目录 | Device SDK Manual

1. 概述

LINK-SDK 主要提供两个功能：

1. MPEG2-TS 封装和上传模块
2. MQTT 消息通道

2 MPEG2-TS 封装和上传模块

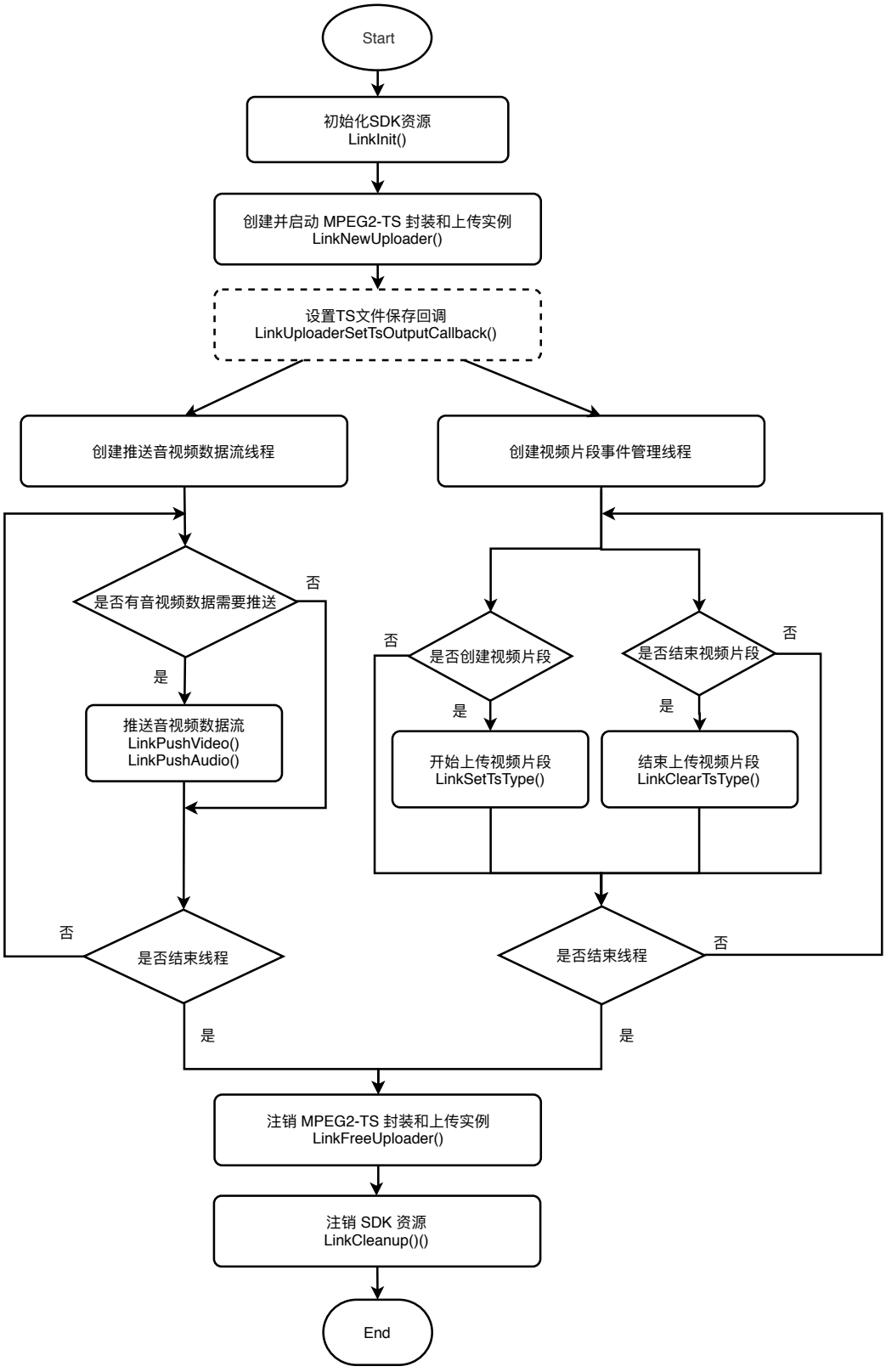
2.1 MPEG2-TS 封装和上传模块介绍

2.1.2 相关概念

- **TS文件**: (MPEG2-TS Format File) 按照 MPEG-2 Transport Stream 格式封装的文件。
- **视频片段**: (Segment) 由一个或多个TS文件组成，为一个上传周期内所有TS文件的集合，在服务端通过 HLS 的方式体现。典型应用场景: 一个移动侦测时间段内的视频。
- **视频缩略图**: (Frame Picture) 每个视频片段对应一个视频缩略图，用于预览当前视频片段。
- **DAK/DSK**: (Device Access Key / Device Secret Key) 是一对密钥对，DAK 和 DSK 的长度为 40 字节，用于设备端与服务端通信过程的鉴权。

MPEG2-TS 封装和上传模块主要应用场景为将经过 IP Camera 编码后的视频数据(H.264，H.265 格式)和音频数据(AAC，G711.A，G711.U 格式)封装为 MPEG2-TS 数据流，用于保存到本地和上传至服务端，其中保存到本地以 TS文件为基本操作单元，上传到服务端以视频片段为基本操作单元。上传至服务端的 TS数据流 通过 Linking 平台提供的各类接口进行操作。

2.1.3 使用说明



MPEG2-TS 封装和上传模块调用时序图

2.2 MPEG2-TS 封装和上传模块接口定义

2.2.1 视频上传操作返回错误码

```
/** @brief LINK 错误码 */
typedef enum _UPLOADER_ERR_CODE {
    LINK_NO_MEMORY =         -1000,  /**< 无法分配内存 */
    LINK_MUTEX_ERROR =       -1100,  /**< mutex相关系统调用返回错误 */
    LINK_COND_ERROR =        -1101,  /**< 条件变量相关系统调用返回错误 */
    LINK_THREAD_ERROR =      -1102,  /**< 线程相关系统调用返回错误 */
    LINK_TIMEOUT =           -2000,  /**< 超时异常 */
    LINK_NO_PUSH =           -2001,  /**< 状态异常 */
    LINK_BUFFER_IS_SMALL =   -2003,  /**< 设置的缓存 buffer 太小 */
    LINK_ARG_TOO_LONG =      -2004,  /**< 参数太长 */
    LINK_ARG_ERROR =         -2100,  /**< 传入的参数错误(null 或值不正确) */
    LINK_JSON_FORMAT =       -2200,  /**< 服务器返回数据包错误 */
    LINK_HTTP_TIME =         -2300,  /**< 从服务器获取标准时间出错 */
    LINK_Q_WRONGSTATE =      -5002,  /**< 缓存 buffer 错误状态 */
    LINK_NOT_INITED =        -5104,  /**< 没有初始化错误 */
    LINK_PAUSED =            -6000,  /**< 暂停标志 */
    LINK_GHTTP_FAIL =        -7000,  /**< GHTTP 错误 */
    LINK_SUCCESS =           0       /**< 操作成功 */
} UPLOADER_ERR_CODE;
```

2.2.2 主要数据结构

视频格式类型

```
/** @brief 视频格式 */
typedef enum {
        LINK_VIDEO_NONE = 0,    /**< 无视频 */
        LINK_VIDEO_H264 = 1,    /**< H264格式 */
        LINK_VIDEO_H265 = 2     /**< H265格式 */
}LinkVideoFormat;
```

音频格式类型

```
/** @brief 音频格式 */
typedef enum {
        LINK_AUDIO_NONE = 0,    /**< 无音频 */
        LINK_AUDIO_PCMU = 1,    /**< G711U格式 */
        LINK_AUDIO_PCMA = 2,    /**< G711A格式 */
        LINK_AUDIO_AAC =  3     /**< AAC格式 */
}LinkAudioFormat;
```

视频片段参数

```
typedef struct _LinkSessionMeta{
        const char **keys;
        int *keylens;
        const char **values;
        int *valuelens;
        int len;
        int isOneShot;
}LinkSessionMeta;
```

上传参数说明，在生成新上传实例时定义

```
/** @brief 上传参数 */
typedef struct _LinkUploadArg {
        LinkAudioFormat nAudioFormat;           /**< 音频格式 */
        size_t nChannels;                       /**< 音频通道数 */
        size_t nSampleRate;                     /**< 音频采样率 */
        LinkVideoFormat nVideoFormat;           /**< 视频格式 */

        const char * pConfigRequestUrl;         /**< 获取业务配置的请求地址 */
        size_t nConfigRequestUrlLen;            /**< 业务配置的请求地址长度 */
        const char *pDeviceAk;                  /**< 设备 APP KEY */
        size_t nDeviceAkLen;                    /**< 设备 APP KEY 长度 */
        const char *pDeviceSk;                  /**< 设备 SECRET KEY */
        size_t nDeviceSkLen;                    /**< 设备 SECRET KEY 长度 */
        void(*getPictureCallback)(void *pUserData, const char *pFilename, int
nFilenameLen);
        void *pGetPictureCallbackUserData;      /**< 图片上传回调函数 */
        UploadStatisticCallback *pUpStatCb;     /*上传结果回调*/
        void *pUpStatCbUserArg;                 /*作为上传结果回调的第一个参数*/

        void * reserved1;                       /**< 预留1 */
        void * reserved2;                       /**< 预留2 */
}LinkUploadArg;
```

2.2.3 视频上传接口说明

2.2.3.1 初始化 SDK 资源

```
/**
 * @brief 初始化上传SDK，此函数必须在任何其他子功能之前调用。
 *
 * @note 此函数不是线程安全函数。
 *
 * @return LINK_SUCCESS 成功；其它值 失败
 */
int LinkInit();
```

2.2.3.2 创建并启动 MPEG2-TS 封装和上传实例

```
/**
 * @brief 创建并启动 MPEG2-TS 封装和上传实例
 *
 * @note 此函数不是线程安全函数。
 *
 * @param[out] pTsMuxUploader 切片上传实例
 * @param[in]  pUserUploadArg 上传需要的参数，用来设置 token,deviceid
 * @return LINK_SUCCESS 成功；其它值 失败
 */
int LinkNewUploader(void ** pUploader,
                    const LinkUploadArg *pUploadArg
                    );
```

2.2.3.3 设置TS文件保存回调

```
/**
 * 设置ts切片数据回调
 *
 * 此函数不是线程安全函数。
 *
 * @param[out] pTsMuxUploader 切片上传实例
 * @param[in]  pTsDataCb 回调函数
 * @param[in]  pUserArg 作为pTsDataCb函数的userCtx参数，返回给用户
 */
void LinkUploaderSetTsOutputCallback(LinkTsMuxUploader *pTsMuxUploader,
                                     LinkTsOutput pTsDataCb,
                                     void * pUserArg
                                     );
```

2.2.3.4 开始生成视频片段

```
/**
 * 设置片段上报的元数据
 *
 * @brief 设置片段上报的元数据,通常使用场景为摄像头检查到移动侦测后调用该接口
 *
 * @param[in] pTsMuxUploader 切片上传实例
 * @param[in] metas 自定义的元数据，key->value结构
 *                  metas->isOneShot 非0，仅上报一次后便不在上报
 * @return LINK_SUCCESS 成功; 其它值 失败
 */
int LinkSetTsType(LinkTsMuxUploader *pTsMuxUploader,
                  LinkSessionMeta *metas
                  );
```

2.2.3.5 结束当前视频片段

```
/**
 * 清空段上报的元数据
 *
 * @brief 清空段上报的元数据，通常使用场景为摄像头检查到移动侦测消失后调用该接口
 *
 * @param[in] pTsMuxUploader 切片上传实例
 */
void LinkClearTsType(IN LinkTsMuxUploader *pTsMuxUploader);
```

2.2.3.6 刷新当前视频片段

```
/**
 * 刷新缓存数据
 *
 * @brief 通知当前没有可上传数据,通常使用场景为摄像头检查到移动侦测后消失调用该接口，以通知上传缓冲的数据
 *
 * 此函数用于当上传结束时，将当前已缓存的资源完成进行上传
 * 例如当移动侦测结束时，暂时不再上传资源，调用函数后会将已缓存的资源完成切片上传
 *
 * @param[in] pTsMuxUploader 切片上传实例
 * @return NULL
 */
void LinkFlushUploader(IN LinkTsMuxUploader *pTsMuxUploader);
```

2.2.3.7 推送视频流数据

```
/**
 * 推送视频流数据
 *
 * @param[in] pTsMuxUploader 切片上传实例
```

```
 * @param[in] pData 视频数据
 * @param[in] nDataLen 视频数据大小
 * @param[in] nTimestamp 视频时间戳，如果存在音频，和音频时间戳一定要对应同一个基点
 * @param[in] nIsKeyFrame 是否是关键帧
 * @param[in] nIsSegStart 是否是新的片段开始
 * @param[in] nFrameSysTime 帧对应的系统时间,单位为m毫秒。通常的使用场景是：开启运动侦测时候，送入预录数据关键帧时候填写该预录视频关键帧对应的系统时间,其它情况可以填0
 *                          就是说，如果这个值大于1548064836000，则使用传入的时间，否则取系统时间
 * @return LINK_SUCCESS 成功; 其它值 失败
 */
int LinkPushVideo(LinkTsMuxUploader *pTsMuxUploader,
                  char * pData,
                  int nDataLen,
                  int64_t nTimestamp,
                  int nIsKeyFrame,
                  int nIsSegStart,
                  int64_t nFrameSysTime
                  );
```

2.2.3.8 推送音频流数据

```
/**
 * 推送音频流数据
 *
 * @param[in] pTsMuxUploader 切片上传实例
 * @param[in] pData 音频数据
 * @param[in] nDataLen 音频数据大小
 * @param[in] nTimestamp 音频时间戳，必须和视频时间戳对应同一个基点
 * @param[in] nFrameSysTime 帧对应的系统时间,单位为m毫秒。目前值填固定的0
 * @return LINK_SUCCESS 成功; 其它值 失败
 */
int LinkPushAudio(LinkTsMuxUploader *pTsMuxUploader,
                  char * pData,
                  int nDataLen,
                  int64_t nTimestamp,
                  int64_t nFrameSysTime
                  );
```

2.2.3.9 注销 MPEG2-TS 封装和上传实例

```
/**
 * 销毁切片上传实例
 *
 * 如果正在上传会停止上传
 *
 * @param[in,out] pTsMuxUploader 切片上传实例
 * @return NULL
 */
void LinkFreeUploader(LinkTsMuxUploader **pTsMuxUploader);
```

2.2.3.10 注销 SDK 资源

```
/**
 * 销毁释放 sdk 资源
 *
 * 此函数不是线程安全函数。
 *
 * @return NULL
 */
void LinkCleanup();
```

2.2.3.11 验证七牛凭证合法性

```
/**
 * 验证七牛凭证合法性
 *
 * @param[in] pAk 设备端的 accessKey
 * @param[in] nAkLen accessKey 长度，最大长度 512 字节
 * @param[in] pSk 设备端的 secretKey
 * @param[in] nSkLen secretKey 长度，最大长度 512 字节
 * @param[in] pToken 访问凭证， 格式为 "ak + ':' + encodedSign + ':' +
encodedPutPolicy"
 * @param[in] nTokenLen Token 长度，最大长度 4096 字节
 * @return LINK_TRUE: 验证成功; LINK_FALSE: 验证失败; LINK_ERROR: 参数错误
 */
int LinkVerify(const char *pAk,
               size_t nAkLen,
               const char *pSk,
               size_t nSkLen,
               const char* pToken,
               size_t nTokenLen
               );
```

3 MQTT 消息模块使用说明

3.1 MQTT 返回错误码

```
 /** @brief MQTT 返回错误码 */
typedef enum _MQTT_ERR_CODE {
    MQTT_SUCCESS =               -3000,  /**< 表示成功调用API接口 */
    MQTT_CONNECT_SUCCESS =       -3001,  /**< 表示连接成功，回调接口返回 */
    MQTT_DISCONNECT_SUCCESS =    -3002,  /**< 表示断开连接成功，回调接口返回 */
    MQTT_ERR_NOMEM =             -3003,  /**< 没有内存 */
    MQTT_ERR_PROTOCOL =          -3004,  /**< 协议错误 */
    MQTT_ERR_INVAL =             -3005,  /**< 无效参数返回的错误 */
    MQTT_ERR_NO_CONN =           -3006,  /**< 没有链接响应 */
    MQTT_ERR_CONN_REFUSED =      -3007,  /**< 链接被拒绝 */
    MQTT_ERR_NOT_FOUND =         -3008,  /**< 没有发现服务器 */
    MQTT_ERR_CONN_LOST =         -3009,  /**< 链接丢失 */
    MQTT_ERR_TLS =               -3010,  /**< TLS 错误 */
    MQTT_ERR_PAYLOAD_SIZE =      -3011,  /**< 数据包大小不正确 */
    MQTT_ERR_NOT_SUPPORTED =     -3012,  /**< 不支持该调用 */
    MQTT_ERR_AUTH =              -3013,  /**< 认证不正确 */
    MQTT_ERR_ACL_DENIED =        -3014,  /**< 没有 ACL 权限 */
    MQTT_ERR_UNKNOWN =           -3015,  /**< 未知错误 */
    MQTT_ERR_PROXY =             -3016,  /**< 代理不正确 */
    MQTT_ERR_OTHERS =            -3017,  /**< 其他错误 */
    MQTT_SUCCESS = 0                     /**< 操作成功 */
} MQTT_ERR_CODE;
```

3.2 MQTT 主要数据结构

MQTT 授权模式

```
/** @brief 授权模式 */
typedef enum _MQTT_AUTH_TYPE{
    MQTT_AUTH_NULL = 0x0,       /**< 无用户名和密码 */
    MQTT_AUTH_USER = 0x1,       /**< 需要设置用户名和密码 */
    MQTT_AUTH_ONEWAY_SSL = 0x2, /**< 需要单向认证 */
    MQTT_AUTH_TWOWAY_SSL = 0x4  /**< 需要双向认证 */
} MQTT_AUTH_TYPE;
```

MQTT 服务等级

```
/** @brief MQTT QOS 等级 */
typedef enum _MQTT_LEVEL {
    MQTT_LEVEL_0 = 0,   /**< MQTT QOS LEVEL 0 */
    MQTT_LEVEL_1 = 1,   /**< MQTT QOS LEVEL 1 */
    MQTT_LEVEL_2 = 2    /**< MQTT QOS LEVEL 2 */
} MQTT_LEVEL;
```

MQTT 用户信息结构体

```
/** @brief MQTT 用户信息结构体 */
typedef struct _MqttUserInfo {
    MQTT_AUTH_TYPE nAuthenicatinMode; /**< 授权模式 */
    char *pUsername;                  /**< 用户名 */
    char *pPassword;                  /**< 密码 */
    char *pHostname;                  /**< MQTT服务器地址 */
    uint16_t nPort;                   /**< MQTT服务器端口 */
    char *pCafile;                    /**< 服务器CA证书文件路径 */
    char *pCertfile;                  /**< 客户端授权证书文件路径，双向认证服务器端需要 */
    char *pKeyfile;                   /**< 客户端授权密钥，双向认证服务器端需要 */
    char *reserved1;                  /**< 预留 */
} MqttUserInfo;
```

MQTT 参数结构体

```
/** @brief MQTT 参数结构体 */
typedef struct _MqttOptions {
    char *pid;                          /**< 客户端id */
    bool bCleanSession;                 /**< 是否清除会话 */
    MqttUserInfo primaryUserInfo;       /**< 首用户信息 */
    int32_t nKeepalive;                 /**< 心跳 */
    MqttCallback callbacks;             /**< 用户回调函数 */
    MQTT_LEVEL nQos;                    /**< 服务质量 */
    bool bRetain;                       /**< 是否保留 */
} MqttOptions;
```

消息回调函数结构体

```
/** @brief 消息回调函数结构体 */
typedef struct _MqttCallback {
    /** 消息回调函数，用来接收订阅消息 */
    void (*OnMessage)(const void* _pInstance, const char* _pTopic, const char*
_pMessage, const size_t _nLength);
    /** 事件回调函数，用来接收关键消息和错误消息 */
    void (*OnEvent)(const void* _pInstance, const int _nCode, const char*
_pReason);
} MqttCallback;
```

3.3 MQTT 接口说明

3.3.1 初始化 MQTT SDK

```
/**
 * @brief 初始化 MQTT SDK
 *
 * @return MQTT_ERR_CODE
 */
int LinkMqttLibInit();
```

3.3.2 注销 MQTT SDK

```
/**
 * @brief 注销 MQTT SDK
 *
 * @return MQTT_ERR_CODE
 */
int LinkMqttLibCleanup();
```

3.3.3 创建一个 MQTT 实例

```
/**
 * @brief 创建一个 MQTT 实例
 *
 * @param[out] pInstance 创建成功的 MQTT 实例指针
 * @param[in]  pMqttpOption 创建的 MQTT 参数
 * @return MQTT_ERR_CODE
 */
int LinkMqttCreateInstance(void ** pInstance,
                           const MqttOptions* pMqttpOption
                           );
```

3.3.4 销毁一个 MQTT 实例

```
/**
 * @brief 销毁一个 MQTT 实例
 *
 * @param[in] pInstance 需要销毁的MQTT实例
 * @return MQTT_ERR_CODE
 */
int LinkMqttDestroyInstance(const void* pInstance);
```

3.3.5 上报 MQTT 消息

```
/**
 * @brief 上报 MQTT 消息
 *
 * @param[in] pInstance MQTT实 例
```

```
 * @param[in] pTopic 发布主题
 * @param[in] nPayloadlen 发布消息长度
 * @param[in] pPayload 发布消息负载
 * @return MQTT_ERR_CODE
 */
int LinkMqttPublish(const void* pInstance,
                    char* pTopic,
                    size_t nPayloadlen,
                    const void* pPayload
                    );
```

3.3.6 订阅 MQTT 消息

```
/**
 * @brief 订阅 MQTT 消息
 *
 * @param[in] pInstance MQTT 实例
 * @param[in] pTopic 订阅主题
 * @return
 */
int LinkMqttSubscribe(const void* pInstance,
                      char* pTopic
                      );
```

3.3.7 取消订阅 MQTT 消息

```
/**
 * 取消订阅 MQTT 消息
 *
 * @param[in] pInstance MQTT 实例
 * @param[in] pTopic 取消订阅主题
 * @return
 */
int LinkMqttUnsubscribe(const void* pInstance,
                        char* pTopic
                        );
```