说明 | SDK 初始化之后设置 token 获取接口 |
| 3 | 参数[input] | ptoken_server_url |
| 4 | 返回值(int) | 成功返回 0，失败返回非 0 值(-1) |

## 2.4. SDK 反初始化接口

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVCloudPush_API_Uninitialize |
| 2 | 接口说明 | 销毁 SDK 全局变量，在 sdk 卸载之前调用 |
| 3 | 参数 | 无 |
| 4 | 返回值(int) | 成功返回 0，失败返回非 0 值(-1) |

## 2.5. 连接推流服务器

| No | | 信息 | 描述 |
|---|---|---|---|
| 1 | | 接口名称 | SVCloudPush_API_Connect |
| 2 | | 接口说明 | 连接推流服务器，开始推流 |
| 3 | | 参数 1 [input] | Const char* Url: 推流 url |
| 4 | | 参数 2[input] | Const char* ptoken |
| 5 | | 参数 3 [input, output] | event_callback pcbFun：消息回调通知接口，详见 2.4.1 |
| 6 | | 返回值(int) | 成功返回 0，失败返回非 0 值(-1) |

补充 URL 规则定义：rtmp://host:port/$appname/$streamname/$time

### 2.5.1. 消息回调接口

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | event_callback |
| 2 | 接口说明 | sdk 推流回调消息通知接口，由 app 调用层设置 |
| 3 | 参数 1 [input] | long lcid： SVCloudPush_API_Connect 返回的 connectid |
| 4 | 参数 2 [input] | E_SV_EVENT_TYPE eventType：回调事件类型，详见 2.4.2 |
| 5 | 参数 3 [input] | long wparam：辅助说明参数，通常为错误码 |
| 6 | 参数 4 [input] | long lparam：消息值 |
| 4 | 返回值(int) | 成功返回 0，失败返回非 0 值(-1) |

### 2.5.2. 回调通知事件枚举(E_SV_EVENT_TYPE)类型定义

| No | 枚举定义 | 枚举值 | 描述 | wparm | lparam |
|---|---|---|---|---|---|
| 1 | E_SV_EVENT_TYPE_NONE | 0x0 | 未知事件类型， | 0 | 0 |
| 2 | E_SV_EVENT_TYPE_RTMP_INVALID_URL | 0x01 | 无效的推流 url | error code | 0 |
| 3 | E_SV_EVENT_TYPE_RTMP_CONNECT_DNS_RESOVLE_FAILED | 0x02 | dns 域名解析失败 | error code | 0 |
| 4 | E_SV_EVENT_TYPE_RTMP_SOCKET_CONNECT_FAILED | 0x03 | 推流连接时 socket 创建连接失败 | error code | 0 |
| 5 | E_SV_EVENT_TYPE_RTMP_HANDSHAKE_FAILED | 0x4 | rtmp 握手失败 | error code | 0 |
| 6 | E_SV_EVENT_TYPE_RTMP_CONNECT_APP_FAILED | 0x05 | Rtmp 连接 app 失败 | error code | 0 |
| 7 | E_SV_EVENT_TYPE_RTMP_PUBLISH_STREAM_FAILED | 0x06 | rtmp 发布流失败 | error code | 0 |
| 8 | E_SV_EVENT_TYPE_RTMP_SET_TIMEOUT_FAILED | 0x07 | rtmp 设置超时失败 | error code | 0 |
| 9 | E_SV_EVENT_TYPE_RTMP_CONNECT_CLOSE | 0x08 | rtmp 推流结束 | 0 | 0 |
| 10 | E_SV_EVENT_TYPE_RTMP_CONNECT_SUCCESS | 0x09 | rtmp 推流连接成功 | 0 | 0 |
| 11 | E_SV_EVENT_TYPE_RTMP_INVALID_PARAMETER | 0x50 | 无效的参数 | error code | 0 |
| 12 | E_SV_EVENT_TYPE_RTMP_AUDIO_ADTS_DATA_ER | 0x51 | 音频 aac 数据 adts 头错误 | error code | 0 |

| | | | | | |
|---|---|---|---|---|---|
| | ROR | | | | |
| 13 | E_SV_EVENT_TYPE_RTMP_VIDEO_DATA_ERROR | 0x52 | 视频数据错误 | error code | 0 |
| 14 | E_SV_EVENT_TYPE_RTMP_AUDIO_DATA_ERROR | 0x53 | 音频数据错误 | error code | 0 |
| 15 | E_SV_EVENT_TYPE_RTMP_SEND_VIDEO_NO_SPS_PPS | 0x54 | 视频 h264/h265 数据没有 sps、pps | 0 | 0 |
| 16 | E_SV_EVENT_TYPE_RTMP_SEND_ERROR_SPS_PPS | 0x55 | 视频 h264/h265 错误的 SPS/PPS | 0 | 0 |
| 17 | E_SV_EVENT_TYPE_RTMP_SEND_ERROR_SPS | 0x56 | SPS 数据错误 | 0 | 0 |
| 18 | E_SV_EVENT_TYPE_RTMP_SEND_ERROR_PPS | 0x57 | PPS 数据错误 | 0 | 0 |
| 19 | E_SV_EVENT_TYPE_RTMP_RESUME_MEDIA_PUSH | 0x58 | Rtmp 恢复推流 | 0 | 0 |
| 20 | E_SV_EVENT_TYPE_RTMP_PAUSE_MEDIA_PUSH | 0x59 | rtmp 暂停推流（网络拥塞，缓冲已满） | 0 | 0 |
| 21 | E_SV_EVENT_TYPE_RTMP_SEND_THREAD_EXIT_WITH_ERROR | 0x60 | Rtmp socket 发送线程因错误退出 | error code | 0 |
| 22 | E_SV_EVENT_TYPE_RTMP_SEND_ERROR_VPS | 0x61 | VPS 数据错误 | error code | 0 |
| 23 | E_SV_EVENT_TYPE_PUSH_SEGMENT_END | 0x102 | 分片推流结束 | 0 | 0 |

## 2.6. 连接推流服务器

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVCloudPush_API_Connect_By_D2eviceSN |
| 2 | 接口说明 | 连接推流服务器，开始推流 |
| 3 | 参数 1 [input] | Const char* pclientid: 与设备绑定的 clientid |
| 4 | 参数 2[input] | Const char*pclient_secret:：与设备绑定的客户端密钥 |
| 5 | 参数 3[input] | Char* pdeviceSN：设备端序列号 |
| 6 | 参数 3 [input, output] | event_callback pcbFun：消息回调通知接口，详见 2.4.1 |
| 7 | 返回值(int) | 成功返回 0，失败返回非 0 值(-1) |

## 2.7. 连接推流服务器

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVCloudPush_API_Connect_By_Device_Token |
| 2 | 接口说明 | 根据设备端口令连接推流服务器 |
| 3 | 参数 1 [input] | Const char* pdevice_token: 设备口令 |
| 4 | 参数 2 [input, output] | event_callbackt pcbFun：消息回调通知接口，详见 2.4.1 |
| 5 | 返回值(int) | 成功返回 0，失败返回非 0 值(-1) |

## 2.8. 直播连接推流服务器

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVCloudPush_API_Live_Connect |
| 2 | 接口说明 | 根据设备端口令连接推流服务器 |
| 3 | 参数 1 [input] | Const char* ptoken_url: 设备口令 |
| 4 | 参数 2 [input] | Const char* pdeviceSN：设备 SN |
| 4 | 参数 3 [input, output] | event_callbackt pcbFun：消息回调通知接口，详见 2.4.1 |
| 5 | 返回值(int) | 成功返回 0，失败返回非 0 值(-1) |

## 2.9. 发送音视频数据

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVCloudPush_API_Send_Packet |
| 2 | 接口说明 | 发送音视频数据包 |
| 3 | 参数 1 [input] | long lcid： SVCloudPush_API_Connect 返回的 connectid |
| 4 | 参数 2 [input] | E_SV_MEDIA_TYPE，媒体数据类型，详见 2.5.1 |
| 5 | 参数 3 [input] | char* pData：媒体数据缓存指针 |
| 6 | 参数 4 [input] | Int nSize：媒体数据缓存大小 |
| 7 | 参数 5 [input] | Unsigned int usTimeStamp：媒体数据时间戳 |
| 4 | 返回值(int) | 成功返回 0，失败返回非 0 值(-1) |

### 2.9.1. 媒体数据编码类型枚举(E_SV_MEDIA_TYPE)类型定义

| No | 枚举定义 | 枚举值 | 描述 |
|---|---|---|---|
| 1 | E_SV_MEDIA_TYPE_UNKNOW | -1 | 未知媒体编码类型 |
| 2 | E_SV_MEDIA_TYPE_H264 | 0 | 视频-h264 编码类型（带起始码） |
| 3 | E_SV_MEDIA_TYPE_H265 | 1 | 视频-h265 编码类型（带起始码） |
| 4 | E_SV_MEDIA_TYPE_MP3 | 2 | 音频-mp3 编码类型 |
| 5 | E_SV_MEDIA_TYPE_AAC | 3 | 音频-AAC 编码类型（带 adts 头） |
| 6 | E_SV_MEDIA_TYPE_PRI_DATA | 4 | 私有媒体数据类型 |

## 2.10. 发送元数据

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVCloudPush_API_Send_Metadata |
| 2 | 接口说明 | 发送元数据 |
| 3 | 参数 1 [input] | long lcid： SVCloudPush_API_Connect 返回的 connectid |
| 4 | 参数 2 [input] | E_SV_METADATA_TYPE，元数据类型，详见 2.6.1 |
| 5 | 参数 3 [input] | wParam：元数据辅助说明参数 |
| 6 | 参数 4 [input] | lParam：元数据消息值 |
| 4 | 返回值(int) | 成功返回 0，失败返回非 0 值(-1) |

### 2.10.1. 元数据类型枚举(E_SV_METADATA_TYPE)类型定义

| No | 枚举定义 | 枚举值 | 描述 | wparam | lparam |
|---|---|---|---|---|---|
| 1 | E_SV_METADATA_TYPE_NONE | -1 | 未知元数据类型 | 0 | 0 |
| 2 | E_SV_METADATA_TYPE_STREAM_START | 0 | 流开始 | Tigger type | 流触发本地时间戳取地址(int64_t*) |
| 3 | E_SV_METADATA_TYPE_STREAM_END | 1 | 流结束 | 告警时间（毫秒/ms） | 流停止本地时间戳取地址(int64_t*) |
| 4 | E_SV_METADATA_TYPE_VIDEO_ENCRYPT | 2 | 视频加密 | 加密方式，详见 2.10.2 定义 | 加密密钥指针 |
| 5 | E_SV_METADATA_TYPE_AUDIO_ENCRYPT | 3 | 音频加密 | 加密方式，详见 2.10.2 定义 | 加密密钥指针 |
| 6 | E_SV_METADATA_TYPE_STREAM_BITRATE | 4 | 媒体码流的码率 | 音频码率 | 视频码率 |
| 7 | E_SV_METADATA_TYPE_STREAM_DROP_FRAMEss | 5 | 丢帧数 | 音频丢帧数 | 视频丢帧数 |
| 8 | E_SV_METADATA_TYPE_STREAM_DISCONNECT | 6 | app 与 camera 失去连接 | error code | 0 |
| 9 | E_SV_METADATA_TYPE_TIGGER_TYPE_CHANGE | 7 | 触发类型改变消息 | 改变之前的触发类型 | 改变之后的触发类型 |

### 2.10.2. 媒体数据加密类型枚举

| No | 枚举定义 | 枚举值 | 描述 |
|---|---|---|---|
| 1 | E_SV_MEDIA_ENCRYPT_NONE | 0 | 不加密 |
| 2 | E_SV_MEDIA | 1 | AES 加密关键帧/I 帧 |

| | | | |
|---|---|---|---|
| | _ENCRYPT_AES_KEYFRAME | | |
| 3 | E_SV_MEDIA _ENCRYPT_ALL_FRAME | 2 | AES 加密所有帧 |

推流触发类型枚举

| No | 枚举定义 | 枚举值 | 描述 |
|---|---|---|---|
| 1 | E_IPC_TIGGER_TYPE_UNKNOW | -1 | 未知的推流触发类型 |
| 2 | E_IPC_TIGGER_TYPE_HUMAN | 0 | 人脸识别触发推流 |
| 3 | E_IPC_TIGGER_TYPE_DEFAULT | 1 | 默认的触发类型（IPC HS002 默认为移动侦测触发，HS003/HS004 默认为人脸触发） |
| 4 | E_IPC_TIGGER_TYPE_FACE | 2 | 人脸识别触发 |
| 5 | E_IPC_TIGGER_TYPE_HUMAN_AND_FACE | 3 | 人形加人脸触发 |
| 6 | E_IPC_TIGGER_TYPE_MOVE_DETECT | 4 | 移动侦测触发 |
| 7 | E_IPC_TIGGER_TYPE_RESERVED | 5 | 5-99 为保留触发类型，以便后续扩展 |
| 8 | E_IPC_TIGGER_TYPE_LIVE | 100 | 调试工具触发直播 |
| 9 | E_IPC_TIGGER_TYPE_IOS_APP | 101 | IOS app 触发直播 |
| 10 | E_IPC_TIGGER_TYPE_ANDROID_APP | 102 | Android app 触发直播 |
| 11 | E_IPC_TIGGER_TYPE_ALEXA | 103 | Alexa 触发直播 |
| 12 | E_IPC_TIGGER_TYPE_WEB | 104 | Web 触发直播 |
| 13 | E_IPC_TIGGER_TYPE_GOOGLE_ASSISTANT | 105 | Google assintant 触发直播 |

## 2.11. 推流连接是否正常连接

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVCloudPush_API_Is_Connect |
| 2 | 接口说明 | 推流是否正常连接 |
| 3 | 参数 1 [input] | lsong lcid： SVCloudPush_API_Conn2ect 返回的 connectid |
| 4 | 返回值(int) | 连接正常返回 1,连接已断开返回 0 |

## 2.12. 关闭推流连接

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVCloudPush_API_Close |
| 2 | 接口说明 | 关闭推流连接 |
| 3 | 参数 1 [input] | long lcid： SVCloudPush_API_Connect 返回的 connectid |
| 4 | 返回值(int) | 连接正常返回 0，失败返回非 0 值(-1) |

## 2.13. 获取推流 SDK 版本信息

| No | 信息 | 描述 |
|---|---|---|
| 1 | 接口名称 | SVCloudPush_API_Version |
| 2 | 接口说明 | 获取推流 SDK 版本信息 |
| 3 | 参数 1 [output] | char* szVersionXml：返回当前 SDK 的版本信息 |
| 4 | 参数 2 [input] | int len：szVersionXml 的缓存长度 |
| 5 | 返回值(int) | 返回当前版本信息的长度 |

## 3. 接入演示代码

```
#include <stdio.h>
#include <vector>
#include <signal.h>
#include <assert.h>
#include <time.h>
#include <unistd.h>
#include "../common/isvpush.h"
#define MAX_CHANNEL_NUM         20
#define
TOKEN_SERVER_URL        "http://10.30.0.200:2118/connection/token/get"
#define CHECK_RESULT(ret) if(ret < 0) {assert(0); return ret;}
#define TEST_DEVICE_SN      "P0201010002012010124000001"
bool g_brunning = false;
void sig_stop(int signo)
{
    printf("sig_stop: signo = %d\n", signo);
    g_brunning = false;
    return;
}
```

```
static int event_callback(long lCID, long eventType, long wparam, long
lparam)
    {
        CAutoLock lock(m_crisec);

        std::map<long, CPushTask*>::iterator it =
m_mconnect_task_list.find(lCID);
        if(it == m_mconnect_task_list.end())
        {
            printf("Invalid connect task id:%ld eventType:%ld,
wparam:%ld, lparam:%ld\n", lCID, eventType, wparam, lparam);
            //assert(0);
            return -1;
        }

        CPushTask* ptask = it->second;
        switch(eventType)
        {
            case E_SV_EVENT_TYPE_RTMP_INVALID_URL:
            case E_SV_EVENT_TYPE_RTMP_CONNECT_DNS_RESOVLE_FAILED:
            case E_SV_EVENT_TYPE_RTMP_SOCKET_CONNECT_FAILED:
            case E_SV_EVENT_TYPE_RTMP_CONNECT_HANDSHAKE_FAILED:
            case E_SV_EVENT_TYPE_RTMP_CONNECT_APP_FAILED:
            case E_SV_EVENT_TYPE_RTMP_PUBLISH_STREAM_FAILED:
            case E_SV_EVENT_TYPE_RTMP_CONNECT_SET_TIMEOUT_FAILED:
            case E_SV_EVENT_TYPE_RTMP_INVALID_PARAMETER:
            {
                printf("lcid:%ld connect error, errorcode:%d\n", lCID,
eventType);
                ptask->m_bconnected = false;
                return 0;
            }
            case E_SV_EVENT_TYPE_RTMP_CONNECT_CLOSE:
            {
                printf("lcid:%ld connection close\n", lCID);
                ptask->m_bconnected = false;
                return 0;
            }
            case E_SV_EVENT_TYPE_RTMP_CONNECT_SUCCESS:
            {
                printf("lcid:%ld connection success\n", lCID);
                return 0;
            }
            case E_SV_EVENT_TYPE_RTMP_AUDIO_ADTS_DATA_ERROR:
```

```
            case E_SV_EVENT_TYPE_RTMP_VIDEO_START_CODE_ERROR:
            case E_SV_EVENT_TYPE_RTMP_RECV_AUDIO_DATA_ERROR:
            case E_SV_EVENT_TYPE_RTMP_SEND_VIDEO_NO_SPS_PPS:
            case E_SV_EVENT_TYPE_RTMP_SEND_ERROR_SPS_PPS:
            case E_SV_EVENT_TYPE_RTMP_SEND_ERROR_SPS:
            case E_SV_EVENT_TYPE_RTMP_SEND_ERROR_PPS:
            {
                printf("lcid:%ld connection error, errorcode:%d\n", lCID,
eventType);
                return 0;
            }
            case E_SV_EVENT_TYPE_RTMP_RESUME_MEDIA_PUSH:
            {
                printf("lcid:%ld connection resume media push\n", lCID);
                return 0;
            }
            case E_SV_EVENT_TYPE_RTMP_PAUSE_MEDIA_PUSH:
            {
                printf("lcid:%ld connection pause media push\n", lCID);
                return 0;
            }
            case E_SV_EVENT_TYPE_RTMP_SEND_THREAD_EXIT_WITH_ERROR:
            {
                printf("lcid:%ld connection error, push thread exit\n",
lCID);
                ptask->m_bconnected = false;
                return 0;
            }
            case E_SV_EVENT_TYPE_ECHOSHOW_NOTIFY_LIVE_OPEN:
            {
                g_brunning = false;
                return 0;
            }
            case E_SV_EVENT_TYPE_ECHOSHOW_NOTIFY_LIVE_CLOSE:
            {
                g_brunning = false;
                return 0;
            }
            default:
            {
                 printf("lcid:%ld connection unknown error,
errorcode:%ld\n", lCID, eventType);
                return 0;
            }
```

```
        }
    }

int main(int argc, char** args)
{
    char ver[256];
    struct timeval tv;
    int ret = 0;
    long lcid = -1;
    char* pkeystring = "vavakey";
    char* pvideodata = NULL;
    int video_data_len = 0;
    char* paudiodata = NULL;
    int audio_data_len = 0;
    int64_t pts = 0;
    int keyflag = 0;
    int encflag = 0;
    // get sdk version
    SVPush_API_Version(ver, 256);
    printf(ver);
    printf("\n sunvalley cloud storage push sdk example\n");

    //sdk initialize
    int ret = SVPush_API_Initialize(1024*1024, MAX_CHANNEL_NUM);
    CHECK_RESULT(ret);

    ret = SVPush_API_Init_log((E_SV_LOG_LEVEL)E_LOG_LEVEL_MAIN,
SV_LOG_OUTPUT_MODE_CONSOLE | SV_LOG_OUTPUT_MODE_FILE, "./log/");
    CHECK_RESULT(ret);

    ret = SVPush_API_Set_Token_Server_Interface(TOKEN_SERVER_URL);
    CHECK_RESULT(ret);

    signal(SIGINT,  sig_stop);
    signal(SIGQUIT, sig_stop);
    signal(SIGTERM, sig_stop);
    // connect to media server
    lcid = SVPush_API_Connect_By_DeviceSN(NULL, NULL, TEST_DEVICE_SN,
event_callback);
    if(lcid < 0)
    {
        printf("push sdk connect media server failed, lcid:%ld,
TEST_DEVICE_SN:%s\n", lcid, TEST_DEVICE_SN);
```

```
        assert(0);
        return -1;
    }

    gettimeofday(&tv, NULL);
    unsigned long long untsamp = ((unsigned long long)tv.tv_sec
/*+3600*16*/) * 1000 + tv.tv_usec / 1000;
    ret = SVPush_API_Send_Metadata(lcid,
E_SV_METADATA_TYPE_VIDEO_ENCRYPT, E_SV_MEDIA_ENCRYPT_AES_KEY_FRAME,
(long)(pkeystring));
    CHECK_RESULT(ret);
    ret = SVPush_API_Send_Metadata(lcid,
E_SV_METADATA_TYPE_AUDIO_ENCRYPT, E_SV_MEDIA_ENCRYPT_AES_ALL_FRAME,
(long)(pkeystring));
    CHECK_RESULT(ret);
    ret = SVPush_API_Send_Metadata(lcid,
E_SV_METADATA_TYPE_STREAM_START, E_IPC_TIGGER_TYPE_FACE,
(long)(&untsamp));
    CHECK_RESULT(ret);


    do
    {
        encflag = 0;
        // read h264 data from file(annexb format)
        // ...
        //send h264 data packet
       ret = SVPush_API_Send_Packet(lcid, E_SV_MEDIA_TYPE_H264,
pvideodata, video_data_len, pts, keyflag, encflag);
        // read aac packet data from file
        // ...
        //send aac data packet(with adts header)
        keyflag = 1;
       ret = SVPush_API_Send_Packet(lcid, E_SV_MEDIA_TYPE_AAC,
paudiodata, audio_data_len, pts, keyflag, encflag);
        usleep(30000);
    } while (g_brunning);
    printf("after while\n");
    gettimeofday(&tv, NULL);
    untsamp = ((unsigned long long)tv.tv_sec /*+3600*16*/) * 1000 +
tv.tv_usec / 1000;
    // segment end alarm time 10000ms
    ret = SVPush_API_Send_Metadata(lcid, E_SV_METADATA_TYPE_STREAM_END,
10000, (long)(&untsamp));
```

```
    CHECK_RESULT(ret);
    SVPush_API_UnInitialize();

    return 0;
```