全志科技
Allwinner Technology

Tina

TRecorder 接口说明文档 v3.0

文档履历

版本号	日期	制/修订人	制/修订记录
V3.0			

目　录

1. 概述

1.1. 编写目的

此文档说明在 tina3.0 平台，如何使用 TRecorder 的接口来开发录像应用程序，方便录像开发人员快速正确地开发。

1.2. 适用范围

本文档目前只适用于 tina3.0 平台的的 Recorder 开发。

1.3. 相关人员

Tina3.0 平台，Recorder 开发人员。

2. TRecorder 状态图及状态说明

2.1. TRecorder 状态图

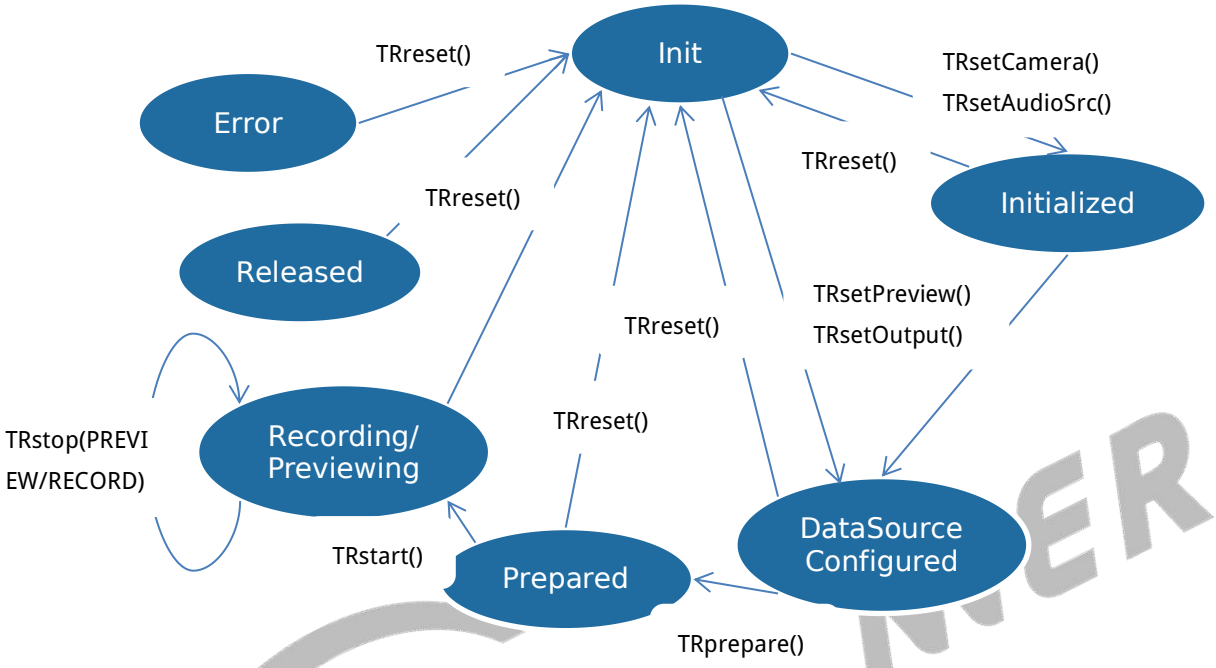


图 2-1 TRecorder 录制状态图

这张状态转换图清晰地描述了 TRecorder 的各个状态，可以通过 TRsetCamera()、TRsetAudioSrc()、TRsetPreview()和 TRsetOutput()函数选择相应的模块，而后通过 TRprepare()、TRstart()函数启动相应模块线程等。

2.2.TRecorder 每个状态简要说明

2.2.1. Init 状态

Init 状态：当调用 CreateTRecorder()创建一个 TRecorder 或者调用了其 TRreset()方法时，TRecorder 处于 reset 状态。

2.2.2. Initialized 状态

这个状态比较简单，调用 TRsetCamera()并且调用了 TRsetAudioSrc 方法就进入 Initialized 状态，表示此时要录制的数据已经设置好了。

2.2.3. DataSourceConfigured 状态

这个状态在调用了 TRsetPreview 和 TRsetOutput 后会进入，主要用来设置录制的显示输出和录制输出。进入此状态后需要配置之前生效的录制数据源和录制显示输出或者录制文件输出的参数。在选择模块的时候，将会通过读取配置文件的形式完成了模块的参数配置，该状态下的参数设置实际为覆盖配置文件的参数配置，此时可以通过调用 TRprepare 进入下一个状态。

2.2.4. Prepared 状态

初始化在 DataSourceConfigured 状态下调用 TRprepare 即可进入该状态。此状态用来确定所有所需的输入输出节点和参数已经全部配置完毕。

2.2.5. Recording/Previewing 状态

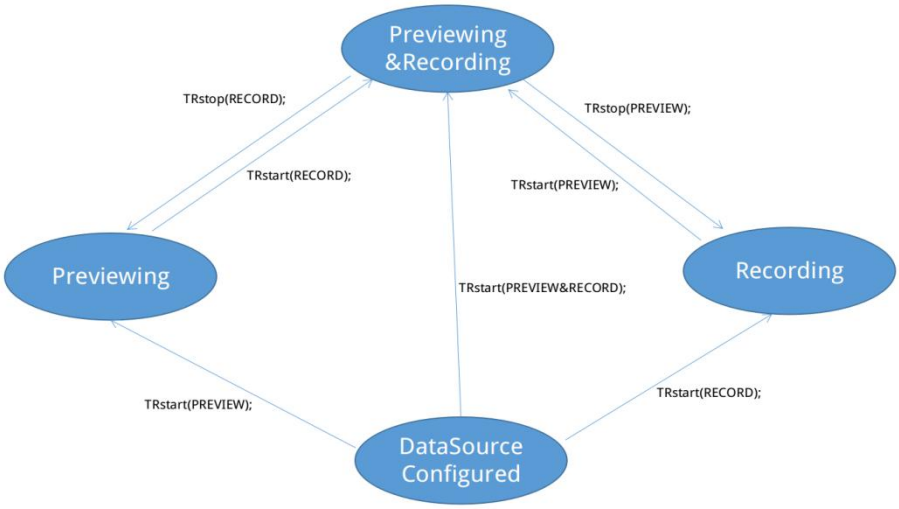
一旦 TRecorder 进入 DataSourceConfigured 状态并且数据源和输出参数已经配置完毕，就可以通过 TRStart 进入此状态。此状态表明在录制或者在预览，在此状态下有三个小状态，分别为：

Recording:只录制状态，用于后台录制等情况

Previewing:只预览状态，此状态用于拍照或者录制时未插入 SD 卡等的预览状态

Previewing&Recording:同时录制和预览状态

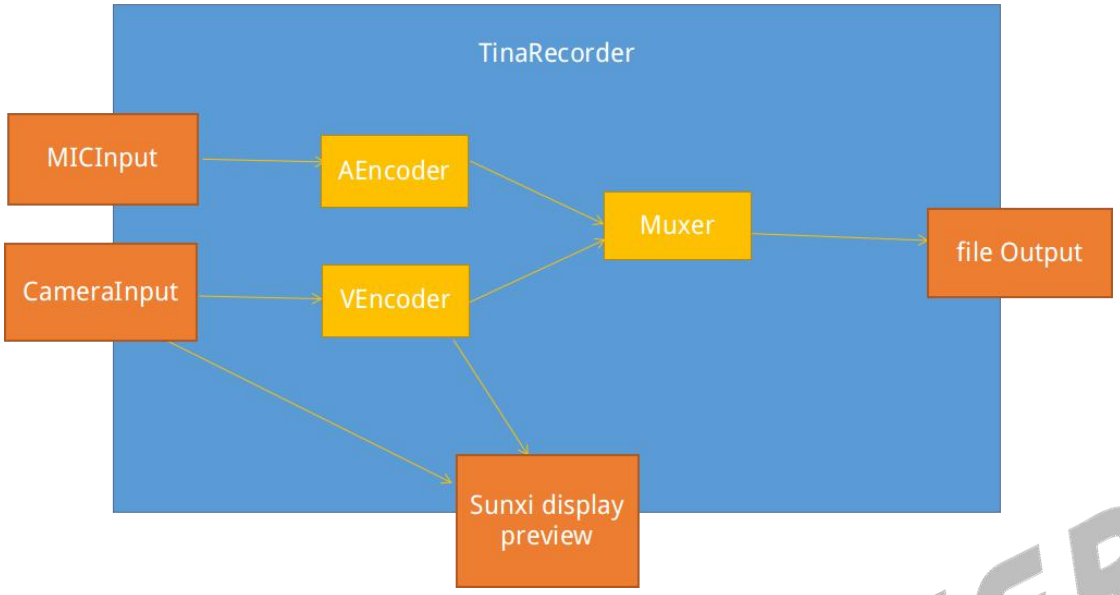
三种状态切换方法如下。



2.2.6. Released 状态

Init 状态下调用 TRrelease 进入 Released 状态，此状态下所有和此 TRRecorder 相关资源都会被释放，如需再次使用，需重新创建 TRRecorder 并重新配置参数。

2.3. TinaRecorder 结构图



如上图所示，trecorder 内部模块的连接如上，当使用外部模块与 trecorder 结合开发时，需要用户有效设置数据通路，外部模块与内部模块数据交互接口的主要函数在 dataqueue.c，该文件实现 module link、data push 等操作。

3. 接口函数说明

根据功能不一样 ,将 TinaRecorder 的接口函数分为九类 :TinaRecorder 端口创建类、TinaRecorder 状态操作类、Camera 操作类、Mic 操作类、显示操作类、编码参数设置类、封装设置类、外部 module 操作类和 Module 数据操作类。前两类主要完成 TinaRecorder 整体框架的搭建，而 Camera 操作类、Mic 操作类、显示操作类和编码参数设置类则主要完成相应模块的参数重覆盖操作，封装设置类实现录制保存文件时的录制时间、保存路径等设置，最后的外部 module 操作类和 Module 数据操作类则是当存在外部 module 与 TinaRecorder 内部模块连接使用时需要操作的 API。

3.1. TinaRecorder 端口创建类

该类接口主要实现 TinaRecorder 的 handle 以及模块 handle 选择、创建等。

3.1.1. CreateTRecorder

函数原型	TrecorderHandle *CreateTRecorder();
功能	创建一个 TRecorder
参数	无
返回值	成功返回 TRecorder 的句柄指针，失败返回 NULL
调用说明	创建一个 TRRecorder，创建成功后 TRRecorder 处于 Init 状态

3.1.2. TRsetCamera

函数原型	int TRsetCamera(TrecorderHandle *hdl,int index);
功能	设置录制的输入源，可设置为前置摄像头或者后置摄像头
参数	Hdl：TRRecorder 的句柄指针 index: typedef enum { T_CAMERA_FRONT, T_CAMERA_BACK, T_CAMERA_DISABLE, T_CAMERA_END, }TcameraIndex; 代表前置，后置，关闭
返回值	成功返回 0，失败返回-1
调用说明	当设为 T_CAMERA_DISABLE 情况下，预览和视频录制都无法打开，仅用于录制音频的情况

3.1.3. TRsetAudioSrc

函数原型	int TRsetAudioSrc(TrecorderHandle *hdl,int index);
功能	设置录制的音频来源

参数	Hdl：TRRecorder 的句柄指针 Index： typedef enum { T_AUDIO_MIC0, T_AUDIO_MIC1, T_AUDIO_MIC_DISABLE, T_AUDIO_MIC_END, }TmicIndex; 分别代表麦克风 0,麦克风 1 和关闭音频输入
返回值	成功返回 0，失败返回-1
调用说明	MIC1 仅在板子上有多个麦克风时需要。可以同时设为 MIC0 或者 MIC1,当同时设置为 MIC0 或者 MIC1 时，PCM 数据将通过拷贝模式分成多个音频通路并输入到 TRRecorder 中

3.1.4. TRsetPreview

函数原型	int TRsetPreview(TrecorderHandle *hdl,int layer);
功能	设置预览画面的图层
参数	Hdl：TRRecorder 的句柄指针 layer: typedef enum { T_DISP_LAYER0, T_DISP_LAYER1, T_DISP_LAYER_DISABLE, T_DISP_LAYER_END, }TdispIndex; 分别代表显示的两个图层
返回值	成功返回 0，失败返回-1
调用说明	当需要双路预览时，不同的预览画面需要显示在不同的图层上。默认 LAYER1 会覆盖住 LAYER0。如需切换，需要通过切换 Zorder 实现。

3.1.5. TRsetOutput

函数原型	int TRsetOutput(TrecorderHandle *hdl,char *url);
功能	设置此 TRRecorder 录制文件的存放位置
参数	Hdl:TRRecorder 的句柄指针 url:录制文件的存放路径
返回值	成功返回 0,失败返回-1
调用说明	此函数仅用于录制一次性文件的初始路径设置。如果行车记录仪方案需要不停切换文件名，则需通过 TRsetRecorderCallback 接口设置 TRRecorder 的回调，并处理 T_RECORD_ONE_FILE_COMPLETE 消息来切换每次录制完毕后的文件。

3.2. TinaRecorder 状态操作类

该类接口实现 TinaRecorder 状态转换、模块硬件初始化，开启、停止模块线程等。

3.2.1. TRstart

函数原型	int TRstart(TrecorderHandle *hdl,int flags);
功能	开始预览/录制
参数	Hdl：TRRecorder 的句柄指针 flags： typedef enum { T_PREVIEW, T_RECORD, T_ALL, }TFlags; 分别代表打开预览，录制，同时打开
返回值	成功返回 0，失败返回-1
调用说明	可以分别打开预览或者录制，具体状态转换见 2.2.5 状态转换图

3.2.2. TRstop

函数原型	int TRstop(TrecorderHandle *hdl,int flags);
功能	关闭预览/录制
参数	Hdl：TRRecorder 的句柄指针 flags： typedef enum { T_PREVIEW, T_RECORD, T_ALL, }TFlags; 分别代表关闭预览，录制，同时关闭
返回值	成功返回 0，失败返回-1
调用说明	可以分别关闭预览或者录制，具体状态转换见 2.2.5 状态转换图

3.2.3. TRrelease

函数原型	int TRrelease(TrecorderHandle *hdl);
功能	释放 TRRecorder
参数	Hdl：TRRecorder 的句柄指针
返回值	成功返回 0，失败返回-1
调用说明	无

3.2.4. TRprepare

函数原型	int TRprepare(TrecorderHandle *hdl);

功能	准备 TRRecorder
参数	Hdl：TRRecorder 的句柄指针
返回值	成功返回 0，失败返回-1
调用说明	无

3.2.5. TRreset

函数原型	int TRreset(TrecorderHandle *hdl);
功能	重置 TRRecorder
参数	Hdl：TRRecorder 的句柄指针
返回值	成功返回 0，失败返回-1
调用说明	除了 Released 状态，在任何状态下都可以调用该函数

3.3. Camera 操作类

该类接口实现 camera 的参数设置。

3.3.1. TRsetCameraInputFormat

函数原型	int TRsetCameraInputFormat(TrecorderHandle *hdl,int format);
功能	设置摄像头的输入格式
参数	Hdl：TRRecorder 的句柄指针 Format： typedef enum { T_CAMERA_YUV420SP,//NV12 T_CAMERA_YVU420SP,//NV21 T_CAMERA_YUV420P,//YU12 T_CAMERA_YVU420P,//YV12 T_CAMERA_YUV422SP, T_CAMERA_YVU422SP, T_CAMERA_YUV422P, T_CAMERA_YVU422P, T_CAMERA_YUYV422, T_CAMERA_UYVY422, T_CAMERA_YVYU422, T_CAMERA_VYUY422, T_CAMERA_ARGB, T_CAMERA_RGBA, T_CAMERA_ABGR, T_CAMERA_BGRA, T_CAMERA_TILE_32X32,//MB32_420 T_CAMERA_TILE_128X32, }TcameraFormat;
返回值	成功返回 0，失败返回-1
调用说明	目前仅支持输入 YUV 格式的摄像头，在 DataSourceConfigured 状态调用

3.3.2. TRCaptureCurrent

函数原型	int TRCaptureCurrent(TrecorderHandle *hdl,TCaptureConfig *config);
功能	对当前的预览画面截图

参数	Hdl：TRRecorder 的句柄指针
	typedef struct
	{
	char capturePath[128];//截图存放位置
	TcaptureFormat captureFormat ;//截图格式
	int captureWidth;//截图宽
	int captureHeight;//截图高
	}TCaptureConfig;

返回值	成功返回 0，失败返回-1
调用说明	在 Previewing/Recording 状态并且小状态处于预览打开的情况下才可调用。

3.3.3. TRsetCameraFramerate

函数原型	int TRsetCameraFramerate(TrecorderHandle *hdl,int framerate);
功能	设置摄像头的帧率
参数	Hdl：TRRecorder 的句柄指针 Framerate：摄像头帧率，此帧率需与实际摄像头帧率保持一致
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态调用

3.3.4. TRsetCameraCaptureSize

函数原型	int TRsetCameraCaptureSize(TrecorderHandle *hdl,int width,int height);
功能	设置摄像头的分辨率
参数	Hdl：TRRecorder 的句柄指针 Width：摄像头的宽 height：摄像头的高
返回值	成功返回 0，失败返回-1
调用说明	设置的分辨率必须被摄像头支持，在 DataSourceConfigured 状态调用

3.3.5. TRsetCameraDiscardRatio

函数原型	int TRsetCameraDiscardRatio(TrecorderHandle *hdl,int ratio);
功能	设置摄像头的丢帧比例
参数	Hdl：TRRecorder 的句柄指针 Ratio：摄像头间隔丢帧的帧数
返回值	成功返回 0，失败返回-1
调用说明	此接口暂未实现

3.3.6. TRsetCameraWaterMarkIndex

函数原型	int TRsetCameraWaterMarkIndex(TrecorderHandle *hdl,int id);
功能	设置水印 ID 号
参数	Hdl：TRRecorder 的句柄指针 Id：水印资源的 ID 号
返回值	成功返回 0，失败返回-1
调用说明	此接口暂未实现

3.3.7. TRsetCameraEnable

函数原型	int TRsetCameraEnable(TrecorderHandle *hdl,int enable);
功能	确认 camera 相关参数已配置完毕
参数	Hdl：TRRecorder 的句柄指针 enable：0 代表 disable，1 代表 enable
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态下调用，确认 camera 参数已设置完毕。真正的设置完毕依然需要应用来确保。此处不做任何检查。

3.4. Mic 操作类

该类函数主要完成麦克风配置。

3.4.1. TRsetMICInputFormat

函数原型	int TRsetMICInputFormat(TrecorderHandle *hdl,int format);
功能	设置麦克风输入的格式
参数	Hdl：TRRecorder 的句柄指针 Format： typedef enum { T_MIC_PCM, T_MIC_END, }TmicFormat;
返回值	成功返回 0，失败返回-1
调用说明	目前仅支持 PCM 输入的麦克风，在 DataSourceConfigured 状态下调用

3.4.2. TRsetMICSampleRate

函数原型	int TRsetMICSampleRate(TrecorderHandle *hdl,int sampleRate);
功能	设置麦克风输入的采样率
参数	Hdl：TRRecorder 的句柄指针 Samplerate：

	麦克风输入的采样率
返回值	成功返回 0，失败返回-1
调用说明	采样率需根据声卡支持的格式设置，在 DataSourceConfigured 状态下调用

3.4.3. TRsetMICChannels

函数原型	int TRsetMICChannels(TrecorderHandle *hdl,int channels);
功能	设置麦克风输入的声道数
参数	Hdl：TRRecorder 的句柄指针 channels： 麦克风输入的声道数
返回值	成功返回 0，失败返回-1
调用说明	声道数需根据声卡支持的格式设置，在 DataSourceConfigured 状态下调用

3.4.4. TRsetMICBitrate

函数原型	int TRsetMICBitrate(TrecorderHandle *hdl,int bitrate);
功能	设置麦克风输入的比特率
参数	Hdl：TRRecorder 的句柄指针 bitrate: 麦克风输入的比特率
返回值	成功返回 0，失败返回-1
调用说明	比特率需根据声卡支持的格式设置，在 DataSourceConfigured 状态下调用

3.4.5. TRsetAudioMute

函数原型	int TRsetAudioMute(TrecorderHandle *hdl,int muteFlag);
功能	在录制时对音频进行静音
参数	Hdl：TRRecorder 的句柄指针 muteFlag：1 表示静音，0 表示非静音
返回值	成功返回 0，失败返回-1
调用说明	在 previewing/recording 状态下调用。

3.4.6. TRsetMICEnable

函数原型	int TRsetMICEnable(TrecorderHandle *hdl,int enable);
功能	确认麦克风相关参数已设置完毕
参数	Hdl：TRRecorder 的句柄指针 Enable：0 代表 disable，1 代表 enable
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态下调用，麦克风参数的设置完毕需要应用确认，此处不做检查。若 enable 为 0,表示此 TRRecorder 不需要设置麦克风参数。

3.5. 显示操作类

该类函数主要完成显示设置。

3.5.1. TRsetPreviewSrcRect

函数原型	int TRsetPreviewSrcRect(TrecorderHandle *hdl,TdispRect *SrcRect);
功能	设置预览时源数据的裁剪矩形

参数	Hdl：TRRecorder 的句柄指针
	SrcRect：
	typedef struct
	{
	int x; //源数据左上角 x 坐标
	int y;//源数据左上角 y 坐标
	int width;//矩形的宽度
	int height;//矩形的高度
	}TdispRect;

返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 和 previewing/recording 状态下都可调用。源数据根据 preview route 的不同大小可能不同。此矩形不能超过源数据的边界。

3.5.2. TRsetPreviewRect

函数原型	int TRsetPreviewRect(TrecorderHandle *hdl,TdispRect *rect);
功能	设置预览时源数据裁剪后的显示位置

参数	Hdl：TRRecorder 的句柄指针
	rect：
	typedef struct
	{
	int x; //源数据左上角 x 坐标
	int y;//源数据左上角 y 坐标
	int width;//矩形的宽度
	int height;//矩形的高度
	}TdispRect;

返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 和 previewing/recording 状态下都可调用。此矩形不能超过屏幕大小的边界

3.5.3. TRsetPreviewRotate

函数原型	int TRsetPreviewRotate(TrecorderHandle *hdl,int angle);
功能	设置预览时的旋转角度

参数	Hdl：TRRecorder 的句柄指针 angel： typedef enum { T_ROTATION_ANGLE_0 = 0, T_ROTATION_ANGLE_90 = 1, T_ROTATION_ANGLE_180 = 2, T_ROTATION_ANGLE_270 = 3 }TrotateDegree;
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态下调用。在不支持硬件旋转的机器中，设置后不能改变

3.5.4. TRsetPreviewRoute

函数原型	int TRsetPreviewRoute(TrecorderHandle *hdl,int route);
功能	设置预览源数据的路径
参数	Hdl：TRRecorder 的句柄指针 typedef enum { T_ROUTE_ISP, //从 camera ISP 缩放而来 T_ROUTE_VE,//从 VE 缩放而来 T_ROUTE_CAMERA,//直接从 CAMERA 而来 }TRoute;
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态下调用。当数据从 VE 而来时，源数据的大小为 camera 源数据大小根据 scaledown ratio 等比例缩放而来。当数据从 camera 而来时，源数据大小和 camera 采集的大小一致。

3.5.5. TRsetPreviewZorder

函数原型	int TRsetPreviewZorder(TrecorderHandle *hdl,int zorder);
功能	设置此 TRRecorder 关联的 preview 显示图层的层序
参数	Hdl：TRRecorder 的句柄指针 zorder： typedef enum { T_PREVIEW_ZORDER_TOP, T_PREVIEW_ZORDER_MIDDLE, T_PREVIEW_ZORDER_BOTTOM, }TZorder;
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 和 previewing/recording 状态下都可调用。遮盖的顺序为 TOP 最上，MIDDLE 居中，BOTTOM 最下。上部的图层会遮盖住下部的图层。

3.5.6. TRsetPreviewEnable

函数原型	int TRsetPreviewEnable(TrecorderHandle *hdl,int enable);
功能	确认预览相关参数设置完毕
参数	Hdl：TRRecorder 的句柄指针 Enable：0 为 disable，1 为 enable
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态下调用。

3.6. 编码参数设置类

该类函数主要完成编码器参数设置。

3.6.1. TRsetOutputFormat

函数原型	int TRsetOutputFormat(TrecorderHandle *hdl,int format);
功能	设置录制输出文件的格式
参数	Hdl：TRRecorder 的句柄指针 format： typedef enum { T_OUTPUT_TS, T_OUTPUT_MOV, T_OUTPUT_JPG, T_OUTPUT_AAC, T_OUTPUT_MP3, T_OUTPUT_END, }ToutputFormat; 目前视频文件支持 TS 和 MOV 两种格式
返回值	成功返回 0，失败返回-1
调用说明	此格式若设为 AAC 和 MP3 格式（纯音频），则配置的视频相关的参数将被抛弃。目前暂不支持 JPG 格式，需要拍照请使用截图函数实现同样功能。在 DataSourceConfigured 状态调用。

3.6.2. TRsetVideoEncoderFormat

函数原型	int TRsetVideoEncoderFormat(TrecorderHandle *hdl,int VFormat);
功能	设置视频编码器的编码格式
参数	Hdl：TRRecorder 的句柄指针 VFormat： typedef enum { T_VIDEO_H264, T_VIDEO_MJPEG, T_VIDEO_END,

	}TvideoEncodeFormat;
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态调用

3.6.3. TRsetVideoEncodeSize

函数原型	int TRsetVideoEncodeSize(TrecorderHandle *hdl,int width,int height);
功能	设置视频编码的大小
参数	Hdl：TRRecorder 的句柄指针 width：编码视频的宽度 height：编码视频的高度
返回值	成功返回 0，失败返回-1
调用说明	如此大小比摄像头的大小大，则通过插值方式放大图像再编码。如此大小比摄像头小，则缩放图像后再编码。在 DataSourceConfigured 状态调用

3.6.4. TRsetEncodeFramerate

函数原型	int TRsetEncodeFramerate(TrecorderHandle *hdl,int framerate);
功能	设置视频编码的帧率
参数	Hdl：TRRecorder 的句柄指针 Framerate：视频编码的帧率
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态调用。如设置的帧率和摄像头帧率不一致，将通过丢帧或者插帧的方式达到此帧率。

3.6.5. TRsetVEScaleDownRatio

函数原型	int TRsetVEScaleDownRatio(TrecorderHandle *hdl,int ratio);
功能	设置 VE 输出给 Preview 数据源的缩放比例
参数	Hdl：TRRecorder 的句柄指针 ratio: 可设为 2,4,8,分别代表 VE 把源数据通过 1/2 缩放，1/4 缩放，1/8 缩放后送给 preview 模块，用于节省内存和带宽等，同时可以控制不同分辨率下的清晰度。
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态或者 Previewing/Recording 状态都可调用。仅当 TRsetPreviewRoute 设置为 T_ROUTE_VE 时生效

3.6.6. TRsetAudioEncoderFormat

函数原型	int TRsetAudioEncoderFormat(TrecorderHandle *hdl,int AFormat);
功能	设置音频编码器的编码格式
参数	Hdl：TRRecorder 的句柄指针 AFormat： typedef enum { T_AUDIO_PCM,

	T_AUDIO_AAC, T_AUDIO_MP3, T_AUDIO_LPCM, T_AUDIO_END, }TaudioEncodeFormat;
返回值	成功返回 0，失败返回-1。
调用说明	在 DataSourceConfigured 状态调用

3.6.7. TRsetEncoderBitRate

函数原型	int TRsetEncoderBitRate(TrecorderHandle *hdl,int bitrate);
功能	设置视频编码的比特率
参数	Hdl：TRRecorder 的句柄指针 Bitrate：视频编码的比特率
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态调用。通过设置此参数可以控制码率。

3.6.8. TRsetRecorderEnable

函数原型	int TRsetRecorderEnable(TrecorderHandle *hdl,int enable);
功能	确认录制相关参数配置完毕并打开录制功能
参数	Hdl：TRRecorder 的句柄指针 enable: 0 表示 disable，1 表示 enable
返回值	成功返回 0，失败返回-1
调用说明	此函数起到应用主动确认的作用，但是最终参数是否的确设置完毕需要调用者自己确认。此函数调用后，将不能再设置录制相关参数。在 DataSourceConfigured 状态调用

3.7.封装设置类

该类函数主要实现文件封装完成回调函数设置、单个文件录制时间、输出文件路径设置等操作。

3.7.1. TRsetRecorderCallback

函数原型	int TRsetRecorderCallback(TrecorderHandle *hdl,TRecorderCallback callback,void* pUserData);
功能	设置 TRRecorder 的消息回调函数
参数	Hdl：TRRecorder 的句柄指针 callback： typedef int (*TRecorderCallback)(void* pUserData,int msgType,void* param); 其中 msgType： typedef enum TrecorderCallbackMsg { T_RECORD_ONE_FILE_COMPLETE = 0, }TrecorderCallbackMsg; 此 callback 用于处理 TRReocrder 发回的消息。目前支持的消息为 FILE_COMPLETE 消息，

	此消息在录制文件长度达到所设时长时发出。处理此消息时需要设置下一个录制文件的文件名。 pUserData:回调发回的用于区分的 userdata
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态调用

3.7.2. TRsetMaxRecordTimeMs

函数原型	int TRsetMaxRecordTimeMs(TrecorderHandle *hdl,int ms);
功能	设置录制单一文件的最大时长
参数	Hdl：TRRecorder 的句柄指针 msec: 存储文件总时长，单位：ms
返回值	成功返回 0，失败返回-1
调用说明	在 DataSourceConfigured 状态调用

3.7.3. TRchangeOutputPath

函数原型	int TRchangeOutputPath(TrecorderHandle *hdl,char* path);
功能	在 TRRecorder 中切换文件录制的命名
参数	Hdl：TRRecorder 的句柄指针 Path：录制的文件路径
返回值	成功返回 0，失败返回-1
调用说明	在 previewing/recording 状态下调用。

3.8. 外部 module 操作类

当存在外部模块与 TinaRecorder 内部模块实现数据通路时，需要调用该类函数将模块数据通路连接起来。

3.8.1. TRmoduleLink

函数原型	int TRmoduleLink(TrecorderHandle *hdl, TmoduleName *moduleName_1, TmoduleName *moduleName_2, ...);
功能	连接相应模块的数据通路
参数	Hdl：TRRecorder 的句柄指针 moduleName_1、moduleName_2： typedef enum { T_CAMERA = 0x01,//camera 模块 T_AUDIO = 0x02,//mic 模块 T_DECODER = 0x04,//解码模块

	T_ENCODER = 0x08,//编码模块 T_SCREEN = 0x10,//显示模块 T_MUXER = 0x20,//封装模块 T_CUSTOM = 0x40,//用户自定义模块 }TmoduleName;
返回值	成功返回 0，失败返回-1
调用说明	在调用该函数的时候 ,需要设置用户自定义模块的 name、inputTyte、outputTyte 等参数。

3.8.2. TRmoduleUnlink

函数原型	int TRmoduleUnlink(TrecorderHandle *hdl, TmoduleName *moduleName_1, TmoduleName *moduleName_2, ...);
功能	断开相应模块的数据通路
参数	Hdl：TRRecorder 的句柄指针 moduleName_1、moduleName_2： typedef enum { T_CAMERA = 0x01,//camera 模块 T_AUDIO = 0x02,//mic 模块 T_DECODER = 0x04,//解码模块 T_ENCODER = 0x08,//编码模块 T_SCREEN = 0x10,//显示模块 T_MUXER = 0x20,//封装模块 T_CUSTOM = 0x40,//用户自定义模块 }TmoduleName;
返回值	成功返回 0，失败返回-1
调用说明	调用该函数之后，模块间数据连接将断开，传输数据需要重新 link。

3.8.3. packetCreate

函数原型	struct modulePacket *packetCreate(int bufSize);
功能	创建一个新的数据包

参数	bufSize：产生数据包的 buf 指针指向内存块的大小，可以为 0。
	struct modulePacket：
	struct modulePacket{
	enum packetType packet_type; /* packet data type */
	cdx_atomic_t ref; /* the number of packet references */
	int OnlyMemFlag;
	union{
	struct freeGroup free;
	struct notifyGroup notify;
	}mode;
	void *buf;
	void *reserved; /* reserved for special use, the default is zero */
	};

返回值	成功返回数据包地址，失败返回 NULL
调用说明	成员变量 OnlyMemFlag 指示该 packet 是否仅仅包含系统内存而没有模块的内部数据，0：包含内部数据，需要下一模块使用完通过 notify 中的 notifySrcFunc 函数告知本模块；1：不包含模块内部数据，当 packet 的 ref 为零时可直接调用 free 中的释放函数释放 packet 而不需要告知本模块

3.8.4. packetDestroy

函数原型	int packetDestroy(struct modulePacket *mPacket);
功能	释放数据包
参数	mPacket：模块使用完毕，待释放的 packet
返回值	成功返回 0，失败返回-1
调用说明	mPacket 中的成员变量 OnlyMemFlag 指示该 packet 是否仅仅包含系统内存而没有模块的内部数据，0：包含内部数据，需要下一模块使用完通过 notify 中的 notifySrcFunc 函数告知本模块；1：不包含模块内部数据，当 packet 的 ref 为零时可直接调用 free 中的释放函数释放 packet 而不需要告知本模块

3.8.5. ModuleSetNotifyCallback

函数原型	int ModuleSetNotifyCallback(struct moduleAttach *module, NotifyCallbackForSinkModule notify, void *handle);
功能	设置模块接收数据时的通知方式

参数	module：模块的句柄指针
	struct moduleAttach{
	struct outputSrc *output; //保存输出模块的相关信息
	AwPoolT *sinkDataPool; /* self module data pool */
	CdxQueueT *sinkQueue; /* self module data queue */
	unsigned int name; /* sele module name */
	unsigned int src_name; /* input data module name */
	unsigned int inputTyte; /* supported input data type */
	unsigned int outputTyte; /* supported output data type */
	unsigned int moduleEnable; /* self module enable flag */
	cdx_atomic_t packetCount; /* queue data packet count*/
	void *freePacketHdl; /* free packet callback handle */
	freePacketCallBack freeFunc; /* free packet callback function */
	sem_t waitReceiveSem; //等待数据到达的信号量
	sem_t waitReturnSem; //等待下一模块使用完数据释放的信号量
	NotifyCallbackForSinkModule notifyFunc; //上游模块发送数据时的通知方式
	void *notifyHdl; //上游模块通知方式使用到的 handle
	};
	notify：
	typedef int (*NotifyCallbackForSinkModule)(void *handle);
	Handle：notify 函数中需要用到的 handle；

返回值	成功返回 0，失败返回-1
调用说明	当上游模块发送数据到该模块时，notify 函数将被调用（由上游模块调用）

3.8.6. NotifyCallbackToSink

函数原型	int NotifyCallbackToSink(void *handle);
功能	Post handle 指向模块的信号量 waitReceiveSem
参数	handle：struct moduleAttach 的句柄指针
返回值	成功返回 0，失败返回-1
调用说明	一般与 ModuleSetNotifyCallback()函数配合使用，当上游模块发送数据时，通过该函数发送 waitReceiveSem 信号量

3.8.7. Module 信号量操作

函数原型	void module_waitReturnSem(void *handle);
	void module_postReturnSem(void *handle);
	void module_waitReceiveSem(void *handle);
	void module_postReceiveSem(void *handle);

功能	操作 module 相应信号量
参数	Handle：struct moduleAttach 的句柄指针
返回值	无
调用说明	Module 可通过信号量实现数据同步

3.9. Module 数据操作类

该类函数为外部模块操作 packet，管理数据包操作等。

3.9.1. module_push

函数原型	int module_push(struct moduleAttach *module, struct modulePacket *mPacket);
功能	将数据包 mPacket 发送到 link 的模块
参数	module：struct moduleAttach 的句柄指针 mPacket：待发送的数据包
返回值	成功发送该 packet 到下游模块的个数，失败返回小于或等于 0
调用说明	可根据返回确认需要等待 packet 释放的个数（返回值代表将有多少个 module 使用该 packet）

3.9.2. module_pop

函数原型	void *module_pop(struct moduleAttach *module);
功能	从模块的数据队列中 pop 一个 packet
参数	module：struct moduleAttach 的句柄指针

返回值	成功返回 struct modulePacket 类型的指针，失败返回 NULL
调用说明	

3.9.3. module_InputQueueEmpty

函数原型	int module_InputQueueEmpty(struct moduleAttach *module);
功能	查询 module 数据队列是否为空
参数	module：struct moduleAttach 的句柄指针
返回值	队列为空返回 1，队列不为空返回 0
调用说明	

3.9.4. alloc_VirPhyBuf

函数原型	int alloc_VirPhyBuf(unsigned char **pVirBuf, unsigned char **pPhyBuf, int length);
功能	申请 length 大小的物理地址连续的内存
参数	pVirBuf：指向申请内存的虚拟地址 pPhyBuf：指向申请内存的物理地址 length：申请的长度
返回值	成功返回 0，失败返回-1
调用说明	

3.9.5. memFlushCache

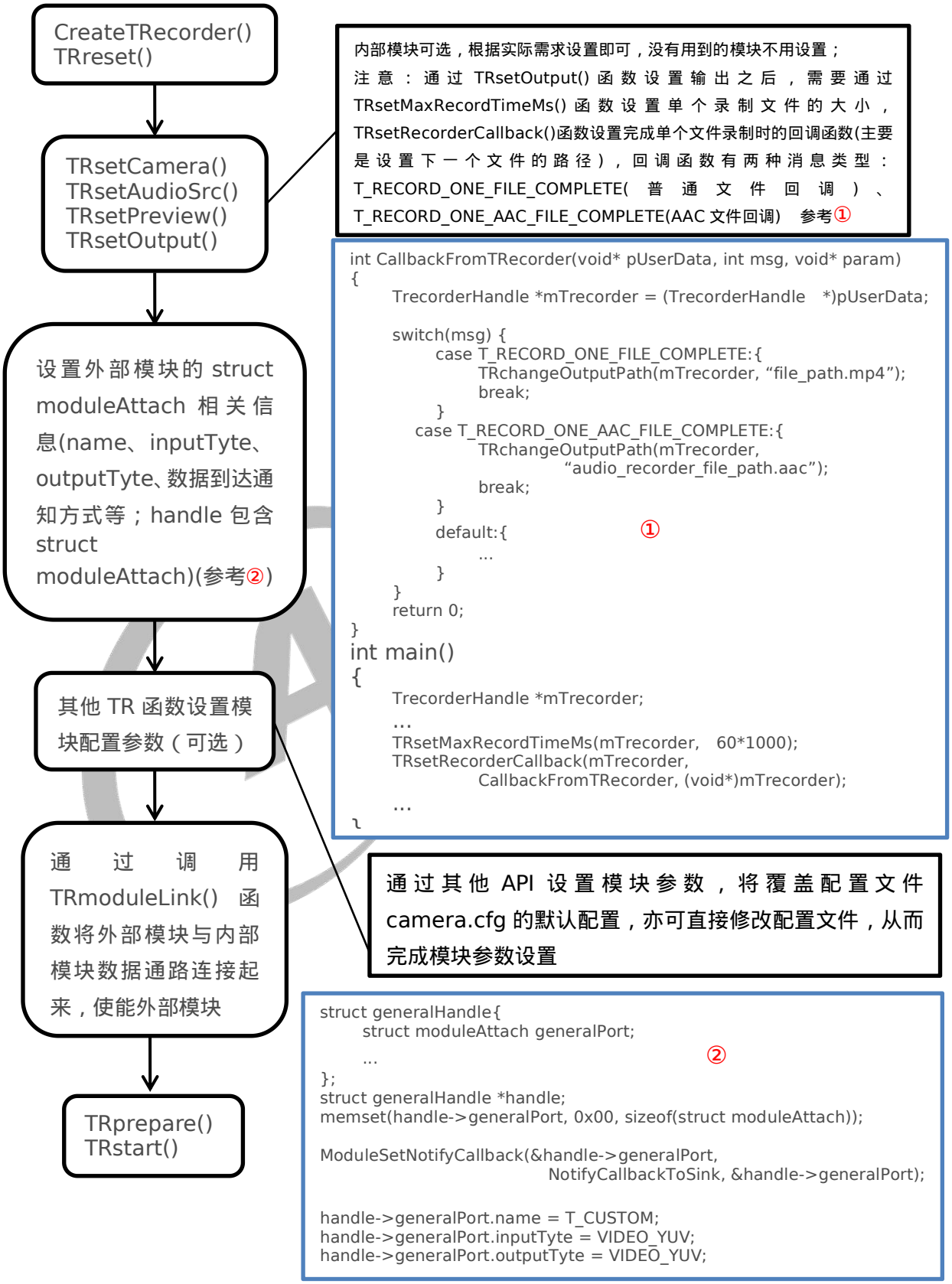
函数原型	void memFlushCache(void *pVirBuf, int nSize);
功能	将 pVirBuf 指向的数据刷出缓冲
参数	pVirBuf：数据的地址 nSize：数据的长度
返回值	无
调用说明	防止数据没有及时同步，同调用该函数确认数据一致

3.9.6. free_VirPhyBuf

函数原型	int free_VirPhyBuf(unsigned char *pVirBuf);
功能	释放通过 alloc_VirPhyBuf()函数申请的物理内存连续的内存
参数	pVirBuf:待释放的内存指针
返回值	成功返回 0，失败返回-1
调用说明	

4. TinaRecorder 模块开发框架

下图为基于 trecorder 使用外部 module 与 trecorder 结合开发程序的流程框架：



5. TinaRecorder 模块开发

5.1. 外部模块开发

trecorder 除了可以通过调用 API 设置各个模块的参数之外 ,还可以通过设置/etc/camera.cfg 文件设置模块参数，在调用 TRsetCamera()、setAudioSrc()、TRsetPreview()和 TRSetOutput()这四个方法的时候，将会在这个 API 函数内部进行配置参数的读取，同时，如果在这之后通过 API 函数设置相应的参数，将覆盖配置文件的参数。

改版之后的 trecorder 新增两个 API 用于 module link：

```
int TRmoduleLink(TrecorderHandle *hdl, TmoduleName *moduleName_1,
                                                    TmoduleName *moduleName_2, ...);
int TRmoduleUnlink(TrecorderHandle *hdl, TmoduleName *moduleName_1,
                                                    TmoduleName *moduleName_2, ...);
```

这两个 API 调用最后需用 NULL 参数结束，其余参数意义如下：

hdl------TrecorderHandle 类型的结构体，该结构体是 trecorder 内部 handle；

moduleName_X------TmoduleName 类型的枚举变量，将通过该变量说明 link 或者 unlink 时模块的连接顺序，它的定义如下：

```
typedef enum
{
    T_CAMERA        = 0x01,             ; 代表 trecorder 内部的摄像头模块
    T_AUDIO         = 0x02,             ; 代表 trecorder 内部的麦克风模块
    T_DECODER       = 0x04,             ; 代表 trecorder 内部的视频解码模块
    T_ENCODER       = 0x08,             ; 代表 trecorder 内部的编码模块
    T_SCREEN        = 0x10,             ; 代表 trecorder 内部的显示模块
    T_MUXER         = 0x20,             ; 代表 trecorder 内部的封装模块
    T_CUSTOM        = 0x40,             ; 代表 trecorder 用户自定义添加的模块
}TmoduleName;
```

下面就用户层自定义的外部模块与 trecorder 内部模块连接使用展开说明：

在外部模块中，都将需要定义一个类型为 struct moduleAttach 的变量，将会通过该变量完成外部模块与 trecorder 内部模块的数据通路连接，该变量声明了外部模块的 name、数据输出类型以及将输出到哪些

模块等信息。所以在设置 trecorder 内部相应的模块之后，将要初始化外部模块中 struct moduleAttach 类型的变量，struct moduleAttach 定义如下：

```
struct moduleAttach{
    struct outputSrc *output;          /* next module data source */
    AwPoolT *sinkDataPool;             /* self module data pool */
    CdxQueueT *sinkQueue;              /* self module data queue */
    unsigned int name;                 /* sele module name */
    unsigned int src_name;             /* input data module name */
    unsigned int inputTyte;            /* supported input data type */
    unsigned int outputTyte;           /* supported output data type */
    unsigned int moduleEnable;         /* self module enable flag */
    cdx_atomic_t packetCount;          /* queue data packet count */
    void *freePacketHdl;               /* free packet callback handle */
    freePacketCallBack freeFunc;       /* free packet callback function   */
    sem_t waitReceiveSem;
    sem_t waitReturnSem;
    NotifyCallbackForSinkModule notifyFunc;
    void *notifyHdl;
};
```

output：如果该模块会将自身处理的数据 push 到下一个模块，output 保存下一个模块相关信息；

sinkDataPool：模块的队列内存缓冲池；

sinkQueue：模块的数据队列，上游模块将会把数据 push 到该队列；

name：模块的名称；

src_name：保存模块的输入源名称；

inputTyte：模块可接收处理的数据类型；

outputTyte：模块将会产生的数据类型；

moduleEnable：模块使能标志位，若为 1 则代表该模块可以接收、处理数据了；

packetCount：模块队列中还有多少数据没有读取；

waitReceiveSem：模块等待数据的信号量；

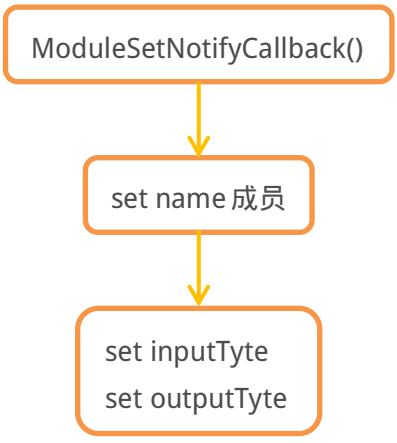
waitReturnSem：模块等待自身产生的数据包使用完毕的信号量；

notifyFunc：通知模块接收到数据的函数指针；

notifyHdl：通知函数的结构体；

struct moduleAttach 初始化流程如下：

1.通过 int ModuleSetNotifyCallback(struct moduleAttach *module, NotifyCallbackForSinkModule notify, void *handle)函数设置当上游模块发送的数据到达本模块时的通知方式，该函数的 module 参数代表本模块，而 notify 参数则为通知的方式 ,而 handle 参数表示 notify 函数中使用到的 handle。（目前 struct moduleAttach 结构体中包含了一个名为 waitReceiveSem 的信号量，可通过该函数设置上游模块发送数据时调用 module_postReceiveSem 函数 post 该信号量，然后在相应模块的处理循环中，通过 module_waitReceiveSem()函数等待上游模块发送该信号量）



struct moduleAttach
初始化流程

2.设置模块的 name：这个变量，外部模块都将统一设置为 T_CUSTOM(在 TRmoduleLink 或者 TRmoduleUnlink 时都将通过这个成员变量获取 struct moduleAttach 的首地址 ,从而完成 link 或 unlink 动作)。

3.设置模块可处理的数据类型，分别包含输入类型与输出类型：inputTyte 代表着可接收的数据类型，outputTyte 代表着模块支持的输出类型(这两个成员变量很重要，在 module link 和 data push 的时候，都将会检查两个模块间数据类型是否支持，只有支持的才可以进行下一步操作，否则失败)。

在完成外部模块的 struct moduleAttach 变量填充之后，再通过 TRmoduleLink()函数完成外部模块与 trecorder 内部模块的数据通路连接，最后还需要通过 modulePort_SetEnable()函数使能外部模块。

5.2. 模块类型

下面将介绍主要的三种模块：src 模块、filter 模块和 sink 模块。

1.src 模块：不接收其他模块的数据，自身将产生数据，传输给下一模块的输出型模块，由于 trecorder 内部部分模块通过物理地址操作相应的数据，src 模块在将数据传给下一模块注意填充物理地址等变量。

在 src 模块的 handle 中，需要定义一个类型为 struct moduleAttach 的变量，初始化需要填充 struct moduleAttach 的变量的 name 为 T_CUSTOM，设置 struct moduleAttach 的变量的 outputTyte 变量（填充值为 enum packetType 类型），最后通过 TRmoduleLink()函数完成相应模块的 link。

而后在 src 模块的 handle 循环中，当自身模块产生数据时，将通过 struct modulePacket *packetCreate(int bufSize)函数产生一个新的 packet。根据 src 模块的输出数据类型，完成 packet 的 buf 变量填充（视频数据使用 struct MediaPacket 填充，音频数据使用 struct AudioPacket 填充，而编码之后的数据使用 CdxMuxerPacketT 填充），而后根据 src 模块的内容完成 OnlyMemFlag、mode 和 packet_type 等变量的填充，通过 int module_push(struct moduleAttach *module, struct modulePacket *mPacket)函数将该 packet push 到下一模块，module_push 函数的返回值代表将该 packet push 到多少个下游模块，最后根据 src 模块的自身情况，等待下游模块返还 buf 或者直接释放相应资源，从而完成一次 handle 循环。

2.filter 模块：接收其他模块的数据经过处理之后，将处理之后的数据传输给下一模块的过滤型模块，由于 trecorder 内部部分模块通过物理内存连续的内存块处理相应的数据，所以 filter 模块在将数据传给下一模块注意填充物理地址等变量。

在 filter 模块的 handle，同样的需要定义一个类型为 struct moduleAttach 的变量，初始化需要通过 ModuleSetNotifyCallback()函数完成 filter 模块的数据到达时的通知形式以及填充 struct moduleAttach 的变量的 name 为 T_CUSTOM，设置 struct moduleAttach 的变量的 inputTyte、outputTyte 变量(填充值为 enum packetType 类型)，最后通过 TRmoduleLink()函数完成相应模块的 link。

而后在 filter 模块的 handle 循环中，当上游模块 push packet 到达 filter 模块通过

ModuleSetNotifyCallback()设置的函数通知 filter 模块的时候(也可以通过 filter 模块 struct moduleAttach 的变量 sinkQueue 是否为空判断是否已经有数据到达)，然后将通过 module_pop()函数 pop packet，通过 packet 的 packet_type 辨别不同的 packet 从而进行相应处理，处理完成之后，将通过 packetDestroy()函数释放该 packet。

同时通过 struct modulePacket *packetCreate(int bufSize)函数产生一个新的 packet。根据 filter 模块的输出数据类型，完成 packet 的 buf 变量填充 (视频数据使用 struct MediaPacket 填充，音频数据使用 struct AudioPacket 填充，而编码之后的数据使用 CdxMuxerPacketT 填充），而后根据 filter 模块的内容完成 OnlyMemFlag、mode 和 packet_type 等变量的填充，通过 int module_push(struct moduleAttach *module, struct modulePacket *mPacket)函数将该 packet push 到下一模块，module_push 函数的返回值代表将该 packet push 到多少个下游模块，最后根据 filter 模块的自身情况，等待下游模块返还 buf 或者直接释放相应资源，从而完成一次 handle 循环。

3.sink 模块：只接收数据，不输出数据的输入型模块。

在 sink 模块的 handle，同样的需要定义一个类型为 struct moduleAttach 的变量，初始化需要通过 ModuleSetNotifyCallback()函数完成 sink 模块的数据到达时的通知形式以及填充 struct moduleAttach 的变量的 name 为 T_CUSTOM，设置 struct moduleAttach 的变量的 inputTyte 变量(填充值为 enum packetType 类型)，最后通过 TRmoduleLink()函数完成相应模块的 link。

而后在 sink 模块的 handle 循环中，当上游模块 push packet 到达 sink 模块通过 ModuleSetNotifyCallback()设置的函数通知 sink 模块的时候(也可以通过 filter 模块 struct moduleAttach 的变量 sinkQueue 是否为空判断是否已经有数据到达)，然后将通过 module_pop()函数 pop packet，通过 packet 的 packet_type 辨别不同的 packet 从而进行相应处理，处理完成之后，将通过 packetDestroy()函数释放该 packet，从而完成一次 handle 循环。

5.3. 模块 packet 管理

trecorder 模块间的数据通过 packet 的形式，所以可以动态的通过 API 将不同的模块有效的连接起来，需要注意的是，在模块 link 的时候，需要填充模块支持的输入输出格式，因为在 link 的内部实现，将会检查两个模块间的数据有效性。同时，在 link 之前，需要通过 ModuleSetNotifyCallback()方法设置 sink 模块是如何接收 src 模块发送 packet 的消息通知。在 trecorder 中，视频数据的 buf 是通过 struct MediaPacket 类型的结构体封装，而音频数据的 buf 则通过 struct AudioPacket 类型的结构体封装，这两种类型的结构体定义在头文件 dataqueue.h，而编码之后的音视频数据则是通过 CdxMuxerPacketT 类型进行封装。同时需要注意的是，由于 trecorder 内部模块（encoder 编码模块）使用物理内存连续的 buf，所以当需要用到编码模块，都将需要填充相应的物理地址（显示模块可以物理地址，也可以虚拟地址）。

packet 的数据形式定义如下：

```
enum packetType{
    VIDEO_RAW       = 0x01,          ；视频类的 RAW 数据
    VIDEO_YUV       = 0x02,          ；视频类的 YUV 数据
    AUDIO_PCM       = 0x04,          ；音频类的 PCM 数据
    VIDEO_ENC       = 0x08,          ；视频类的编码之后的视频数据
    AUDIO_ENC       = 0x10,          ；音频类的编码之后的音频数据
    SCALE_YUV       = 0x20,          ；视频类的缩放 YUV 数据(YUV 缩放之后的小图)
    CUSTOM_TYPE     = 0X80,          ；用户层自定义的数据类型
};
```

Struct packet 的定义如下

```
struct modulePacket{
    enum packetType packet_type;    /* packet data type */
    cdx_atomic_t ref;               /* the number of packet references */ (详见①)
    int OnlyMemFlag;                                                      (详见②)
    union{
        struct freeGroup free;                                            (详见③)
            struct notifyGroup notify;                                    (详见④)
    }mode;
    void *buf;                                                            (详见⑤)
    void *reserved;                 /* reserved for special use, the default is zero */
};
```

详注：

①该成员指示这个 packet 被多少个模块引用；

②该成员变量指示该 packet 的 buf 指向的数据仅为申请的内存而不包含模块的自身数据：

0：包含内部数据，需要下一模块使用完通过 notify 中的 notifySrcFunc 函数告知本模块；

1：不包含模块内部数据，当 packet 的 ref 为零时可直接调用 free 函数释放 packet 而不需要告知本模块。

③该成员变量为当 OnlyMemFlag 为 1 时，packet 的引用计数值 ref 为零时通过调用该函数释放内存；

④该成员变量为当 OnlyMemFlag 为 0 时，释放 packet 将会通过 notify 中函数通知发出该 packet 的模块；

⑤指向该 packet 的 buf 内存 ,在通过 struct modulePacket *packetCreate(int bufSize)函数产生一个新的 packet，而参数 bufSize 代表 packet 的 buf 指向内存的大小；

5.4. TinaRecorder 内部模块 packet 形式

下表为 trecorder 内部各模块的输入输出情况：

module	Module type	inputTyte	outputTyte	Input buf type	Output buf type
camera	src	无	VIDEO_YUV | VIDEO_ENC | VIDEO_RAW	无	struct MediaPacket
audio	src	无	AUDIO_PCM	无	Struct AudioPacket
decoder	filter	VIDEO_ENC | AUDIO_ENC	VIDEO_YUV | AUDIO_PCM	struct MediaPacket	struct MediaPacket
screen	sink	VIDEO_YUV | SCALE_YUV	无	struct MediaPacket	无
encoder	filter	AUDIO_PCM | VIDEO_YUV | SCALE_YUV	AUDIO_ENC | VIDEO_ENC | SCALE_YUV	StructMediaPacket/ struct AudioPacket	CdxMuxerPacketT
muxer	sink	AUDIO_ENC | VIDEO_ENC	无	CdxMuxerPacketT	无

6. TinaRecorder 配置文件说明

TinaRecorder 的配置文件 camera.cfg 在机器端的存在路径为/etc/camera.cfg，在机器端可通过修改该配置文件，修改 trecorder 的模块参数(在应用层，可通过其他 TR 函数修改模块配置参数，从而达到覆盖配置文件参数的作用)。下面将介绍配置文件的配置项。

```
camera_id = 0                                   ; camera 索引
;--------------------------------------------------------------------------
; 2 for usb sensor
; 1 for raw sensor (need isp)
; 0 for yuv sensor
;--------------------------------------------------------------------------
camera_type = 0                                 ; camera 物理硬件类型
video_enable = 1
video_width = 1920                              ; camera 分辨率
video_height = 1080
video_framerate = 30                            ; camera 输出帧率
;-------------------------------------------------------------------------------------------------------------
video_format 为 camera 输出格式 ,支持 YVU420SP、YUV420SP、YUV420P、YVU420P、YUV422SP、
YVU422SP、YUV422P、YVU422P、YUYV422、UYVY422、YVYU422、VYUY422、MJPEG、H264
video_format = YVU420SP
;-------------------------------------------------------------------------------------------------------------
video_rotation = 0                              ; ISP 处理旋转角度
video_use_wm = 1                                ; 使能水印标志位
video_wm_pos_x = 20                             ; 水印的添加位置
video_wm_pos_y = 20
;--------------------------------------------------------------------------
; scale down need isp
;--------------------------------------------------------------------------
video_scale_down_enable = 0                     ; 使能 ISP 以及缩放图的大小
video_sub_width =
video_sub_height =
video_buf_num = 3                               ; camera buf 的个数

audio_enable = 1
;-------------------------------------------------------------------------------------------------------------
audio 的输出格式、支持 AAC、PCM、MP3、LPCM
;-------------------------------------------------------------------------------------------------------------
audio_format = PCM
audio_channels = 2                              ; audio 通道数
audio_samplerate = 8000                         ; audio 采样率
audio_samplebits = 16                           ; audio 采样位数

display_enable = 1
```

```
display_rect_x = 0                                  ；显示的位置与大小
display_rect_y = 0
display_rect_width = 1280
display_rect_height = 800
;-------------------------------------------------------------------------
; 0 ZORDER_TOP
; 1 ZORDER_MIDDLE
; 2 ZORDER_BOTTOM
;-------------------------------------------------------------------------
display_zorder = 2                                  ；显示的图层
;-------------------------------------------------------------------------
; 0 ROTATION_ANGLE_0
; 1 ROTATION_ANGLE_90
; 2 ROTATION_ANGLE_180
; 3 ROTATION_ANGLE_270
;-------------------------------------------------------------------------
display_rotation = 0                                ；显示的角度

;-------------------------------------------------------------------------------------------------------------------
显示的数据源大小(在没有设置 camera 却使用显示时需要配置该项)
;-------------------------------------------------------------------------------------------------------------------
display_src_width = 640
display_src_height = 480

encoder_enable = 1

encoder_voutput_enable = 1                          ；编码的视频使能
encoder_voutput_type = H264                         ；编码视频格式
encoder_voutput_width = 1920                        ；编码的输出分辨率
encoder_voutput_height = 1080
encoder_voutput_framerate = 30                      ；编码输出帧率
encoder_voutput_bitrate = 16000000                  ；视频编码的码率
encoder_scale_down_ratio = 2                        ；缩放的倍数，默认应为 0

encoder_aoutput_enable = 1                          ；编码的音频使能
encoder_aoutput_type = AAC                          ；编码音频格式
encoder_aoutput_channels = 2                        ；编码音频通道数
encoder_aoutput_samplerate = 8000                   ；编码音频的采样率
encoder_aoutput_samplerbits = 16                    ；audio 采样精度
encoder_aoutput_bitrate = 3200                      ；audio 输出比特率

muxer_enable = 1
;-------------------------------------------------------------------------------------------------------------------
muxer 的封装格式，支持 TS、MOV、JPG、AAC、MP3
;-------------------------------------------------------------------------------------------------------------------
muxer_output_type = MOV
```

注意事项：

1.使用到音频数据的，需要先使用其他工具打通音频数据通路（否则可能出现其他的一些问题，设置音频通路的工具，比如 alsa-ucm-aw）；

2.如果在预览的时候，没有报任何错误，但却没有正常显示，这个时候检查时候有共用 Zorder，通过 cat sys/class/disp/disp/attr/sys 即可查看相应信息，关注 z 那一列的相应信息即可知道是否共用 Zorder，相关的修改 Zorder 即可正常显示；

3.如果使用到水印文件，将需要将水印文件拷贝到机器端的/rom/etc/res 目录下，且水印文件命名为 wm_540p_*，拷贝动作在相应的 Makefile 完成，这个可参考相关 demo 的实现；

4.配置文件 camera.cfg 中说明的编码缩放设置，默认将缩放之后的数据送往显示模块，同时不建议使用编码器缩放；

5.trecorder 在封装线程处通过限制 muxer 队列的大小而阻塞编码完成的数据 push 到 muxer 队列 ,这里通过宏定义实现队列的大小限制：MUXER_MoreVIDEO1080P_QUEUE_MbSIZE、MUXER_LessVIDEO1080P_QUEUE_MbSIZE 和 MUXER_AUDIO_QUEUE_KbSIZE。其中 MUXER_MoreVIDEO1080P_QUEUE_MbSIZE 为编码完成后的视频分辨率大于 1080P 时队列的大小，单位为 MB，默认为 4MB，而 MUXER_LessVIDEO1080P_QUEUE_MbSIZE 为编码后的视频分辨率小于 1080P 时队列的限制大小，单位为 MB，默认为 3MB，MUXER_AUDIO_QUEUE_KbSIZE 则为编码后的音频数据队列大小，单位为 KB，默认为 12KB(这三个宏定义实现在 TinaRecorder.h，可根据实际情况适当修改);

6.由于封装线程需要将编码之后大量数据填充到其他介质，比如 TF 卡，这种将会导致 trecorder 框架变慢，所以将提高系统将数据刷新到外存的频率，设置了/proc/sys/vm/dirty_background_bytes 参数，该值设置为 DIRTY_BACKGROUND_BYTES，默认为 4MB，在封装线程退出的时候也将还原系统原来配置。DIRTY_BACKGROUND_BYTES 定义在 TinaRecorder.h，可根据实际情况修改该值；

7.建议 uvc camera 使用编码格式输出，输出接解码器再送显示或者编码等操作；

8.Trecorder 内部设置了 debug 屏蔽模式，可通过修改 Trecorder.c 中的 TR_log_mask 值，以保证相应模块 log 信息正常输出；

9.在 TinaRecorder.c 文件定义了 SAVE_DISP_SRC、SAVE_AUDIO_SRC，可通过修改这两个宏定义以 debug display 和 audio 的数据源是否正确；

10.如果是单独录音测试，音频编码仅支持 AAC 格式输出；

11.在使用自定义的外部模块与 trecorder 内部模块 link 的时候，/etc/camera.cfg 的 camera 类型一定需要与实际硬件设备一致（即 camera.cfg 中的 camera_type 值）；

12.在自定义 link 的过程，如果没有设置摄像头作为显示输入，却会用到显示模块显示其他来源的数据，需要准确设置 camera.cfg 文件中的 display_src_width 和 display_src_height 值(代表显示数据源的分辨率)；

13.Trecorder 使用开发说明可参照《Tina 平台 trecorder 接口相关 demo 使用说明文档》；

7. 参考 demo 的路径

package/allwinner/tina_multimedia_demo/trecorderdemo
package/allwinner/tina_multimedia_demo/moduledemo

8. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgment to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.