MI AO API

Version 2.11

© 2020 SigmaStar Technology Corp. All rights reserved.

SigmaStar Technology makes no representations or warranties including, for example but not limited to, warranties of merchantability, fitness for a particular purpose, non-infringement of any intellectual property right or the accuracy or completeness of this document, and reserves the right to make changes without further notice to any products herein to improve reliability, function or design. No responsibility is assumed by SigmaStar Technology arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others.

SigmaStar is a trademark of SigmaStar Technology Corp. Other trademarks or names herein are only for identification purposes only and owned by their respective owners.

REVISION HISTORY

Revision No.	Description	Date
2.03	• Initial release	04/12/2018
2.04	• Updated for accuracy	02/25/2019
2.05	• Added description regarding audio algorithm	03/25/2019
2.06	• Added MI_AO_SetChnParam and MI_AO_GetChnParam	03/30/2019
2.07	• Updated audio supporting sample rate	06/17/2019
2.08	• Added src gain setting API	07/04/2019
2.09	• Fixed the wrong sample rate • Updated MI_AUDIO_Frame_t structure	07/31/2019
2.10	• Updated I2S mode and Mclk • Fixed the wrong description of AGC	10/26/2019
2.11	• Prefect this document	01/02/2020

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

TABLE OF CONTENTS

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

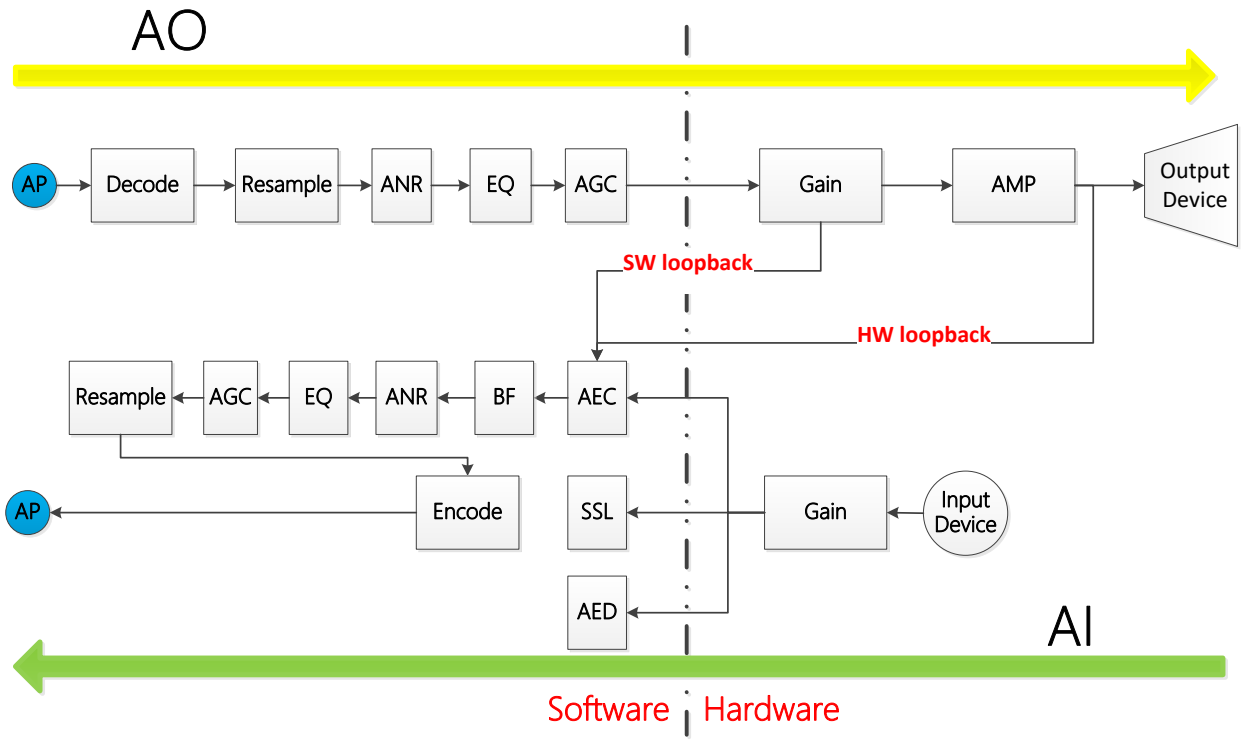
Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

1. 概述

1.1. 模块说明

音频输出（Audio Output, AO）主要实现配置及启用音频输出设备、发送音频帧数据、以及声学算法处理等功能。声学算法处理主要包括：重采样、降噪、高通滤波、均衡器、自动增益控制等。

1.2. 流程框图



Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

1.3. 关键字说明

- Device
  与其他模块的 Device 概念不同，AO 的 Device 指代的是不同的外部输出设备，如 Line out/I2S TX/HDMI 等。
- Channel
  AO 的 Channel 指代的是物理上的声道数。
- SRC
  SRC(Sample Rate Conversion)，即 resample，重采样。
- AGC
  AGC(Automatic Gain Control)，自动增益控制，用于控制输出增益。
- EQ
  EQ(Equalizer)，均衡器处理，用于对特定频段进行处理。
- ANR
  ANR(Acoustic Noise Reduction)，降噪，用于去除环境中持续存在，频率固定的噪声。
- HPF
  HPF(High-Pass Filtering)，高通滤波

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

2. API 参考

2.1. 功能模块 API

API 名	功能
MI_AO_SetPubAttr	设置 AO 设备属性
MI_AO_GetPubAttr	获取 AO 设备属性
MI_AO_Enable	启用 AO 设备
MI_AO_Disable	禁用 AO 设备
MI_AO_EnableChn	启用 AO 通道
MI_AO_DisableChn	禁用 AO 通道
MI_AO_SendFrame	发送音频帧
MI_AO_EnableReSmp	启用 AO 重采样
MI_AO_DisableReSmp	禁用 AO 重采样。
MI_AO_PauseChn	暂停 AO 通道
MI_AO_ResumeChn	恢复 AO 通道
MI_AO_ClearChnBuf	清除 AO 通道中当前的音频数据缓存
MI_AO_QueryChnStat	查询 AO 通道中当前的音频数据缓存状态
MI_AO_SetVolume	设置 AO 设备音量大小
MI_AO_GetVolume	获取 AO 设备音量大小
MI_AO_SetMute	设置 AO 设备静音状态
MI_AO_GetMute	获取 AO 设备静音状态
MI_AO_ClrPubAttr	清除 AO 设备属性
MI_AO_SetVqeAttr	设置 AO 的声音质量增强功能相关属性
MI_AO_GetVqeAttr	获取 AO 的声音质量增强功能相关属性
MI_AO_EnableVqe	使能 AO 的声音质量增强功能
MI_AO_DisableVqe	禁用 AO 的声音质量增强功能
MI_AO_SetAdecAttr	设置 AO 解码功能相关属性
MI_AO_GetAdecAttr	获取 AO 解码功能相关属性。
MI_AO_EnableAdec	使能 AO 解码功能。
MI_AO_DisableAdec	禁用 AO 解码功能。
MI_AO_SetChnParam	设置 AO 通道属性
MI_AO_GetChnParam	获取 AO 通道属性

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

API 名	功能
MI_AO_SetSrcGain	设置 AO 设备的回声参考增益

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

2.1.1 MI_AO_SetPubAttr

➢ 功能

设置 AO 设备属性。

➢ 语法

MI_S32 MI_AO_SetPubAttr(MI_AUDIO_DEV AoDevId, MI_AUDIO_Attr_t *pstAttr);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
pstAttr	AO 设备属性指针。	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

音频输出设备属性包括采样率、采样精度、工作模式、每帧的采样点数、通道数目和 I2S 的配置参数。若需要对接 Codec，这些属性应与待对接 Codec 要求一致。

- 采样率
  采样率指一秒内的采样点数，采样率越高表明失真度越小，处理的数据量也就随之增加。一般来说语音使用 8k 采样率，音频使用 32k 或以上的采样率；目前仅支持 8/11.025/12/16/22.05/24/32/44.1/48KHz 采样率。若需要对接 Codec，设置时请确认对接的 Audio Codec 是否支持所要设定的采样率。
- 采样精度
  采样精度指某个通道的采样点数据宽度。目前采样精度仅支持 16bit。
- 工作模式
  音频输入目前支持 I²S 主模式、I²S 从模式、Tdm 主模式、Tdm 从模式，但芯片支持的工作模式不尽相同。
- 每帧的采样点数
  当音频采样率较高时，建议相应地增加每帧的采样点数目。若发现采集到的声音断续，则需要增大每帧的采样点数。
- 通道数目
  通道数目指声道数，1 为单声道，2 为立体声。
- I2S 的配置参数
  I2S 的配置参数指定 I2S Mclk 的频率、I2S 传输的数据格式、I2S 使用的是 4-wire mode 还是 6-wire mode。

➢ 举例

下面的代码实现初始化 AO 设备、送一帧音频数据、反初始化 AO 设备

```
1.  MI_S32 ret;
2.  MI_AUDIO_Attr_t stAttr;
3.  MI_AUDIO_Attr_t stGetAttr;
4.  MI_AUDIO_DEV AoDevId = 0;
5.  MI_AO_CHN AoChn = 0;
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
MI_U8 u8Buf[1024];
MI_AUDIO_Frame_t stAoSendFrame;

stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_MASTER;
stAttr.u32PtNumPerFrm = 1024;
stAttr.u32ChnCnt = 1;

MI_SYS_Init();

/* set ao public attr*/
ret = MI_AO_SetPubAttr(AoDevId, &stAttr);
if(MI_SUCCESS != ret)
{
    printf("set ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

ret = MI_AO_GetPubAttr(AoDevId, &stGetAttr);
if(MI_SUCCESS != ret)
{
    printf("get ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

/* enable ao device*/
ret = MI_AO_Enable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("enable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

ret = MI_AO_EnableChn(AoDevId, AoChn);
if (MI_SUCCESS != ret)
{
    printf("enable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

memset(&stAoSendFrame, 0x0, sizeof(MI_AUDIO_Frame_t));
stAoSendFrame.u32Len = 1024;
stAoSendFrame.apVirAddr[0] = u8Buf;
stAoSendFrame.apVirAddr[1] = NULL;

do{
    ret = MI_AO_SendFrame(AoDevId, AoChn, &stAoSendFrame, -1);
}while(ret == MI_AO_ERR_NOBUF);

ret = MI_AO_DisableChn(AoDevId, AoChn);
if (MI_SUCCESS != ret)
{
    printf("disable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* disable ao device */
ret = MI_AO_Disable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("disable ao dev %d err:0x%x\n", AoDevId, ret);
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
69.     return ret;
70. }
71.
72. MI_SYS_Exit();
```

2.1.2 MI_AO_GetPubAttr

➢ 功能

获取 AO 设备属性。

➢ 语法

MI_S32 MI_AO_GetPubAttr(MI_AUDIO_DEV AoDevId, MI_AUDIO_Attr_t *pstAttr);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
pstAttr	AO 设备属性指针。	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

- 获取的属性为前一次配置的属性。
- 如果从来没有配置过属性，则返回失败。

➢ 举例

请参考 MI_AO_SetPubAttr 举例部分。

2.1.3 MI_AO_Enable

➢ 功能

启用 AO 设备。

➢ 语法

MI_S32 MI_AO_Enable(MI_AUDIO_DEV AoDevId);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 返回值

返回值 { 0 成功。
非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

- 必须在启用前配置 AO 设备属性，否则返回属性未配置错误
- 如果 AO 设备已经处于启用状态，则直接返回成功。

➢ 举例

请参考 MI_AO_SetPubAttr 的举例部分。

2.1.4 MI_AO_Disable

➢ 功能

禁用 AO 设备。

➢ 语法

MI_S32 MI_AO_Disable(MI_AUDIO_DEV AoDevId);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入

➢ 返回值

返回值 { 0 成功。
非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

- 如果 AO 设备已经处于禁用状态，则直接返回成功
- 禁用 AO 设备前必须先禁用该设备下已启用的所有 AO 通道

➢ 举例

请参考 MI_AO_SetPubAttr 的举例部分。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

2.1.5 MI_AO_EnableChn

- 功能

  启用 AO 通道

- 语法

  MI_S32 MI_AO_EnableChn(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

- 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 仅支持使用通道0。	输入

- 返回值

  返回值：
  - 0　成功。
  - 非 0　失败，参照错误码。

- 依赖
  - 头文件: mi_ao.h
  - 库文件: libmi_ao.a/libmi_ao.so

※ 注意
- 启用 AO 通道前，必须先启用其所属的 AO 设备，否则返回设备未启动的错误码

- 举例

  请参考 MI_AO_SetPubAttr 的举例部分。

2.1.6 MI_AO_DisableChn

- 功能

  禁用 AI 通道

- 语法

  MI_S32 MI_AO_DisableChn(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

- 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 仅支持使用通道0。	输入

- 返回值
  - 0　成功。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

返回值

非 0　失败，参照错误码。

➢ 依赖

- 头文件:　mi_ao.h
- 库文件:　libmi_ao.a/libmi_ao.so

※ 注意

在禁用 AO 通道前，需要将该通道上使能的音频算法禁用。

➢ 举例

请参考 MI_AO_SetPubAttr 的举例部分。

2.1.7 MI_AO_SendFrame

➢ 功能

发送 AO 音频帧。

➢ 语法

MI_S32 MI_AO_SendFrame(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn, MI_AUDIO_Frame_t *pstData, MI_S32 s32MilliSec);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 仅支持使用通道0。	输入
pstData	音频帧结构体指针。	输入
s32MilliSec	设置数据的超时时间 -1 表示阻塞模式，无数据时一直等待； 0 表示非阻塞模式，无数据时则报错返回； >0表示阻塞s32MilliSec毫秒，超时则报错返回。	输入

➢ 返回值

返回值
- 0　成功。
- 非 0　失败，参照错误码。

➢ 依赖

- 头文件:　mi_ao.h
- 库文件:　libmi_ao.a/libmi_ao.so

※ 注意

- s32MilliSec 的值必须大于等于-1，等于-1 时采用阻塞模式获取数据，等于 0 时采用非阻塞模式获取数据，大于 0 时，阻塞 s32MilliSec 毫秒后，没有数据则返回超时并报错
- 调用该接口发送音频帧到 AO 输出时，必须先使能对应的 AO 通道。

Copyright © 2020 SigmaStar Technology Corp.　All rights reserved.

➢ 举例

请参考 MI_AO_SetPubAttr 的举例部分。

2.1.8 MI_AO_EnableReSmp

➢ 功能

启用 AO 重采样

➢ 语法

MI_S32 MI_AO_EnableReSmp(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn, MI_AUDIO_SampleRate_e eInSampleRate);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输入通道号。 仅支持使用通道0。	输入
eInSampleRate	音频重采样的输入采样率。	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so libSRC_LINUX.so

※ 注意

- 应该在启用 AO 通道之后，调用此接口启用重采样功能。

➢ 举例

下面的代码实现开启和关闭重采样，输入数据为 16K，重采样至 8K 播放。

```
MI_S32 ret;
MI_AUDIO_Attr_t stAttr;
MI_AUDIO_Attr_t stGetAttr;
MI_AUDIO_DEV AoDevId = 0;
MI_AO_CHN AoChn = 0;
MI_U8 u8Buf[1024];
MI_AUDIO_Frame_t stAoSendFrame;
MI_AUDIO_SampleRate_e eInSampleRate = E_MI_AUDIO_SAMPLE_RATE_16000;

stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_MASTER;
stAttr.u32PtNumPerFrm = 1024;
stAttr.u32ChnCnt = 1;
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
16.
17. MI_SYS_Init();
18.
19. /* set ao public attr*/
20. ret = MI_AO_SetPubAttr(AoDevId, &stAttr);
21. if(MI_SUCCESS != ret)
22. {
23.     printf("set ao %d attr err:0x%x\n", AoDevId, ret);
24.     return ret;
25. }
26.
27. ret = MI_AO_GetPubAttr(AoDevId, &stGetAttr);
28. if(MI_SUCCESS != ret)
29. {
30.     printf("get ao %d attr err:0x%x\n", AoDevId, ret);
31.     return ret;
32. }
33.
34. /* enable ao device*/
35. ret = MI_AO_Enable(AoDevId);
36. if(MI_SUCCESS != ret)
37. {
38.     printf("enable ao dev %d err:0x%x\n", AoDevId, ret);
39.     return ret;
40. }
41.
42. ret = MI_AO_EnableChn(AoDevId, AoChn);
43. if (MI_SUCCESS != ret)
44. {
45.     printf("enable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
46.     return ret;
47. }
48.
49. ret = MI_AO_EnableReSmp(AoDevId, AoChn, eInSampleRate);
50. if (MI_SUCCESS != ret)
51. {
52.     printf("enable resample ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
53.     return ret;
54. }
55.
56. memset(&stAoSendFrame, 0x0, sizeof(MI_AUDIO_Frame_t));
57. stAoSendFrame.u32Len = 1024;
58. stAoSendFrame.apVirAddr[0] = u8Buf;
59. stAoSendFrame.apVirAddr[1] = NULL;
60.
61. do{
62.     ret = MI_AO_SendFrame(AoDevId, AoChn, &stAoSendFrame, -1);
63. }while(ret == MI_AO_ERR_NOBUF);
64.
65. ret = MI_AO_DisableReSmp(AoDevId, AoChn);
66. if (MI_SUCCESS != ret)
67. {
68.     printf("disable resample ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
69.     return ret;
70. }
71.
72. ret = MI_AO_DisableChn(AoDevId, AoChn);
73. if (MI_SUCCESS != ret)
74. {
75.     printf("disable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
76.     return ret;
77. }
78.
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
/* disable ao device */
ret = MI_AO_Disable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("disable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

MI_SYS_Exit();
```

2.1.9 MI_AO_DisableReSmp

➢ 功能

禁用 AO 重采样

➢ 语法

MI_S32 MI_AO_DisableReSmp(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输入通道号。 仅支持使用通道0。	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so libSRC_LINUX.so

※ 注意

- 不再使用 AO 重采样功能的话，应该调用此接口将其禁用。

➢ 举例

请参考 MI_AO_EnableReSmp 举例部分。

2.1.10 MI_AO_PauseChn

➢ 功能

暂停 AO 通道

➢ 语法

MI_S32 MI_AO_PauseChn(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输入通道号。 仅支持使用通道0。	输入

➢ 返回值

返回值
- 0　　成功。
- 非 0　失败，参照错误码。

➢ 依赖

- 头文件:　mi_ao.h
- 库文件:　libmi_ao.a/libmi_ao.so

※ 注意

- AO 通道为禁用状态时，不允许调用此接口暂停 AO 通道。

➢ 举例

下面的代码实现暂停和恢复 AO 播放。

```
1.  MI_S32 ret;
2.  MI_AUDIO_Attr_t stAttr;
3.  MI_AUDIO_Attr_t stGetAttr;
4.  MI_AUDIO_DEV AoDevId = 0;
5.  MI_AO_CHN AoChn = 0;
6.  MI_U8 u8Buf[1024];
7.  MI_AUDIO_Frame_t stAoSendFrame;
8.
9.  stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
10. stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
11. stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
12. stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_MASTER;
13. stAttr.u32PtNumPerFrm = 1024;
14. stAttr.u32ChnCnt = 1;
15.
16. MI_SYS_Init();
17.
18. /* set ao public attr*/
19. ret = MI_AO_SetPubAttr(AoDevId, &stAttr);
20. if(MI_SUCCESS != ret)
21. {
22.     printf("set ao %d attr err:0x%x\n", AoDevId, ret);
23.     return ret;
24. }
25.
26. ret = MI_AO_GetPubAttr(AoDevId, &stGetAttr);
27. if(MI_SUCCESS != ret)
28. {
29.     printf("get ao %d attr err:0x%x\n", AoDevId, ret);
30.     return ret;
31. }
32.
33. /* enable ao device*/
34. ret = MI_AO_Enable(AoDevId);
35. if(MI_SUCCESS != ret)
36. {
```


Copyright © 2020 SigmaStar Technology Corp.　All rights reserved.

```
37.     printf("enable ao dev %d err:0x%x\n", AoDevId, ret);
38.     return ret;
39. }
40.
41. ret = MI_AO_EnableChn(AoDevId, AoChn);
42. if (MI_SUCCESS != ret)
43. {
44.     printf("enable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
45.     return ret;
46. }
47.
48. memset(&stAoSendFrame, 0x0, sizeof(MI_AUDIO_Frame_t));
49. stAoSendFrame.u32Len = 1024;
50. stAoSendFrame.apVirAddr[0] = u8Buf;
51. stAoSendFrame.apVirAddr[1] = NULL;
52.
53. do{
54.     ret = MI_AO_SendFrame(AoDevId, AoChn, &stAoSendFrame, -1);
55. }while(ret == MI_AO_ERR_NOBUF);
56.
57. ret = MI_AO_PauseChn(AoDevId, AoChn);
58. if (MI_SUCCESS != ret)
59. {
60.     printf("pause ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
61.     return ret;
62. }
63.
64. ret = MI_AO_ResumeChn(AoDevId, AoChn);
65. if (MI_SUCCESS != ret)
66. {
67.     printf("resume ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
68.     return ret;
69. }
70.
71. ret = MI_AO_DisableChn(AoDevId, AoChn);
72. if (MI_SUCCESS != ret)
73. {
74.     printf("disable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
75.     return ret;
76. }
77.
78. /* disable ao device */
79. ret = MI_AO_Disable(AoDevId);
80. if(MI_SUCCESS != ret)
81. {
82.     printf("disable ao dev %d err:0x%x\n", AoDevId, ret);
83.     return ret;
84. }
85.
86. MI_SYS_Exit();
```

2.1.11 MI_AO_ResumeChn

- 功能

  恢复 AO 通道。

- 语法

  MI_S32 MI_AO_ResumeChn (MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输入通道号。 仅支持使用通道0。	输入

➢ 返回值

返回值
- 0　成功。
- 非 0　失败，参照错误码。

➢ 依赖

- 头文件:　mi_ao.h
- 库文件:　libmi_ao.a/libmi_ao.so

※ 注意

- AO 通道暂停后可以通过调用此接口重新恢复。
- AO 通道为暂停状态或使能状态下，调用此接口返回成功；否则调用将返回错误。

➢ 举例

请参考 MI_AO_PauseChn 举例部分。

2.1.12 MI_AO_ClearChnBuf

➢ 功能

清除 AO 通道中当前的音频数据缓存。

➢ 语法

MI_S32 MI_AO_ClearChnBuf (MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输入通道号。 仅支持使用通道0。	输入

➢ 返回值

返回值
- 0　成功。
- 非 0　失败，参照错误码。

➢ 依赖

- 头文件:　mi_ao.h
- 库文件:　libmi_ao.a/libmi_ao.so

Copyright © 2020 SigmaStar Technology Corp.　All rights reserved.

※ 注意

- AO 通道成功启用后再调用此接口。

➢ 举例

下面的代码实现获取 AO 通道缓存情况并清除缓存。

```
MI_S32 ret;
MI_AUDIO_Attr_t stAttr;
MI_AUDIO_Attr_t stGetAttr;
MI_AUDIO_DEV AoDevId = 0;
MI_AO_CHN AoChn = 0;
MI_U8 u8Buf[1024];
MI_AUDIO_Frame_t stAoSendFrame;
MI_AO_ChnState_t stStatus;

stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_MASTER;
stAttr.u32PtNumPerFrm = 1024;
stAttr.u32ChnCnt = 1;

MI_SYS_Init();

/* set ao public attr*/
ret = MI_AO_SetPubAttr(AoDevId, &stAttr);
if(MI_SUCCESS != ret)
{
    printf("set ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

ret = MI_AO_GetPubAttr(AoDevId, &stGetAttr);
if(MI_SUCCESS != ret)
{
    printf("get ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

/* enable ao device*/
ret = MI_AO_Enable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("enable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

/* enable ao chn */
ret = MI_AO_EnableChn(AoDevId, AoChn);
if (MI_SUCCESS != ret)
{
    printf("enable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* send frame */
memset(&stAoSendFrame, 0x0, sizeof(MI_AUDIO_Frame_t));
stAoSendFrame.u32Len = 1024;
stAoSendFrame.apVirAddr[0] = u8Buf;
stAoSendFrame.apVirAddr[1] = NULL;

```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
do{
    ret = MI_AO_SendFrame(AoDevId, AoChn, &stAoSendFrame, -1);
}while(ret == MI_AO_ERR_NOBUF);

/* get chn stat */
ret = MI_AO_QueryChnStat(AoDevId, AoChn, &stStatus);
if (MI_SUCCESS != ret)
{
    printf("query chn status ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* clear chn buf */
ret = MI_AO_ClearChnBuf(AoDevId, AoChn);
if (MI_SUCCESS != ret)
{
    printf("clear chn buf ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* disable ao chn */
ret = MI_AO_DisableChn(AoDevId, AoChn);
if (MI_SUCCESS != ret)
{
    printf("disable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* disable ao device */
ret = MI_AO_Disable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("disable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

MI_SYS_Exit();
```

2.1.13 MI_AO_QueryChnStat

- 功能

  查询 AO 通道中当前的音频数据缓存状态。

- 语法

  MI_S32 MI_AO_QueryChnStat(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn, MI_AO_ChnState_t *pstStatus);

- 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 仅支持使用通道0。	输入
pstStatus	缓存状态结构体指针。	输出

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 返回值

返回值 { 0 成功。
非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

- AO 通道成功启用后再调用此接口。

➢ 举例

请参考 MI_AO_ClearChnBuf 举例部分。

2.1.14 MI_AO_SetVolume

➢ 功能

设置 AO 设备音量大小。

➢ 语法

MI_S32 MI_AO_SetVolume(MI_AUDIO_DEV AoDevId, MI_S32 s32VolumeDb);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
s32VolumeDb	音频设备音量大小（-60 – 30，以dB为单位）。	输入

➢ 返回值

返回值 { 0 成功。
非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

- AO 设备成功启用后再调用此接口。

➢ 举例

下面的代码实现设置或获取 AO 的音量参数

```
1.  MI_S32 ret;
2.  MI_AUDIO_Attr_t stAttr;
3.  MI_AUDIO_Attr_t stGetAttr;
4.  MI_AUDIO_DEV AoDevId = 0;
5.  MI_AO_CHN AoChn = 0;
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
6.  MI_U8 u8Buf[1024];
7.  MI_AUDIO_Frame_t stAoSendFrame;
8.  MI_S32 s32VolumeDb = 0;
9.
10. stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
11. stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
12. stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
13. stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_MASTER;
14. stAttr.u32PtNumPerFrm = 1024;
15. stAttr.u32ChnCnt = 1;
16.
17. MI_SYS_Init();
18.
19. /* set ao public attr*/
20. ret = MI_AO_SetPubAttr(AoDevId, &stAttr);
21. if(MI_SUCCESS != ret)
22. {
23.     printf("set ao %d attr err:0x%x\n", AoDevId, ret);
24.     return ret;
25. }
26.
27. ret = MI_AO_GetPubAttr(AoDevId, &stGetAttr);
28. if(MI_SUCCESS != ret)
29. {
30.     printf("get ao %d attr err:0x%x\n", AoDevId, ret);
31.     return ret;
32. }
33.
34. /* enable ao device*/
35. ret = MI_AO_Enable(AoDevId);
36. if(MI_SUCCESS != ret)
37. {
38.     printf("enable ao dev %d err:0x%x\n", AoDevId, ret);
39.     return ret;
40. }
41.
42. /* enable ao chn */
43. ret = MI_AO_EnableChn(AoDevId, AoChn);
44. if (MI_SUCCESS != ret)
45. {
46.     printf("enable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
47.     return ret;
48. }
49.
50. /* set ao volume */
51. ret = MI_S32 MI_AO_SetVolume(AoDevId, s32VolumeDb);
52. if (MI_SUCCESS != ret)
53. {
54.     printf("set volume ao dev %d err:0x%x\n", AoDevId, ret);
55.     return ret;
56. }
57.
58. /* get ao volume */
59. ret = MI_S32 MI_AO_GetVolume(AoDevId, &s32VolumeDb);
60. if (MI_SUCCESS != ret)
61. {
62.     printf("get volume ao dev %d err:0x%x\n", AoDevId, ret);
63.     return ret;
64. }
65.
66.
67.
68. /* send frame */
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
memset(&stAoSendFrame, 0x0, sizeof(MI_AUDIO_Frame_t));
stAoSendFrame.u32Len = 1024;
stAoSendFrame.apVirAddr[0] = u8Buf;
stAoSendFrame.apVirAddr[1] = NULL;

do{
    ret = MI_AO_SendFrame(AoDevId, AoChn, &stAoSendFrame, -1);
}while(ret == MI_AO_ERR_NOBUF);


/* disable ao chn */
ret = MI_AO_DisableChn(AoDevId, AoChn);
if (MI_SUCCESS != ret)
{
    printf("disable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* disable ao device */
ret = MI_AO_Disable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("disable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

MI_SYS_Exit();
```

2.1.15 MI_AO_GetVolume

➢ 功能

获取 AO 设备音量大小。

➢ 语法

MI_S32 MI_AO_GetVolume(MI_AUDIO_DEV AoDevId, MI_S32 *ps32VolumeDb);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
ps32VolumeDb	音频设备音量大小指针	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

- AO 设备成功启用后再调用此接口。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 举例

请参考 MI_AO_SetVolume 的举例部分。

2.1.16 MI_AO_SetMute

➢ 功能

设置 AO 设备静音状态。

➢ 语法

MI_S32 MI_AO_SetMute(MI_AUDIO_DEV AoDevId, MI_BOOL bEnable);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
bEnable	音频设备是否启用静音。 TRUE：启用静音功能； FALSE：关闭静音功能。	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

- AO 设备成功启用后再调用此接口。

➢ 举例

下面的代码实现获取和设置静音状态。

```
1.  MI_S32 ret;
2.  MI_AUDIO_Attr_t stAttr;
3.  MI_AUDIO_Attr_t stGetAttr;
4.  MI_AUDIO_DEV AoDevId = 0;
5.  MI_AO_CHN AoChn = 0;
6.  MI_U8 u8Buf[1024];
7.  MI_AUDIO_Frame_t stAoSendFrame;
8.  MI_BOOL bMute = TRUE;
9.
10. stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
11. stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
12. stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
13. stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_MASTER;
14. stAttr.u32PtNumPerFrm = 1024;
15. stAttr.u32ChnCnt = 1;
16.
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
MI_SYS_Init();

/* set ao public attr*/
ret = MI_AO_SetPubAttr(AoDevId, &stAttr);
if(MI_SUCCESS != ret)
{
    printf("set ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

ret = MI_AO_GetPubAttr(AoDevId, &stGetAttr);
if(MI_SUCCESS != ret)
{
    printf("get ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

/* enable ao device*/
ret = MI_AO_Enable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("enable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

/* enable ao chn */
ret = MI_AO_EnableChn(AoDevId, AoChn);
if (MI_SUCCESS != ret)
{
    printf("enable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* set ao mute */
ret = MI_AO_SetMute(AoDevId, bMute);
if (MI_SUCCESS != ret)
{
    printf("mute ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

/* get ao mute status */
ret = MI_AO_GetMute(AoDevId, &bMute);
if (MI_SUCCESS != ret)
{
    printf("get mute status ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

/* send frame */
memset(&stAoSendFrame, 0x0, sizeof(MI_AUDIO_Frame_t));
stAoSendFrame.u32Len = 1024;
stAoSendFrame.apVirAddr[0] = u8Buf;
stAoSendFrame.apVirAddr[1] = NULL;

do{
    ret = MI_AO_SendFrame(AoDevId, AoChn, &stAoSendFrame, -1);
}while(ret == MI_AO_ERR_NOBUF);


/* disable ao chn */
ret = MI_AO_DisableChn(AoDevId, AoChn);
if (MI_SUCCESS != ret)
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
{
    printf("disable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* disable ao device */
ret = MI_AO_Disable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("disable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

MI_SYS_Exit();
```

2.1.17 MI_AO_GetMute

➢ 功能

获取 AO 设备静音状态。

➢ 语法

MI_S32 MI_AO_GetMute(MI_AUDIO_DEV AoDevId, MI_BOOL *pbEnable);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
pbEnable	音频设备静音状态指针。	输出

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

- AO 设备成功启用后再调用此接口。

➢ 举例

请参考 MI_AO_SetMute 举例部分

2.1.18 MI_AO_ClrPubAttr

➢ 功能

清除 AO 设备属性。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 语法

MI_S32 MI_AO_ClrPubAttr(MI_AUDIO_DEV AoDevId);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

- 清除设备属性前，需要先停止设备。

➢ 举例

下面的代码实现设置和清除 AO 设备的属性。

```
1.  MI_S32 ret;
2.  MI_AUDIO_Attr_t stAttr;
3.  MI_AUDIO_DEV AoDevId = 0;
4.
5.  stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
6.  stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
7.  stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
8.  stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_MASTER;
9.  stAttr.u32PtNumPerFrm = 1024;
10. stAttr.u32ChnCnt = 1;
11.
12. MI_SYS_Init();
13.
14. /* set ao public attr*/
15. ret = MI_AO_SetPubAttr(AoDevId, &stAttr);
16. if(MI_SUCCESS != ret)
17. {
18.     printf("set ao %d attr err:0x%x\n", AoDevId, ret);
19.     return ret;
20. }
21.
22. /* clear ao public attr */
23. ret = MI_AO_ClrPubAttr(AoDevId);
24. if (MI_SUCCESS != ret)
25. {
26.     printf("clear ao %d attr err:0x%x\n", AoDevId, ret);
27.     return ret;
28. }
29.
30. MI_SYS_Exit();
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

2.1.19 MI_AO_SetVqeAttr

➢ 功能

设置 AO 的声音质量增强功能相关属性。

➢ 语法

MI_S32 MI_AO_SetVqeAttr(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn, MI_AO_VqeConfig_t *pstVqeConfig);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 仅支持使用通道0。	输入
pstVqeConfig	音频输出声音质量增强配置结构体指针	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so libAPC_LINUX.so

※ 注意

- 启用声音质量增强功能前必须先设置相对应 AO 通道的声音质量增强功能相关属性。
- 设置 AO 的声音质量增强功能相关属性前，必须先使能对应的 AO 通道。
- 相同 AO 信道的声音质量增强功能不支持动态设置属性，重新设置 AO 通道的声音质量增强功能相关属性时，需要先关闭 AO 通道的声音质量功能，再设置 AO 通道的声音质量增强功能相关属性。
- 在设置声音质量增强功能属性时，可通过配置相应的声音质量增强功能属性来选择使能其中的部分功能。
- Vqe 所有算法均支持 8K/16K，仅 ANR/AGC/EQ 支持 48K

➢ 举例

下面的代码实现设置和使能声音质量增强功能。

```
1.  MI_S32 ret;
2.  MI_AUDIO_Attr_t stAttr;
3.  MI_AUDIO_Attr_t stGetAttr;
4.  MI_AUDIO_DEV AoDevId = 0;
5.  MI_AO_CHN AoChn = 0;
6.  MI_U8 u8Buf[1024];
7.  MI_AUDIO_Frame_t stAoSendFrame;.
8.  MI_AO_VqeConfig_t stAoSetVqeConfig, stAoGetVqeConfig;
9.
10. MI_AUDIO_HpfConfig_t stHpfCfg = {
11.     .eMode = E_MI_AUDIO_ALGORITHM_MODE_USER,
12.     .eHpfFreq = E_MI_AUDIO_HPF_FREQ_150,
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
};

MI_AUDIO_AnrConfig_t stAnrCfg = {
    .eMode = E_MI_AUDIO_ALGORITHM_MODE_MUSIC,
    .u32NrIntensity = 15,
    .u32NrSmoothLevel = 10,
    .eNrSpeed = E_MI_AUDIO_NR_SPEED_MID,
};

MI_AUDIO_AgcConfig_t stAgcCfg = {
    .eMode = E_MI_AUDIO_ALGORITHM_MODE_USER,
    .s32NoiseGateDb = -60,
    .s32TargetLevelDb = -3,
    .stAgcGainInfo = {
        .s32GainInit = 0,
        .s32GainMax = 20,
        .s32GainMin = 0,
    },
    .u32AttackTime = 1,
    .s16Compression_ratio_input = {-80, -60, -40, -25, 0},
    .s16Compression_ratio_output = {-80, -30, -15, -10, -3},
    .u32DropGainMax = 12,
    .u32NoiseGateAttenuationDb = 0,
    .u32ReleaseTime = 3,
};

MI_AUDIO_EqConfig_t stEqCfg = {
    .eMode = E_MI_AUDIO_ALGORITHM_MODE_DEFAULT,
    .s16EqGainDb = {[0 ... 128] = 3},
};

stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_MASTER;
stAttr.u32PtNumPerFrm = 1024;
stAttr.u32ChnCnt = 1;

MI_SYS_Init();

/* set ao public attr*/
ret = MI_AO_SetPubAttr(AoDevId, &stAttr);
if(MI_SUCCESS != ret)
{
    printf("set ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

/* get ao public attr */
ret = MI_AO_GetPubAttr(AoDevId, &stGetAttr);
if(MI_SUCCESS != ret)
{
    printf("get ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

/* enable ao device*/
ret = MI_AO_Enable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("enable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
76.
77. /* enable ao chn */
78. ret = MI_AO_EnableChn(AoDevId, AoChn);
79. if (MI_SUCCESS != ret)
80. {
81.     printf("enable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
82.     return ret;
83. }
84.
85. stAoSetVqeConfig.bAgcOpen = TRUE;
86. stAoSetVqeConfig.bAnrOpen = TRUE;
87. stAoSetVqeConfig.bEqOpen = TRUE;
88. stAoSetVqeConfig.bHpfOpen = TRUE;
89. stAoSetVqeConfig.s32FrameSample = 128;
90. stAoSetVqeConfig.s32WorkSampleRate = E_MI_AUDIO_SAMPLE_RATE_8000;
91. memcpy(&stAoSetVqeConfig.stAgcCfg, &stAgcCfg, sizeof(MI_AUDIO_AgcConfig_t));
92. memcpy(&stAoSetVqeConfig.stAnrCfg, &stAnrCfg, sizeof(MI_AUDIO_AnrConfig_t));
93. memcpy(&stAoSetVqeConfig.stEqCfg, &stEqCfg, sizeof(MI_AUDIO_EqConfig_t));
94. memcpy(&stAoSetVqeConfig.stHpfCfg, &stHpfCfg, sizeof(MI_AUDIO_HpfConfig_t));
95.
96. /* set vqe attr */
97. ret = MI_AO_SetVqeAttr(AoDevId, AoChn, &stAoSetVqeConfig);
98. if (MI_SUCCESS != s32Ret)
99. {
100.     printf("set vqe attr ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
101.     return ret;
102. }
103.
104. /* get vqe attr */
105. ret = MI_AO_GetVqeAttr(AoDevId, AoChn, &stAoGetVqeConfig);
106. if (MI_SUCCESS != s32Ret)
107. {
108.     printf("get vqe attr ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
109.     return ret;
110. }
111.
112. /* enable vqe attr */
113. ret = MI_AO_EnableVqe(AoDevId, AoChn);
114. if (MI_SUCCESS != s32Ret)
115. {
116.     printf("enable vqe ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
117.     return ret;
118. }
119.
120. /* send frame */
121. memset(&stAoSendFrame, 0x0, sizeof(MI_AUDIO_Frame_t));
122. stAoSendFrame.u32Len = 1024;
123. stAoSendFrame.apVirAddr[0] = u8Buf;
124. stAoSendFrame.apVirAddr[1] = NULL;
125.
126. do{
127.     ret = MI_AO_SendFrame(AoDevId, AoChn, &stAoSendFrame, -1);
128. }while(ret == MI_AO_ERR_NOBUF);
129.
130. /* disable vqe attr */
131. ret = MI_AO_DisableVqe(AoDevId, AoChn);
132. if (MI_SUCCESS != s32Ret)
133. {
134.     printf("disable vqe ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
135.     return ret;
136. }
137.
138. /* disable ao chn */
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
ret = MI_AO_DisableChn(AoDevId, AoChn);
if (MI_SUCCESS != ret)
{
    printf("disable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* disable ao device */
ret = MI_AO_Disable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("disable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

MI_SYS_Exit();
```

2.1.20 MI_AO_GetVqeAttr

➢ 功能

获取 AO 的声音质量增强功能相关属性。

➢ 语法

MI_S32 MI_AO_GetVqeAttr(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn, MI_AO_VqeConfig_t *pstVqeConfig);

➢ 形参

参数名称	描述	输入/输出
AiDevId	音频设备号	输入
AiChn	音频输入通道号。 仅支持使用通道0。	输入
pstVqeConfig	音频输出声音质量增强配置结构体指针	输出

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so libAPC_LINUX.so

※ 注意

- 获取声音质量增强功能相关属性前必须先设置相对应 AO 通道的声音质量增强功能相关属性

➢ 举例

请参考 MI_AO_SetVqeAttr 举例部分。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

2.1.21 MI_AO_EnableVqe

➢ 功能

使能 AO 的声音质量增强功能。

➢ 语法

MI_S32 MI_AO_EnableVqe(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 仅支持使用通道0。	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so libAPC_LINUX.so

※ 注意

- 启用声音质量增强功能前必须先启用相对应的 AO 通道。
- 多次使能相同 AO 通道的声音质量增强功能时，返回成功。
- 禁用 AO 通道后，如果重新启用 AO 通道，并使用声音质量增强功能，需调用此接口重新启用声音质量增强功能。
- Vqe 支持 8K 16K

➢ 举例

请参考 MI_AO_SetVqeAttr 举例部分。

2.1.22 MI_AO_DisableVqe

➢ 功能

禁用 AO 的声音质量增强功能。

➢ 语法

MI_S32 MI_AO_DisableVqe(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 仅支持使用通道0。	输入

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 返回值

返回值 { 0 成功。; 非 0 失败，参照错误码。}

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so libAPC_LINUX.so

※ 注意

- 不再使用 AO 声音质量增强功能时，应该调用此接口将其禁用。
- 多次禁用相同 AO 通道的声音质量增强功能，返回成功。

➢ 举例

请参考 MI_AO_SetVqeAttr 举例部分。

2.1.23 MI_AO_SetAdecAttr

➢ 功能

设置 AO 解码功能相关属性。

➢ 语法

MI_S32 MI_AO_SetAdecAttr (MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn, MI_AO_AdecConfig_t *pstAdecConfig);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 仅支持使用通道0。	输入
pstAdecConfig	音频解码配置结构体指针	输入

➢ 返回值

返回值 { 0 成功。; 非 0 失败，参照错误码。}

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so libg711.a libg726.a

※ 注意

设置 AO 的解码功能相关属性前，必须先使能对应的 AO 通道。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 举例

下面的代码实现解码功能的设置和使能。

```
MI_S32 ret;
MI_AUDIO_Attr_t stAttr;
MI_AUDIO_Attr_t stGetAttr;
MI_AUDIO_DEV AoDevId = 0;
MI_AO_CHN AoChn = 0;
MI_U8 u8Buf[1024];
MI_AUDIO_Frame_t stAoSendFrame;
MI_AO_AdecConfig_t stAoSetAdecConfig, stAoGetAdecConfig;


stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_MASTER;
stAttr.u32PtNumPerFrm = 1024;
stAttr.u32ChnCnt = 1;

MI_SYS_Init();

/* set ao public attr*/
ret = MI_AO_SetPubAttr(AoDevId, &stAttr);
if(MI_SUCCESS != ret)
{
    printf("set ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

/* get ao public attr */
ret = MI_AO_GetPubAttr(AoDevId, &stGetAttr);
if(MI_SUCCESS != ret)
{
    printf("get ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

/* enable ao device*/
ret = MI_AO_Enable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("enable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

/* enable ao chn */
ret = MI_AO_EnableChn(AoDevId, AoChn);
if (MI_SUCCESS != ret)
{
    printf("enable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

memset(&stAoSetAdecConfig, 0x0, sizeof(MI_AO_AdecConfig_t));
stAoSetAdecConfig.eAdecType = E_MI_AUDIO_ADEC_TYPE_G711A;
stAoSetAdecConfig.stAdecG711Cfg.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAoSetAdecConfig.stAdecG711Cfg.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;

/* set adec attr */
ret = MI_AO_SetAdecAttr(AoDevId, AoChn, &stAoSetAdecConfig);
if (MI_SUCCESS != s32Ret)
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
{
    printf("set adec attr ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* get adec attr */
ret = MI_AO_GetAdecAttr(AoDevId, AoChn, &stAoGetAdecConfig);
if (MI_SUCCESS != s32Ret)
{
    printf("get adec attr ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* enable adec */
ret = MI_AO_EnableAdec(AoDevId, AoChn);
if (MI_SUCCESS != s32Ret)
{
    printf("enable adec ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* send frame */
memset(&stAoSendFrame, 0x0, sizeof(MI_AUDIO_Frame_t));
stAoSendFrame.u32Len = 1024;
stAoSendFrame.apVirAddr[0] = u8Buf;
stAoSendFrame.apVirAddr[1] = NULL;

do{
    ret = MI_AO_SendFrame(AoDevId, AoChn, &stAoSendFrame, -1);
}while(ret == MI_AO_ERR_NOBUF);

/* disable adec */
ret = MI_AO_DisableAdec(AoDevId, AoChn);
if (MI_SUCCESS != s32Ret)
{
    printf("disable adec ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* disable ao chn */
ret = MI_AO_DisableChn(AoDevId, AoChn);
if (MI_SUCCESS != ret)
{
    printf("disable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

/* disable ao device */
ret = MI_AO_Disable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("disable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

MI_SYS_Exit();
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

2.1.24 MI_AO_GetAdecAttr

➢ 功能

获取 AO 解码功能相关属性。

➢ 语法

MI_S32 MI_AO_GetAdecAttr (MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn, MI_AO_AdecConfig_t *pstAdecConfig);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 仅支持使用通道0。	输入
pstAdecConfig	音频解码配置结构体指针	输出

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so libg711.a libg726.a

※ 注意

无。

➢ 举例

请参考 MI_AO_SetAdecAttr 举例部分。

2.1.25 MI_AO_EnableAdec

➢ 功能

使能 AO 解码功能。

➢ 语法

MI_AO_EnableAdec (MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 仅支持使用通道0。	输入

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 返回值

返回值 { 0 成功。
非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so libg711.a libg726.a

※ 注意

使能 AO 的解码功能相关前，必须先设置对应 AO 通道的解码参数。

➢ 举例

请参考 MI_AO_SetAdecAttr 举例部分。

2.1.26 MI_AO_DisableAdec

➢ 功能

禁用 AO 解码功能。

➢ 语法

MI_AO_DisableAdec (MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 仅支持使用通道0。	输入

➢ 返回值

返回值 { 0 成功。
非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so libg711.a libg726.a

※ 注意

无。

➢ 举例

请参考 MI_AO_SetAdecAttr 举例部分。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

2.1.27 MI_AO_SetChnParam

➢ 功能

设置音频通道参数。

➢ 语法

MI_S32 MI_AO_SetChnParam(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn, MI_AO_ChnParam_t *pstChnParam);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 取值范围：[0, MI_AUDIO_MAX_CHN_NUM)。	输入
pstChnParam	音频通道参数结构体指针	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

无。

➢ 举例

下面的代码实现通道参数的设置和获取。

```
1.  MI_S32 ret;
2.  MI_AUDIO_Attr_t stAttr;
3.  MI_AUDIO_Attr_t stGetAttr;
4.  MI_AUDIO_DEV AoDevId = 0;
5.  MI_AO_CHN AoChn = 0;
6.  MI_U8 u8Buf[1024];
7.  MI_AUDIO_Frame_t stAoSendFrame;
8.  MI_AO_ChnParam_t stChnParam;
9.
10.
11. stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
12. stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
13. stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
14. stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_MASTER;
15. stAttr.u32PtNumPerFrm = 1024;
16. stAttr.u32ChnCnt = 1;
17.
18. MI_SYS_Init();
19.
20. /* set ao public attr*/
21. ret = MI_AO_SetPubAttr(AoDevId, &stAttr);
22. if(MI_SUCCESS != ret)
23. {
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
24.     printf("set ao %d attr err:0x%x\n", AoDevId, ret);
25.     return ret;
26. }
27.
28. /* get ao public attr */
29. ret = MI_AO_GetPubAttr(AoDevId, &stGetAttr);
30. if(MI_SUCCESS != ret)
31. {
32.     printf("get ao %d attr err:0x%x\n", AoDevId, ret);
33.     return ret;
34. }
35.
36. /* enable ao device*/
37. ret = MI_AO_Enable(AoDevId);
38. if(MI_SUCCESS != ret)
39. {
40.     printf("enable ao dev %d err:0x%x\n", AoDevId, ret);
41.     return ret;
42. }
43.
44. /* enable ao chn */
45. ret = MI_AO_EnableChn(AoDevId, AoChn);
46. if (MI_SUCCESS != ret)
47. {
48.     printf("enable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
49.     return ret;
50. }
51.
52. memset(&stChnParam, 0x0, sizeof(stChnParam));
53. stChnParam.stChnGain.bEnableGainSet = TRUE;
54. stChnParam.stChnGain.s16Gain = 0;
55.
56. /* set chn param */
57. ret = MI_AO_SetChnParam(AoDevId, AoChn, &stChnParam);
58. if (MI_SUCCESS != ret)
59. {
60.     printf("set chn param ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
61.     return ret;
62. }
63.
64. memset(&stChnParam, 0x0, sizeof(stChnParam));
65.
66. ret = MI_AO_GetChnParam(AoDevId, AoChn, &stChnParam);
67. if (MI_SUCCESS != ret)
68. {
69.     printf("get chn param ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
70.     return ret;
71. }
72.
73. /* send frame */
74. memset(&stAoSendFrame, 0x0, sizeof(MI_AUDIO_Frame_t));
75. stAoSendFrame.u32Len = 1024;
76. stAoSendFrame.apVirAddr[0] = u8Buf;
77. stAoSendFrame.apVirAddr[1] = NULL;
78.
79. do{
80.     ret = MI_AO_SendFrame(AoDevId, AoChn, &stAoSendFrame, -1);
81. }while(ret == MI_AO_ERR_NOBUF);
82.
83. /* disable ao chn */
84. ret = MI_AO_DisableChn(AoDevId, AoChn);
85. if (MI_SUCCESS != ret)
86. {
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
87.     printf("disable ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
88.     return ret;
89. }
90.
91. /* disable ao device */
92. ret = MI_AO_Disable(AoDevId);
93. if(MI_SUCCESS != ret)
94. {
95.     printf("disable ao dev %d err:0x%x\n", AoDevId, ret);
96.     return ret;
97. }
98.
99. MI_SYS_Exit();
```

2.1.28 MI_AO_GetChnParam

➢ 功能

获取音频通道参数。

➢ 语法

MI_S32 MI_AO_GetChnParam(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn, MI_AO_ChnParam_t *pstChnParam);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
AoChn	音频输出通道号。 取值范围：[0, MI_AUDIO_MAX_CHN_NUM)。	输入
pstChnParam	音频通道参数结构体指针	输出

➢ 返回值

返回值
- 0　成功。
- 非 0　失败，参照错误码。

➢ 依赖

- 头文件:　mi_ao.h
- 库文件:　libmi_ao.a/libmi_ao.so

※ 注意

无。

➢ 举例

请参考 MI_AO_SetChnParam 举例部分。

Copyright © 2020 SigmaStar Technology Corp.　All rights reserved.

2.1.29 MI_AO_SetSrcGain

➢ 功能

设置 AO 数据的回声参考增益。

➢ 语法

MI_S32 MI_AO_SetSrcGain(MI_AUDIO_DEV AoDevId, MI_S32 s32VolumeDb);

➢ 形参

参数名称	描述	输入/输出
AoDevId	音频设备号	输入
s32VolumeDb	音频设备音量大小（-60 - 30dB）。	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_ao.h
- 库文件: libmi_ao.a/libmi_ao.so

※ 注意

- AO 设备成功启用后再调用此接口。

➢ 举例

下面的代码实现设置 Src gian。

```
MI_S32 ret;
MI_AUDIO_Attr_t stAttr;
MI_AUDIO_Attr_t stGetAttr;
MI_AUDIO_DEV AoDevId = 0;
MI_AO_CHN AoChn = 0;
MI_U8 u8Buf[1024];
MI_AUDIO_Frame_t stAoSendFrame;

stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_MASTER;
stAttr.u32PtNumPerFrm = 1024;
stAttr.u32ChnCnt = 1;

MI_SYS_Init();

/* set ao public attr*/
ret = MI_AO_SetPubAttr(AoDevId, &stAttr);
if(MI_SUCCESS != ret)
{
    printf("set ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

/* get ao public attr */
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

```
ret = MI_AO_GetPubAttr(AoDevId, &stGetAttr);
if(MI_SUCCESS != ret)
{
    printf("get ao %d attr err:0x%x\n", AoDevId, ret);
    return ret;
}

/* enable ao device*/
ret = MI_AO_Enable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("enable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

ret = MI_AO_SetSrcGain(AoDevId, 0);
if(MI_SUCCESS != ret)
{
    printf("set src gain ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

/* disable ao device */
ret = MI_AO_Disable(AoDevId);
if(MI_SUCCESS != ret)
{
    printf("disable ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

MI_SYS_Exit();
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

3. AO 数据类型

AO 模块相关数据类型定义如下：

MI_AUDIO_DEV	定义音频输入/输出设备编号
MI_AUDIO_MAX_CHN_NUM	定义音频输入/输出设备的最大通道数
MI_AO_CHN	定义音频输出通道
MI_AUDIO_SampleRate_e	定义音频采样率
MI_AUDIO_Bitwidth_e	定义音频采样精度
MI_AUDIO_Mode_e	定义音频输入输出工作模式
MI_AUDIO_SoundMode_e	定义音频声道模式
MI_AUDIO_Attr_t	定义音频输入输出设备属性结构体
MI_AO_ChnState_t	音频输出通道的数据缓存状态结构体
MI_AUDIO_Frame_t	定义音频帧数据结构体
MI_AUDIO_AecFrame_t	定义回声抵消参考帧信息结构体
MI_AUDIO_SaveFileInfo_t	定义音频保存文件功能配置信息结构体
MI_AO_VqeConfig_t	定义音频输出声音质量增强配置信息结构体
MI_AUDIO_HpfConfig_t	定义音频高通滤波功能配置信息结构体
MI_AUDIO_HpfFreq_e	定义音频高通滤波截止频率
MI_AUDIO_AnrConfig_t	定义音频语音降噪功能配置信息结构体
MI_AUDIO_AgcConfig_t	定义音频自动增益控制配置信息结构体
MI_AUDIO_EqConfig_t	定义音频均衡器功能配置信息结构体
MI_AO_AdecConfig_t	定义解码功能配置信息结构体。
MI_AUDIO_AlgorithmMode_e	定义音频算法的运行模式
MI_AO_ChnParam_t	定义音频通道属性结构体
MI_AO_ChnGainConfig_t	定义音频通道增益设置结构体

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

3.1. MI_AUDIO_DEV

➢ 说明

定义音频输入/输出设备编号。

➢ 定义

typedef MI_S32 MI_AUDIO_DEV

※ 注意事项

以下为 chip 的 AI/AO Dev ID 和物理设备的对照表

Dev ID	AI Dev	AO Dev
0	Amic	Line out
1	Dmic	I2S TX
2	I2S RX	HDMI（TAIYAKI 系列芯片支持）
3	Line in	HDMI + Line out（TAIYAKI 系列芯片支持）
4	Amic + I2S RX（TAKOYAKI 系列芯片支持）	
5	Dmic + I2S RX（TAKOYAKI 系列芯片支持）	

以下为不同系列 chip 的规格差异

	Pretzel	Macaron	TAIYAKI	TAKOYAKI	Pudding	Ispahan
Line out	支持 2 路 8/16/32/48KHz 采样率	支持 2 路 8/16/32/48KHz 采样率	支持 2 路 8/16/32/48KHz 采样率	支持 2 路 8/16/32/48KHz 采样率	支持 2 路 8/16/32/48KHz 采样率	支持 2 路 8/16/32/48KHz 采样率
I2S TX	支持标准 I2S 模式和 TDM 模式，TDM 模式可扩展到 8 路，支援 4-wire 和 6-wire 模式，可提供 Mclk。支持 8/16/32/48KHz 采样率。	只支持标准 I2S 模式，且只能作为 master，仅支持 4-wire 模式，不可提供 Mclk。支持 8/16/32/48KHz 采样率。	只支持标准 I2S 模式，且只能作为 master，仅支持 4-wire 模式，不可提供 Mclk。支持 8/16/32/48KHz 采样率。	只支持标准 I2S 模式，且只能作为 master，仅支持 4-wire 模式，不可提供 Mclk。支持 8/16/32/48KHz 采样率。	支持标准 I2S 模式和 TDM 模式，TDM 模式可扩展到 8 路，支援 4-wire 和 6-wire 模式，可提供 Mclk。支持 8/16/32/48KHz 采样率。	只支持标准 I2S 模式，且只能作为 master，仅支持 4-wire 模式，不可提供 Mclk。支持 8/16/32/48KHz 采样率。
HDMI	不支持	不支持	支持	不支持	不支持	不支持
HDMI + Line out	不支持	不支持	支持	不支持	不支持	不支持

➢ 相关数据类型及接口

无。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

3.2. MI_AUDIO_MAX_CHN_NUM

➢ 说明

定义音频输入/输出设备的最大通道数。

➢ 定义

#define MI_AUDIO_MAX_CHN_NUM 16

※ 注意事项

无。

➢ 相关数据类型及接口

无。

3.3. MI_AO_CHN

➢ 说明

定义音频输出通道。

➢ 定义

typedef MI_S32 MI_AO_CHN

※ 注意事项

无。

➢ 相关数据类型及接口

无。

3.4. MI_AUDIO_SampleRate_e

➢ 说明

定义音频采样率。

➢ 定义

```
typedef enum
{
    E_MI_AUDIO_SAMPLE_RATE_8000 = 8000, /* 8kHz sampling rate */
    E_MI_AUDIO_SAMPLE_RATE_11052 = 11025, /* 11.025kHz sampling rate */
    E_MI_AUDIO_SAMPLE_RATE_12000 = 12000, /* 12kHz sampling rate */
    E_MI_AUDIO_SAMPLE_RATE_16000 = 16000, /* 16kHz sampling rate */
    E_MI_AUDIO_SAMPLE_RATE_22050 = 22050, /* 22.05kHz sampling rate */
    E_MI_AUDIO_SAMPLE_RATE_24000 = 24000, /* 24kHz sampling rate */
    E_MI_AUDIO_SAMPLE_RATE_32000 = 32000, /* 32kHz sampling rate */
    E_MI_AUDIO_SAMPLE_RATE_44100 = 44100, /* 44.1kHz sampling rate */
    E_MI_AUDIO_SAMPLE_RATE_48000 = 48000, /* 48kHz sampling rate */
    E_MI_AUDIO_SAMPLE_RATE_96000 = 96000, /* 96kHz sampling rate */
    E_MI_AUDIO_SAMPLE_RATE_INVALID,
}MI_AUDIO_SampleRate_e;;
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 成员

成员名称	描述
E_MI_AUDIO_SAMPLE_RATE_8000	8kHz 采样率
E_MI_AUDIO_SAMPLE_RATE_11025	11.025kHz 采样率
E_MI_AUDIO_SAMPLE_RATE_12000	12kHz 采样率
E_MI_AUDIO_SAMPLE_RATE_16000	16kHz 采样率
E_MI_AUDIO_SAMPLE_RATE_22050	22.05kHz 采样率
E_MI_AUDIO_SAMPLE_RATE_24000	24kHz 采样率
E_MI_AUDIO_SAMPLE_RATE_32000	32kHz 采样率
E_MI_AUDIO_SAMPLE_RATE_44100	44.1kHz 采样率
E_MI_AUDIO_SAMPLE_RATE_48000	48kHz 采样率
E_MI_AUDIO_SAMPLE_RATE_96000	96kHz 采样率

※ 注意事项

这里枚举值不是从 0 开始，而是与实际的采样率值相同。

➢ 相关数据类型及接口

MI_AUDIO_Attr_t。

3.5. MI_AUDIO_Bitwidth_e

➢ 说明

定义音频采样精度。

➢ 定义

```
typedef enum
{
    E_MI_AUDIO_BIT_WIDTH_16 =0, /* 16bit width */
    E_MI_AUDIO_BIT_WIDTH_24 =1, /* 24bit width */
    E_MI_AUDIO_BIT_WIDTH_MAX,
}MI_AUDIO_BitWidth_e;
```

➢ 成员

成员名称	描述
E_MI_AUDIO_BIT_WIDTH_16	采样精度为 16bit 位宽
E_MI_AUDIO_BIT_WIDTH_24	采样精度为 24bit 位宽

※ 注意事项

目前软件只支持 16bit 位宽。

➢ 相关数据类型及接口

无。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

3.6. MI_AUDIO_Mode_e

➢ 说明

定义音频输入输出设备工作模式。

➢ 定义

```
typedef enum
{
    E_MI_AUDIO_MODE_I2S_MASTER,    /* I2S master mode */
    E_MI_AUDIO_MODE_I2S_SLAVE,     /* I2S slave mode */
    E_MI_AUDIO_MODE_TDM_MASTER, /* TDM master mode */
    E_MI_AUDIO_MODE_TDM_SLAVE,    /* TDM slave mode */
    E_MI_AUDIO_MODE_MAX,
}MI_AUDIO_Mode_e;
```

➢ 成员

成员名称	描述
E_MI_AUDIO_MODE_I2S_MASTER	I2S 主模式
E_MI_AUDIO_MODE_I2S_SLAVE	I2S 从模式
E_MI_AUDIO_MODE_TDM_MASTER	TDM 主模式
E_MI_AUDIO_MODE_TDM_SLAVE	TDM 从模式

※ 注意事项

主模式与从模式是否支持会依据不同的芯片而有区别。

➢ 相关数据类型及接口

MI_AUDIO_Attr_t。

3.7. MI_AUDIO_SoundMode_e

➢ 说明

定义音频声道模式。

➢ 定义

```
typedef enum
{
    E_MI_AUDIO_SOUND_MODE_MONO =0,   /* mono */
    E_MI_AUDIO_SOUND_MODE_STEREO =1, /* stereo */
    E_MI_AUDIO_SOUND_MODE_QUEUE =2,  /* queue */
     E_MI_AUDIO_SOUND_MODE_MAX,
}MI_AUDIO_SoundMode_e
```

➢ 成员

成员名称	描述
E_MI_AUDIO_SOUND_MODE_MONO	单声道。
E_MI_AUDIO_SOUND_MODE_STEREO	双声道。
E_MI_AUDIO_SOUND_MODE_QUEUE	此模式仅应用在音频采集

※ 注意事项

无。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 相关数据类型及接口

MI_AUDIO_Attr_t。

3.8. MI_AUDIO_HpfFreq_e

➢ 说明

定义音频高通滤波截止频率。

➢ 定义

```
typedef enum
{
    E_MI_AUDIO_HPF_FREQ_80 = 80, /* 80Hz */
    E_MI_AUDIO_HPF_FREQ_120 = 120, /* 120Hz */
    E_MI_AUDIO_HPF_FREQ_150 = 150, /* 150Hz */
    E_MI_AUDIO_HPF_FREQ_BUTT,
} MI_AUDIO_HpfFreq_e;
```

➢ 成员

成员名称	描述
E_MI_AUDIO_HPF_FREQ_80	截止频率为 80Hz。
E_MI_AUDIO_HPF_FREQ_120	截止频率为 120Hz。
E_MI_AUDIO_HPF_FREQ_150	截止频率为 150Hz。

※ 注意事项

无

➢ 相关数据类型及接口

MI_AO_VqeConfig_t

3.9. MI_AUDIO_AdecType_e

➢ 说明

定义音频解码类型。

➢ 定义

```
typedef enum
{
    E_MI_AUDIO_ADEC_TYPE_G711A = 0,
    E_MI_AUDIO_ADEC_TYPE_G711U,
    E_MI_AUDIO_ADEC_TYPE_G726,
    E_MI_AUDIO_ADEC_TYPE_INVALID,
}MI_AUDIO_AdecType_e;
```

➢ 成员

成员名称	描述
E_MI_AUDIO_ADEC_TYPE_G711A	G711A 解码。
E_MI_AUDIO_ADEC_TYPE_G711U	G711U 解码。
E_MI_AUDIO_ADEC_TYPE_G726	G726 解码。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

※ 注意事项

无

➢ 相关数据类型及接口

MI_AO_AdecConfig

3.10. MI_AUDIO_G726Mode_e

➢ 说明

定义 G726 工作模式。

➢ 定义

```
typedef enum{
    E_MI_AUDIO_G726_MODE_16 = 0,
    E_MI_AUDIO_G726_MODE_24,
    E_MI_AUDIO_G726_MODE_32,
    E_MI_AUDIO_G726_MODE_40,
    E_MI_AUDIO_G726_MODE_INVALID,
}MI_AUDIO_G726Mode_e;
```

➢ 成员

成员名称	描述
E_MI_AUDIO_G726_MODE_16	G726 16K 比特率模式。
E_MI_AUDIO_G726_MODE_24	G726 24K 比特率模式。
E_MI_AUDIO_G726_MODE_32	G726 32K 比特率模式。
E_MI_AUDIO_G726_MODE_40	G726 40K 比特率模式

※ 注意事项

无

➢ 相关数据类型及接口

MI_AO_AdecConfig_t

3.11. MI_AUDIO_I2sFmt_e

➢ 说明

I2S 格式设定。

➢ 定义

```
typedef enum
{
    E_MI_AUDIO_I2S_FMT_I2S_MSB,
    E_MI_AUDIO_I2S_FMT_LEFT_JUSTIFY_MSB,
}MI_AUDIO_I2sFmt_e;
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 成员

成员名称	描述
E_MI_AUDIO_I2S_FMT_I2S_MSB	I2S 标准格式，最高位优先
E_MI_AUDIO_I2S_FMT_LEFT_JUSTIFY_MSB	I2S 左对齐格式，最高位优先

※ 注意事项

无

➢ 相关数据类型及接口

MI_AUDIO_I2sConfig_t

3.12. MI_AUDIO_I2sMclk_e

➢ 说明

I2S MCLK 设定

➢ 定义

```
typedef enum{
    E_MI_AUDIO_I2S_MCLK_0,
    E_MI_AUDIO_I2S_MCLK_12_288M,
    E_MI_AUDIO_I2S_MCLK_16_384M,
    E_MI_AUDIO_I2S_MCLK_18_432M,
    E_MI_AUDIO_I2S_MCLK_24_576M,
    E_MI_AUDIO_I2S_MCLK_24M,
    E_MI_AUDIO_I2S_MCLK_48M,
}MI_AUDIO_I2sMclk_e;
```

➢ 成员

成员名称	描述
E_MI_AUDIO_I2S_MCLK_0	关闭 MCLK
E_MI_AUDIO_I2S_MCLK_12_288M	设置 MCLK 为 12.88M
E_MI_AUDIO_I2S_MCLK_16_384M	设置 MCLK 为 16.384M
E_MI_AUDIO_I2S_MCLK_18_432M	设置 MCLK 为 18.432M
E_MI_AUDIO_I2S_MCLK_24_576M	设置 MCLK 为 24.576M
E_MI_AUDIO_I2S_MCLK_24M	设置 MCLK 为 24M
E_MI_AUDIO_I2S_MCLK_48M	设置 MCLK 为 48M

※ 注意事项

无。

➢ 相关数据类型及接口

MI_AUDIO_I2sConfig_t

3.13. MI_AUDIO_I2sConfig_t

➢ 说明

定义 I2S 属性结构体。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 定义

```
typedef struct MI_AUDIO_I2sConfig_s
{
    MI_AUDIO_I2sFmt_e eFmt;
    MI_AUDIO_I2sMclk_e eMclk;
    MI_BOOL bSyncClock;
}MI_AUDIO_I2sConfig_t;
```

➢ 成员

成员名称	描述
eFmt	I2S 格式设置。 静态属性。
eMclk	I2S MCLK 时钟频率。 静态属性。
bSyncClock	AO 同步 AI 时钟 静态属性。

※ 注意事项

无。

➢ 相关数据类型及接口

MI_AUDIO_Attr_t

3.14. MI_AUDIO_Attr_t

➢ 说明

定义音频输入输出设备属性结构体。

➢ 定义

```
typedef struct MI_AUDIO_Attr_s
{
    MI_AUDIO_SampleRate_e eSamplerate; /*sample rate*/
    MI_AUDIO_BitWidth_e eBitwidth; /*bitwidth*/
    MI_AUDIO_Mode_e eWorkmode; /*master or slave mode*/
    MI_AUDIO_SoundMode_e eSoundmode; /*momo or stereo*/
    MI_U32 u32FrmNum; /*frame num in buffer*/
    MI_U32 u32PtNumPerFrm; /*number of samples*/
    MI_U32 u32CodecChnCnt; /*channel number on Codec */
    MI_U32 u32ChnCnt;
    union{
        MI_AUDIO_I2sConfig_t stI2sConfig;
    }WorkModeSetting;
}MI_AUDIO_Attr_t;
```

➢ 成员

成员名称	描述
eSamplerate	音频采样率。 静态属性。
eBitwidth	音频采样精度（从模式下，此参数必须和音频 AD/DA 的采样精度匹配）。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

成员名称	描述
	静态属性。
eWorkmode	音频输入输出工作模式。 静态属性。
eSoundmode	音频声道模式。 静态属性。
u32FrmNum	缓存帧数目。 保留，未使用。
u32PtNumPerFrm	每帧的采样点个数。 取值范围为：128，128*2,…,128*N。 静态属性。
u32CodecChnCnt	支持的 codec 通道数目。 保留，未使用。
u32ChnCnt	支持的通道数目，实际可使能的最大通道数。取值：1、2、4、8、16。（输入最多支持 MI_AUDIO_MAX_CHN_NUM 个通道，输出最多支持 2 个通道）
MI_AUDIO_I2sConfig_t stI2sConfig;	设置 I2S 工作属性

※ 注意事项

无。

➢ 相关数据类型及接口

MI_AO_SetPubAttr

3.15. MI_AO_ChnState_t

➢ 说明

音频输出通道的数据缓存状态结构体。

➢ 定义

```
typedef struct MI_AO_ChnState_s
{
    MI_U32 u32ChnTotalNum;
    MI_U32 u32ChnFreeNum;
    MI_U32 u32ChnBusyNum;
} MI_AO_ChnState_t;
```

➢ 成员

成员名称	描述
u32ChnTotalNum	输出通道总的缓存字节数。
u32ChnFreeNum	可用的空闲缓存字节数。
u32ChnBusyNum	被占用缓存字节数。

※ 注意事项

无。

➢ 相关数据类型及接口

无。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

3.16. MI_AUDIO_Frame_t

➢ 说明

定义音频帧结构体。

➢ 定义

```
typedef struct MI_AUDIO_Frame_s
{
    MI_AUDIO_BitWidth_e eBitwidth; /*audio frame bitwidth*/
    MI_AUDIO_SoundMode_e eSoundmode; /*audio frame momo or stereo mode*/
    void *apVirAddr[2];
    MI_U64 u64TimeStamp;/*audio frame timestamp*/
    MI_U32 u32Seq; /*audio frame seq*/
    MI_U32 u32Len; /*data lenth per channel in frame*/
    MI_U32 au32PoolId[2];
}MI_AUDIO_Frame_t;
```

➢ 成员

成员名称	描述
eBitwidth	音频采样精度
eSoundmode	音频声道模式。
pVirAddr[2]	音频帧数据虚拟地址。
u64TimeStamp	音频帧时间戳。 以 μs 为单位
u32Seq	音频帧序号。
u32Len	音频帧长度。 以 byte 为单位。
u32PoolId[2]	音频帧缓存池 ID。

※ 注意事项

- u32Len（音频帧长度）指整个缓冲区的数据长度。
- 单声道数据直接存放，采样点数为 u32PtNumPerFrm，长度为 u32Len；立体声数据按左右声道交错存放，长度为 u32Len。

➢ 相关数据类型及接口

无。

3.17. MI_AUDIO_AecFrame_t

➢ 说明

定义音频回声抵消参考帧信息结构体。

➢ 定义

```
typedef struct MI_AUDIO_AecFrame_s
{
    MI_AUDIO_Frame_t stRefFrame; /* aec reference audio frame */
    MI_BOOL bValid; /* whether frame is valid */
}MI_AUDIO_AecFrame_t;
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 成员

成员名称	描述
stRefFrame	回声抵消参考帧结构体。
bValid	参考帧有效的标志。 取值范围： TRUE：参考帧有效。 FALSE：参考帧无效，无效时不能使用此参考帧进行回声抵消。

※ 注意事项

无。

➢ 相关数据类型及接口

无。

3.18. MI_AUDIO_SaveFileInfo_t

➢ 说明

定义音频保存文件功能配置信息结构体。

➢ 定义

```
typedef struct MI_AUDIO_SaveFileInfo_s
{
    MI_BOOL bCfg;
    MI_U8 szFilePath[256];
    MI_U32 u32FileSize; /*in KB*/
} MI_AUDIO_SaveFileInfo_t
```

➢ 成员

成员名称	描述
bCfg	配置使能开关。
szFilePath	音频文件的保存路径
u32FileSize	文件大小，取值范围[1,10240]KB。

※ 注意事项

无

➢ 相关数据类型及接口

MI_AI_SaveFile

3.19. MI_AO_VqeConfig_t

➢ 说明

定义音频输出声音质量增强配置信息结构体。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 定义

```
typedef struct MI_AO_VqeConfig_s
{
    MI_BOOL                 bHpfOpen;
    MI_BOOL                 bAnrOpen;
    MI_BOOL                 bAgcOpen;
    MI_BOOL                 bEqOpen;
    MI_S32                  s32WorkSampleRate;
    MI_S32                  s32FrameSample;
    MI_AUDIO_HpfConfig_t    stHpfCfg;
    MI_AUDIO_AnrConfig_t    stAnrCfg;
    MI_AUDIO_AgcConfig_t    stAgcCfg;
    MI_AUDIO_EqConfig_t     stEqCfg;
} MI_AO_VqeConfig_t;
```

➢ 成员

成员名称	描述
bHpfOpen	高通滤波功能是否使能标志。
bAnrOpen	语音降噪功能是否使能标志。
bAgcOpen	自动增益控制功能是否使能标志
bEqOpen	均衡器功能是否使能标志
s32WorkSampleRate	工作采样频率。该参数为内部功能算法工作采样率。取值范围：8KHz/16KHz。默认值为 8KHz。
s32FrameSample	VQE 的帧长，即采样点数目。只支持 128。
stHpfCfg	高通滤波功能相关配置信息。
stAnrCfg	语音降噪功能相关配置信息。
stAgcCfg	自动增益控制相关配置信息。
stEqCfg	均衡器相关配置信息。

※ 注意事项

无。

➢ 相关数据类型及接口

无。

3.20. MI_AUDIO_HpfConfig_t

➢ 说明

定义音频高通滤波功能配置信息结构体。

➢ 定义

```
typedef struct MI_AUDIO_HpfConfig_s
{
    MI_AUDIO_AlgorithmMode_e eMode;
    MI_AUDIO_HpfFreq_e eHpfFreq; /*freq to be processed*/
} MI_AUDIO_HpfConfig_t;
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 成员

成员名称	描述
eMode	音频算法的运行模式
eHpfFreq	高通滤波截止频率选择。 80：截止频率为 80Hz； 120：截止频率为 120Hz； 150：截止频率为 150Hz。 默认值 150。

※ 注意事项

无。

➢ 相关数据类型及接口

MI_AO_VqeConfig_t

3.21. MI_AUDIO_AnrConfig_t

➢ 说明

定义音频语音降噪功能配置信息结构体。

➢ 定义

```
typedef struct MI_AUDIO_AnrConfig_s
{
    MI_AUDIO_AlgorithmMode_e eMode;
    MI_U32      u32NrIntensity;
    MI_U32      u32NrSmoothLevel;
    MI_AUDIO_NrSpeed_e eNrSpeed;
} MI_AUDIO_AnrConfig_t;
```

➢ 成员

成员名称	描述
eMode	音频算法运行模式 注：Anr 的模式选择将会在一定程度上影响 Agc 的功能
u32NrIntensity	降噪力度配置，配置值越大降噪力度越高，但同时也会带来细节音的丢失/损伤。 范围[0,30]；步长 1 默认值 20。
u32NrSmoothLevel	平滑化程度 范围[0,10]；步长 1 默认值 10。
eNrSpeed	噪声收敛速度，低速，中速，高速 默认值中速。

※ 注意事项

在 Anr 和 Agc 都有使能的情况下，当 Anr 设定为 user mode 时，Agc 会对音频数据做频域处理，会评估出语音信号再做相应的增减，而当 Anr 设定为 default/music mode 时，Agc 会对音频数据做时域处理，对全频段的数据进行增减。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 相关数据类型及接口
MI_AO_VqeConfig_t

3.22. MI_AUDIO_NrSpeed_e

➢ 说明
定义噪声收敛速度

➢ 定义
```
typedef enum
 {
        E_MI_AUDIO_NR_SPEED_LOW,
        E_MI_AUDIO_NR_SPEED_MID,
        E_MI_AUDIO_NR_SPEED_HIGH
}MI_AUDIO_NrSpeed_e;
```

➢ 成员

成员名称	描述
E_MI_AUDIO_NR_SPEED_LOW	低速。
E_MI_AUDIO_NR_SPEED_MID	中速。
E_MI_AUDIO_NR_SPEED_HIGH	高速。

※ 注意事项
无

➢ 相关数据类型及接口
MI_AO_VqeConfig_t

3.23. MI_AUDIO_AgcConfig_t

➢ 说明
定义音频自动增益控制配置信息结构体。

➢ 定义
```
typedef struct MI_AUDIO_AgcConfig_s
{
    MI_AUDIO_AlgorithmMode_e eMode;
    AgcGainInfo_t stAgcGainInfo;
    MI_U32       u32DropGainMax;
    MI_U32       u32AttackTime;
    MI_U32       u32ReleaseTime;
    MI_S16       s16Compression_ratio_input[5];
    MI_S16       s16Compression_ratio_output[5];
    MI_S32       s32TargetLevelDb;
    MI_S32       s32NoiseGateDb;
    MI_U32       u32NoiseGateAttenuationDb;
} MI_AUDIO_AgcConfig_t;
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 成员

成员名称	描述
eMode	音频算法的运行模式
stAgcGainInfo	定义 AGC 增益的最大、最小和初始值
u32DropGainMax	增益下降的最大值，防止输出饱和 范围[0,60]；步长 1 默认值 55。
u32AttackTime	增益下降时间区间长度，以 16 毫秒为 1 单位 范围[1,20]；步长 1 默认值 0。
u32ReleaseTime	增益上升时间区间长度，以 16 毫秒为 1 单位 范围[1,20] ；步长 1 默认值 0。
s16Compression_ratio_input[5]	输入压缩比，必须配合 s16Compression_ratio_output 使用，与 u32CompressionRatio 相比，透过多个转折点实现多斜率的曲线 范围[-80,0]；步长 1
s16Compression_ratio_output[5]	输出压缩比，必须配合 s16Compression_ratio_input 使用，与 u32CompressionRatio 相比，透过多个转折点实现多斜率的曲线 范围[-80,0]；步长 1
s32TargetLevelDb	目标电平，经过处理后的最大电平门限 范围[-80,0]dB；步长 1 默认值 0。
s32NoiseGateDb	噪声底值 范围[-80,0]；步长 1 注：当值为-80，噪声底值将不起作用 默认值-55。
u32NoiseGateAttenuationDb	当噪声底值起效果时，输入源的衰减百分比 范围[0,100]；步长 1 默认值 0。

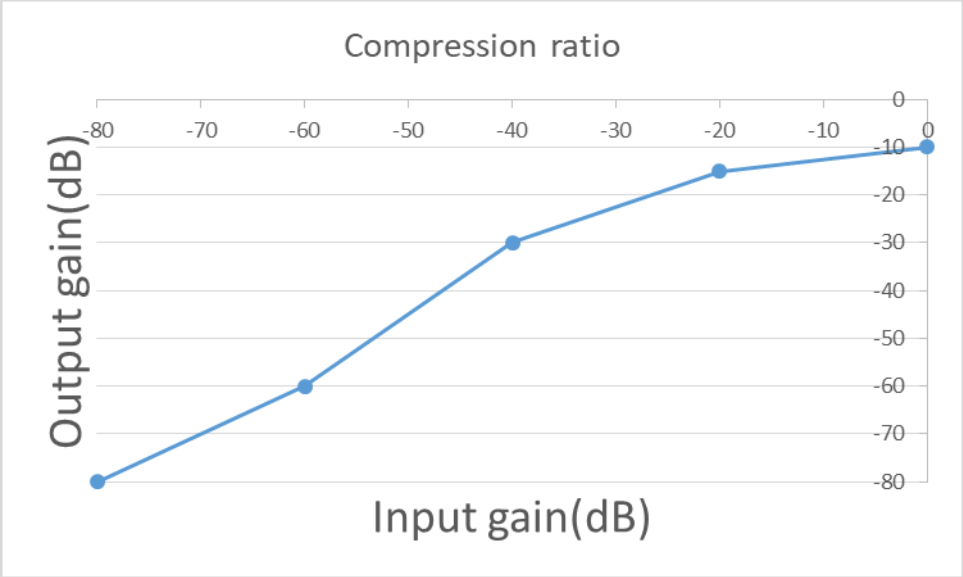
※ 注意事项

在 Anr 和 Agc 都有使能的情况下，当 Anr 设定为 user mode 时，Agc 会对音频数据做频域处理，会评估出语音信号再做相应的增减，而当 Anr 设定为 default/music mode 时，Agc 会对音频数据做时域处理，对全频段的数据进行增减。

而 s16Compression_ratio_input 和 s16Compression_ratio_output 则需要根据所需要的增益曲线来设定。

如下面的折线图所示，在输入增益为-80~0dB 划分为四段斜率，-80dB~-60dB 范围内保持原来的增益，斜率为 1，-60dB~-40dB 范围内需要稍微提高增益，斜率为 1.5，-40dB~-20dB 范围内斜率为 1.25，-20dB~0dB 范围内斜率为 0.25。根据曲线的转折点对 s16Compression_ratio_input 和 s16Compression_ratio_output 设置，若不需要那么多段曲线，则将数组不需要的部分填 0。

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.



➢ 相关数据类型及接口

MI_AO_VqeConfig_t

3.24. AgcGainInfo_t

➢ 说明

AGC 增益的取值

➢ 定义

```
typedef struct AgcGainInfo_s{
    MI_S32    s32GainMax;
    MI_S32    S32GainMin;
    MI_S32    s32GainInit;
}AgcGainInfo_t;
```

➢ 成员

成员名称	描述
s32GainMax	增益最大值 范围[0,60]；步长 1 默认值 15。
s32GainMin	增益最小值 范围[-20,30]；步长 1 默认值 0。
s32GainInit	增益初始值 范围[-20,60]；步长 1 默认值 0。

※ 注意事项

无

➢ 相关数据类型及接口

MI_AO_VqeConfig_t

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

3.25. MI_AUDIO_EqConfig_t

➢ 说明

定义音频均衡器功能配置信息结构体。

➢ 定义

```
typedef struct MI_AUDIO_EqConfig_s
{
    MI_AUDIO_AlgorithmMode_e eMode;
    MI_S16        s16EqGainDb[129];
} MI_AUDIO_EqConfig_t;
```

➢ 成员

成员名称	描述
eMode	音频算法的运行模式
s16EqGainDb[129]	均衡器增益调节取值,将当前采样率的频率范围分成 129 个频率范围来进行调节 范围[-50,20]；步长 1 默认值 0。 如：当前采样率为 16K，对应的最高频率为 8K，8000 / 129 ≈ 62Hz,则单个调节的频率范围为 62Hz,将 0-8K 划分成{0-1＊62Hz，1-2＊62Hz,2-3＊62Hz,…,128-129＊62Hz} = {0-62Hz，62-124Hz，124-186Hz，…，7938-8000Hz}，每段对应一个增益值

※ 注意事项

无

➢ 相关数据类型及接口

MI_AO_VqeConfig_t

3.26. MI_AO_AdecConfig_t

➢ 说明

定义解码功能配置信息结构体。

➢ 定义

```
typedef struct MI_AO_AdecConfig_s
{
    MI_AUDIO_AdecType_e    eAdecType;
        union
        {
            MI_AUDIO_AdecG711Config_t stAdecG711Cfg;
             MI_AUDIO_AdecG726Config_s    stAdecG726Cfg;
        };
}MI_AO_AdecConfig_t;
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 成员

成员名称	描述
eAdecType	音频解码类型。
stAdecG711Cfg	G711 解码相关配置信息。
stAdecG726Cfg	G726 解码相关配置信息。

※ 注意事项

无

➢ 相关数据类型及接口

MI_AO_SetAdecAttr

3.27. MI_AUDIO_AdecG711Config_t

➢ 说明

定义 G711 解码功能配置信息结构体。

➢ 定义

```
typedef struct MI_AUDIO_AdecG711Config_s{
    MI_AUDIO_SampleRate_e eSamplerate;
    MI_AUDIO_SoundMode_e eSoundmode;
}MI_AUDIO_AdecG711Config_t;
```

➢ 成员

成员名称	描述
eSamplerate	音频采样率。
eSoundmode	音频声道模式。

※ 注意事项

无

➢ 相关数据类型及接口

MI_AO_SetAdecAttr

3.28. MI_AUDIO_AdecG726Config_s

➢ 说明

定义 G711 解码功能配置信息结构体。

➢ 定义

```
typedef struct MI_AUDIO_AdecG726Config_s{
    MI_AUDIO_SampleRate_e eSamplerate;
    MI_AUDIO_SoundMode_e eSoundmode;
    MI_AUDIO_G726Mode_e   eG726Mode;
}MI_AUDIO_AdecG726Config_t;
```


Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

➢ 成员

成员名称	描述
eSamplerate	音频采样率。
eSoundmode	音频声道模式。
eG726Mode	G726 工作模式。

※ 注意事项

无

➢ 相关数据类型及接口

MI_AO_SetAdecAttr

3.29. MI_AUDIO_AlgorithmMode_e

➢ 说明

音频算法运行的模式。

➢ 定义

```
typedef enum
{
    E_MI_AUDIO_ALGORITHM_MODE_DEFAULT,
    E_MI_AUDIO_ALGORITHM_MODE_USER,
    E_MI_AUDIO_ALGORITHM_MODE_MUSIC,
    E_MI_AUDIO_ALGORITHM_MODE_INVALID,
}MI_AUDIO_AlgorithmMode_e;
```

➢ 成员

成员名称	描述
E_MI_AUDIO_ALGORITHM_MODE_DEFAULT	默认运行模式 当使用该模式时，将使用算法的默认参数
E_MI_AUDIO_ALGORITHM_MODE_USER	用户模式 当使用该模式时，需要用户重新设定所有参数
E_MI_AUDIO_ALGORITHM_MODE_MUSIC	音乐模式 仅有 Anr 具有此模式，当为此模式时，Agc 不会进行 speech enhancment（语音增强）处理

※ 注意事项

在 Anr 和 Agc 都有使能的情况下，当 Anr 设定为 user mode 时，Agc 会对音频数据做频域处理，会评估出语音信号再做相应的增减，而当 Anr 设定为 default/music mode 时，Agc 会对音频数据做时域处理，对全频段的数据进行增减。

➢ 相关数据类型及接口

MI_AUDIO_HpfConfig_t
MI_AUDIO_AnrConfig_t
MI_AUDIO_AgcConfig_t
MI_AUDIO_EqConfig_t

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

3.30. MI_AO_ChnParam_t

➢ 说明

定义音频通道参数设置结构体。

➢ 定义

```
typedef struct MI_AO_ChnParam_s
{
    MI_AO_ChnGainConfig_t stChnGain;
    MI_U32 u32Reserved;
} MI_AO_ChnParam_t;
```

➢ 成员

成员名称	描述
stChnGain	音频通道增益设置结构体
u32Reserved	保留，不使用

※ 注意事项

无。

➢ 相关数据类型及接口

MI_AO_SetChnParam
MI_AO_GetChnParam

3.31. MI_AO_ChnGainConfig_t

➢ 说明

定义音频通道增益设置结构体。

➢ 定义

```
typedef struct MI_AO_ChnGainConfig_s
{
    MI_BOOL bEnableGainSet;
    MI_S16 s16Gain;
}MI_AO_ChnGainConfig_t;
```

➢ 成员

成员名称	描述
bEnableGainSet	是否使能增益设置
s16Gain	增益（-60 ~ 30dB）

※ 注意事项

无。

➢ 相关数据类型及接口

MI_AO_ChnParam_t

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.

4. 错误码

AO API 错误码如表 3-1 所示：

表 3-1 AO API 错误码

宏定义	描述
MI_AO_ERR_INVALID_DEVID	音频输出设备号无效
MI_AO_ERR_INVALID_CHNID	音频输出信道号无效
MI_AO_ERR_ILLEGAL_PARAM	音频输出参数设置无效
MI_AO_ERR_NOT_ENABLED	音频输出设备或信道没有使能
MI_AO_ERR_NULL_PTR	输入参数空指标错误
MI_AO_ERR_NOT_CONFIG	音频输出设备属性未设置
MI_AO_ERR_NOT_SUPPORT	操作不支持
MI_AO_ERR_NOT_PERM	操作不允许
MI_AO_ERR_NOMEM	分配内存失败
MI_AO_ERR_NOBUF	音频输出缓存不足
MI_AO_ERR_BUF_EMPTY	音频输出缓存为空
MI_AO_ERR_BUF_FULL	音频输出缓存为满
MI_AO_ERR_SYS_NOTREADY	音频输出系统未初始化
MI_AO_ERR_BUSY	音频输出系统忙碌
MI_AO_ERR_VQE_ERR	音频输出 VQE 算法处理失败
MI_AO_ERR_ADEC_ERR	音频输出解码算法处理失败

Copyright © 2020 SigmaStar Technology Corp. All rights reserved.