

MI AI API

Version 2.13

© 2019 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	<ul style="list-style-type: none"> Initial release 	04/12/2018
2.04	<ul style="list-style-type: none"> Updated for accuracy 	02/23/2019
2.05	<ul style="list-style-type: none"> Added MI_AI_SetAedAttr, MI_AI_GetAedAttr, MI_AI_EnableAed, MI_AI_DisableAed, and MI_AI_GetAedResult 	03/07/2019
2.06	<ul style="list-style-type: none"> Added description regarding audio algorithm 	03/25/2019
2.07	<ul style="list-style-type: none"> Added MI_AI_SetExtAecChn and updated MI_AI_SetChnParam and MI_AI_GetChnParam 	03/30/2019
2.08	<ul style="list-style-type: none"> Added support for SSL and BF 	06/04/2019
2.09	<ul style="list-style-type: none"> Updated audio supporting sample rate 	06/17/2019
2.10	<ul style="list-style-type: none"> Updated MI_AUDIO_Frame_t structure Fixed the wrong sample rate Fixed some incorrect instructions 	07/31/2019
2.11	<ul style="list-style-type: none"> Updated I2S mode and Mclk Fixed the wrong description of AEC and AGC 	10/26/2019
2.12	<ul style="list-style-type: none"> Added API for setting bf angle 	12/04/2019
2.13	<ul style="list-style-type: none"> Added mute API for channel Added demo code for API 	12/30/2019

TABLE OF CONTENTS

REVISION HISTORY	i
TABLE OF CONTENTS.....	ii
1. 概述.....	v
1.1. 模块说明	v
1.2. 流程框图	v
1.3. 关键字说明.....	vi
2. API 参考	1
2.1. 功能模块 API	1
2.1.1 MI_AI_SetPubAttr	3
2.1.2 MI_AI_GetPubAttr	5
2.1.3 MI_AI_Enable	6
2.1.4 MI_AI_Disable	7
2.1.5 MI_AI_EnableChn	7
2.1.6 MI_AI_DisableChn.....	8
2.1.7 MI_AI_GetFrame.....	9
2.1.8 MI_AI_ReleaseFrame	10
2.1.9 MI_AI_SetChnParam	10
2.1.10 MI_AI_GetChnParam.....	12
2.1.11 MI_AI_EnableReSmp.....	13
2.1.12 MI_AI_DisableReSmp	15
2.1.13 MI_AI_SetVqeAttr	16
2.1.14 MI_AI_GetVqeAttr	19
2.1.15 MI_AI_EnableVqe.....	19
2.1.16 MI_AI_DisableVqe.....	20
2.1.17 MI_AI_ClrPubAttr	21
2.1.18 MI_AI_SaveFile	22
2.1.19 MI_AI_SetVqeVolume.....	23
2.1.20 MI_AI_GetVqeVolume	25
2.1.21 MI_AI_SetAencAttr.....	26
2.1.22 MI_AI_GetAencAttr	28
2.1.23 MI_AI_EnableAenc	29
2.1.24 MI_AI_DisableAenc	30
2.1.25 MI_AI_SetAedAttr	30
2.1.26 MI_AI_GetAedAttr	33
2.1.27 MI_AI_EnableAed.....	34
2.1.28 MI_AI_DisableAed.....	35
2.1.29 MI_AI_GetAedResult	35
2.1.30 MI_AI_SetExtAecChn.....	36
2.1.31 MI_AI_SetSslInitAttr.....	38
2.1.32 MI_AI_GetSslInitAttr	41
2.1.33 MI_AI_SetSslConfigAttr	42
2.1.34 MI_AI_GetSslConfigAttr	43

2.1.35	MI_AI_EnableSsl	44
2.1.36	MI_AI_DisableSsl	44
2.1.37	MI_AI_GetSslDoa	45
2.1.38	MI_AI_SetBfInitAttr	46
2.1.39	MI_AI_GetBfInitAttr	49
2.1.40	MI_AI_SetBfConfigAttr	49
2.1.41	MI_AI_GetBfConfigAttr	50
2.1.42	MI_AI_EnableBf	51
2.1.43	MI_AI_DisableBf	51
2.1.44	MI_AI_SetBfAngle	52
2.1.45	MI_AI_SetMute	53
2.1.46	MI_AI_GetMute	55
3.	AI 数据类型	56
3.1.	MI_AUDIO_DEV	57
3.2.	MI_AUDIO_MAX_CHN_NUM	58
3.3.	MI_AI_CHN	58
3.4.	MI_AUDIO_SampleRate_e	58
3.5.	MI_AUDIO_Bitwidth_e	59
3.6.	MI_AUDIO_Mode_e	60
3.7.	MI_AUDIO_SoundMode_e	60
3.8.	MI_AUDIO_AencType_e	61
3.9.	MI_AUDIO_G726Mode_e	61
3.10.	MI_AUDIO_I2sFmt_e	62
3.11.	MI_AUDIO_I2sMclk_e	63
3.12.	MI_AUDIO_I2sConfig_t	63
3.13.	MI_AUDIO_Attr_t	64
3.14.	MI_AI_ChnParam_t	65
3.15.	MI_AUDIO_Frame_t	66
3.16.	MI_AUDIO_AecFrame_t	67
3.17.	MI_AUDIO_SaveFileInfo_t	67
3.18.	MI_AI_VqeConfig_t	68
3.19.	MI_AUDIO_HpfConfig_t	69
3.20.	MI_AUDIO_HpfFreq_e	70
3.21.	MI_AI_AecConfig_t	70
3.22.	MI_AUDIO_AnrcConfig_t	71
3.23.	MI_AUDIO_NrSpeed_e	72
3.24.	MI_AUDIO_AgcConfig_t	72
3.25.	AgcGainInfo_t	74
3.26.	MI_AUDIO_EqConfig_t	75
3.27.	MI_AI_AencConfig_t	75
3.28.	MI_AUDIO_AencG711Config_t	76
3.29.	MI_AUDIO_AencG726Config_t	77
3.30.	MI_AUDIO_AlgorithmMode_e	77
3.31.	MI_AI_AedConfig_t	78

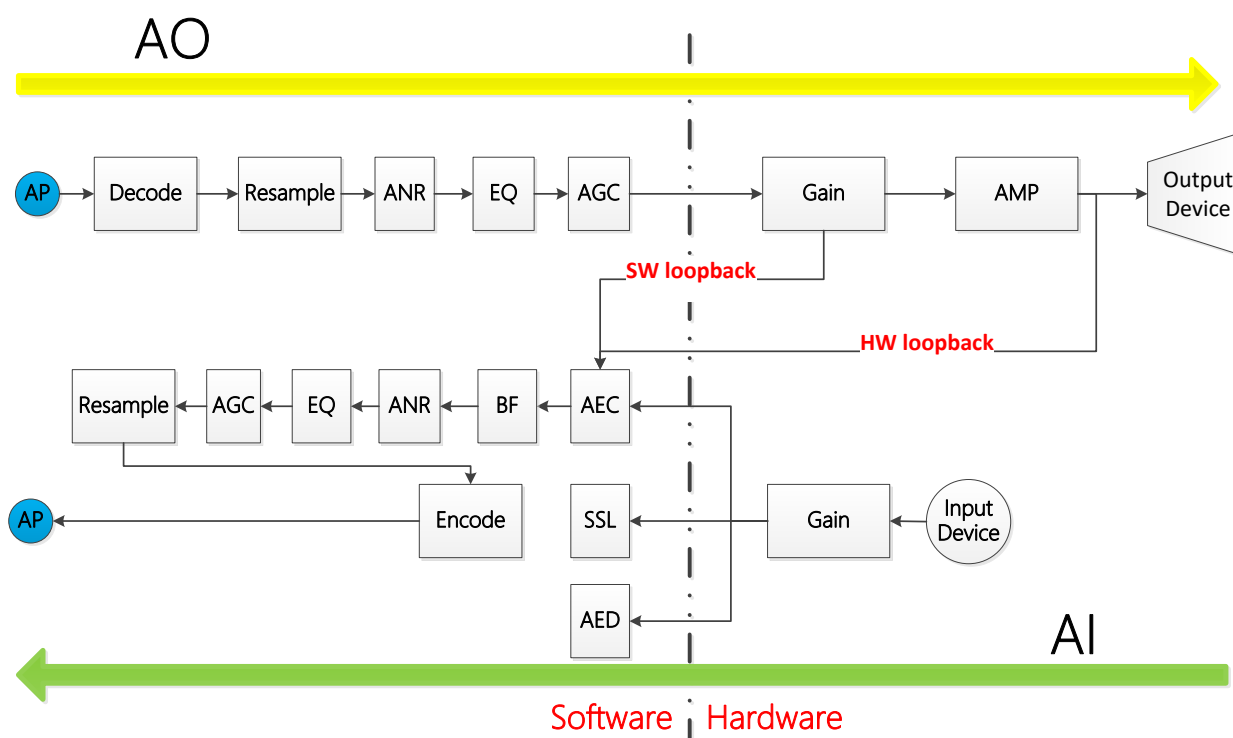
3.32. MI_AUDIO_AedSensitivity_e.....	78
3.33. MI_AI_AedResult_t.....	79
3.34. MI_AI_ChnGainConfig_t.....	80
3.35. MI_AI_SslInitAttr_t.....	80
3.36. MI_AI_SslConfigAttr_t.....	81
3.37. MI_AI_BfInitAttr_t.....	82
3.38. MI_AI_BfConfigAttr_t.....	82
4. 错误码	84

1. 概述

1.1. 模块说明

音频输入 (Audio Input, AI) 主要实现配置及启用音频输入设备、获取音频帧数据、以及声学算法处理等功能。声学算法处理主要包括：重采样、回声消除、降噪、高通滤波、均衡器、自动增益控制、声音事件检测、声源定位、波束成形等。

1.2. 流程框图



1.3. 关键字说明

- Device
与其他模块的 Device 概念不同，AI 的 Device 指代的是不同的外部输入设备，如 Amic/Dmic/I2S RX/Line in 等。
- Channel
AI 的 Channel 指代的是软件概念上的声道数。
- 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。
- BF
BF(beamforming)，波束成形算法，用于麦克风阵列，对声音进行增强。
- AEC
AEC(Acoustic Echo Cancellation)，回声消除。
- SSL
SSL(Sound Source Localization)，声源定位，用于麦克风阵列，识别声音的方向
- AED
AED(Acoustic Event Detection)，声音事件监测，目前仅能对婴儿哭声以及分贝过大的声音进行检测。
- HPF
HPF(High-Pass Filtering)，高通滤波

2. API 参考

2.1. 功能模块 API

API 名	功能
MI_AI_SetPubAttr	设置 AI 设备属性
MI_AI_GetPubAttr	获取 AI 设备属性
MI_AI_Enable	启用 AI 设备
MI_AI_Disable	禁用 AI 设备
MI_AI_EnableChn	启用 AI 通道
MI_AI_DisableChn	禁用 AI 通道
MI_AI_GetFrame	获取音频帧
MI_AI_ReleaseFrame	释放音频帧
MI_AI_SetChnParam	设置 AI 通道参数
MI_AI_GetChnParam	获取 AI 通道参数
MI_AI_EnableReSmp	启用 AI 重采样
MI_AI_DisableReSmp	禁用 AI 重采样。
MI_AI_SetVqeAttr	设置 AI 通道的声音质量增强功能相关属性
MI_AI_GetVqeAttr	获取 AI 通道的声音质量增强功能相关属性
MI_AI_EnableVqe	使能 AI 通道的声音质量增强功能
MI_AI_DisableVqe	禁用 AI 通道的声音质量增强功能
MI_AI_ClrPubAttr	清除 AI 设备属性
MI_AI_SaveFile	开启音频输入保存文件功能
MI_AI_SetVqeVolume	设置 AI 通道的音量大小
MI_AI_GetVqeVolume	获取 AI 通道的音量大小
MI_AI_SetAencAttr	设置 AI 通道编码功能相关属性
MI_AI_GetAencAttr	获取 AI 通道编码功能相关属性
MI_AI_EnableAenc	使能 AI 通道编码功能
MI_AI_DisableAenc	禁止 AI 通道编码功能
MI_AI_SetAedAttr	设置 AI 通道声音事件检测相关属性
MI_AI_GetAedAttr	获取 AI 通道声音事件检测相关属性
MI_AI_EnableAed	使能 AI 通道声音检测功能
MI_AI_DisableAed	禁止 AI 通道声音检测功能
MI_AI_GetAedResult	获取 AI 通道声音检测结果

API 名	功能
MI_AI_SetExtAecChn	设置 AI 通道回声消除功能参考的外部 AI 通道
MI_AI_SetSslInitAttr	设置 AI 通道声源定位功能的初始化参数
MI_AI_GetSslInitAttr	获取 AI 通道声源定位功能的初始化参数
MI_AI_SetSslConfigAttr	设置 AI 通道声源定位功能的配置参数
MI_AI_GetSslConfigAttr	获取 AI 通道声源定位功能的配置参数
MI_AI_EnableSsl	使能 AI 通道声源定位功能
MI_AI_DisableSsl	禁止 AI 通道声源定位功能
MI_AI_GetSslDoa	获取 AI 通道声源定位功能的结果
MI_AI_SetBfInitAttr	设置 AI 通道波束形成功能的初始化参数
MI_AI_GetBfInitAttr	获取 AI 通道波束形成功能的初始化参数
MI_AI_SetBfConfigAttr	设置 AI 通道波束形成功能的配置参数
MI_AI_GetBfConfigAttr	获取 AI 通道波束形成功能的配置参数
MI_AI_EnableBf	使能 AI 通道波束成形功能
MI_AI_DisableBf	禁止 AI 通道波束成形功能
MI_AI_SetBfAngle	设置 AI 通道波束成形功能的作用角度
MI_AI_SetMute	设置 AI 通道的静音参数
MI_AI_GetMute	获取 AI 通道的静音参数

2.1.1 MI_AI_SetPubAttr

➤ 功能

设置 AI 设备属性。

➤ 语法

```
MI_S32 MI_AI_SetPubAttr(MI_AUDIO_DEV AiDevId, MI_AUDIO_Attr_t *pstAttr);
```

➤ 形参

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

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so

※ 注意

音频输入设备的属性决定了输入数据的格式，输入设备属性包括采样率、采样精度、工作模式、通道的数据格式、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 8/16/32/48KHz 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。
- 通道的数据格式
数据格式表示通道中的数据排列方式。Mono 表示 AI 通道的数据是一个物理通道数据。Stereo 表示 AI 通道的数据是两个物理通道数据交织在一起，组成立体声。Queue 表示一个通道中有多个物理通道数据首尾相接存放。在 Queue 模式下启用算法时，所有通道使用相同的算法参数。
- 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和通道配置的 buffer 数目。
- 通道数目
通道数是指当前输入设备在软件概念中的通道数。通道的数量以及通道的数据格式决定了要使用多少物理通道。

- 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

下面的代码实现从 AI 通道取一帧数据，然后释放的功能。

```

1. MI_S32 ret;
2. MI_AUDIO_Attr_t stAttr;
3. MI_AUDIO_Dev AiDevId = 0;
4. MI_AI_CHN AiChn = 0;
5. MI_SYS_ChnPort_t stChnPort;
6. MI_S32 s32Fd;
7. fd_set readFdSet;
8. struct timeval stTimeOut;
9. MI_AUDIO_Frame_t stAiChFrame;
10. MI_AUDIO_AecFrame_t stAecFrame;
11. MI_SYS_ChnPort_t stAiChnOutputPort;
12.
13. MI_SYS_Init();
14.
15. stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
16. stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
17. stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
18. stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
19. stAttr.u32PtNumPerFrm = 160;
20. stAttr.u32ChnCnt = 1;
21.
22. /* set public attribute of AI device */
23. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
24. if (MI_SUCCESS != ret)
25. {
26.     printf("set ai %d attr err:0x%x\n", AiDevId, ret);
27.     return ret;
28. }
29.
30. /* get public attribute of AI device */
31. ret = MI_AI_GetPubAttr(AiDevId, &stAttr);
32. if (MI_SUCCESS != ret)
33. {
34.     printf("get ai %d attr err:0x%x\n", AiDevId, ret);
35.     return ret;
36. }
37.
38. /* enable AI device */
39. ret = MI_AI_Enable(AiDevId);
40. if (MI_SUCCESS != ret)
41. {
42.     printf("enable ai %d err:0x%x\n", AiDevId, ret);
43.     return ret;
44. }
45.
46. /* enable AI Channel */
47. ret = MI_AI_EnableChn(AiDevId, AiChn);
48. if (MI_SUCCESS != ret)
49. {
50.     printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
51.     return ret;
52. }
53.
54. /* set buffer depth */

```

```

55. stAiChnOutputPort.eModId = E_MI_MODULE_ID_AI;
56. stAiChnOutputPort.u32DevId = AiDevId;
57. stAiChnOutputPort.u32ChnId = AiChn;
58. stAiChnOutputPort.u32PortId = 0;
59. MI_SYS_SetChnOutputPortDepth(&stAiChnOutputPort, 1, 8);
60.
61. /* get port fd */
62. stChnPort.eModId = E_MI_MODULE_ID_AI;
63. stChnPort.u32DevId = AiDevId;
64. stChnPort.u32ChnId = AiChn;
65. stChnPort.u32PortId = 0;
66. ret = MI_SYS_GetFd(&stChnPort, &s32Fd);
67. if (MI_SUCCESS != ret)
68. {
69.     printf("Dev%d Chn%d failed to call MI_SYS_GetFd!!!\n", AiDevId, AiChn);
70.     return ret;
71. }
72.
73. /* select 100ms */
74. FD_ZERO(&readFdSet);
75. FD_SET(s32Fd, &readFdSet);
76. stTimeOut.tv_sec = 0;
77. stTimeOut.tv_usec = 100 * 1000;
78. ret = select(s32Fd + 1, &readFdSet, NULL, NULL, &stTimeOut);
79. if (FD_ISSET(s32Fd, &readFdSet))
80. {
81.     ret = MI_AI_GetFrame(AiDevId, AiChn, &stAiChFrame, &stAecFrame, 0);
82.     if (MI_SUCCESS == ret)
83.     {
84.         /* do something */
85.         MI_AI_ReleaseFrame(AiDevId, AiChn, &stAiChFrame, &stAecFrame);
86.     }
87. }
88.
89. /* disable AI Channel */
90. ret = MI_AI_DisableChn(AiDevId, AiChn);
91. if (MI_SUCCESS != ret)
92. {
93.     printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
94.     return ret;
95. }
96.
97. /* disable AI Device */
98. ret = MI_AI_Disable(AiDevId);
99. if (MI_SUCCESS != ret)
100. {
101.     printf("disable ai %d err:0x%x\n", AiDevId, ret);
102.     return ret;
103. }
104.
105. MI_SYS_Exit();

```

2.1.2 MI_AI_GetPubAttr

➤ 功能

获取 AI 设备属性。

➤ 语法

```
MI_S32 MI_AI_GetPubAttr( MI\_AUDIO\_DEV AiDevId, MI\_AUDIO\_Attr\_t*pstAttr);
```

➤ 形参

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

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照[错误码](#)。} \end{array} \right.$

➤ 依赖

- 头文件: [mi_ai.h](#)
- 库文件: [libmi_ai.a/libmi_ai.so](#)

※ 注意

- 获取的属性为当前配置的属性。
- 如果从来没有配置过属性，则返回失败。

➤ 举例

请参考 [MI_AI_SetPubAttr](#) 举例部分。

2.1.3 MI_AI_Enable

➤ 功能

启用 AI 设备。

➤ 语法

```
MI_S32 MI_AI_Enable(MI\_AUDIO\_DEV AiDevId);
```

➤ 形参

参数名称	描述	输入/输出
AiDevId	音频设备号	输入

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照[错误码](#)。} \end{array} \right.$

➤ 依赖

- 头文件: [mi_ai.h](#)
- 库文件: [libmi_ai.a/libmi_ai.so](#)

※ 注意

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

➤ 举例

请参考 [MI_AI_SetPubAttr](#) 举例部分。

2.1.4 MI_AI_Disable

➤ 功能

禁用 AI 设备。

➤ 语法

```
MI_S32 MI_AI_Disable(MI_AUDIO_DEV AiDevId);
```

➤ 形参

参数名称	描述	输入/输出
AiDevId	音频设备号	输入

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 0} & \text{失败，参照错误码。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so

※ 注意

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

➤ 举例

请参考 [MI_AI_SetPubAttr](#) 举例部分。

2.1.5 MI_AI_EnableChn

➤ 功能

启用 AI 通道

➤ 语法

```
MI_S32 MI_AI_EnableChn(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);
```

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so

※ 注意

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

➤ 举例

请参考 [MI_AI_SetPubAttr](#) 举例部分。

2.1.6 MI_AI_DisableChn

➤ 功能

禁用 AI 通道

➤ 语法

MI_S32 MI_AI_DisableChn(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so

※ 注意

- 如果 AI 通道已经处于禁用状态，则直接返回成功。
- 禁用 AI 通道前必须先禁用该 AI 通道已经使能的音频算法。

➤ 举例

请参考 [MI_AI_SetPubAttr](#) 举例部分。

2.1.7 MI_AI_GetFrame

➤ 功能

获取音频帧

➤ 语法

```
MI_S32 MI_AI_GetFrame(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AUDIO_Frame_t *pstFrm,
MI_AUDIO_AecFrame_t *pstAecFrm, MI_S32 s32MilliSec);
```

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstFrm	音频帧结构体指针。	输出
pstAecFrm	回声消除参考帧结构体指针。	输出
s32MilliSec	获取数据的超时时间 -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 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so

※ 注意

- 在获取音频帧前，必须先使能 AI 通道，否则返回错误。
- 如果需要获取回声消除参考帧，pstAecFrm 不能是空指针，如果不想获取回声抵消参考帧 pstAecFrm 置为空指针即可。
- 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。
- 本接口支持 select/poll 操作，建议使用 select/poll 操作代替 s32MilliSec。

- 举例
请参考 [MI_AI_SetPubAttr](#) 举例部分。

2.1.8 MI_AI_ReleaseFrame

- 功能
释放音频帧
- 语法
MI_S32 MI_AI_ReleaseFrame(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AUDIO_Frame_t *pstFrm, MI_AUDIO_AecFrame_t *pstAecFrm);
- 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。 支持的通道范围由AI设备属性中的最大通道个数u32ChnCnt决定。	输入
pstFrm	音频帧结构体指针。	输入
pstAecFrm	回声抵消参考帧结构体指针。	输入

- 返回值
返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

- 依赖
 - 头文件: mi_ai.h
 - 库文件: libmi_ai.a/libmi_ai.so

- ※ 注意
无

- 举例
请参考 [MI_AI_SetPubAttr](#) 举例部分。

2.1.9 MI_AI_SetChnParam

- 功能
设置 AI 通道参数
- 语法
MI_S32 MI_AI_SetChnParam(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AI_ChnParam_t *pstChnParam);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstChnParam	音频通道参数结构体指针。	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so

※ 注意

需要设置 AI 通道参数，必须先配置和使能 AI 设备。

➤ 举例

下面的代码实现配置和获取 AI 通道参数。

```

1. MI_S32 ret;
2. MI_AUDIO_DEV AiDevId = 0;
3. MI_AI_CHN AiChn = 0;
4. MI_AUDIO_Attr_t stAttr;
5. MI_AI_ChnParam_t stChnParam;
6.
7. MI_SYS_Init();
8.
9. memset(&stAttr, 0x0, sizeof(MI_AUDIO_Attr_t));
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_SLAVE;
14. stAttr.u32PtNumPerFrm = 160;
15. stAttr.u32ChnCnt = 1;
16.
17. /* set public attribute of AI device */
18. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
19. if (MI_SUCCESS != ret)
20. {
21.     printf("set Dev%d attr err:0x%x\n", AiDevId, ret);
22.     return ret;
23. }
24.
25. /* enable AI device */
26. ret = MI_AI_Enable(AiDevId);
27. if (MI_SUCCESS != ret)
28. {
29.     printf("enable Dev%d err:0x%x\n", AiDevId, ret);
30.     return ret;
31. }
32.
33. /* enable AI Channel */

```

```

34. ret = MI_AI_EnableChn(AiDevId, AiChn);
35. if (MI_SUCCESS != ret)
36. {
37.     printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
38.     return ret;
39. }
40.
41. memset(&stChnParam, 0x0, sizeof(stChnParam));
42. stChnParam.stChnGain.bEnableGainSet = TRUE;
43. stChnParam.stChnGain.s16FrontGain = 0;
44. stChnParam.stChnGain.s16RearGain = 0;
45.
46. ret = MI_AI_SetChnParam(AiDevId, AiChn, &stChnParam);
47. if (MI_SUCCESS != ret)
48. {
49.     printf("set Dev%d Chn%d param err:0x%x\n", AiDevId, AiChn, ret);
50.     return ret;
51. }
52.
53. /* get channel param */
54. ret = MI_AI_GetChnParam(AiDevId, AiChn, &stChnParam);
55. if (MI_SUCCESS != ret)
56. {
57.     printf("get Dev%d Chn%d param err:0x%x\n", AiDevId, AiChn, ret);
58.     return ret;
59. }
60.
61. /* disable AI Channel */
62. ret = MI_AI_DisableChn(AiDevId, AiChn);
63. if (MI_SUCCESS != ret)
64. {
65.     printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
66.     return ret;
67. }
68.
69. /* disable AI Device */
70. ret = MI_AI_Disable(AiDevId);
71. if (MI_SUCCESS != ret)
72. {
73.     printf("disable ai %d err:0x%x\n", AiDevId, ret);
74.     return ret;
75. }
76.
77. MI_SYS_Exit();

```

2.1.10 MI_AI_GetChnParam

➤ 功能

获取 AI 通道参数

➤ 语法

MI_S32 MI_AI_GetChnParam(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AI_ChnParam_t *pstChnParam);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstChnParam	音频通道参数结构体指针。	输出

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so

※ 注意

无

➤ 举例

请参考 [MI_AI_SetChnParam](#) 举例部分。

2.1.11 MI_AI_EnableReSmp

➤ 功能

启用 AI 重采样

➤ 语法

MI_S32 MI_AI_EnableReSmp(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AUDIO_SampleRate_e eOutSampleRate);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
eOutSampleRate	音频重采样的输出采样率。	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libSRC_LINUX.so

※ 注意

- 在启用 AI 通道之后, 调用此接口启用重采样功能。
- 若需要使能波束成形功能, 需在使能波束成形功能后再启用重采样。

➤ 举例

下面的代码实现 AI 从 8K 到 16K 的重采样

```
1. MI_S32 ret;
2. MI_AUDIO_DEV AiDevId = 0;
3. MI_AI_CHN AiChn = 0;
4. MI_AUDIO_Attr_t stAttr;
5.
6. MI_SYS_Init();
7.
8. memset(&stAttr, 0x0, sizeof(MI_AUDIO_Attr_t));
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_SLAVE;
13. stAttr.u32PtNumPerFrm = 160;
14. stAttr.u32ChnCnt = 1;
15.
16. /* set public attribute of AI device */
17. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
18. if (MI_SUCCESS != ret)
19. {
20.     printf("set Dev%d attr err:0x%x\n", AiDevId, ret);
21.     return ret;
22. }
23.
24. /* enable AI device */
25. ret = MI_AI_Enable(AiDevId);
26. if (MI_SUCCESS != ret)
27. {
28.     printf("enable Dev%d err:0x%x\n", AiDevId, ret);
29.     return ret;
30. }
31.
32. /* enable AI Channel */
33. ret = MI_AI_EnableChn(AiDevId, AiChn);
34. if (MI_SUCCESS != ret)
35. {
36.     printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
37.     return ret;
38. }
39.
40. ret = MI_AI_EnableReSmp(AiDevId, AiChn, E_MI_AUDIO_SAMPLE_RATE_16000);
41. if (MI_SUCCESS != ret)
42. {
43.     printf("resample Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
44.     return ret;
45. }
46.
47. ret = MI_AI_DisableReSmp(AiDevId, AiChn);
48. if (MI_SUCCESS != ret)
49. {
```

```

50.     printf("disable resample Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
51.     return ret;
52. }
53.
54. /* disable AI Channel */
55. ret = MI_AI_DisableChn(AiDevId, AiChn);
56. if (MI_SUCCESS != ret)
57. {
58.     printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
59.     return ret;
60. }
61.
62. /* disable AI Device */
63. ret = MI_AI_Disable(AiDevId);
64. if (MI_SUCCESS != ret)
65. {
66.     printf("disable ai %d err:0x%x\n", AiDevId, ret);
67.     return ret;
68. }
69.
70. MI_SYS_Exit();
71.

```

2.1.12 MI_AI_DisableReSmp

➤ 功能

禁用 AI 重采样

➤ 语法

MI_S32 MI_AI_DisableReSmp(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 0} & \text{失败，参照错误码。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libSRC_LINUX.so

※ 注意

- 不再使用 AI 重采样功能的话，应该调用此接口将其禁用。

- 举例
请参考 [MI_AI_EnableReSmp](#) 举例部分

2.1.13 MI_AI_SetVqeAttr

- 功能
设置 AI 的声音质量增强功能相关属性
- 语法
MI_S32 MI_AI_SetVqeAttr(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn, MI_AI_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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
AoDevId	用于回声抵消的AO设备号。	输入
AoChn	用于回声抵消的AO通道号。 支持的通道范围由AO设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstVqeConfig	音频输入声音质量增强配置结构体指针	输入

- 返回值
返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 0} & \text{失败，参照错误码。} \end{array} \right.$

- 依赖
 - 头文件: mi_ai.h
 - 库文件: libmi_ai.a/libmi_ai.so libAEC_LINUX.so libAPC_LINUX.so

- ※ 注意
 - 设置 AI 的声音质量增强功能相关属性前，必须先使能对应的 AI 通道。
 - Vqe 包含的所有算法均支持 8/16K 采样率，仅 ANR/AGC/EQ 支持 48K

- 举例
下面代码实现使能和禁用声音质量增强属性

```

1. MI_S32 ret;
2. MI_AUDIO_DEV AiDevId = 0;
3. MI_AI_CHN AiChn = 0;
4. MI_AUDIO_Attr_t stAttr;
5. MI_AI_VqeConfig_t stVqeConfig;
6. MI_U32 u32AecSupfreq[] = {4,6,36,49,50,51};
7. MI_U32 u32AecSupIntensity[] = {5,4,4,5,10,10};
8. MI_S16 s16Compression_ratio_input[] = {-80, -60, -40, -20, 0};
9. MI_S16 s16Compression_ratio_output[] = {-80, -60, -30, -15, -10};

```



```

10.
11. MI_SYS_Init();
12.
13. memset(&stAttr, 0x0, sizeof(MI_AUDIO_Attr_t));
14. stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
15. stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
16. stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
17. stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
18. stAttr.u32PtNumPerFrm = 160;
19. stAttr.u32ChnCnt = 1;
20.
21. /* set public attribute of AI device */
22. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
23. if (MI_SUCCESS != ret)
24. {
25.     printf("set Dev%d attr err:0x%x\n", AiDevId, ret);
26.     return ret;
27. }
28.
29. /* enable AI device */
30. ret = MI_AI_Enable(AiDevId);
31. if (MI_SUCCESS != ret)
32. {
33.     printf("enable Dev%d err:0x%x\n", AiDevId, ret);
34.     return ret;
35. }
36.
37. /* enable AI Channel */
38. ret = MI_AI_EnableChn(AiDevId, AiChn);
39. if (MI_SUCCESS != ret)
40. {
41.     printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
42.     return ret;
43. }
44.
45. /* set vqe attr */
46. memset(&stVqeConfig, 0x0, sizeof(stVqeConfig));
47. stVqeConfig.bHpfOpen = TRUE;
48. stVqeConfig.bAecOpen = TRUE;
49. stVqeConfig.bAnrOpen = TRUE;
50. stVqeConfig.bAgcOpen = TRUE;
51. stVqeConfig.bEqOpen = TRUE;
52. stVqeConfig.u32ChnNum = 1;
53. stVqeConfig.s32WorkSampleRate = 8000;
54. stVqeConfig.s32FrameSample = 128;
55. stVqeConfig.stHpfCfg.eMode = E_MI_AUDIO_ALGORITHM_MODE_USER;
56. stVqeConfig.stHpfCfg.eHpfFreq = E_MI_AUDIO_HP_FREQ_150;
57.
58. stVqeConfig.stAecCfg.bComfortNoiseEnable = TRUE;
59. stVqeConfig.stAecCfg.s16DelaySample = 0;
60. memcpy(stVqeConfig.stAecCfg.u32AecSupfreq, u32AecSupfreq, sizeof(u32AecSupfreq));
61. memcpy(stVqeConfig.stAecCfg.u32AecSupIntensity, u32AecSupIntensity, sizeof(u32AecSupIntensity));
62.
63. stVqeConfig.stAnrCfg.eMode = E_MI_AUDIO_ALGORITHM_MODE_MUSIC;
64. stVqeConfig.stAnrCfg.u32NrIntensity = 30;
65. stVqeConfig.stAnrCfg.u32NrSmoothLevel = 10;
66. stVqeConfig.stAnrCfg.eNrSpeed = E_MI_AUDIO_NR_SPEED_MID;
67.
68. stVqeConfig.stAgcCfg.eMode = E_MI_AUDIO_ALGORITHM_MODE_USER;
69. stVqeConfig.stAgcCfg.stAgcGainInfo.s32GainMax = 15;
70. stVqeConfig.stAgcCfg.stAgcGainInfo.s32GainMin = 0;
71. stVqeConfig.stAgcCfg.stAgcGainInfo.s32GainInit = 0;

```

```

72. stVqeConfig.stAgcCfg.u32DropGainMax = 55;
73. stVqeConfig.stAgcCfg.u32AttackTime = 1;
74. stVqeConfig.stAgcCfg.u32ReleaseTime = 6;
75. memcpy(stVqeConfig.stAgcCfg.s16Compression_ratio_input, s16Compression_ratio_input, sizeof(s
    16Compression_ratio_input));
76. memcpy(stVqeConfig.stAgcCfg.s16Compression_ratio_output, s16Compression_ratio_output, sizeof
    (s16Compression_ratio_output));
77. stVqeConfig.stAgcCfg.s32TargetLevelDb = -3;
78. stVqeConfig.stAgcCfg.s32NoiseGateDb = -80;
79. stVqeConfig.stAgcCfg.u32NoiseGateAttenuationDb = 0;
80.
81. stVqeConfig.stEqCfg.eMode = E_MI_AUDIO_ALGORITHM_MODE_USER;
82.
83. ret = MI_AI_SetVqeAttr(AiDevId, AiChn, 0, 0, &stVqeConfig);
84. if (MI_SUCCESS != ret)
85. {
86.     printf("set vqe attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
87.     return ret;
88. }
89.
90. ret = MI_AI_GetVqeAttr(AiDevId, AiChn, 0, 0, &stVqeConfig);
91. if (MI_SUCCESS != ret)
92. {
93.     printf("get vqe attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
94.     return ret;
95. }
96.
97. /* enable vqe */
98. ret = MI_AI_EnableVqe(AiDevId, AiChn);
99. if (MI_SUCCESS != ret)
100. {
101.     printf("enable vqe Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
102.     return ret;
103. }
104.
105. /* disable vqe */
106. ret = MI_AI_DisableVqe(AiDevId, AiChn);
107. if (MI_SUCCESS != ret)
108. {
109.     printf("disable vqe Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
110.     return ret;
111. }
112.
113. /* disable AI Channel */
114. ret = MI_AI_DisableChn(AiDevId, AiChn);
115. if (MI_SUCCESS != ret)
116. {
117.     printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
118.     return ret;
119. }
120.
121. /* disable AI Device */
122. ret = MI_AI_Disable(AiDevId);
123. if (MI_SUCCESS != ret)
124. {
125.     printf("disable ai %d err:0x%x\n", AiDevId, ret);
126.     return ret;
127. }
128.
129. MI_SYS_Exit();
130.

```

2.1.14 MI_AI_GetVqeAttr

➤ 功能

获取 AI 的声音质量增强功能相关属性。

➤ 语法

```
MI_S32 MI_AI_GetVqeAttr(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AI_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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstVqeConfig	音频输入声音质量增强配置结构体指针	输出

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 0} & \text{失败，参照错误码。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libAEC_LINUX.so libAPC_LINUX.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetVqeAttr](#) 举例部分。

2.1.15 MI_AI_EnableVqe

➤ 功能

使能 AI 的声音质量增强功能。

➤ 语法

```
MI_S32 MI_AI_EnableVqe(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);
```

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libAPC_LINUX.so libAEC_LINUX.so

※ 注意

- 启用声音质量增强功能前必须先使能相对应 AI 通道和设置相对应 AI 通道的声音质量增强功能相关属性。
- 多次使能相同 AI 通道的声音质量增强功能时，返回成功。

➤ 举例

请参考 [MI_AI_SetVqeAttr](#) 举例部分。

2.1.16 MI_AI_DisableVqe

➤ 功能

禁用 AI 的声音质量增强功能。

➤ 语法

MI_S32 MI_AI_DisableVqe(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

- 依赖
 - 头文件: mi_ai.h
 - 库文件: libmi_ai.a/libmi_ai.so libAPC_LINUX.so libAPC_LINUX.so
- ※ 注意
 - 不再使用 AI 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 AI 通道的声音质量增强功能, 返回成功。
- 举例

请参考 [MI_AI_SetVqeAttr](#) 举例部分。

2.1.17 MI_AI_ClrPubAttr

- 功能

清除 AI 设备属性。
- 语法


```
MI_S32 MI_AI_ClrPubAttr(MI_AUDIO_DEV AiDevId);
```

- 形参

参数名称	描述	输入/输出
AiDevId	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_ai.h
 - 库文件: libmi_ai.a/libmi_ai.so
- ※ 注意
 - 清除设备属性前, 需要先停止设备。
- 举例

下面的代码实现了清除设备属性。

```

1. MI_S32 ret;
2. MI_AUDIO_DEV AiDevId = 0;
3. MI_AI_CHN AiChn = 0;
4. MI_AUDIO_Attr_t stAttr;
5.
6. memset(&stAttr, 0x0, sizeof(MI_AUDIO_Attr_t));
7. stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
8. stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
9. stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
10. stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
```

```

11. stAttr.u32PtNumPerFrm = 160;
12. stAttr.u32ChnCnt = 1;
13.
14. /* set public attribute of AI device */
15. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
16. if (MI_SUCCESS != ret)
17. {
18.     printf("set Dev%d attr err:0x%x\n", AiDevId, ret);
19.     return ret;
20. }
21.
22. /* clean public attr of AI device*/
23. ret = MI_AI_ClrPubAttr(AiDevId);
24. if (MI_SUCCESS != ret)
25. {
26.     printf("clean Dev%d attr err:0x%x\n", AiDevId, ret);
27.     return ret;
28. }

```

2.1.18 MI_AI_SaveFile

➤ 功能

开启音频输入保存文件功能

➤ 语法

MI_S32 MI_AI_SaveFile(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AUDIO_SaveFileInfo_t *pstSaveFileInfo);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstSaveFileInfo	音频保存文件属性结构体指针。	输入

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 0} & \text{失败，参照错误码。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so

※ 注意

- 无。

➤ 举例

无。

2.1.19 MI_AI_SetVqeVolume

➤ 功能

设置 AI 通道的音量大小

➤ 语法

MI_S32 MI_AI_SetVqeVolume(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_S32 s32VolumeDb);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
s32VolumeDb	AI通道的音量大小。	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so

※ 注意

下表描述了 s32VolumeDb 在各个 device 下对应的增益（dB）。对于 Amic 和 Linein 设备，s32VolumeDb 表示该设备的模拟增益。对于 Macaron 系列、Pudding 系列、Ispahan 系列芯片 s32VolumeDb 即 Dmic 所对应的增益。

s32VolumeDb	Amic(dB)	Line in(dB)	Pretzel 系列 TAIYAKI 系列 TAKOYAKI 系列 Dmic(dB)
0	-6	-6	0
1	-3	-3	6
2	0	0	12
3	3	3	18
4	6	6	24
5	9	9	
6	12	12	
7	15	15	
8	18		
9	21		
10	24		
11	27		
12	30		

s32VolumeDb	Amic(dB)	Line in(dB)	Pretzel 系列 TAIYAKI 系列 TAKOYAKI 系列 Dmic(dB)
13	33		
14	36		
15	39		
16	42		
17	45		
18	48		
19	51		
20	54		
21	57		

- 举例
下面的代码实现增益设置和获取功能

```

1. MI_S32 ret;
2. MI_AUDIO_DEV AiDevId = 0;
3. MI_AI_CHN AiChn = 0;
4. MI_AUDIO_Attr_t stAttr;
5. MI_S32 s32VolumeDb;
6.
7. MI_SYS_Init();
8.
9. memset(&stAttr, 0x0, sizeof(MI_AUDIO_Attr_t));
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_SLAVE;
14. stAttr.u32PtNumPerFrm = 160;
15. stAttr.u32ChnCnt = 1;
16.
17. /* set public attribute of AI device */
18. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
19. if (MI_SUCCESS != ret)
20. {
21.     printf("set Dev%d attr err:0x%x\n", AiDevId, ret);
22.     return ret;
23. }
24.
25. /* enable AI device */
26. ret = MI_AI_Enable(AiDevId);
27. if (MI_SUCCESS != ret)
28. {
29.     printf("enable Dev%d err:0x%x\n", AiDevId, ret);
30.     return ret;
31. }
32.
33. /* enable AI Channel */
34. ret = MI_AI_EnableChn(AiDevId, AiChn);
35. if (MI_SUCCESS != ret)
36. {
37.     printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
38.     return ret;
39. }
40.

```



```

41. /* set channel volume */
42. s32VolumeDb = 12;
43. ret = MI_AI_SetVqeVolume(AiDevId, AiChn, s32VolumeDb);
44. if (MI_SUCCESS != ret)
45. {
46.     printf("set volume Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
47.     return ret;
48. }
49.
50. /* get channel volume */
51. ret = MI_AI_GetVqeVolume(AiDevId, AiChn, &s32VolumeDb);
52. if (MI_SUCCESS != ret)
53. {
54.     printf("get volume Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
55.     return ret;
56. }
57.
58. /* disable AI Channel */
59. ret = MI_AI_DisableChn(AiDevId, AiChn);
60. if (MI_SUCCESS != ret)
61. {
62.     printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
63.     return ret;
64. }
65.
66. /* disable AI Device */
67. ret = MI_AI_Disable(AiDevId);
68. if (MI_SUCCESS != ret)
69. {
70.     printf("disable ai %d err:0x%x\n", AiDevId, ret);
71.     return ret;
72. }
73.
74. MI_SYS_Exit();
75.

```

2.1.20 MI_AI_GetVqeVolume

➤ 功能

获取 AI 通道的音量大小

➤ 语法

MI_S32 MI_AI_GetVqeVolume(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_S32 *ps32VolumeDb);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
ps32VolumeDb	AI通道的音量大小。	输出

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetVqeVolume](#) 举例部分

2.1.21 MI_AI_SetAencAttr

➤ 功能

设置 AI 编码功能相关属性

➤ 语法

MI_S32 MI_AI_SetAencAttr (MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AI_AencConfig_t *pstAencConfig);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstAencConfig	音频编码配置结构体指针	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libg711.so libg726.so

※ 注意

需要设置编码功能时，需要先使能 AI 通道。

➤ 举例

下面的代码实现使能和禁用编码参数的功能。

```
1. MI_S32 ret;
2. MI_AUDIO_DEV AiDevId = 0;
3. MI_AI_CHN AiChn = 0;
4. MI_AUDIO_Attr_t stAttr;
5. MI_AI_AencConfig_t stAiAencConfig;
6.
7. MI_SYS_Init();
8.
9. memset(&stAttr, 0x0, sizeof(MI_AUDIO_Attr_t));
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_SLAVE;
14. stAttr.u32PtNumPerFrm = 160;
15. stAttr.u32ChnCnt = 1;
16.
17. /* set public attribute of AI device */
18. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
19. if (MI_SUCCESS != ret)
20. {
21.     printf("set Dev%d attr err:0x%x\n", AiDevId, ret);
22.     return ret;
23. }
24.
25. /* enable AI device */
26. ret = MI_AI_Enable(AiDevId);
27. if (MI_SUCCESS != ret)
28. {
29.     printf("enable Dev%d err:0x%x\n", AiDevId, ret);
30.     return ret;
31. }
32.
33. /* enable AI Channel */
34. ret = MI_AI_EnableChn(AiDevId, AiChn);
35. if (MI_SUCCESS != ret)
36. {
37.     printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
38.     return ret;
39. }
40.
41. /* set aenc attr */
42. memset(&stAiAencConfig, 0x0, sizeof(stAiAencConfig));
43. stAiAencConfig.eAencType = E_MI_AUDIO_AENC_TYPE_G711A;
44. ret = MI_AI_SetAencAttr(AiDevId, AiChn, &stAiAencConfig);
45. if (MI_SUCCESS != ret)
46. {
47.     printf("set aenc attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
48.     return ret;
49. }
50.
51. /* get aenc attr */
52. memset(&stAiAencConfig, 0x0, sizeof(stAiAencConfig));
53. ret = MI_AI_GetAencAttr(AiDevId, AiChn, &stAiAencConfig);
54. if (MI_SUCCESS != ret)
55. {
56.     printf("get aenc attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
57.     return ret;
58. }
59.
```

```
60. /* enable aenc */
61. ret = MI_AI_EnableAenc(AiDevId, AiChn);
62. if (MI_SUCCESS != ret)
63. {
64.     printf("enable aenc Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
65.     return ret;
66. }
67.
68. /* disable aenc */
69. ret = MI_AI_DisableAenc(AiDevId, AiChn);
70. if (MI_SUCCESS != ret)
71. {
72.     printf("disable aenc Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
73.     return ret;
74. }
75.
76. /* disable AI Channel */
77. ret = MI_AI_DisableChn(AiDevId, AiChn);
78. if (MI_SUCCESS != ret)
79. {
80.     printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
81.     return ret;
82. }
83.
84. /* disable AI Device */
85. ret = MI_AI_Disable(AiDevId);
86. if (MI_SUCCESS != ret)
87. {
88.     printf("disable ai %d err:0x%x\n", AiDevId, ret);
89.     return ret;
90. }
91.
92. MI_SYS_Exit();
93.
```

2.1.22 MI_AI_GetAencAttr

➤ 功能

获取 AI 编码功能相关属性

➤ 语法

MI_S32 MI_AI_GetAencAttr (MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AI_AencConfig_t *pstAencConfig);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstAencConfig	音频编码配置结构体指针	输出

- 返回值

{	0	成功。
}	非 0	失败，参照 错误码 。
- 依赖
 - 头文件: mi_ai.h
 - 库文件: libmi_ai.a/libmi_ai.so libg711.so libg726.so
- ※ 注意

无。
- 举例

请参考 [MI_AI_SetAencAttr](#) 举例部分。

2.1.23 MI_AI_EnableAenc

- 功能

使能 AI 编码功能。
- 语法

MI_S32 MI_AI_EnableAenc (MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);
- 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

- 返回值

{	0	成功。
}	非 0	失败，参照 错误码 。
- 依赖
 - 头文件: mi_ai.h
 - 库文件: libmi_ai.a/libmi_ai.so libg711.so libg726.so
- ※ 注意

在使能编码功能前，必须先使能相应的 AI 通道和设置编码属性。
- 举例

请参考 [MI_AI_SetAencAttr](#) 举例部分。

2.1.24 MI_AI_DisableAenc

➤ 功能

禁用 AI 编码功能。

➤ 语法

MI_S32 MI_AI_DisableAenc ([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 0} & \text{失败，参照[错误码](#)。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libg711.so libg726.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetAencAttr](#) 举例部分。

2.1.25 MI_AI_SetAedAttr

➤ 功能

设置 AI 声音事件检测功能。

➤ 语法

MI_S32 MI_AI_SetAedAttr([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, [MI_AI_AedConfig_t](#) *pstAedConfig);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstAedConfig	声音事件检测配置结构体指针	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 0} & \text{失败，参照[错误码](#)。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libAED_LINUX.so

※ 注意

要设置声音事件检测功能参数，需要先使能 AI 通道。

➤ 举例

下面的代码实现设置声音检测参数、使能声音检测、获取声音检测结果等功能。

```
1. MI_S32 ret;
2. MI_AUDIO_Attr_t stAttr;
3. MI_AUDIO_Dev AiDevId = 0;
4. MI_AI_CHN AiChn = 0;
5. MI_SYS_ChnPort_t stChnPort;
6. MI_S32 s32Fd;
7. fd_set readFdSet;
8. struct timeval stTimeOut;
9. MI_AUDIO_Frame_t stAiChFrame;
10. MI_AUDIO_AecFrame_t stAecFrame;
11. MI_SYS_ChnPort_t stAiChnOutputPort;
12. MI_AI_AedConfig_t stAiAedConfig;
13. MI_AI_AedResult_t stAedResult;
14.
15. MI_SYS_Init();
16.
17. stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
18. stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
19. stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
20. stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
21. stAttr.u32PtNumPerFrm = 160;
22. stAttr.u32ChnCnt = 1;
23.
24. /* set public attribute of AI device */
25. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
26. if (MI_SUCCESS != ret)
27. {
28.     printf("set ai %d attr err:0x%x\n", AiDevId, ret);
29.     return ret;
30. }
31.
32. /* get public attribute of AI device */
33. ret = MI_AI_GetPubAttr(AiDevId, &stAttr);
34. if (MI_SUCCESS != ret)
35. {
36.     printf("get ai %d attr err:0x%x\n", AiDevId, ret);
37.     return ret;
38. }
39.
40. /* enable AI device */
41. ret = MI_AI_Enable(AiDevId);
```

```

42. if (MI_SUCCESS != ret)
43. {
44.     printf("enable ai %d err:0x%x\n", AiDevId, ret);
45.     return ret;
46. }
47.
48. /* enable AI Channel */
49. ret = MI_AI_EnableChn(AiDevId, AiChn);
50. if (MI_SUCCESS != ret)
51. {
52.     printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
53.     return ret;
54. }
55.
56. memset(&stAiAedConfig, 0x0, sizeof(stAiAedConfig));
57. stAiAedConfig.bEnableNr = TRUE;
58. stAiAedConfig.eSensitivity = E_MI_AUDIO_AED_SEN_HIGH;
59. stAiAedConfig.s32OperatingPoint = -5;
60. stAiAedConfig.s32VadThresholdDb = -40;
61. stAiAedConfig.s32LsdThresholdDb = -15;
62.
63. /* set aed attr */
64. ret = MI_AI_SetAedAttr(AiDevId, AiChn, &stAiAedConfig);
65. if (MI_SUCCESS != ret)
66. {
67.     printf("set aed attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
68.     return ret;
69. }
70.
71. /* get aed attr */
72. ret = MI_AI_GetAedAttr(AiDevId, AiChn, &stAiAedConfig);
73. if (MI_SUCCESS != ret)
74. {
75.     printf("get aed attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
76.     return ret;
77. }
78.
79. /* enable aed */
80. ret = MI_AI_EnableAed(AiDevId, AiChn);
81. if (MI_SUCCESS != ret)
82. {
83.     printf("enable aed Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
84.     return ret;
85. }
86.
87. /* set buffer depth */
88. stAiChnOutputPort.eModId = E_MI_MODULE_ID_AI;
89. stAiChnOutputPort.u32DevId = AiDevId;
90. stAiChnOutputPort.u32ChnId = AiChn;
91. stAiChnOutputPort.u32PortId = 0;
92. MI_SYS_SetChnOutputPortDepth(&stAiChnOutputPort, 1, 8);
93.
94. /* get port fd */
95. stChnPort.eModId = E_MI_MODULE_ID_AI;
96. stChnPort.u32DevId = AiDevId;
97. stChnPort.u32ChnId = AiChn;
98. stChnPort.u32PortId = 0;
99. ret = MI_SYS_GetFd(&stChnPort, &s32Fd);
100. if (MI_SUCCESS != ret)
101. {
102.     printf("Dev%d Chn%d failed to call MI_SYS_GetFd!!!\n", AiDevId, AiChn);
103.     return ret;
104. }

```



```

105.
106.  /* select 100ms */
107.  FD_ZERO(&readFdSet);
108.  FD_SET(s32Fd, &readFdSet);
109.  stTimeOut.tv_sec = 0;
110.  stTimeOut.tv_usec = 100 * 1000;
111.  ret = select(s32Fd + 1, &readFdSet, NULL, NULL, &stTimeOut);
112.  if (FD_ISSET(s32Fd, &readFdSet))
113.  {
114.      ret = MI_AI_GetFrame(AiDevId, AiChn, &stAiChFrame, &stAecFrame, 0);
115.      if (MI_SUCCESS == ret)
116.      {
117.          /* get aed result */
118.          MI_AI_GetAedResult(AiDevId, AiChn, &stAedResult);
119.
120.          MI_AI_ReleaseFrame(AiDevId, AiChn, &stAiChFrame, &stAecFrame);
121.      }
122.  }
123.
124.  /* disable aed */
125.  ret = MI_AI_DisableAed(AiDevId, AiChn);
126.  if (MI_SUCCESS != ret)
127.  {
128.      printf("disable aed Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
129.      return ret;
130.  }
131.
132.  /* disable AI Channel */
133.  ret = MI_AI_DisableChn(AiDevId, AiChn);
134.  if (MI_SUCCESS != ret)
135.  {
136.      printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
137.      return ret;
138.  }
139.
140.  /* disable AI Device */
141.  ret = MI_AI_Disable(AiDevId);
142.  if (MI_SUCCESS != ret)
143.  {
144.      printf("disable ai %d err:0x%x\n", AiDevId, ret);
145.      return ret;
146.  }
147.
148.  MI_SYS_Exit();

```

2.1.26 MI_AI_GetAedAttr

➤ 功能

获取 AI 声音事件检测功能配置。

➤ 语法

MI_S32 MI_AI_GetAedAttr([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, [MI_AI_AedConfig_t](#) *pstAedConfig);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstAedConfig	声音事件检测配置结构体指针	输出

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libAED_LINUX.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetAedAttr](#) 举例部分

2.1.27 MI_AI_EnableAed

➤ 功能

使能 AI 声音事件检测功能。

➤ 语法

MI_S32 MI_AI_EnableAed([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libAED_LINUX.so

※ 注意

在使能声音事件检测功能前，必须先使能 AI 通道和设置声音事件检测属性。

➤ 举例

请参考 [MI_AI_SetAedAttr](#) 举例部分

2.1.28 MI_AI_DisableAed

➤ 功能

禁止 AI 声音事件检测功能。

➤ 语法

MI_S32 MI_AI_DisableAed([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libAED_LINUX.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetAedAttr](#) 举例部分

2.1.29 MI_AI_GetAedResult

➤ 功能

获取 AI 声音事件检测结果。

➤ 语法

MI_S32 MI_AI_GetAedResult([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, [MI_AI_AedResult_t](#) *pstAedResult);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstAedResult	声音事件检测结果结构体指针	输出

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libAED_LINUX.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetAedAttr](#) 举例部分

2.1.30 MI_AI_SetExtAecChn

➤ 功能

设置回声消除功能参考的 AI 通道。

➤ 语法

MI_S32 MI_AI_SetExtAecChn([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, [MI_AI_CHN](#) AiAECSndChn);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
AiAECSndChn	参考的AI通道号	输入

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so

※ 注意

设置外部回声参考通道时，需要在通道使能前进行设置，目前仅支持在 **Mono** 模式下调用此接口。

➤ 举例

下面的代码进行外部 **AEC** 参考通道的设置。

```
1. MI_S32 ret;
2. MI_AUDIO_Attr_t stAttr;
3. MI_AUDIO_Dev AiDevId = 0;
4. MI_AI_CHN AiChn = 0;
5. MI_AI_CHN ExtAecChn = 0;
6.
7. MI_SYS_Init();
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_SLAVE;
13. stAttr.u32PtNumPerFrm = 160;
14. stAttr.u32ChnCnt = 1;
15.
16. /* set public attribute of AI device */
17. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
18. if (MI_SUCCESS != ret)
19. {
20.     printf("set ai %d attr err:0x%x\n", AiDevId, ret);
21.     return ret;
22. }
23.
24. /* get public attribute of AI device */
25. ret = MI_AI_GetPubAttr(AiDevId, &stAttr);
26. if (MI_SUCCESS != ret)
27. {
28.     printf("get ai %d attr err:0x%x\n", AiDevId, ret);
29.     return ret;
30. }
31.
32. /* enable AI device */
33. ret = MI_AI_Enable(AiDevId);
34. if (MI_SUCCESS != ret)
35. {
36.     printf("enable ai %d err:0x%x\n", AiDevId, ret);
37.     return ret;
38. }
39.
40.
41. /* set ext Aec Chn */
42. ret = MI_AI_SetExtAecChn(AiDevId, AiChn, ExtAecChn);
43. if (MI_SUCCESS != ret)
44. {
45.     printf("set ext aec chn Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
46.     return ret;
47. }
48.
49. /* enable AI Channel */
```

```
50. ret = MI_AI_EnableChn(AiDevId, AiChn);
51. if (MI_SUCCESS != ret)
52. {
53.     printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
54.     return ret;
55. }
56.
57. /* disable AI Channel */
58. ret = MI_AI_DisableChn(AiDevId, AiChn);
59. if (MI_SUCCESS != ret)
60. {
61.     printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
62.     return ret;
63. }
64.
65. /* disable AI Device */
66. ret = MI_AI_Disable(AiDevId);
67. if (MI_SUCCESS != ret)
68. {
69.     printf("disable ai %d err:0x%x\n", AiDevId, ret);
70.     return ret;
71. }
72.
73. MI_SYS_Exit();
```

2.1.31 MI_AI_SetSslInitAttr

➤ 功能

设置声源定位功能的初始化参数。

➤ 语法

MI_S32 MI_AI_SetSslInitAttr([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, [MI_AI_SslInitAttr_t](#)*
pstSslInitAttr);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstSslInitAttr	声源定位功能初始化结构体指针	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

※ 注意

声源定位的初始化参数只能在使能前进行设置，使能后不能再进行设置。

➤ 举例

下面的代码实现声源定位参数的设置，使能声源定位和获取声源定位结果。

```

1. MI_S32 ret;
2. MI_AUDIO_Attr_t stAttr;
3. MI_AUDIO_Dev AiDevId = 0;
4. MI_AI_CHN AiChn = 0;
5. MI_SYS_ChnPort_t stChnPort;
6. MI_S32 s32Fd;
7. fd_set readFdSet;
8. struct timeval stTimeOut;
9. MI_AUDIO_Frame_t stAiChFrame;
10. MI_AUDIO_AecFrame_t stAecFrame;
11. MI_SYS_ChnPort_t stAiChnOutputPort;
12. MI_S32 s32Doa;
13.
14. MI_AI_SslInitAttr_t stSslInit = {
15.     .bBfMode = FALSE,
16.     .u32MicDistance = 3,
17. };
18.
19. MI_AI_SslConfigAttr_t stSslConfig = {
20.     .s32Temperature = 25,
21.     .s32NoiseGateDbfs = -40,
22.     .s32DirectionFrameNum = 300,
23. };
24.
25. MI_AI_SslInitAttr_t stAiGetSslInitAttr;
26. MI_AI_SslConfigAttr_t stAiGetSslConfigAttr;
27.
28. MI_SYS_Init();
29.
30. stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
31. stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
32. stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_STEREO;
33. stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
34. stAttr.u32PtNumPerFrm = 160;
35. stAttr.u32ChnCnt = 1;
36.
37. /* set public attribute of AI device */
38. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
39. if (MI_SUCCESS != ret)
40. {
41.     printf("set ai %d attr err:0x%x\n", AiDevId, ret);
42.     return ret;
43. }
44.
45. /* get public attribute of AI device */
46. ret = MI_AI_GetPubAttr(AiDevId, &stAttr);
47. if (MI_SUCCESS != ret)
48. {
49.     printf("get ai %d attr err:0x%x\n", AiDevId, ret);
50.     return ret;
51. }
52.
53. /* enable AI device */
54. ret = MI_AI_Enable(AiDevId);
55. if (MI_SUCCESS != ret)
56. {

```

```

57.     printf("enable ai %d err:0x%x\n", AiDevId, ret);
58.     return ret;
59. }
60.
61. /* enable AI Channel */
62. ret = MI_AI_EnableChn(AiDevId, AiChn);
63. if (MI_SUCCESS != ret)
64. {
65.     printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
66.     return ret;
67. }
68.
69. /* set ssl init attr */
70. ret = MI_AI_SetSslInitAttr(AiDevId, AiChn, &stSslInit);
71. if (MI_SUCCESS != ret)
72. {
73.     printf("set ssl init attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
74.     return ret;
75. }
76.
77. /* get ssl init attr */
78. ret = MI_AI_GetSslInitAttr(AiDevId, AiChn, &stAiGetSslInitAttr);
79. if (MI_SUCCESS != ret)
80. {
81.     printf("get ssl init attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
82.     return ret;
83. }
84.
85. /* set ssl config attr */
86. ret = MI_AI_SetSslConfigAttr(AiDevId, AiChn, &stSslConfig);
87. if (MI_SUCCESS != ret)
88. {
89.     printf("set ssl config attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
90.     return ret;
91. }
92.
93. /* get ssl config attr */
94. ret = MI_AI_GetSslConfigAttr(AiDevId, AiChn, &stAiGetSslConfigAttr);
95. if (MI_SUCCESS != ret)
96. {
97.     printf("get ssl config attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
98.     return ret;
99. }
100.
101. /* enable ssl */
102. ret = MI_AI_EnableSsl(AiDevId, AiChn);
103. if (MI_SUCCESS != ret)
104. {
105.     printf("enable ssl Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
106.     return ret;
107. }
108.
109. /* set buffer depth */
110. stAiChnOutputPort.eModId = E_MI_MODULE_ID_AI;
111. stAiChnOutputPort.u32DevId = AiDevId;
112. stAiChnOutputPort.u32ChnId = AiChn;
113. stAiChnOutputPort.u32PortId = 0;
114. MI_SYS_SetChnOutputPortDepth(&stAiChnOutputPort, 1, 8);
115.
116. /* get port fd */
117. stChnPort.eModId = E_MI_MODULE_ID_AI;
118. stChnPort.u32DevId = AiDevId;
119. stChnPort.u32ChnId = AiChn;

```



```

120. stChnPort.u32PortId = 0;
121. ret = MI_SYS_GetFd(&stChnPort, &s32Fd);
122. if (MI_SUCCESS != ret)
123. {
124.     printf("Dev%d Chn%d failed to call MI_SYS_GetFd!!!\n", AiDevId, AiChn);
125.     return ret;
126. }
127.
128. /* select 100ms */
129. FD_ZERO(&readFdSet);
130. FD_SET(s32Fd, &readFdSet);
131. stTimeOut.tv_sec = 0;
132. stTimeOut.tv_usec = 100 * 1000;
133. ret = select(s32Fd + 1, &readFdSet, NULL, NULL, &stTimeOut);
134. if (FD_ISSET(s32Fd, &readFdSet))
135. {
136.     ret = MI_AI_GetFrame(AiDevId, AiChn, &stAiChFrame, &stAecFrame, 0);
137.     if (MI_SUCCESS == ret)
138.     {
139.         /* get ssl result */
140.         MI_AI_GetSslDoa(AiDevId, AiChn, &s32Doa);
141.         MI_AI_ReleaseFrame(AiDevId, AiChn, &stAiChFrame, &stAecFrame);
142.     }
143. }
144.
145. /* disable ssl */
146. ret = MI_AI_DiableSsl(AiDevId, AiChn);
147. if (MI_SUCCESS != ret)
148. {
149.     printf("disable ssl Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
150.     return ret;
151. }
152.
153. /* disable AI Channel */
154. ret = MI_AI_DisableChn(AiDevId, AiChn);
155. if (MI_SUCCESS != ret)
156. {
157.     printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
158.     return ret;
159. }
160.
161. /* disable AI Device */
162. ret = MI_AI_Disable(AiDevId);
163. if (MI_SUCCESS != ret)
164. {
165.     printf("disable ai %d err:0x%x\n", AiDevId, ret);
166.     return ret;
167. }
168.
169. MI_SYS_Exit();

```

2.1.32 MI_AI_GetSslInitAttr

➤ 功能

获取声源定位功能的初始化参数。

➤ 语法

MI_S32 MI_AI_GetSslInitAttr([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, [MI_AI_SslInitAttr_t](#)* pstSslInitAttr);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数u32ChnCnt决定。	输入
pstSslInitAttr	声源定位功能初始化结构体指针	输出

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetSslInitAttr](#) 举例部分

2.1.33 MI_AI_SetSslConfigAttr

➤ 功能

设置声源定位功能的配置参数。

➤ 语法

MI_S32 MI_AI_SetSslConfigAttr([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, [MI_AI_SslConfigAttr_t](#)* pstSslConfigAttr);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数u32ChnCnt决定。	输入
pstSslConfigAttr	声源定位功能配置结构体指针	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

※ 注意

可以对每一帧数据进行设置。

➤ 举例

请参考 [MI_AI_SetSslInitAttr](#) 举例部分

2.1.34 MI_AI_GetSslConfigAttr

➤ 功能

获取声源定位功能的配置参数。

➤ 语法

MI_S32 MI_AI_GetSslConfigAttr([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, [MI_AI_SslConfigAttr_t](#)* pstSslConfigAttr);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstSslConfigAttr	声源定位功能配置结构体指针	输出

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetSslInitAttr](#) 举例部分

2.1.35 MI_AI_EnableSsl

➤ 功能

使能声源定位功能。

➤ 语法

MI_S32 MI_AI_EnableSsl([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

※ 注意

在使能声源定位功能前，必须先使能 AI 通道以及设置声源定位的初始化参数和配置参数。

➤ 举例

请参考 [MI_AI_SetSslInitAttr](#) 举例部分

2.1.36 MI_AI_DisableSsl

➤ 功能

禁止声源定位功能。

➤ 语法

MI_S32 MI_AI_DisableSsl([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetSslInitAttr](#) 举例部分

2.1.37 MI_AI_GetSslDoa

➤ 功能

获取声源定位功能的检测结果。

➤ 语法

MI_S32 MI_AI_GetSslDoa([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, MI_S32 *ps32SslDoa);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
ps32SslDoa	声源定位功能的检测结果	输出

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetSslInitAttr](#) 举例部分

2.1.38 MI_AI_SetBfInitAttr

➤ 功能

设置波束成形功能的初始化参数。

➤ 语法

MI_S32 MI_AI_SetBfInitAttr([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, [MI_AI_BfInitAttr_t](#)* pstBfInitAttr);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstBfInitAttr	波束成形功能初始化结构体指针	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照[错误码](#)。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.so/libmi_ai.a libBF_LINUX.so

※ 注意

波束成形初始化参数只能在使能前进行设置。

➤ 举例

下面的代码实现波束成形功能的参数设置和使能。

```
1. MI_S32 ret;
2. MI_AUDIO_Attr_t stAttr;
3. MI_AUDIO_Dev AiDevId = 0;
4. MI_AI_CHN AiChn = 0;
5. MI_AI_BfInitAttr_t stBfInit = {
6.     .u32ChanCnt = 2,
7.     .u32MicDistance = 3,
8. };
9. MI_AI_BfConfigAttr_t stBfConfig = {
10.     .s32Temperature = 25,
11.     .s32NoiseGateDbfs = -40,
12.     .s32NoiseSuppressionMode = 8,
13.     .s32NoiseEstimation = 1,
14.     .outputGain = 0.7,
15. };
16. MI_AI_BfInitAttr_t stAiGetBfInitAttr;
17. MI_AI_BfConfigAttr_t stAiGetBfConfigAttr;
18. MI_BOOL bAiSetBfDoa = TRUE;
19. MI_S32 s32AiBfDoa = 0;
20.
```

```

21. MI_SYS_Init();
22.
23. stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
24. stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
25. stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_STEREO;
26. stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
27. stAttr.u32PtNumPerFrm = 160;
28. stAttr.u32ChnCnt = 1;
29.
30. /* set public attribute of AI device */
31. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
32. if (MI_SUCCESS != ret)
33. {
34.     printf("set ai %d attr err:0x%x\n", AiDevId, ret);
35.     return ret;
36. }
37.
38. /* get public attribute of AI device */
39. ret = MI_AI_GetPubAttr(AiDevId, &stAttr);
40. if (MI_SUCCESS != ret)
41. {
42.     printf("get ai %d attr err:0x%x\n", AiDevId, ret);
43.     return ret;
44. }
45.
46. /* enable AI device */
47. ret = MI_AI_Enable(AiDevId);
48. if (MI_SUCCESS != ret)
49. {
50.     printf("enable ai %d err:0x%x\n", AiDevId, ret);
51.     return ret;
52. }
53.
54. /* enable AI Channel */
55. ret = MI_AI_EnableChn(AiDevId, AiChn);
56. if (MI_SUCCESS != ret)
57. {
58.     printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
59.     return ret;
60. }
61.
62. /* set bf init attr */
63. ret = MI_AI_SetBfInitAttr(AiDevId, AiChn, &stBfInit);
64. if (MI_SUCCESS != ret)
65. {
66.     printf("set bf init attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
67.     return ret;
68. }
69.
70. /* get bf init attr */
71. ret = MI_AI_GetBfInitAttr(AiDevId, AiChn, &stAiGetBfInitAttr);
72. if (MI_SUCCESS != ret)
73. {
74.     printf("get bf init attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
75.     return ret;
76. }
77.
78. /* set bf config attr */
79. ret = MI_AI_SetBfConfigAttr(AiDevId, AiChn, &stBfConfig);
80. if (MI_SUCCESS != ret)
81. {
82.     printf("set bf config attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
83.     return ret;

```

```

84. }
85.
86. /* get bf config attr */
87. ret = MI_AI_GetBfConfigAttr(AiDevId, AiChn, &stAiGetBfConfigAttr);
88. if (MI_SUCCESS != ret)
89. {
90.     printf("get bf config attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
91.     return ret;
92. }
93.
94. /* set bf angle */
95. if (bAiSetBfDoa)
96. {
97.     ret = MI_AI_SetBfAngle(AiDevId, AiChn, s32AiBfDoa);
98.     if (MI_SUCCESS != ret)
99.     {
100.         printf("set bf doa Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
101.         return ret;
102.     }
103. }
104.
105. /* enable bf */
106. ret = MI_AI_EnableBf(AiDevId, AiChn);
107. if (MI_SUCCESS != ret)
108. {
109.     printf("enable bf Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
110.     return ret;
111. }
112.
113. /* do something */
114.
115. /* enable bf */
116. ret = MI_AI_DisableBf(AiDevId, AiChn);
117. if (MI_SUCCESS != ret)
118. {
119.     printf("disable bf Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
120.     return ret;
121. }
122.
123. /* disable AI Channel */
124. ret = MI_AI_DisableChn(AiDevId, AiChn);
125. if (MI_SUCCESS != ret)
126. {
127.     printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
128.     return ret;
129. }
130.
131. /* disable AI Device */
132. ret = MI_AI_Disable(AiDevId);
133. if (MI_SUCCESS != ret)
134. {
135.     printf("disable ai %d err:0x%x\n", AiDevId, ret);
136.     return ret;
137. }
138.
139. MI_SYS_Exit();

```


2.1.39 MI_AI_GetBfInitAttr

➤ 功能

获取波束成形功能的初始化参数。

➤ 语法

```
MI_S32 MI_AI_GetBfInitAttr(MI\_AUDIO\_DEV AiDevId, MI\_AI\_CHN AiChn, MI\_AI\_BfInitAttr\_t*  
pstBfInitAttr);
```

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstBfInitAttr	波束成形功能初始化结构体指针	输出

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.so/libmi_ai.a libBF_LINUX.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetBfInitAttr](#) 举例部分。

2.1.40 MI_AI_SetBfConfigAttr

➤ 功能

设置波束成形功能的配置参数。

➤ 语法

```
MI_S32 MI_AI_SetBfConfigAttr(MI\_AUDIO\_DEV AiDevId, MI\_AI\_CHN AiChn, MI\_AI\_BfConfigAttr\_t*  
pstBfConfigAttr);
```

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pstBfInitAttr	波束成形功能配置结构体指针	输入

- 返回值
- | | |
|-----|--|
| 返回值 | $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$ |
|-----|--|
- 依赖
- 头文件: mi_ai.h
 - 库文件: libmi_ai.so/libmi_ai.a libBF_LINUX.so
- ※ 注意
- 无。
- 举例
- 请参考 [MI_AI_SetBfInitAttr](#) 举例部分。

2.1.41 MI_AI_GetBfConfigAttr

- 功能
- 获取波束成形功能的配置参数。
- 语法
- MI_S32 MI_AI_GetBfConfigAttr([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, [MI_AI_BfConfigAttr_t](#)* pstBfConfigAttr);
- 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。
支持的通道范围由AI设备属性中的最大通道个数
u32ChnCnt决定。 | 输入 |
| pstBfConfigAttr | 波束成形功能配置结构体指针 | 输出 |
- 返回值
- | | |
|-----|--|
| 返回值 | $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$ |
|-----|--|
- 依赖
- 头文件: mi_ai.h
 - 库文件: libmi_ai.so/libmi_ai.a libBF_LINUX.so
- ※ 注意
- 无。

- 举例
请参考 [MI_AI_SetBfInitAttr](#) 举例部分。

2.1.42 MI_AI_EnableBf

- 功能
使能波束成形功能。
- 语法
MI_S32 MI_AI_EnableBf([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn);
- 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

- 返回值

返回值	$\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{array} \right.$
-----	--
- 依赖
 - 头文件: mi_ai.h
 - 库文件: libmi_ai.so/libmi_ai.a libBF_LINUX.so
- ※ 注意
无。
- 举例
请参考 [MI_AI_SetBfInitAttr](#) 举例部分。

2.1.43 MI_AI_DisableBf

- 功能
禁止波束成形功能。
- 语法
MI_S32 MI_AI_DisableBf([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.so/libmi_ai.a libBF_LINUX.so

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetBfInitAttr](#) 举例部分。

2.1.44 MI_AI_SetBfAngle

➤ 功能

设置波束成形功能的作用角度。

➤ 语法

MI_S32 MI_AI_SetBfAngle([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, MI_S32 s32BfAngle);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
s32BfAngle	角度	输入

➤ 返回值

返回值 $\begin{cases} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{cases}$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.so/libmi_ai.a libBF_LINUX.so

※ 注意
无。

➤ 举例
请参考 [MI_AI_SetBfInitAttr](#) 举例部分。

2.1.45 MI_AI_SetMute

➤ 功能
设置 AI 通道的静音参数。

➤ 语法
MI_S32 MI_AI_SetMute([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, MI_BOOL bMute);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
bMute	静音参数	输入

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 } 0 & \text{失败，参照错误码。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.so/libmi_ai.a

※ 注意
无。

➤ 举例
请下面的代码实现设置和获取 AI 通道的静音状态。

```
1. MI_S32 ret;
2. MI_AUDIO_Attr_t stAttr;
3. MI_AUDIO_DEV AiDevId = 0;
4. MI_AI_CHN AiChn = 0;
5. MI_BOOL bMute = TRUE;
6.
7. MI_SYS_Init();
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_STEREO;
12. stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
```

```

13. stAttr.u32PtNumPerFrm = 160;
14. stAttr.u32ChnCnt = 1;
15.
16. /* set public attribute of AI device */
17. ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
18. if (MI_SUCCESS != ret)
19. {
20.     printf("set ai %d attr err:0x%x\n", AiDevId, ret);
21.     return ret;
22. }
23.
24. /* get public attribute of AI device */
25. ret = MI_AI_GetPubAttr(AiDevId, &stAttr);
26. if (MI_SUCCESS != ret)
27. {
28.     printf("get ai %d attr err:0x%x\n", AiDevId, ret);
29.     return ret;
30. }
31.
32. /* enable AI device */
33. ret = MI_AI_Enable(AiDevId);
34. if (MI_SUCCESS != ret)
35. {
36.     printf("enable ai %d err:0x%x\n", AiDevId, ret);
37.     return ret;
38. }
39.
40. /* enable AI Channel */
41. ret = MI_AI_EnableChn(AiDevId, AiChn);
42. if (MI_SUCCESS != ret)
43. {
44.     printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
45.     return ret;
46. }
47.
48. /* set mute status */
49. ret = MI_AI_SetMute(AiDevId, AiChn, bMute);
50. if (MI_SUCCESS != ret)
51. {
52.     printf("set mute status Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
53.     return ret;
54. }
55.
56. /* get mute status */
57. ret = MI_AI_GetMute(AiDevId, AiChn, &bMute);
58. if (MI_SUCCESS != ret)
59. {
60.     printf("get mute status Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
61.     return ret;
62. }
63.
64. /* disable AI Channel */
65. ret = MI_AI_DisableChn(AiDevId, AiChn);
66. if (MI_SUCCESS != ret)
67. {
68.     printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
69.     return ret;
70. }
71.
72. /* disable AI Device */
73. ret = MI_AI_Disable(AiDevId);
74. if (MI_SUCCESS != ret)
75. {

```

```
76.     printf("disable ai %d err:0x%x\n", AiDevId, ret);  
77.     return ret;  
78. }  
79.  
80. MI_SYS_Exit();
```

2.1.46 MI_AI_GetMute

➤ 功能

获取 AI 通道的静音参数。

➤ 语法

MI_S32 MI_AI_GetMute([MI_AUDIO_DEV](#) AiDevId, [MI_AI_CHN](#) AiChn, MI_BOOL *pbMute);

➤ 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。 支持的通道范围由AI设备属性中的最大通道个数 u32ChnCnt决定。	输入
pbMute	静音参数	输出

➤ 返回值

返回值 $\left\{ \begin{array}{ll} 0 & \text{成功。} \\ \text{非 0} & \text{失败，参照[错误码](#)。} \end{array} \right.$

➤ 依赖

- 头文件: mi_ai.h
- 库文件: libmi_ai.so/libmi_ai.a

※ 注意

无。

➤ 举例

请参考 [MI_AI_SetMute](#) 举例部分。

3. AI 数据类型

AI 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_AI_CHN	定义音频输入通道
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_AI_ChnParam_t	定义通道参数结构体
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_AI_VqeConfig_t	定义音频输入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_AI_AecConfig_t	定义音频回声抵消配置信息结构体
MI_AUDIO_AnrcConfig_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_AI_AecConfig_t	定义音频回音消除功能配置信息结构体
MI_AI_AencConfig_t	定义音频编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_AI_AedConfig_t	定义声音事件检测功能配置信息结构体
MI_AUDIO_AedSensitivity_e	定义声音事件检测的灵敏度
MI_AI_AedResult_t	定义声音事件检测的结果
MI_AI_ChnGainConfig_t	定义音频通道增益设置结构体
MI_AI_SslInitAttr_t	定义音频声源定位功能的初始化信息结构体
MI_AI_SslConfigAttr_t	定义音频声源定位功能的配置信息结构体
MI_AI_BfInitAttr_t	定义音频波束成形功能的初始化信息结构体
MI_AI_BfConfigAttr_t	定义音频波束成形功能的配置信息结构体

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
Amic	支持 2 路 8/16/32/48KHz 采样率	支持 2 路 8/16/32/48KHz 采样率	支持 1 路 8/16/32/48KHz 采样率	支持 1 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
Dmic	标准 Dmic 接口, 支持 4 路 8/16/32/48KHz 采样率	非标准 Dmic 接口, 支持 2 路 8/16/32KHz 采样率	标准 Dmic 接口, 支持 4 路 8/16/32/48KHz 采样率	标准 Dmic 接口, 支持 4 路 8/16/32/48KHz 采样率	非标准 Dmic 接口, 支持 2 路 8/16/32KHz 采样率	非标准 Dmic 接口, 支持 2 路 8/16/32KHz 采样率
I2S RX	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。
Line in	支持 2 路 8/16/32/48KHz 采样率	支持 2 路 8/16/32/48KHz 采样率	支持 1 路 8/16/32/48KHz 采样率	支持 1 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

➤ 相关数据类型及接口

无。

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_AI_CHN

- 说明
定义音频输入通道。
- 定义

```
typedef MI_S32 MI_AI_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_11025 = 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;
```

➤ 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。
AI 仅支持 8/16/32/48kHz。

➤ 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。

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, /*all data in One chn */
    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	所有音频数据按通道顺序排列在 1 个通道的缓冲区（针对音频采集时使用）

※ 注意事项

当 sound mode 为 E_MI_AUDIO_SOUND_MODE_QUEUE 时，只需要对通道 0 做参数设置，其他通道会使用与通道 0 相同的参数，但是各个通道的处理流程相互独立。同时各个通道增益也是相互独立，需要用户分别设置。假设目前是 4 个通道，用户拿到的 buffer 大小为 4nByte，则各个通道的数据量分别为 $4n / 4 = n\text{Byte}$ 。数据排列如下：

Ch0, Ch0, Ch0, Ch0, Ch0, Ch0, Ch0, Ch0.....

Ch1, Ch1, Ch1, Ch1, Ch1, Ch1, Ch1, Ch1.....

Ch2, Ch2, Ch2, Ch2, Ch2, Ch2, Ch2, Ch2.....

Ch3, Ch3, Ch3, Ch3, Ch3, Ch3, Ch3, Ch3.....

➤ 相关数据类型及接口

[MI_AUDIO_Attr_t](#)

3.8. MI_AUDIO_AencType_e

➤ 说明

定义音频编码类型。

➤ 定义

```
typedef enum
{
    E_MI_AUDIO_AENC_TYPE_G711A = 0,
    E_MI_AUDIO_AENC_TYPE_G711U,
    E_MI_AUDIO_AENC_TYPE_G726,
    E_MI_AUDIO_AENC_TYPE_INVALID,
}MI_AUDIO_AencType_e;
```

➤ 成员

成员名称	描述
E_MI_AUDIO_AENC_TYPE_G711A	G711A 编码。
E_MI_AUDIO_AENC_TYPE_G711U	G711U 编码。
E_MI_AUDIO_AENC_TYPE_G726	G726 编码。

※ 注意事项

无

➤ 相关数据类型及接口

[MI_AUDIO_AencG726Config_t](#)

3.9. 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_AUDIO_AencG726Config_t](#)

3.10. 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;
```

➤ 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](#)

3.11. 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](#)

3.12. MI_AUDIO_I2sConfig_t

➤ 说明

定义 I2S 属性结构体。

➤ 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 时钟设置。 静态属性。
bSyncClock	AI 同步 A0 时钟 静态属性。

※ 注意事项
无。

➤ 相关数据类型及接口
[MI_AUDIO_Attr_t](#)

3.13. 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)。 静态属性。
eWorkmode	音频 I2S 的工作模式。 静态属性。
eSoundmode	音频声道的数据格式。 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。 注意: 如不使用 sigmastar 提供的音频算法, 可不必 128 对齐。
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_AI_SetPubAttr](#)

3.14. MI_AI_ChnParam_t

➤ 说明
定义通道参数结构体。

➤ 定义

```
typedef struct MI_AI_ChnParam_s
{
    MI\_AI\_ChnGainConfig\_t stChnGain;
    MI_U32 u32Reserved;
} MI_AI_ChnParam_t
```

➤ 成员

成员名称	描述
stChnGain	音频通道的增益设定。
u32Reserved	保留, 未使用。

※ 注意事项
无。

- 相关数据类型及接口

[MI_AI_SetChnParam](#)

[MI_AI_GetChnParam](#)

3.15. 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[MI_AUDIO_MAX_CHN_NUM];
    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];
    void *apSrcPcmVirAddr[MI_AUDIO_MAX_CHN_NUM]; /* Ai original pcm date from ai channel */
    MI_U32 u32SrcPcmLen;
}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[MI_AUDIO_MAX_CHN_NUM]	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。
apSrcPcmVirAddr[MI_AUDIO_MAX_CHN_NUM]	音频帧原始数据虚拟地址
u32SrcPcmLen	音频帧原始数据长度。 以 byte 为单位。

※ 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，数据格式为 L, R, L, R, L, R……，采样点数为 u32PtNumPerFrm，长度为 u32Len。当 sound mode 为 queue mode 时，数据按通道顺序存放，各个通道的数据长度为 u32Len / 通道数，采样点数为 u32PtNumPerFrm，数据格式为：
Chn0, Chn0, Chn0, Chn0, Chn0, Chn0……
Chn1, Chn1, Chn1, Chn1, Chn1, Chn1……
Chn2, Chn2, Chn2, Chn2, Chn2, Chn2……
Chn3, Chn3, Chn3, Chn3, Chn3, Chn3……
……
- 当没有开启任何算法时，pVirAddr 等价于 apSrcPcmVirAddr，u32Len 等价于 u32SrcPcmLen。当有开启算法时，pVirAddr 返回的是算法处理后的数据，apSrcPcmVirAddr 返回的是 Ai 采集到的原始 pcm 数据。

- 相关数据类型及接口
无。

3.16. 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;
```

- 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。<br>取值范围：<br>TRUE：参考帧有效。<br>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。

### 3.17. 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](#)

### 3.18. MI\_AI\_VqeConfig\_t

➤ 说明

定义音频输入声音质量增强配置信息结构体。

➤ 定义

```
typedef struct MI_AI_VqeConfig_s
{
 MI_BOOL bHpfOpen;
 MI_BOOL bAecOpen;
 MI_BOOL bAnrOpen;
 MI_BOOL bAgcOpen;
 MI_BOOL bEqOpen;
 MI_U32 u32ChnNum;
 MI_S32 s32WorkSampleRate;
 MI_S32 s32FrameSample;
 MI_AUDIO_HpfConfig_t stHpfCfg;
 MI_AI_AecConfig_t stAecCfg;
 MI_AUDIO_AnrcConfig_t stAnrcCfg;
 MI_AUDIO_AgcConfig_t stAgcCfg;
 MI_AUDIO_EqConfig_t stEqCfg;
}MI_AI_VqeConfig_t;
```

➤ 成员

| 成员名称     | 描述              |
|----------|-----------------|
| bHpfOpen | 高通滤波功能是否使能标志。   |
| bAecOpen | 回声抵消功能是否使能标志。   |
| 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。    |

| 成员名称              | 描述                                               |
|-------------------|--------------------------------------------------|
| u32ChnNum         | 数据的声道数。                                          |
| 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。只能设置 128                          |
| stHpfCfg          | 高通滤波功能相关配置信息。                                    |
| stAecCfg          | 回声抵消功能相关配置信息。                                    |
| 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。

### 3.19. 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;
```

➤ 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。<br>80：截止频率为 80Hz；<br>120：截止频率为 120Hz；<br>150：截止频率为 150Hz。<br>默认值 150。 |

※ 注意事项  
无。

➤ 相关数据类型及接口  
[MI\\_AI\\_VqeConfig\\_t](#)

## 3.20. 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

默认配置为 150Hz

### ➤ 相关数据类型及接口

[MI\\_AI\\_VqeConfig\\_t](#)

## 3.21. MI\_AI\_AecConfig\_t

### ➤ 说明

定义音频回声抵消配置信息结构体。

### ➤ 定义

```
typedef struct MI_AI_AecConfig_s
{
 MI_BOOL bComfortNoiseEnable;
 MI_S16 s16DelaySample;
 MI_U32 u32AecSupfreq[6];
 MI_U32 u32AecSupIntensity[7];
 MI_S32 s32Reserved;
} MI_AI_AecConfig_t;
```

### ➤ 成员

| 成员名称                | 描述                                                                             |
|---------------------|--------------------------------------------------------------------------------|
| bComfortNoiseEnable | 是否添加噪音。<br>0：不添加；<br>1：添加。                                                     |
| s16DelaySample      | 采样点样本延迟个数。<br>仅在 AEC 为立体声处理时有效，默认值为 0。                                         |
| u32AecSupfreq       | 回声消除保护频率范围，后 1 个数据必须大于等于前 1 个数据。<br>如：u32AecSupfreq[0] = 10，则：u32AecSupfreq[1] |

| 成员名称               | 描述                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                    | <p>必须大于等于 10。</p> <p>当前采样率对应的最高频率平均分成 128 份，频率范围则是对应多少份组成一个频带。</p> <p>如：当前采样率为 16K，对应的最大频率为 8K，每一份为 <math>8000 / 128 \approx 62.5\text{Hz}</math>，在推荐值 {4,6,36,49,50,51} 的设定下，保护范围为 {0~4 * 62.5Hz, 4~6 * 62.5Hz, 6~36 * 62.5Hz, 36~49 * 62.5Hz, 49~50 * 62.5Hz, 50~51 * 62.5Hz, 51~127 * 62.5Hz} = {0~250Hz, 250~375Hz, 375~2250Hz, 2250~3062.5Hz, 3062.5~3125Hz, 3125~3187.5Hz, 3187.5Hz~8000Hz}</p> <p>范围 [1, 127]；步长 1</p> <p>推荐值 {4, 6, 36, 49, 50, 51}</p> |
| u32AecSupIntensity | <p>回音消除保护力度，数值越小保护效果越强。此参数与 u32AecSupfreq 相对应，u32AecSupIntensity[0] 对应 0~u32AecSupfreq[0]，u32AecSupIntensity[1] 对应 u32AecSupfreq[0]~u32AecSupfreq[1]，以此类推。</p> <p>范围 [0, 15]；步长 1</p> <p>推荐值 {5, 4, 4, 5, 10, 10, 10}</p>                                                                                                                                                                                                                                       |

※ 注意事项  
无

➤ 相关数据类型及接口  
[MI\\_AI\\_VqeConfig\\_t](#)

## 3.22. MI\_AUDIO\_AnrcConfig\_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_AnrcConfig_s
{
 MI_AUDIO_AlgorithmMode_e eMode;
 MI_U32 u32NrIntensity;
 MI_U32 u32NrSmoothLevel;
 MI_AUDIO_NrSpeed_e eNrSpeed;
} MI_AUDIO_AnrcConfig_t;
```

➤ 成员

| 成员名称           | 描述                                                                                |
|----------------|-----------------------------------------------------------------------------------|
| eMode          | <p>音频算法的运行模式</p> <p>注：Anr 的模式选择将会在一定程度上影响 Agc 的功能</p>                             |
| u32NrIntensity | <p>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。</p> <p>范围 [0, 30]；步长 1</p> <p>默认值 20。</p> |

| 成员名称             | 描述                                        |
|------------------|-------------------------------------------|
| u32NrSmoothLevel | 平滑化程度，值越大越平滑<br>范围[0, 10]；步长 1<br>默认值 10。 |
| eNrSpeed         | 噪声收敛速度，低速，中速，高速<br>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。

➤ 相关数据类型及接口

[MI\\_AI\\_VqeConfig\\_t](#)

### 3.23. 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\\_AI\\_VqeConfig\\_t](#)

### 3.24. 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;
```

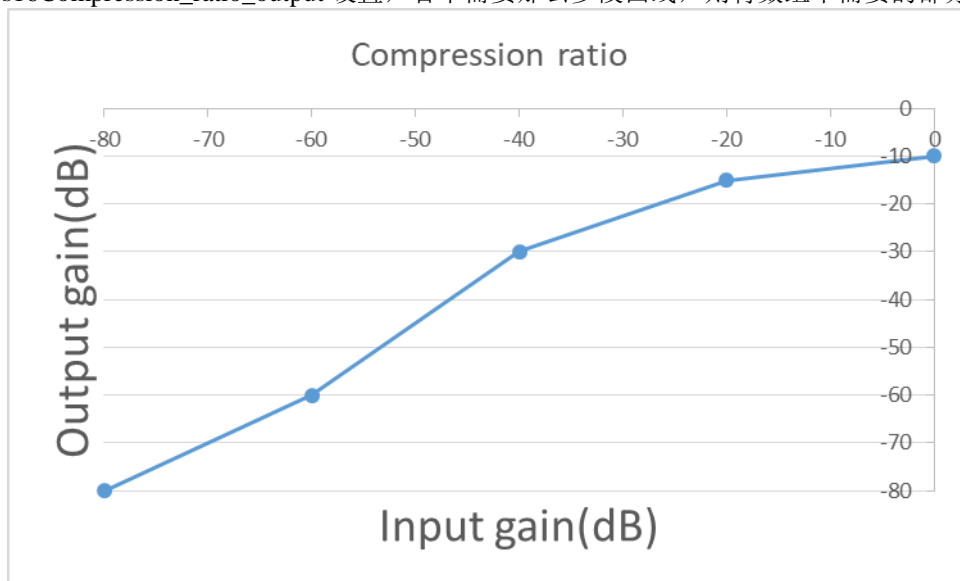
➤ 成员

| 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<br>范围[0, 60]；步长 1<br>默认值 55。                                            |
| u32AttackTime                  | 增益下降时间区间长度，以 16 毫秒为 1 单位<br>范围[1, 20]；步长 1<br>默认值 0。                                    |
| u32ReleaseTime                 | 增益上升时间区间长度，以 16 毫秒为 1 单位<br>范围[1, 20]；步长 1<br>默认值 0。                                    |
| s16Compression_ratio_input[5]  | 输入压缩比，必须配合<br>s16Compression_ratio_output 使用，透过多个<br>转折点实现多斜率的曲线<br>范围[-80, 0]dBFS；步长 1 |
| s16Compression_ratio_output[5] | 输出压缩比，必须配合<br>s16Compression_ratio_input 使用，透过多个转<br>折点实现多斜率的曲线<br>范围[-80, 0]dBFS；步长 1  |
| s32TargetLevelDb               | 目标电平，经过处理后的最大电平门限<br>范围[-80, 0]dB；步长 1<br>默认值 0。                                        |
| s32NoiseGateDb                 | 噪声底值<br>范围[-80, 0]；步长 1<br>注：当值为-80，噪声底值将不起作用<br>默认值-55。                                |
| u32NoiseGateAttenuationDb      | 当噪声底值起效果时，输入源的衰减百分比<br>范围[0, 100]；步长 1<br>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。



## ➤ 相关数据类型及接口

MI\_AI\_VqeConfig\_t

## 3.25. 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<br>范围[0, 60]；步长 1<br>默认值 15。  |
| s32GainMin | 增益最小值<br>范围[-20, 30]；步长 1<br>默认值 0。 |

| 成员名称        | 描述                                  |
|-------------|-------------------------------------|
| s32GainInit | 增益初始值<br>范围[-20, 60]；步长 1<br>默认值 0。 |

※ 注意事项  
无

➤ 相关数据类型及接口  
[MI\\_AI\\_VqeConfig\\_t](#)

### 3.26. 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<br>范围[-50, 20]；步长 1<br>默认值 0。<br>如：当前采样率为 16K, 对应的最高频率为 8K, $8000 / 129 \approx 62\text{Hz}$ , 则单个调节的频率范围为 62Hz, 将 0-8K 划分成 $\{0-1 * 62\text{Hz}, 1-2 * 62\text{Hz}, 2-3 * 62\text{Hz}, \dots, 128-129 * 62\text{Hz}\} = \{0-62\text{Hz}, 62-124\text{Hz}, 124-186\text{Hz}, \dots, 7938-8000\text{Hz}\}$ , 每段对应一个增益值 |

※ 注意事项  
无

➤ 相关数据类型及接口  
[MI\\_AI\\_VqeConfig\\_t](#)

### 3.27. MI\_AI\_AencConfig\_t

➤ 说明  
定义音频编码功能配置信息结构体。

➤ 定义

```
typedef struct MI_AI_AencConfig_s
{
 MI_AUDIO_AencType_e eAencType;
 union
 {
 MI_AUDIO_AencG711Config_t stAencG711Cfg;
 MI_AUDIO_AencG726Config_t stAencG726Cfg;
 };
}MI_AI_AencConfig_t;
```

➤ 成员

| 成员名称          | 描述             |
|---------------|----------------|
| eAencType     | 音频编码类型。        |
| stAencG711Cfg | G711 编码相关配置信息。 |
| stAencG726Cfg | G726 编码相关配置信息。 |

※ 注意事项  
无

➤ 相关数据类型及接口

[MI\\_AI\\_SetAencAttr](#)

## 3.28. MI\_AUDIO\_AencG711Config\_t

➤ 说明

定义音频编码功能配置信息结构体。

➤ 定义

```
typedef struct MI_AUDIO_AencG711Config_s
{
 MI_AUDIO_SampleRate_e eSamplerate;
 MI_AUDIO_SoundMode_e eSoundmode;
}MI_AUDIO_AencG711Config_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\\_AI\\_SetAencAttr](#)

### 3.29. MI\_AUDIO\_AencG726Config\_t

➤ 说明

定义音频编码功能配置信息结构体。

➤ 定义

```
typedef struct MI_AUDIO_AencG726Config_s
{
 MI_AUDIO_SampleRate_e eSamplerate;
 MI_AUDIO_SoundMode_e eSoundmode;
 MI_AUDIO_G726Mode_e eG726Mode;
}MI_AUDIO_AencG726Config_t;
```

➤ 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\\_AI\\_SetAencAttr](#)

### 3.30. 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<br>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<br>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<br>仅有 Anr 具有此模式，当为此模式时，Agc 不会进行 <b>speech enhancement</b> （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\\_AnrcConfig\\_t](#) , [MI\\_AUDIO\\_AgcConfig\\_t](#) , [MI\\_AUDIO\\_EqConfig\\_t](#)

### 3.31. MI\_AI\_AedConfig\_t

➤ 说明

声音事件检测功能配置信息结构体。

➤ 定义

```
typedef struct MI_AI_AedConfig_s
{
 MI_BOOL bEnableNr;
 MI_AUDIO_AedSensitivity_e eSensitivity;
 MI_S32 s32OperatingPoint;
 MI_S32 s32VadThresholdDb;
 MI_S32 s32LsdThresholdDb;
}MI_AI_AedConfig_t;
```

➤ 成员

| 成员名称              | 描述                                                                |
|-------------------|-------------------------------------------------------------------|
| bEnableNr         | 是否启用声音事件检测的降噪功能                                                   |
| eSensitivity      | 声音事件检测功能的灵敏度                                                      |
| s32OperatingPoint | 操作点<br>范围[-10, 10]，步长为 1<br>默认值为 0<br>注：提高操作点将会降低误报率，减小操作点将会降低漏测率 |
| s32VadThresholdDb | Vad 的阈值 (dB)<br>范围[-80, 0]，步长为 1<br>默认值为-40                       |
| s32LsdThresholdDb | Lsd 的阈值 (dB)<br>范围[-80, 0]，步长为 1<br>默认值为-15                       |

※ 注意事项

无

➤ 相关数据类型及接口

[MI\\_AI\\_SetAedAttr](#), [MI\\_AI\\_GetAedAttr](#)

### 3.32. MI\_AUDIO\_AedSensitivity\_e

➤ 说明

声音事件检测的灵敏度

➤ 定义

```
typedef enum
{
 E_MI_AUDIO_AED_SEN_LOW,
 E_MI_AUDIO_AED_SEN_MID,
 E_MI_AUDIO_AED_SEN_HIGH,
 E_MI_AUDIO_AED_SEN_INVALID,
}MI_AUDIO_AedSensitivity_e;
```

➤ 成员

| 成员名称                    | 描述    |
|-------------------------|-------|
| E_MI_AUDIO_AED_SEN_LOW  | 低灵敏度  |
| E_MI_AUDIO_AED_SEN_MID  | 中等灵敏度 |
| E_MI_AUDIO_AED_SEN_HIGH | 高灵敏度  |

※ 注意事项  
无

➤ 相关数据类型及接口  
[MI\\_AI\\_AedConfig\\_t](#)

### 3.33. MI\_AI\_AedResult\_t

➤ 说明

声音事件检测结果结构体。

➤ 定义

```
typedef struct MI_AI_AedResult_s
{
 MI_BOOL bAcousticEventDetected;
 MI_BOOL bLoudSoundDetected;
}MI_AI_AedResult_t;
```

➤ 成员

| 成员名称                   | 描述         |
|------------------------|------------|
| bAcousticEventDetected | 是否检测到声音事件  |
| bLoudSoundDetected     | 是否检测到高分贝声音 |

※ 注意事项  
无

➤ 相关数据类型及接口  
[MI\\_AI\\_GetAedResult](#)

### 3.34. MI\_AI\_ChnGainConfig\_t

➤ 说明

音频通道增益设置结构体。

➤ 定义

```
typedef struct MI_AI_ChnGainConfig_s
{
 MI_BOOL bEnableGainSet;
 MI_S16 s16FrontGain;
 MI_S16 s16RearGain;
}MI_AI_ChnGainConfig_t;
```

➤ 成员

| 成员名称           | 描述       |
|----------------|----------|
| bEnableGainSet | 是否使能增益设置 |
| s16FrontGain   | 前级增益     |
| s16RearGain    | 后级增益     |

※ 注意事项

以下表格描述了各个 device 对 s16FrontGain 和 s16RearGain 的定义

| AI Device ID | s16FrontGain          | s16RearGain              |
|--------------|-----------------------|--------------------------|
| Amic         | Amic 的模拟增益 (0 - 21)   | Amic 的数字增益 (-60 - 30)    |
| Dmic         | Dmic 的增益              | 不起作用                     |
|              | Pretzel 系列 (0 - 4)    |                          |
|              | Macaron 系列 (-60 - 30) |                          |
|              | TAIYAKI 系列 (0 - 4)    |                          |
|              | TAKOYAKI 系列 (0 - 4)   |                          |
|              | Pudding 系列 (-60 - 30) |                          |
|              | Ispahan 系列 (-60 - 30) |                          |
| I2S_RX       | 不起作用                  | 不起作用                     |
| Line in      | Line in 的模拟增益 (0 - 7) | Line in 的数字增益 (-60 - 30) |

➤ 相关数据类型及接口

[MI\\_AI\\_ChnParam\\_t](#)

### 3.35. MI\_AI\_SslInitAttr\_t

➤ 说明

音频声源定位功能初始化结构体。

➤ 定义

```
typedef struct MI_AI_SslInitAttr_s
{
 MI_U32 u32MicDistance;
 MI_BOOL bBfMode;
} MI_AI_SslInitAttr_t;
```



➤ 成员

| 成员名称           | 描述                                                         |
|----------------|------------------------------------------------------------|
| u32MicDistance | 麦克风的间距（cm）<br>步长为 1。                                       |
| bBfMode        | 是否为 beamforming 模式<br>FALSE 为机械转动模式，TRUE 为 beamforming 模式。 |

※ 注意事项  
无

➤ 相关数据类型及接口

[MI\\_AI\\_SetSslInitAttr](#), [MI\\_AI\\_GetSslInitAttr](#)

### 3.36. MI\_AI\_SslConfigAttr\_t

➤ 说明

音频声源定位功能配置结构体。

➤ 定义

```
typedef struct MI_AI_SslConfigAttr_s
{
 MI_S32 s32Temperature;
 MI_S32 s32NoiseGateDbfs;
 MI_S32 s32DirectionFrameNum;
} MI_AI_SslConfigAttr_t;
```

➤ 成员

| 成员名称                 | 描述                                                                                                                                                                                                             |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| s32Temperature       | 温度（摄氏度）<br>摄氏度 = (5/9)*(华氏度 - 32)<br>步长为 1                                                                                                                                                                     |
| s32NoiseGateDbfs     | 噪音增益门限值（dB）<br>注意：低于此值将会把该帧作为噪声部分处理。<br>步长为 1                                                                                                                                                                  |
| s32DirectionFrameNum | 声源定位功能检测的帧数<br>步长为 50<br>注意：进行声源定位检测的帧数，数值必须为 50 的倍数，声源定位处理的一帧数据为 128 个采样点。检测一次的时间 = $s32DirectionFrameNum * 128 / \text{采样率}$ 。如：当前采样率为 48K，s32DirectionFrameNum 设置成 300，则检测时间 = $300 * 128 / 48000 = 0.8(s)$ |

※ 注意事项  
无

➤ 相关数据类型及接口

[MI\\_AI\\_SetSslConfigAttr](#), [MI\\_AI\\_GetSslConfigAttr](#)

### 3.37. MI\_AI\_BfInitAttr\_t

➤ 说明

音频波束成形功能初始化结构体。

➤ 定义

```
typedef struct MI_AI_BfInitAttr_s
{
 MI_U32 u32MicDistance;
 MI_U32 u32ChanCnt;
} MI_AI_BfInitAttr_t;
```

➤ 成员

| 成员名称           | 描述                   |
|----------------|----------------------|
| u32MicDistance | 麦克风的间距（cm）<br>步长为 1。 |
| u32ChanCnt     | 通道数。<br>注意：只能设置为 2。  |

※ 注意事项  
无

➤ 相关数据类型及接口

[MI\\_AI\\_SetBfInitAttr](#), [MI\\_AI\\_GetBfInitAttr](#)

### 3.38. MI\_AI\_BfConfigAttr\_t

➤ 说明

音频波束成形功能配置结构体。

➤ 定义

```
typedef struct MI_AI_BfConfigAttr_s
{
 MI_S32 s32Temperature;
 MI_S32 s32NoiseGateDbfs;
 MI_S32 s32NoiseSupressionMode;
 MI_S32 s32NoiseEstimation;
 MI_FLOAT outputGain;
} MI_AI_BfConfigAttr_t;
```

➤ 成员

| 成员名称                   | 描述                                                                                           |
|------------------------|----------------------------------------------------------------------------------------------|
| s32Temperature         | 温度（摄氏度）<br>摄氏度 = $(5/9) * (\text{华氏度} - 32)$<br>步长为 1                                        |
| s32NoiseGateDbfs       | 噪音增益门限值（dB）<br>注意：低于此值将会把该帧作为噪声部分处理。如果此值设置得太高，将会导致语音失真，如果设置得太低，会影响 BF 的效果。<br>推荐值为 -44，步长为 1 |
| s32NoiseSupressionMode | 噪声抑制模式                                                                                       |

| 成员名称               | 描述                                                                                   |
|--------------------|--------------------------------------------------------------------------------------|
|                    | 范围[0, 15]，步长为 1<br>推荐值为 8<br>注意：此值越大，噪声消除强度越强。如果此值设置得太高，将会导致语音失真，如果设置得太低，会影响 BF 的效果。 |
| s32NoiseEstimation | 噪声的估计模式<br>取值为 0 或 1。<br>0 为自适应模式<br>1 为平均模式<br>关键词定位预处理推荐使用 0<br>语音通讯推荐使用 1         |
| outputGain         | 输出增益<br>范围[0, 1]<br>推荐值 0.7<br>注意：设定为 1，会增加 4dB                                      |

※ 注意事项  
无

➤ 相关数据类型及接口

[MI\\_AI\\_SetBfConfigAttr](#), [MI\\_AI\\_GetBfConfigAttr](#)

## 4. 错误码

AI API 错误码如表 3-1 所示：

表 3-1 AI API 错误码

| 宏定义                     | 描述              |
|-------------------------|-----------------|
| MI_AI_ERR_INVALID_DEVID | 音频输入设备号无效       |
| MI_AI_ERR_INVALID_CHNID | 音频输入信道号无效       |
| MI_AI_ERR_ILLEGAL_PARAM | 音频输入参数设置无效      |
| MI_AI_ERR_NOT_ENABLED   | 音频输入设备或通道没有使能   |
| MI_AI_ERR_NULL_PTR      | 输入参数空指标错误       |
| MI_AI_ERR_NOT_CONFIG    | 音频输入设备属性未设置     |
| MI_AI_ERR_NOT_SUPPORT   | 操作不支持           |
| MI_AI_ERR_NOT_PERM      | 操作不允许           |
| MI_AI_ERR_NOMEM         | 分配内存失败          |
| MI_AI_ERR_NOBUF         | 音频输入缓存不足        |
| MI_AI_ERR_BUF_EMPTY     | 音频输入缓存为空        |
| MI_AI_ERR_BUF_FULL      | 音频输入缓存为满        |
| MI_AI_ERR_SYS_NOTREADY  | 音频输入系统未初始化      |
| MI_AI_ERR_BUSY          | 音频输入系统忙碌        |
| MI_AI_ERR_VQE_ERR       | 音频输入 VQE 算法处理失败 |
| MI_AI_ERR_AENC_ERR      | 音频输入编码算法处理失败    |
| MI_AI_ERR_AED_ERR       | 声音检测算法处理失败      |
| MI_AI_ERR_SSL_ERR       | 声源定位算法处理失败      |
| MI_AI_ERR_BF_ERR        | 波束成形算法处理失败      |