关于本手册

《snc8600 audio framework及快速开发指南》的目标读者群体是使用 snc8600芯片的应用开发工程师。本手册提供了关于 snc8600软件framework的说明，以及客户定制化接口及模板文件， 旨在帮助客户快速上手snc8600软件包并能够搭建基于九音标准样例的定制化音频应用。

Contents

[**1** **整体架构** 3](#_Toc126939895)

[**2 中间层- Audio Manager详解** 4](#_Toc126939896)

[**2.1 Device/Session架构** 4](#_Toc126939897)

[**2.2 Device** 5](#_Toc126939898)

[**2.3 Session** 5](#_Toc126939899)

[**2.4 处理逻辑** 6](#_Toc126939900)

[**3 应用层 - 定制Audio应用** 9](#_Toc126939901)

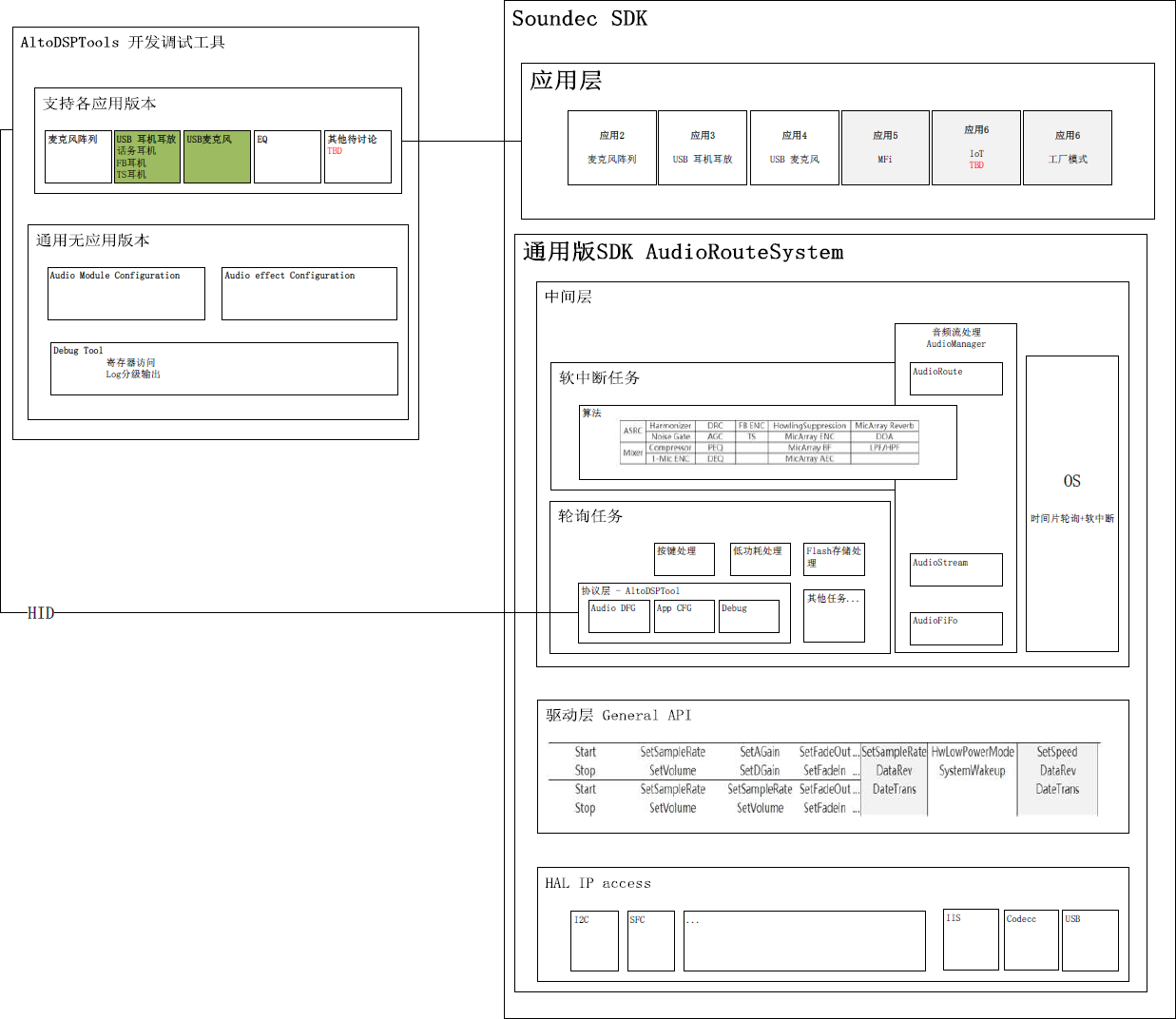
[**3.1 新增项目配置** 10](#_Toc126939902)

[**3.2 新增配置文件和音频描述文件** 10](#_Toc126939903)

[**3.3 修改配置文件** 10](#_Toc126939904)

[**3.4 修改音频通路描述文件** 14](#_Toc126939905)

[**4.** **算法集成** 18](#_Toc126939906)

1. **整体架构**

DspTools提供PC侧的开发调试功能， Soundec SDK则提供固件侧完整的软件运行环境。

Soundec SDK又分为上层的应用层，中间的framework层以及底层驱动层：

1. 底层提供完整的hal及驱动层， hal层以库文件形式提供，driver层提供设备的统一接口，挂载，启动，各种set及get接口
2. 中间层以Audio Manager为核心， 通过Audio Device 管理音频硬件设备的挂载和驱动， 并通过 Audio Session 管理设备间PCM数据流的输入输出
3. 应用层提供了面向用户场景的基于audio device/session的模板配置文件，使开发者可以进行快速扩展，实现定制的音频配置。

对于大多数开发者而言，直接使用应用层提供的模板配置文件进行扩展，可以快速搭建自己的音频链路，实现特定的应用场景，是sdk推荐的使用方式。 当然开发者也可以绕过应用层，自行使用Audio Manager的接口来创建对应的音频通道，这需要对sdk的底层代码有一定的熟悉，难以做到快速开发。

1. **中间层- Audio Manager详解**

Audio Manger是整个中间层framework层的核心，面向音频处理，采用 Device/Session设计， 根据应用的音频通路需求，开发者只需要新建和配置 音频会话描述文件，系统即可通过 AudioManager 自动加载对应的音频 Device， 从而完成音频模块的便携式配置，实现多入多出音频流通路的自由组合。

1. **Device/Session架构**



**Audio Manager**

**Audio Session**

Input

**Input**

**Device**

Push

Pop

**FiFo**

**Output**

**Device**

Output

DriverConfig

DriverConfig

DriverState

DriverState

Ops

Ops

**Driver Layer**

**Driver**

**Driver**

Deviceinformation

Deviceinformation

1. **Device**



注册设备的音频格式（采样率、位宽等）

注册设备的收发属性（接口能力、数据流处理、DMA 配置等）

注册设备状态查询接口

注册设备的初始化及启动管理接口

**Driver**

Ops

DriverState

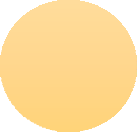
DriverConfig

**Device**

Deviceinformation

1. **Session**

* 单入单出



**Session**

*Capture Stream*

*Playback Stream*

Input Device1

*Push*

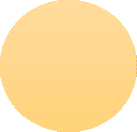
Capture

FiFo1

*Pop*

Output Device1

* 单入多出



**Session**

*Playback Stream*

*Pop*

*Capture Stream*

Output Device1

Input Device1

*Push*

Capture

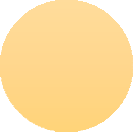
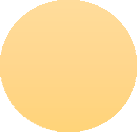
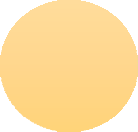
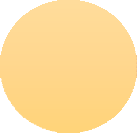
FiFo1

*Playback Stream*

*Pop*

Output Device2

* 多入多出 Session 通路



**Session**

*Capture Stream*

*Playback Stream*

Input Device1

*Push*

Capture

FiFo1

**Processor**

Result

FiFo

*Pop*

Output Device1

*Capture Stream*

*Playback Stream*

Input Device2

*Push*

Capture

FiFo2

*Pop*

Output Device2

*Capture Stream*

Input Device3

*Push*

Capture

FiFo3

1. **处理逻辑**

Audio Manager会根据开发者提供的配置文件执行如下操作

* 1. **注册硬件/加载音频设备**

硬件设备的注册接口见文件audio\_hw\_desc.c，通过其中auDeviceIfs 数组定义了包含Codec(ADC&DAC), I2S, USB（Mic&Speaker）等音频设备及其描述接口

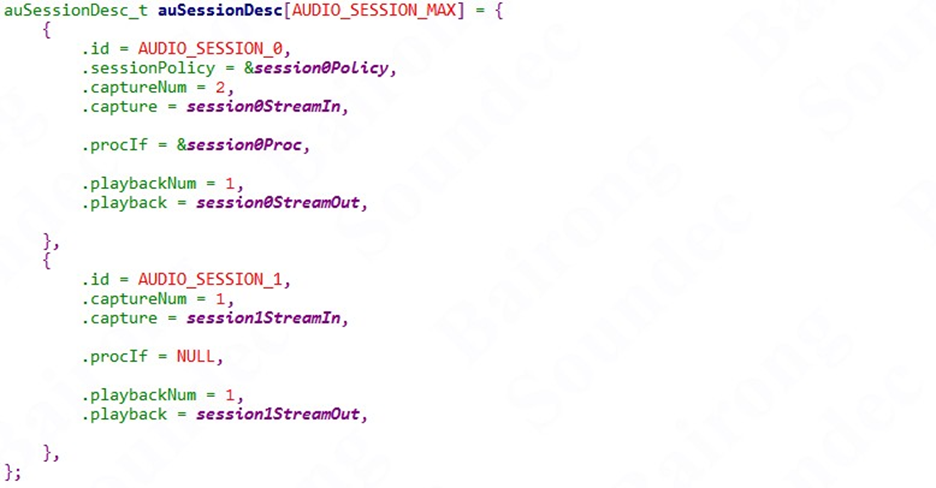
**Hal layer**



* 1. **注册 Session/打开对应通路**

音频通路 Session 的注册接口见文件 audio\_session\_desc\_xx.c，以双麦会议音箱应用为例，其音频通路配置文件为audio\_session\_desc\_2mic\_meetingbox.c 。此应用支持两路音频通路：AUDIO\_SESSION\_0 和 AUDIO\_SESSION\_1。

* AUDIO\_SESSION\_0 支持 2 路输入（capture），1 路输出（playback），输入信号经过（session0Proc）处理后输出；
* AUDIO\_SESSION\_1 支持 1 路输入（capture），1 路输出（playback），输入信号不需要处理，直接输出：



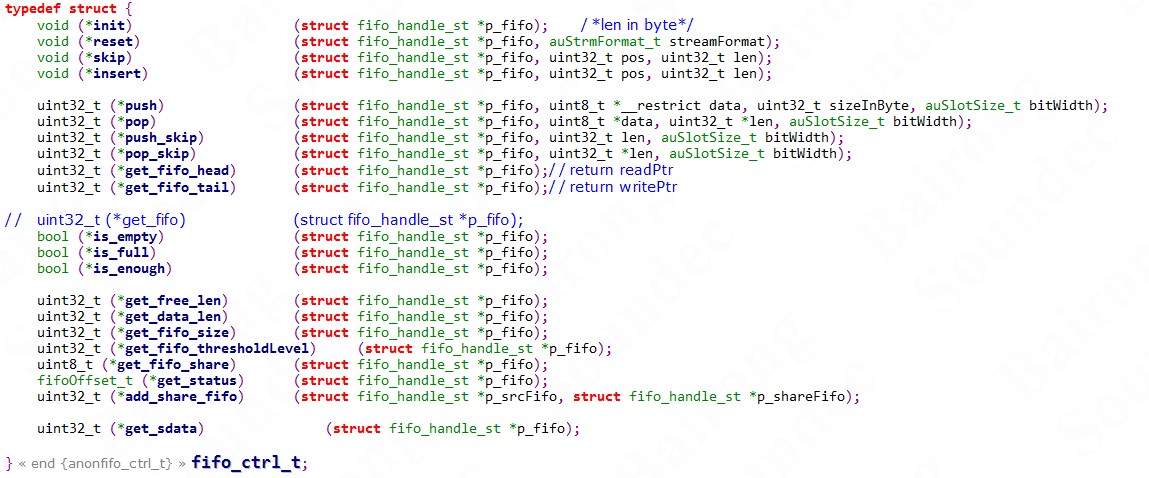
通过调用 audioManager\_init()接口，系统会根据 Session 中的通路描述，以及capture 和playback 中的格式说明，自动挂载对应的音频设备，完成 Audio manager 的初始化。通过调用audioManager\_open\_session(sessionID) 接口，实现相关通路的开启。调用audioManager\_close\_session(sessionID)接口则会关闭对应通路。

* 1. **数据流的读写处理**

经过 audioManager\_init()的调用，所有已挂载的音频设备的数据接收和发送，都将在统一的接口中进行处理：fifo\_push/fifo\_pop

* 1. **数据缓存处理和优化**

所有音频数据的缓存区的创建、数据缓存、格式转换、优化管理，通过 fifo\_manager 实现；



1. **应用层 - 定制Audio应用**

应用层提供了很多标准样例音频应用以及相应的模板文件（音频会话描述文件），开发者基本无需考虑上一节描述的中间层和底层驱动层的内部实现逻辑，也不需要考虑音频设备的初始化调用等流程，而只需要根据 架构的要求，在 模板配置文件中描述音频走向及处理格式：音频的数据来源、音频源数量、源数据的处理（注册算法处理函数）、选择输出设备，即可生成自己的定制音频应用。

1. **新增项目配置**

根据不同的硬件设计及资源，配置板级资源：boardConfig.h，在其中定义并使能项目配置 #define D\_CONFIG\_XXXX



1. **新增配置文件和音频描述文件**

为了支持客制化应用的快速开发，参考 default\_config 目录，创建配置文件和音频描述文件：

配置文件：user\_config\_xxx.h

音频描述文件：audio\_session\_desc\_xxx.c

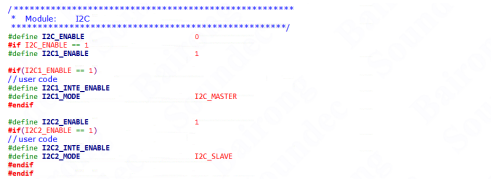
1. **修改配置文件**

以默认配置文件 user\_config\_default.h 为例，根据相应的模块说明配置相关功能。详细功能开关说明可以 参考对应的 ApplicationNote 文件。



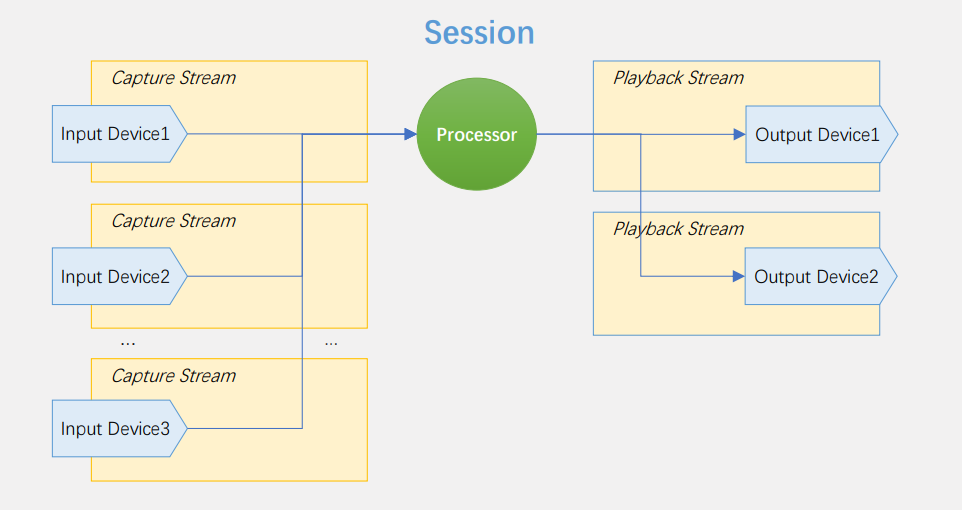






1. **修改音频通路描述文件**

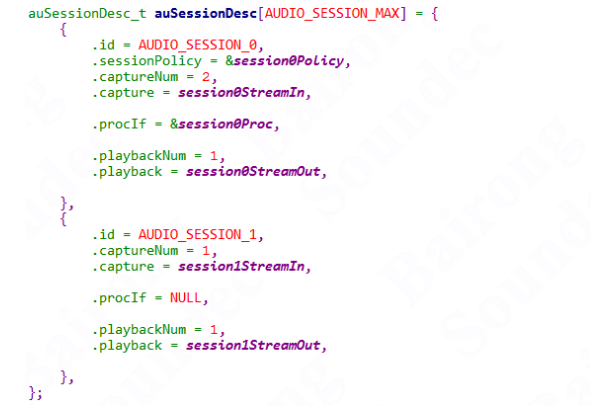
根据音频通路模型，描述每一路音频配置。每个通路均支持多路音频输入和多路音频输出，用户根据 应用需求分别在 capture 和 playback 中描述输入和输出流的信息。



1. **描述音频信号通路**

以双麦会议音箱应用为例，音频描述文件为audio\_session\_desc\_2mic\_meetingbox.c 其配 置支持两路音频通路：AUDIO\_SESSION\_0 和 AUDIO\_SESSION\_1。

* AUDIO\_SESSION\_0 支持 2 路输入（capture），1 路输出（playback），输入信号经过（session0Proc） 处理后输出；
* AUDIO\_SESSION\_1 支持 1 路输入（capture），1 路输出（playback），输入信号不需要处理，直接输 出：

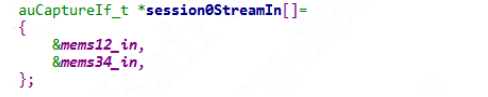


1. **配置音频信号**

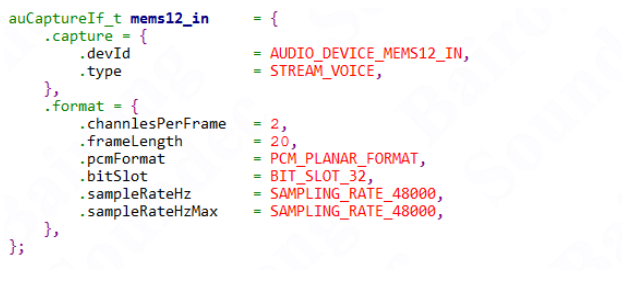
以双麦会议音箱应用为例，配置音频流的输入、处理和输出信息：

**1)音频流的输入配置——Capture**

AudioManager 使用 Capture 相关定义描述输入源信息，其中： mems12\_in 作为音箱参考信号的输入源，mems34\_in 作为两路数字麦克风的输入源;



通过 auCaptureIf\_t 配置接口，分别描述输入设备的硬件信息和音频格式，以 mems12\_in 为例：



1）Capture 输入源配置

a) devId 选择硬件设备为 ADC12（AUDIO\_DEVICE\_ MEMS12\_IN）

b) 音频流类型为 Voice

2） Format 音频格式

a) channelsPerFrame = 2 双声道

b) frameLength = 20，注意此处 frameLength 以时间 ms（毫秒）为单位，用于表示 Mems12 的数据 缓存区大小

c) pcmFormat = PCM\_PACKED\_FORMAT，描述音频流的数据格式，支持两种存放格式：

◆声道交错存储：PCM\_PACKED\_FORMAT，如 LRLR…LR

◆ 声道独立存储：PLANAR\_FORMAT，如 LLL…RRR

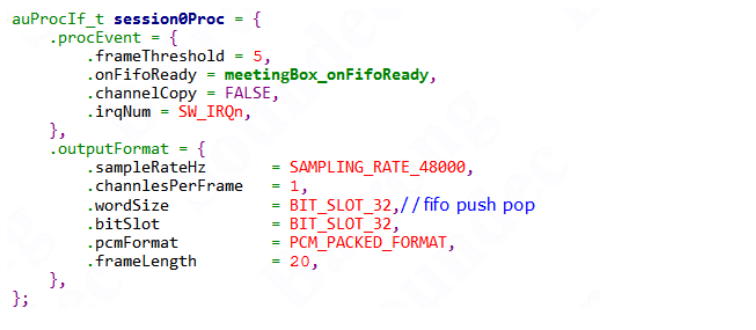
d) bitSlot：默认数据位宽

e) sampleRateHz：默认采样率，单位为 Hz f) sampleRateHzMax：最大采样率，单位为 Hz

[注意]：如果输入信号需要算法处理，音频格式根据算法需求配置

**2)音频流的处理配置——Proc**

AudioManager 使用 Proc 相关定义描述音频流的处理信息： mems12\_in 和 mems34\_in 信号送入 session0Proc 中描述的 onFifoReady 回调函数进行处理。 通过 auProcIf\_t 接口，描述当前 session 通路上的算法信息，其中： proEvent 用于描述算法对输入端的信号需求；outputFormat 用于描述算法输出格式信息。

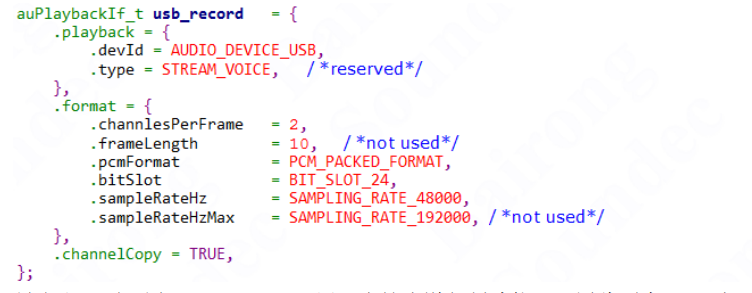


**3)音频流的输出配置——Playback**

AudioManager 使用 Playback 相关定义描述音频流的输出信息，以下描述表示 Session0 的信号通过 USB 录音通道输出。



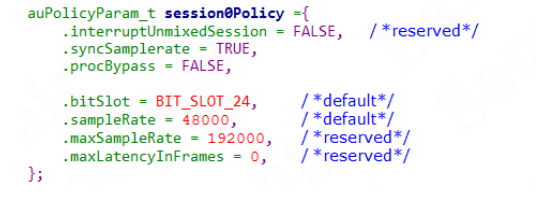
通过 auPlaybackIf\_t 配置接口，描述输出设备的硬件信息、音频格式，以usb\_record 为例，其中关于 Playback 的设备信息和音频格式描述参考 Capture：



其中配置 channelCopy：用于支持声道复制功能，配置此项为 true 表示将单声道音频源用双声道输出。

**4)配置通路的通用属性**

每个通路通过 auPolicyPatam\_t 结构体来定义和存储通路上的音频设置和处理控制，操作对整个音频 通路(Session)有效。部分成员未定义使用功能，用于以后扩展



1. **算法集成**

SDK本身可以提供无缝集成的自研算法，开发者也可以利用现有sdk提供的auProcIf\_t接口开发集成自己的算法。SDK提供了Hifi3 DSP开发所需的Cadence Nature DSP以及FPU库和头文件，开发者可以很方便的在sdk的xplorer开发环境中进行算法的开发或移植。