MI AI API

Version 2.13

REVISION HISTORY

Revision No.	Description	Date
2.03	• Initial release	04/12/2018
2.04	• Updated for accuracy	02/23/2019
2.05	• Added MI_AI_SetAedAttr, MI_AI_GetAedAttr, MI_AI_EnableAed, MI_AI_DisableAed, and MI_AI_GetAedResult	03/07/2019
2.06	• Added description regarding audio algorithm	03/25/2019
2.07	• Added MI_AI_SetExtAecChn and updated MI_AI_SetChnParam and MI_AI_GetChnParam	03/30/2019
2.08	• Added support for SSL and BF	06/04/2019
2.09	• Updated audio supporting sample rate	06/17/2019
2.10	• Updated MI_AUDIO_Frame_t structure • Fixed the wrong sample rate • Fixed some incorrect instructions	07/31/2019
2.11	• Updated I2S mode and Mclk • Fixed the wrong description of AEC and AGC	10/26/2019
2.12	• Added API for setting bf angle	12/04/2019
2.13	• Added mute API for channel • Added demo code for API	12/30/2019

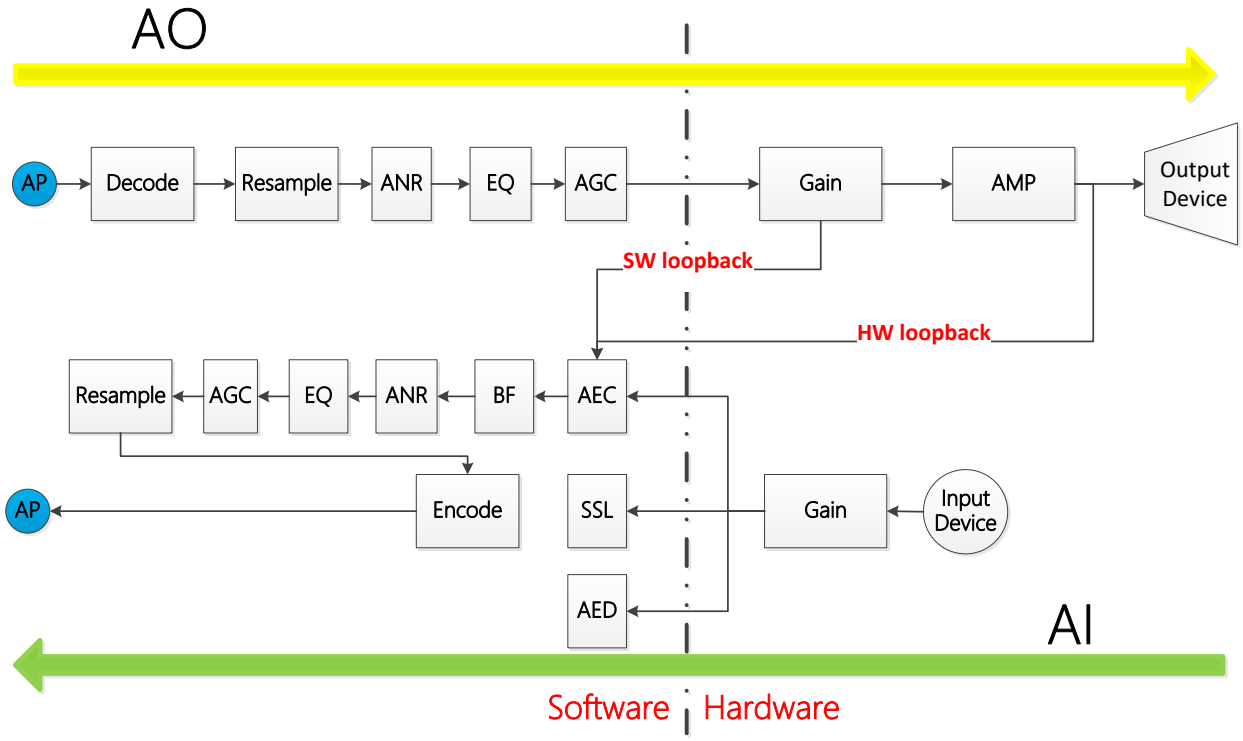
TABLE OF CONTENTS

1. SUMMARY

1.1. Module Description

Audio Input (AI) is mainly used to configure and enable Audio Input devices, obtain Audio frame data, and perform acoustic algorithm processing. Acoustic algorithm processing mainly includes Sample Rate Conversion, Acoustic Echo Cancellation, Acoustic Noise Reduction, High-Pass Filtering, Equalizer, Automatic Gain Control, Sound Event Detection, Sound Source Location, Beamforming and so on.

1.2. Flow Block Diagram



1.3. Keyword Description

- Device
  Different from Device concepts of other modules, AI Device refers to different external input devices such as Amic/Dmic/I2S RX/Line in.
- Channel
  AI Channel refers to the number of tracks in the software concept.
- SRC
  SRC refers to Sample Rate Conversion.
- AGC
  AGC refers to Automatic Gain Control), and is used to control the output gain.
- EQ
  EQ refers to Equalizer, and is used to process specific frequencies.
- ANR
  ANR refers to Acoustic Noise Reduction, and is used to remove persistent, constant frequency noise in the environment.
- BF
  BF refers to beamforming, and is used in microphone arrays to enhance sound.
- AEC
  AEC refers to Acoustic Echo Cancellation.
- SSL
  SSL refers to Sound Source Localization, and is used in microphone arrays to identify the direction of sound.
- AED
  AED refers to Acoustic Event Detection. Currently, only baby cries and loud sound can be detected.
- HPF
  HPF refers to High-Pass Filtering.

2. API REFERENCE

2.1. Overview

Audio Input (AI) mainly implements functions such as configuring and enabling audio input devices and acquiring audio frame data

2.2. API List

API Name	Features
MI_AI_SetPubAttr	Set AI device properties
MI_AI_GetPubAttr	Get AI device properties
MI_AI_Enable	Enable AI device
MI_AI_Disable	Disable AI device
MI_AI_EnableChn	Enable AI channel
MI_AI_DisableChn	Disable AI channel
MI_AI_GetFrame	Get audio frames
MI_AI_ReleaseFrame	Release audio frame
MI_AI_SetChnParam	Set AI channel parameters
MI_AI_GetChnParam	Get AI channel parameters
MI_AI_EnableReSmp	Enable AI channel resampling
MI_AI_DisableReSmp	Disable AI channel resampling.
MI_AI_SetVqeAttr	Set the sound quality enhancement related properties of AI channel
MI_AI_GetVqeAttr	Get the sound quality enhancement related properties of AI channel
MI_AI_EnableVqe	Enable the sound quality enhancements of AI channel
MI_AI_DisableVqe	Disable the sound quality enhancements of AI channel
MI_AI_ClrPubAttr	Clear AI device properties
MI_AI_SaveFile	Turn on audio input to save files
MI_AI_SetVqeVolume	Set the volume level of AI channel
MI_AI_GetVqeVolume	Get the volume level of AI channel
MI_AI_SetAencAttr	Set encoding function related properties of AI channel
MI_AI_GetAencAttr	Get encoding function related properties of AI channel
MI_AI_EnableAenc	Enable encoding of AI channel
MI_AI_DisableAenc	Disable encoding of AI channel

API Name	Features
MI_AI_SetAedAttr	Set sound event detection function related properties of AI channel.
MI_AI_GetAedAttr	Get sound event detection function related properties of AI channel.
MI_AI_EnableAed	Enable sound event detection of AI channel.
MI_AI_DisableAed	Disable sound event detection of AI channel.
MI_AI_GetAedResult	Get the sound event detection result of AI channel.
MI_AI_SetExtAecChn	Set the external echo cancellation function reference AI channel
MI_AI_SetSslInitAttr	Set Sound source localization function initialized properties of AI channel
MI_AI_GetSslInitAttr	Get Sound source localization function initialized properties of AI channel
MI_AI_SetSslConfigAttr	Set Sound source localization function configured properties of AI channel
MI_AI_GetSslConfigAttr	Get Sound source localization function configured properties of AI channel
MI_AI_EnableSsl	Enable Sound source localization function of AI channel
MI_AI_DisableSsl	Disable Sound source localization function of AI channel
MI_AI_GetSslDoa	Get the Sound source localization result of AI channel
MI_AI_SetBfInitAttr	Set Beamforming function initialized properties of AI channel
MI_AI_GetBfInitAttr	Get Beamforming function initialized properties of AI channel
MI_AI_SetBfConfigAttr	Set Beamforming function configured properties of AI channel
MI_AI_GetBfConfigAttr	Get Beamforming function configured properties of AI channel
MI_AI_EnableBf	Enable Beamforming function of AI channel
MI_AI_DisableBf	Disable Beamforming function of AI channel
MI_AI_SetBfAngle	Set fixed Angle for Beamforming function of AI channel
MI_AI_SetMute	Set Mute status of channel
MI_AI_GetMute	Get Mute status of channel

2.2.1 MI_AI_SetPubAttr

- Features

  Set the AI device properties.

- Syntax

  MI_S32 MI_AI_SetPubAttr(MI_AUDIO_DEV AiDevId, MI_AUDIO_Attr_t *pstAttr);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
pstAttr	AI device property pointer	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note

  Audio input device determines the format of input data attributes. The device attributes includes sampling rate, sampling precision, input operation mode, data format of channel, number of sampling points per frame channels, channel count of device, and configuration of I2S. When Codec is used, these attributes should be consistent with the requirements of the docking Codec configuration.

  Sampling rate

  The sampling rate refers to the number of sampling points in one second. The higher the sampling rate, the smaller the distortion, and the more data is processed. Generally speaking, 8k sampling rate is used for speech and 32k or more for audio. Currently, only 8/16/32/48KHz sampling rate is supported. When Codec is used, make sure whether the docked Audio Codec supports the sampling rate to be set.

  Sampling accuracy

  Sampling accuracy refers to the sampling point data width of a channel and determines the channel distribution of the entire device. The sampling accuracy supports 16 bits.

  Operating mode

  Audio input and output currently supports I2S master mode, I2S slave mode, Tdm master mode and Tdm slave mode, but the content supported by each audio device may be different.

  Data format of channel

  The data format indicates data arrangement in channel. Mono indicates that the data of AI channel is one physical data channel. Stereo indicates that the data of AI channel is two interlaced physical data channels. Queue indicates that multiple physical channels of data exist in one channel. When the algorithm is enabled in Queue mode, all channels use the same set of parameters.

Number of samples per frame

When the audio sampling rate is high, it is recommended to increase the number of sampling points per frame accordingly. If the sound is intermittent, you can increase the number of samples per frame and the buffer count as deemed necessary. It is recommended that the number of samples per frame be set at about 16 frames per second.

Number of channels

The number of channels refers to the number of channels in the software concept of the current input device. The number of channels, together with the data format of channel, determines how many physical channels to use.

Configuration of I2S

The configuration parameters of I2S specify the frequency of I2S MCLK, the data format of I2S transmission, and whether I2S uses 4-wire mode or 6-wire mode.

- Example

  The following code shows how to implement the function of taking a data frame from the AI channel and releasing it.

```
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)
```

```
{
    printf("enable ai %d err:0x%x\n", AiDevId, ret);
    return ret;
}

/* enable AI Channel */
ret = MI_AI_EnableChn(AiDevId, AiChn);
if (MI_SUCCESS != ret)
{
    printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* set buffer depth */
stAiChnOutputPort.eModId = E_MI_MODULE_ID_AI;
stAiChnOutputPort.u32DevId = AiDevId;
stAiChnOutputPort.u32ChnId = AiChn;
stAiChnOutputPortu32PortId = 0;
MI_SYS_SetChnOutputPortDepth(&stAiChnOutputPort, 1, 8);

/* get port fd */
stChnPort.eModId = E_MI_MODULE_ID_AI;
stChnPort.u32DevId = AiDevId;
stChnPort.u32ChnId = AiChn;
stChnPort.u32PortId = 0;
ret = MI_SYS_GetFd(&stChnPort, &s32Fd);
if (MI_SUCCESS != ret)
{
    printf("Dev%d Chn%d failed to call MI_SYS_GetFd!!!\n", AiDevId, AiChn);
    return ret;
}

/* select 100ms */
FD_ZERO(&readFdSet);
FD_SET(s32Fd, &readFdSet);
stTimeOut.tv_sec = 0;
stTimeOut.tv_usec = 100 * 1000;
ret = select(s32Fd + 1, &readFdSet, NULL, NULL, &stTimeOut);
if (FD_ISSET(s32Fd, &readFdSet))
{
    ret = MI_AI_GetFrame(AiDevId, AiChn, &stAiChFrame, &stAecFrame, 0);
    if (MI_SUCCESS == ret)
    {
        /* do something */
    MI_AI_ReleaseFrame(AiDevId, AiChn, &stAiChFrame, &stAecFrame);
     }
}

/* disable AI Channel */
ret = MI_AI_DisableChn(AiDevId, AiChn);
if (MI_SUCCESS != ret)
{
    printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* disable AI Device */
ret = MI_AI_Disable(AiDevId);
if (MI_SUCCESS != ret)
{
    printf("disable ai %d err:0x%x\n", AiDevId, ret);
```

```
    return ret;
}

MI_SYS_Exit();
```

2.2.2 MI_AI_GetPubAttr

- Features

  Get the AI device properties.

- Syntax

  MI_S32 MI_AI_GetPubAttr(MI_AUDIO_DEV AiDevId, MI_AUDIO_Attr_t*pstAttr);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
pstAttr	AI device property pointer	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note
  - The obtained property is the property of the previous configuration
  - If the property has never been configured, it returns a failure.

- Example

  Please refer to the MI_AI_SetPubAttr example section.

2.2.3 MI_AI_Enable

- Features

  Enable AI devices.

- Syntax

  MI_S32 MI_AI_Enable(MI_AUDIO_DEV AiDevId);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note
  - The AI device attribute must be configured before enabling, otherwise the return attribute is not configured incorrectly.
  - If the AI device is already enabled, it will return Success directly.

- Example

  Please refer to the MI_AI_SetPubAttr example section.

2.2.4 MI_AI_Disable

- Features

  Disable AI device.

- Syntax

  MI_S32 MI_AI_Disable(MI_AUDIO_DEV AiDevId);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note
  - If the AI device is already disabled, it will return Success directly.
  - All AI channels enabled under this device must be disabled before disabling the AI device.

- Example

  Please refer to the MI_AI_SetPubAttr example section.

2.2.5 MI_AI_EnableChn

- Features

  Enable AI channel

- Syntax

  MI_S32 MI_AI_EnableChn(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels in the AI device attribute u32ChnCnt.	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note

  Before enabling the AI channel, you must first enable the AI device to which it belongs, otherwise it will return the error code that the device is not started.

- Example

  Please refer to the MI_AI_SetPubAttr example section.

2.2.6 MI_AI_DisableChn

- Features

  AI channel disabled

- Syntax

  MI_S32 MI_AI_DisableChn(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels in the AI device attribute u32ChnCnt.	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note
  - If the channel has already been disabled, success is returned.
  - All AI algorithms enabled under the channel must be disabled before disabling the AI channel.

- Example

  Please refer to the MI_AI_SetPubAttr example section.

2.2.7 MI_AI_GetFrame

- Features

  Get audio frames.

- Syntax

  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);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt.	Input
pstFrm	Audio frame structure pointer	Output
pstAecFrm	The echo cancels the reference frame structure body pointer.	Output
s32MilliSec	Timeout for getting data: -1 indicates blocking mode, waiting for no data; 0 means non-blocking mode, when there is no data, it will return an error; >0 means to block s32MilliSec milliseconds, and the timeout will return an error.	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note
  - Before obtaining audio frame data, you must first enable the corresponding AI channel.
  - If you need to obtain an echo cancellation reference frame, pstAecFrm cannot be a null pointer. If you do not want to get the echo cancellation reference frame pstAecFrm, you can set it to a null pointer.
  - s32MilliSec value must be greater than equal to -1, -1 is equal to the data acquired using the blocking mode, the data acquired is equal to non-blocking mode 0, is greater than 0, the blocking s32MilliSec milliseconds, and no data timeout then return error.
  - This interface supports select operations. It is recommended to use the select/poll operation instead of timeout parameter.

- Example

  Please refer to the MI_AI_SetPubAttr example section.

2.2.8 MI_AI_ReleaseFrame

- Features

  Release audio frame

- Syntax

  MI_S32 MI_AI_ReleaseFrame(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AUDIO_Frame_t *pstFrm, MI_AUDIO_AecFrame_t *pstAecFrm);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt and channel mode eSoundmode in the AI device properties.	Input
pstFrm	Audio frame structure pointer	Input
pstAecFrm	The echo cancels the reference frame structure body pointer.	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetPubAttr example section.

2.2.9 MI_AI_SetChnParam

- Features

  Set AI channel parameters

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt.	Input
pstChnParam	Audio parameter structure pointer.	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note

  To set AI channel parameter, configure and enable AI device first.

- Example

  The following code shows how to configure and obtain AI channel parameters.

```
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;
```

```
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
stAttr.u32PtNumPerFrm = 160;
stAttr.u32ChnCnt = 1;

/* set public attribute of AI device */
ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
if (MI_SUCCESS != ret)
{
    printf("set Dev%d attr err:0x%x\n", AiDevId, ret);
    return ret;
}

/* enable AI device */
ret = MI_AI_Enable(AiDevId);
if (MI_SUCCESS != ret)
{
    printf("enable Dev%d err:0x%x\n", AiDevId, ret);
    return ret;
}

/* enable AI Channel */
ret = MI_AI_EnableChn(AiDevId, AiChn);
if (MI_SUCCESS != ret)
{
    printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

memset(&stChnParam, 0x0, sizeof(stChnParam));
stChnParam.stChnGain.bEnableGainSet = TRUE;
stChnParam.stChnGain.s16FrontGain = 0;
stChnParam.stChnGain.s16RearGain = 0;

ret = MI_AI_SetChnParam(AiDevId, AiChn, &stChnParam);
if (MI_SUCCESS != ret)
{
    printf("set Dev%d Chn%d param err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* get channel param */
ret = MI_AI_GetChnParam(AiDevId, AiChn, &stChnParam);
if (MI_SUCCESS != ret)
{
    printf("get Dev%d Chn%d param err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* disable AI Channel */
ret = MI_AI_DisableChn(AiDevId, AiChn);
if (MI_SUCCESS != ret)
{
    printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* disable AI Device */
ret = MI_AI_Disable(AiDevId);
if (MI_SUCCESS != ret)
{
```

```
73.     printf("disable ai %d err:0x%x\n", AiDevId, ret);
74.     return ret;
75. }
76.
77. MI_SYS_Exit();
```

2.2.10 MI_AI_GetChnParam

- Features

  Get AI channel parameters

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt and the channel mode eSoun dmode in the AI device properties.	Input
pstChnParam	Audio parameter structure pointer.	Output

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai,so

- Note

  None.

- Example

  Please refer to the MI_AI_SetChnParam example section.

2.2.11 MI_AI_EnableReSmp

- Features

  Enable resampling of AI channel.

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt.	Input
eOutSampleRate	Output sample rate for audio resampling.	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libSRC_LINUX.so

- Note
  - After enabling the AI channels, call this interface to enable the resampling function.
  - If enabling beamforming is required, resampling should be enabled after enabling beamforming.

- Example

  The following code shows how to implement AI resampling from 8K to 16K.

```
MI_S32 ret;
MI_AUDIO_DEV AiDevId = 0;
MI_AI_CHN AiChn = 0;
MI_AUDIO_Attr_t stAttr;

MI_SYS_Init();

memset(&stAttr, 0x0, sizeof(MI_AUDIO_Attr_t));
stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
stAttr.u32PtNumPerFrm = 160;
stAttr.u32ChnCnt = 1;

/* set public attribute of AI device */
ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
if (MI_SUCCESS != ret)
{
    printf("set Dev%d attr err:0x%x\n", AiDevId, ret);
    return ret;
}

/* enable AI device */
ret = MI_AI_Enable(AiDevId);
if (MI_SUCCESS != ret)
{
```

```
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.
```

2.2.12 MI_AI_DisableReSmp

- Features

  Disable resampling of AI channel

- Syntax

  MI_S32 MI_AI_DisableReSmp(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt.	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libSRC_LINUX.so

- Note
  - If you no longer use the AI resampling feature, you should call this interface to disable it.

- Example

  Please refer to the MI_AI_EnableReSmp example section.

2.2.13 MI_AI_SetVqeAttr

- Features

  Set the sound quality enhancement related properties of AI channel

- Syntax

  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);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt of AI device.	Input
AoDevId	The AO device number used for echo cancellation.	Input
AoChn	The AO channel number used for echo cancellation. The supported channel range is determined by the maximum number of channels u32ChnCnt of AO device.	Input
pstVqeConfig	Audio input sound quality enhancement configuration structure pointer	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libAEC_LINUX.so libAPC_LINUX.so

- Note
  - Before setting the AI sound quality enhancement function related properties, you must first enable the corresponding AI channel.
  - All algorithms included in Vqe support 8/16k sampling rate, while only ANR/AGC/EQ support 48K.

- Example

  The following code shows how to enable and disable sound quality enhancement.

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

MI_SYS_Init();

memset(&stAttr, 0x0, sizeof(MI_AUDIO_Attr_t));
stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
stAttr.u32PtNumPerFrm = 160;
stAttr.u32ChnCnt = 1;

/* set public attribute of AI device */
ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
if (MI_SUCCESS != ret)
{
    printf("set Dev%d attr err:0x%x\n", AiDevId, ret);
    return ret;
}

/* enable AI device */
ret = MI_AI_Enable(AiDevId);
if (MI_SUCCESS != ret)
{
    printf("enable Dev%d err:0x%x\n", AiDevId, ret);
    return ret;
}

/* enable AI Channel */
ret = MI_AI_EnableChn(AiDevId, AiChn);
if (MI_SUCCESS != ret)
```

```
{
    printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* set vqe attr */
memset(&stVqeConfig, 0x0, sizeof(stVqeConfig));
stVqeConfig.bHpfOpen = TRUE;
stVqeConfig.bAecOpen = TRUE;
stVqeConfig.bAnrOpen = TRUE;
stVqeConfig.bAgcOpen = TRUE;
stVqeConfig.bEqOpen = TRUE;
stVqeConfig.u32ChnNum = 1;
stVqeConfig.s32WorkSampleRate = 8000;
stVqeConfig.s32FrameSample = 128;
stVqeConfig.stHpfCfg.eMode = E_MI_AUDIO_ALGORITHM_MODE_USER;
stVqeConfig.stHpfCfg.eHpfFreq = E_MI_AUDIO_HPF_FREQ_150;

stVqeConfig.stAecCfg.bComfortNoiseEnable = TRUE;
stVqeConfig.stAecCfg.s16DelaySample = 0;
memcpy(stVqeConfig.stAecCfg.u32AecSupfreq, u32AecSupfreq, sizeof(u32AecSupfreq));
memcpy(stVqeConfig.stAecCfg.u32AecSupIntensity, u32AecSupIntensity, sizeof(u32AecSupIntensity));

stVqeConfig.stAnrCfg.eMode = E_MI_AUDIO_ALGORITHM_MODE_MUSIC;
stVqeConfig.stAnrCfg.u32NrIntensity = 30;
stVqeConfig.stAnrCfg.u32NrSmoothLevel = 10;
stVqeConfig.stAnrCfg.eNrSpeed = E_MI_AUDIO_NR_SPEED_MID;

stVqeConfig.stAgcCfg.eMode = E_MI_AUDIO_ALGORITHM_MODE_USER;
stVqeConfig.stAgcCfg.stAgcGainInfo.s32GainMax = 15;
stVqeConfig.stAgcCfg.stAgcGainInfo.s32GainMin = 0;
stVqeConfig.stAgcCfg.stAgcGainInfo.s32GainInit = 0;
stVqeConfig.stAgcCfg.u32DropGainMax = 55;
stVqeConfig.stAgcCfg.u32AttackTime = 1;
stVqeConfig.stAgcCfg.u32ReleaseTime = 6;
memcpy(stVqeConfig.stAgcCfg.s16Compression_ratio_input, s16Compression_ratio_input, sizeof(s16Compression
_ratio_input));
memcpy(stVqeConfig.stAgcCfg.s16Compression_ratio_output, s16Compression_ratio_output, sizeof(s16Compressi
on_ratio_output));
stVqeConfig.stAgcCfg.s32TargetLevelDb = -3;
stVqeConfig.stAgcCfg.s32NoiseGateDb = -80;
stVqeConfig.stAgcCfg.u32NoiseGateAttenuationDb = 0;

stVqeConfig.stEqCfg.eMode = E_MI_AUDIO_ALGORITHM_MODE_USER;

ret = MI_AI_SetVqeAttr(AiDevId, AiChn, 0, 0, &stVqeConfig);
if (MI_SUCCESS != ret)
{
    printf("set vqe attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

ret = MI_AI_GetVqeAttr(AiDevId, AiChn, 0, 0, &stVqeConfig);
if (MI_SUCCESS != ret)
{
    printf("get vqe attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* enable vqe */
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.
```

2.2.14 MI_AI_GetVqeAttr

- Features

  Get the sound quality enhancement related properties of AI channel.

- Syntax

  MI_S32 MI_AI_GetVqeAttr(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AI_VqeConfig_t *pstVqeConfig);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstVqeConfig	Audio input sound quality enhancement configuration structure pointer	Output

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details
- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libAEC_LINUX.so libAPC_LINUX.so
- Note

  None.
- Example

  Please refer to the MI_AI_SetVqeAttr example section.

2.2.15 MI_AI_EnableVqe

- Features

  Enable sound quality enhancements of AI channel.
- Syntax

  MI_S32 MI_AI_EnableVqe(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);
- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details
- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libAEC_LINUX.so libAPC_LINUX.so
- Note
  - Before enabling the sound quality enhancement function, you must first enable the corresponding AI channel and set the relevant attributes of the sound quality enhancement function of the corresponding AI channel.
  - When the sound quality enhancement function of the same AI channel is enabled multiple times, the return is successful.
- Example

  Please refer to the MI_AI_SetVqeAttr example section.

2.2.16 MI_AI_DisableVqe

- Features

  Disable sound quality enhancements of AI channel

- Syntax

  MI_S32 MI_AI_DisableVqe(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libAEC_LINUX.so libAPC_LINUX.so

- Note
  - When the AI sound quality enhancement is no longer used, this interface should be called to disable it.
  - The sound quality enhancement function of the same AI channel is disabled multiple times and the return is successful.

- Example

  Please refer to the MI_AI_SetVqeAttr example section.

2.2.17 MI_AI_ClrPubAttr

- Features

  Clear AI device properties.

- Syntax

  MI_S32 MI_AI_ClrPubAttr(MI_AUDIO_DEV AiDevId);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note

  Before you can clear device properties, you need to stop the device first.

- Example

  The following code shows how to implement clearing device properties.

```
MI_S32 ret;
MI_AUDIO_DEV AiDevId = 0;
MI_AI_CHN AiChn = 0;
MI_AUDIO_Attr_t stAttr;

memset(&stAttr, 0x0, sizeof(MI_AUDIO_Attr_t));
stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_MONO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
stAttr.u32PtNumPerFrm = 160;
stAttr.u32ChnCnt = 1;

/* set public attribute of AI device */
ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
if (MI_SUCCESS != ret)
{
    printf("set Dev%d attr err:0x%x\n", AiDevId, ret);
    return ret;
}

/* clean public attr of AI device*/
ret = MI_AI_ClrPubAttr(AiDevId);
if (MI_SUCCESS != ret)
{
    printf("clean Dev%d attr err:0x%x\n", AiDevId, ret);
    return ret;
}
```

2.2.18 MI_AI_SaveFile

- Features

  Turn on audio input to save files.

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. Value range: [0, AUIDO_MAX_CHN_NUM]	Input
pstSaveFileInfo	Audio save file attribute structure pointer.	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note

  None.

- Example

  None.

2.2.19 MI_AI_SetVqeVolume

- Features

  Set the volume level of AI channel

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
s32VolumeDb	The volume level.	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note

The following table describes the corresponding gain (DB) of s32volumedb under each device. For Amic and Line in devices, s32volumedb represents the analog gain of the device. For macaron series, pudding series and Ispahan series chips, s32volumedb is the gain corresponding to Dmic.

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

- Example

The following code shows how to implement the function of setting and obtaining gain.

```
MI_S32 ret;
MI_AUDIO_DEV AiDevId = 0;
MI_AI_CHN AiChn = 0;
MI_AUDIO_Attr_t stAttr;
MI_S32 s32VolumeDb;

MI_SYS_Init();

memset(&stAttr, 0x0, sizeof(MI_AUDIO_Attr_t));
stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
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.
```

2.2.20 MI_AI_GetVqeVolume

- Features

  Get the volume level of AI channel

- Syntax

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

- Parameter

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt.	Input
ps32VolumeDb	The volume level of AI channel.	Output

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetVqeVolume example section.

2.2.21 MI_AI_SetAencAttr

- Features

  Set encoding function related properties of AI channel

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstAencConfig	Audio coding configuration structure pointer	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libg711.so libg726.so

- Note

  To set the encoding function, enable the AI channel first.

- Example

  The following code shows how to enable and disable the function of encoding parameters.

```
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.
```

```
/* enable AI Channel */
ret = MI_AI_EnableChn(AiDevId, AiChn);
if (MI_SUCCESS != ret)
{
    printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* set aenc attr */
memset(&stAiAencConfig, 0x0, sizeof(stAiAencConfig));
stAiAencConfig.eAencType = E_MI_AUDIO_AENC_TYPE_G711A;
ret = MI_AI_SetAencAttr(AiDevId, AiChn, &stAiAencConfig);
if (MI_SUCCESS != ret)
{
    printf("set aenc attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* get aenc attr */
memset(&stAiAencConfig, 0x0, sizeof(stAiAencConfig));
ret = MI_AI_GetAencAttr(AiDevId, AiChn, &stAiAencConfig);
if (MI_SUCCESS != ret)
{
    printf("get aenc attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* enable aenc */
ret = MI_AI_EnableAenc(AiDevId, AiChn);
if (MI_SUCCESS != ret)
{
    printf("enable aenc Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* disable aenc */
ret = MI_AI_DisableAenc(AiDevId, AiChn);
if (MI_SUCCESS != ret)
{
    printf("disable aenc Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* disable AI Channel */
ret = MI_AI_DisableChn(AiDevId, AiChn);
if (MI_SUCCESS != ret)
{
    printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* disable AI Device */
ret = MI_AI_Disable(AiDevId);
if (MI_SUCCESS != ret)
{
    printf("disable ai %d err:0x%x\n", AiDevId, ret);
    return ret;
}

MI_SYS_Exit();

```

2.2.22 MI_AI_GetAencAttr

- Features

  Get encoding function related properties of AI channel

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstAencConfig	Audio coding configuration structure pointer	Output

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libg711.so libg726.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetAnecAttr example section.

2.2.23 MI_AI_EnableAenc

- Features

  Enable encoding of AI channel.

- Syntax

  MI_S32 MI_AI_EnableAenc (MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libg711.so libg726.so

- Note

  Before enabling the coding function, you must enable the corresponding AI channel and set the encoding attributes.

- Example

  Please refer to the MI_AI_SetAnecAttr example section.

2.2.24 MI_AI_DisableAenc

- Features

  Disable encoding of AI channel.

- Syntax

  MI_S32 MI_AI_DisableAenc (MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libg711.so libg726.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetAnecAttr example section.

2.2.25 MI_AI_SetAedAttr

- Features

  Set sound event detection function related properties of AI channel.

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstAedConfig	Sound event detecting configuration structure pointer	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libAED_LINUX.so

- Note

  To set the sound event detection function, enable the AI channel first.

- Example

  The following code shows how to set sound event detection parameters, enable sound event detection, and obtain sound event detection results.

```
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)
```

```
{
    printf("enable aed Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* set buffer depth */
stAiChnOutputPort.eModId = E_MI_MODULE_ID_AI;
stAiChnOutputPort.u32DevId = AiDevId;
stAiChnOutputPort.u32ChnId = AiChn;
stAiChnOutputPortu32PortId = 0;
MI_SYS_SetChnOutputPortDepth(&stAiChnOutputPort, 1, 8);

/* get port fd */
stChnPort.eModId = E_MI_MODULE_ID_AI;
stChnPort.u32DevId = AiDevId;
stChnPort.u32ChnId = AiChn;
stChnPort.u32PortId = 0;
ret = MI_SYS_GetFd(&stChnPort, &s32Fd);
if (MI_SUCCESS != ret)
{
    printf("Dev%d Chn%d failed to call MI_SYS_GetFd!!!\n", AiDevId, AiChn);
    return ret;
}

/* select 100ms */
FD_ZERO(&readFdSet);
FD_SET(s32Fd, &readFdSet);
stTimeOut.tv_sec = 0;
stTimeOut.tv_usec = 100 * 1000;
ret = select(s32Fd + 1, &readFdSet, NULL, NULL, &stTimeOut);
if (FD_ISSET(s32Fd, &readFdSet))
{
    ret = MI_AI_GetFrame(AiDevId, AiChn, &stAiChFrame, &stAecFrame, 0);
    if (MI_SUCCESS == ret)
    {
        /* get aed result */
        MI_AI_GetAedResult(AiDevId, AiChn, &stAedResult);

        MI_AI_ReleaseFrame(AiDevId, AiChn, &stAiChFrame, &stAecFrame);
    }
}

/* disable aed */
ret = MI_AI_DisableAed(AiDevId, AiChn);
if (MI_SUCCESS != ret)
{
    printf("disable aed Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* disable AI Channel */
ret = MI_AI_DisableChn(AiDevId, AiChn);
if (MI_SUCCESS != ret)
{
    printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* disable AI Device */
ret = MI_AI_Disable(AiDevId);
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();
```

2.2.26 MI_AI_GetAedAttr

- Features

  Get sound event detection function related properties of AI channel.

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstAedConfig	Sound event detecting configuration structure pointer	Output

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libAED_LINUX.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetAedAttr example section.

2.2.27 MI_AI_EnableAed

- Features

  Enable sound event detection of AI channel.

- Syntax

  MI_S32 MI_AI_EnableAed(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libAED_LINUX.so

- Note

  Before enabling the sound event detection function, the AI channel must be enabled and the sound event detection property must be set first.

- Example

  Please refer to the MI_AI_SetAedAttr example section.

2.2.28 MI_AI_DisableAed

- Features

  Disable sound event detection of AI channel.

- Syntax

  MI_S32 MI_AI_DisableAed(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libAED_LINUX.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetAedAttr example section.

2.2.29 MI_AI_GetAedResult

- Features

  Get the sound event detection result of AI channel.

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstAedResult	Sound event detection result structure pointer.	Output

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libAED_LINUX.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetAedAttr example section.

2.2.30 MI_AI_SetExtAecChn

- Features

  Set the external echo cancellation function reference AI channel

- Syntax

  MI_S32 MI_AI_SetExtAecChn(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AI_CHN AiAECSndChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
AiAECSndChn	The external echo cancellation function reference AI channel.	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h

- Note

  The external echo reference channel needs to be set before the channel is enabled. This interface is currently supported in Mono mode only.

- Example

  The following code shows how to set the external AEC reference channel.

```
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. /* set ext Aec Chn */
41. ret = MI_AI_SetExtAecChn(AiDevId, AiChn, ExtAecChn);
42. if (MI_SUCCESS != ret)
43. {
44.     printf("set ext aec chn Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, 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. /* disable AI Channel */
57. ret = MI_AI_DisableChn(AiDevId, AiChn);
58. if (MI_SUCCESS != ret)
59. {
60.     printf("disable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
61.     return ret;
62. }
63.
64. /* disable AI Device */
65. ret = MI_AI_Disable(AiDevId);
66. if (MI_SUCCESS != ret)
67. {
68.     printf("disable ai %d err:0x%x\n", AiDevId, ret);
69.     return ret;
70. }
71.
72. MI_SYS_Exit();
```

2.2.31 MI_AI_SetSslInitAttr

- Features

  Set Sound source localization function initialized properties of AI channel

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstSslInitAttr	Sound source localization function initialized structure pointer	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

- Note

  Initialization parameters for sound source location can only be set before enabling.

- Example

  The following code shows how to set the sound source location parameters to enable the sound source location and obtain the sound source location results.

```
MI_S32 ret;
MI_AUDIO_Attr_t stAttr;
MI_AUDIO_Dev AiDevId = 0;
MI_AI_CHN AiChn = 0;
MI_SYS_ChnPort_t stChnPort;
MI_S32 s32Fd;
fd_set readFdSet;
struct timeval stTimeOut;
MI_AUDIO_Frame_t stAiChFrame;
MI_AUDIO_AecFrame_t stAecFrame;
MI_SYS_ChnPort_t stAiChnOutputPort;
MI_S32 s32Doa;

MI_AI_SslInitAttr_t stSslInit = {
    .bBfMode = FALSE,
      .u32MicDistance = 3,
};

MI_AI_SslConfigAttr_t stSslConfig = {
    .s32Temperature = 25,
    .s32NoiseGateDbfs = -40,
    .s32DirectionFrameNum = 300,
};

MI_AI_SslInitAttr_t stAiGetSslInitAttr;
MI_AI_SslConfigAttr_t stAiGetSslConfigAttr;

MI_SYS_Init();

```

```
stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_STEREO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
stAttr.u32PtNumPerFrm = 160;
stAttr.u32ChnCnt = 1;

/* set public attribute of AI device */
ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
if (MI_SUCCESS != ret)
{
    printf("set ai %d attr err:0x%x\n", AiDevId, ret);
    return ret;
}

/* get public attribute of AI device */
ret = MI_AI_GetPubAttr(AiDevId, &stAttr);
if (MI_SUCCESS != ret)
{
    printf("get ai %d attr err:0x%x\n", AiDevId, ret);
    return ret;
}

/* enable AI device */
ret = MI_AI_Enable(AiDevId);
if (MI_SUCCESS != ret)
{
    printf("enable ai %d err:0x%x\n", AiDevId, ret);
    return ret;
}

/* enable AI Channel */
ret = MI_AI_EnableChn(AiDevId, AiChn);
if (MI_SUCCESS != ret)
{
    printf("enable Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* set ssl init attr */
ret = MI_AI_SetSslInitAttr(AiDevId, AiChn, &stSslInit);
if (MI_SUCCESS != ret)
{
    printf("set ssl init attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* get ssl init attr */
ret = MI_AI_GetSslInitAttr(AiDevId, AiChn, &stAiGetSslInitAttr);
if (MI_SUCCESS != ret)
{
    printf("get ssl init attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    return ret;
}

/* set ssl config attr */
ret = MI_AI_SetSslConfigAttr(AiDevId, AiChn, &stSslConfig);
if (MI_SUCCESS != ret)
{
    printf("set ssl config attr Dev%d Chn%d err:0x%x\n", AiDevId, AiChn, ret);
    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.stAiChnOutputPortu32PortId = 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();
```

2.2.32 MI_AI_GetSslInitAttr

- Features

  Get Sound source localization function initialized properties of AI channel

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstSslInitAttr	Sound source localization function initialized structure pointer	Output

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetSslInitAttr example section.

2.2.33 MI_AI_SetSslConfigAttr

- Features

  Set Sound source localization function configured properties of AI channel

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstSslConfigAttr	Sound source localization function configured structure pointer	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

- Note

  You can set configured structure for each frame.

- Example

  Please refer to the MI_AI_SetSslInitAttr example section.

2.2.34 MI_AI_GetSslConfigAttr

- Features

  Get Sound source localization function configured properties

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstSslConfigAttr	Sound source localization function configured structure pointer	Output

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetSslInitAttr example section.

2.2.35 MI_AI_EnableSsl

- Features

  Enable Sound source localization function.

- Syntax

  MI_S32 MI_AI_EnableSsl(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

- Note

  Before enabling the sound source localization function, it is necessary to enable the AI channel and set the initialization parameters and configuration parameters of the sound source localization.

- Example

  Please refer to the MI_AI_SetSslInitAttr example section.

2.2.36 MI_AI_DisableSsl

- Features

  Disable Sound source localization function.

- Syntax

  MI_S32 MI_AI_DisableSsl(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetSslInitAttr example section.

2.2.37 MI_AI_GetSslDoa

- Features

  Get the Sound source localization result.

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
ps32SslDoa	Sound source localization result pointer	Output

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libSSL_LINUX.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetSslInitAttr example section.

2.2.38 MI_AI_SetBfInitAttr

- Features

  Set Beamforming function initialized properties of AI channel.

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstBfInitAttr	Beamforming function initialized structure pointer	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libBF_LINUX.so

- Note

  Initialization parameters for beamforming can only be set before enabling.

- Example

  The following code shows how to set the parameters of beamforming function and enable the beamforming function.

```
MI_S32 ret;
MI_AUDIO_Attr_t stAttr;
MI_AUDIO_Dev AiDevId = 0;
MI_AI_CHN AiChn = 0;
MI_AI_BfInitAttr_t stBfInit = {
    .u32ChanCnt = 2,
    .u32MicDistance = 3,
};
MI_AI_BfConfigAttr_t stBfConfig = {
    .s32Temperature = 25,
    .s32NoiseGateDbfs = -40,
    .s32NoiseSupressionMode = 8,
    .s32NoiseEstimation = 1,
    .outputGain = 0.7,
};
MI_AI_BfInitAttr_t stAiGetBfInitAttr;
MI_AI_BfConfigAttr_t stAiGetBfConfigAttr;
MI_BOOL bAiSetBfDoa = TRUE;
MI_S32 s32AiBfDoa = 0;

MI_SYS_Init();

stAttr.eBitwidth = E_MI_AUDIO_BIT_WIDTH_16;
stAttr.eSamplerate = E_MI_AUDIO_SAMPLE_RATE_8000;
stAttr.eSoundmode = E_MI_AUDIO_SOUND_MODE_STEREO;
stAttr.eWorkmode = E_MI_AUDIO_MODE_I2S_SLAVE;
stAttr.u32PtNumPerFrm = 160;
stAttr.u32ChnCnt = 1;

/* set public attribute of AI device */
ret = MI_AI_SetPubAttr(AiDevId, &stAttr);
if (MI_SUCCESS != ret)
{
    printf("set ai %d attr err:0x%x\n", AiDevId, ret);
    return ret;
}

/* get public attribute of AI device */
ret = MI_AI_GetPubAttr(AiDevId, &stAttr);
if (MI_SUCCESS != ret)
{
    printf("get ai %d attr err:0x%x\n", AiDevId, ret);
    return ret;
}

/* enable AI device */
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();
```

2.2.39 MI_AI_GetBfInitAttr

- Features

  Get Beamforming function initialized properties of AI channel.

- Syntax

  MI_S32 MI_AI_GetBfInitAttr(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AI_BfInitAttr_t* pstBfInitAttr);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstBfInitAttr	Beamforming function initialized structure pointer	Output

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libBF_LINUX.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetBfInitAttr example section.

2.2.40 MI_AI_SetBfConfigAttr

- Features

  Set Beamforming function configured properties of AI channel.

- Syntax

  MI_S32 MI_AI_SetBfConfigAttr(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AI_BfConfigAttr_t* pstBfConfigAttr);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstBfConfigAttr	Beamforming function configured structure pointer	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libBF_LINUX.so

- Note

  You can set configured structure for each frame.

2.2.41 MI_AI_GetBfConfigAttr

- Features

  Get Beamforming function configured properties of AI channel.

- Syntax

  MI_S32 MI_AI_GetBfConfigAttr(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_AI_BfConfigAttr_t* pstBfConfigAttr);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
pstBfConfigAttr	Beamforming function configured structure pointer	Output

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libBF_LINUX.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetBfInitAttr example section.

2.2.42 MI_AI_EnableBf

- Features

  Enable Beamforming function of AI channel.

- Syntax

  MI_S32 MI_AI_EnableBf(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libBF_LINUX.so

- Note

  Before enabling the beamforming function, it is necessary to enable the AI channel and set the initialization parameters and configuration parameters of the beamforming function.

- Example

  Please refer to the MI_AI_SetBfInitAttr example section.

2.2.43 MI_AI_DisableBf

- Features

  Disable Beamforming function of AI channel.

- Syntax

  MI_S32 MI_AI_DisableBf(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libBF_LINUX.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetBfInitAttr example section.

2.2.44 MI_AI_SetBfAngle

- Features

  Set fixed angle for Beamforming function of AI channel.

- Syntax

  MI_S32 MI_AI_SetBfDoa(MI_AUDIO_DEV AiDevId, MI_AI_CHN AiChn, MI_S32 s32BfAngle);

- Parameters

Parameter Name	Description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. The supported channel range is determined by the maximum number of channels u32ChnCnt	Input
s32BfAngle	Fixed angel of Beamforming	Input

- Return value
  - Zero: Successful
  - Non-zero: Failed, see error code for details

- Dependency
  - Header: mi_ai.h
  - Library: libmi_ai.a/libmi_ai.so libBF_LINUX.so

- Note

  None.

- Example

  Please refer to the MI_AI_SetBfInitAttr example section.

3. AI DATA TYPE

The AI module related data types are defined as follows:

MI_AUDIO_DEV	Define the audio input / output device number
MI_AUDIO_MAX_CHN_NUM	Define the maximum number of channels for audio input / output devices
MI_AI_CHN	Define the audio input channel
MI_AUDIO_SampleRate_e	Define the audio sample rate
MI_AUDIO_Bitwidth_e	Define audio sampling accuracy
MI_AUDIO_Mode_e	Define the audio input and output working mode
MI_AUDIO_SoundMode_e	Define audio channel mode
MI_AUDIO_Attr_t	Defining audio input/output device attribute structures
MI_AI_ChnParam_t	Define channel parameter structure
MI_AUDIO_Frame_t	Defining an audio frame data structure
MI_AUDIO_AecFrame_t	Defining echo cancellation reference frame information structure
MI_AUDIO_SaveFileInfo_t	Define audio save file function configuration information structure
MI_AI_VqeConfig_t	Defining audio input sound quality enhancement configuration information structure
MI_AUDIO_HpfConfig_t	Defining an audio high-pass filtering function configuration information structure
MI_AUDIO_HpfFreq_e	Define the audio high pass filter cutoff frequency
MI_AI_AecConfig_t	Defining an audio echo cancellation configuration information structure
MI_AUDIO_AnrConfig_t	Define audio voice noise reduction function configuration information structure
MI_AUDIO_AgcConfig_t	Defining an audio automatic gain control configuration information structure
MI_AUDIO_EqConfig_t	Define the audio equalizer function configuration information structure
MI_AI_AecConfig_t	Defining audio echo cancellation function configuration information structure
MI_AI_AencConfig_t	Defining an audio coding function configuration information structure
MI_AI_AedConfig_t	Define the sound event detection function configuration information structure.
MI_AUDIO_AedSensitivity_e	Define the sensitivity of sound event detection

MI_AI_AedResult_t	Define the sound event detection result structure.
MI_AI_ChnGainConfig_t	Define the channel gain setting structure
MI_AI_SslInitAttr_t	Define the Sound source localization initialized structure
MI_AI_SslConfigAttr_t	Define the Sound source localization configured structure
MI_AI_BfInitAttr_t	Define the Beamforming initialized structure
MI_AI_BfConfigAttr_t	Define the Beamforming configured structure

3.1. MI_AUDIO_DEV

- Description

  Define the audio input/output device number

- Definition

  typedef MI_S32 MI_AUDIO_DEV

- Note

  The following table is a comparison table of chip's AI/AO Dev ID and physical device.

Dev ID	AI Dev	AO Dev
0	Amic	Line out
1	Dmic	I2S TX
2	I2S RX	HDMI (TAIYAKI series chip support)
3	Line in	HDMI + Line out (TAIYAKI series chip support)
4	Amic + I2S RX (Takoyaki series chip support)	
5	Dmic + I2S RX (Takoyaki series chip support)	

The following table is the specifications of different series of chips

	Pretzel	Macaron	TAIYAKI	TAKOYAKI	Pudding	Ispahan
Amic	Support 2 channel 8/16/32/48KHz sampling rate	Support 2 channel 8/16/32/48KHz sampling rate	Support 1 channel 8/16/32/48KHz sampling rate	Support 1 channel 8/16/32/48KHz sampling rate	Support 2 channel 8/16/32/48KHz sampling rate	Support 2 channel 8/16/32/48KHz sampling rate
Dmic	Standard dmic interface, supporting 4 channels 8/16/32/48KHz sampling rate	Non standard dmic interface, supporting 2 channel 8/16/32kHz sampling rates	Standard dmic interface, supporting 4 channels 8/16/32/48KHz sampling rate	Standard dmic interface, supporting 4 channels 8/16/32/48KHz sampling rate	Non standard dmic interface, supporting 2 channel 8/16/32kHz sampling rates	Non standard dmic interface, supporting 2 channel 8/16/32kHz sampling rates
I2S RX	It supports standard I2S mode and TDM mode. TDM mode can be extended to 8 channels. It supports 4-wire and 6-wire modes and can provide MCLK. It supports 8/16/32/48KHz sampling rate.	Only standard I2S mode is supported and can only be used as master, only 4-wire mode is supported, MCLK is not available. Support 8/16/32/48KHz sampling rate.	Only standard I2S mode is supported and can only be used as master, only 4-wire mode is supported, MCLK is not available. Support 8/16/32/48KHz sampling rate.	Only standard I2S mode is supported and can only be used as master, only 4-wire mode is supported, MCLK is not available. Support 8/16/32/48KHz sampling rate.	It supports standard I2S mode and TDM mode. TDM mode can be extended to 8 channels. It supports 4-wire and 6-wire modes and can provide MCLK. It support 8/16/32/48KHz sampling rate.	Only standard I2S mode is supported and can only be used as master, only 4-wire mode is supported, MCLK is not available. Support 8/16/32/48KHz sampling rate.
Line in	Support 2 channel 8/16/32/48KHz sampling rate	Support 2 channel 8/16/32/48KHz sampling rate	Support 1 channel 8/16/32/48KHz sampling rate	Support 1 channel 8/16/32/48KHz sampling rate	Support 2 channel 8/16/32/48KHz sampling rate	Support 2 channel 8/16/32/48KHz sampling rate

- Related data types and interfaces
  None.

3.2. MI_AUDIO_MAX_CHN_NUM

- Description
  Defines the maximum number of channels for an audio input / output device.

- Definition
  #define MI_AUDIO_MAX_CHN_NUM 16

- Note
  None.

- Related data types and interfaces
  None.

3.3. MI_AI_CHN

- Description
  Define the audio input channel.

- Definition
  typedef MI_S32 MI_AI_CHN

- Note
  None.

- Related data types and interfaces
  None.

3.4. MI_AUDIO_SampleRate_e

- Description
  Define the audio sample rate.

- Definition

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

- Member

Member Name	Description
E_MI_AUDIO_SAMPLE_RATE_8000	8kHz sampling rate
E_MI_AUDIO_SAMPLE_RATE_11025	11.025kHz sampling rate
E_MI_AUDIO_SAMPLE_RATE_12000	12kHz sampling rate
E_MI_AUDIO_SAMPLE_RATE_16000	16kHz sampling rate
E_MI_AUDIO_SAMPLE_RATE_22050	22.05kHz sampling rate
E_MI_AUDIO_SAMPLE_RATE_24000	24kHz sampling rate
E_MI_AUDIO_SAMPLE_RATE_32000	32kHz sampling rate
E_MI_AUDIO_SAMPLE_RATE_44100	44.1kHz sampling rate
E_MI_AUDIO_SAMPLE_RATE_48000	48kHz sampling rate
E_MI_AUDIO_SAMPLE_RATE_96000	96kHz sampling rate

- Note

The enumeration value here does not start at 0 but it is the same as the actual sample rate value.
Audio input only supports 8/16/32/48kHz sampling rate.

- Related data types and interfaces

MI_AUDIO_Attr_t.

3.5. MI_AUDIO_Bitwidth_e

- Description

Define audio sampling accuracy.

- Definition

```
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;
```

- Member

Member Name	Description
E_MI_AUDIO_BIT_WIDTH_16	Sampling accuracy is 16bit width
E_MI_AUDIO_BIT_WIDTH_24	Sampling accuracy is 24 bit width

- Note

Currently the software only supports 16bit bit width.

- Related data types and interfaces

None.

3.6. MI_AUDIO_Mode_e

- Description

Define the audio input and output device operating mode.

- Definition

```
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;
```

- Member

Member Name	Description
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

- Note

Whether the master mode and the slave mode is supported depends on the chip involved.

- Related data types and interfaces

MI_AUDIO_Attr_t

3.7. MI_AUDIO_SoundMode_e

- Description

  Define the audio channel mode.

- Definition

```
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
```

- Member

Member Name	Description
E_MI_AUDIO_SOUND_MODE_MONO	Mono
E_MI_AUDIO_SOUND_MODE_STEREO	Two channels.
E_MI_AUDIO_SOUND_MODE_QUEUE	All audio data is arranged in order in one channel for use in audio capture

- Note

  When the sound mode is E_MI_AUDIO_SOUND_MODE_QUEUE, only channel 0 needs to be set as the parameter. Other channels will use the same parameter as channel 0, but the processing of each channel is independent of each other. At the same time, each channel gain is independent of each other, which needs to be set separately by the user. Suppose there are 4 channels at present, and the buffer size of the user is 4nByte, then the data size of each channel is 4n / 4 = nByte. The data are arranged as follows:
  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……

- Related data types and interfaces

  MI_AUDIO_Attr_t

3.8. MI_AUDIO_AencType_e

- Description

  Define the audio encoding type.

- Definition

```
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;
```

- Member

Member Name	Description
E_MI_AUDIO_AENC_TYPE_G711A	G711A encoding
E_MI_AUDIO_AENC_TYPE_G711U	G711U encoding
E_MI_AUDIO_AENC_TYPE_G726	G726 encoding

- Note

  None.

- Related data types and interfaces

  MI_AUDIO_AencG726Config_t

3.9. MI_AUDIO_G726Mode_e

- Description

  Define the G726 operating mode.

- Definition

```
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;
```

- Member

Member Name	Description
E_MI_AUDIO_G726_MODE_16	G726 16K bit rate mode.
E_MI_AUDIO_G726_MODE_24	G726 24K bit rate mode.
E_MI_AUDIO_G726_MODE_32	G726 32K bit rate mode.
E_MI_AUDIO_G726_MODE_40	G726 40K bit rate mode.

- Note

  None.

- Related data types and interfaces

  MI_AUDIO_AencG726Config_t

3.10. MI_AUDIO_I2sFmt_e

- Description

  I2S format setting

- Definition

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

- Member

Member Name	Description
E_MI_AUDIO_I2S_FMT_I2S_MSB	I2S standard format, highest priority
E_MI_AUDIO_I2S_FMT_LEFT_JUSTIFY_MSB	I2S left-aligned format, highest priority

- Note

None.

- Related data types and interfaces

MI_AUDIO_I2sConfig_t

3.11. MI_AUDIO_I2sMclk_e

- Description

I2S MCLK setting

- Definition

```
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;
```

- Member

Member Name	Description
E_MI_AUDIO_I2S_MCLK_0	Turn off MCLK
E_MI_AUDIO_I2S_MCLK_12_288M	Set MCLK to 12.88M
E_MI_AUDIO_I2S_MCLK_16_384M	Set MCLK to 16.384M
E_MI_AUDIO_I2S_MCLK_18_432M	Set MCLK to 18.432M
E_MI_AUDIO_I2S_MCLK_24_576M	Set MCLK to 24.576M
E_MI_AUDIO_I2S_MCLK_24M	Set MCLK to 24M
E_MI_AUDIO_I2S_MCLK_48M	Set MCLK to 48M

- Note

None.

- Related data types and interfaces
  MI_AUDIO_I2sConfig_t

3.12. MI_AUDIO_I2sConfig_t

- Description
  Define the I2S attribute structure.

- Definition

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

- Member

Member Name	Description
eFmt	I2S format settings, static attribute.
eMclk	I2S MCLK clock setting, static attribute.
bSyncClock	The AI synchronizes the AO clock. Static attribute.

- Note
  None.

- Related data types and interfaces
  MI_AUDIO_Attr_t

3.13. MI_AUDIO_Attr_t

- Description
  Define the audio input and output device attribute structure.

- Definition

```
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;
```

- Member

Member Name	Description
eSamplerate	Audio sample rate. Static attribute.
eBitwidth	Audio sampling accuracy (in slave mode, this parameter must match the sampling accuracy of the audio AD/DA). Static attribute.
eWorkmode	Audio input and output working mode. Static attribute.
eSoundmode	Audio channel mode. Static attribute.
u32FrmNum	The number of cached frames. Reserved, unused.
u32PtNumPerFrm	The number of sampling points per frame. The value ranges from: 128, 128*2, …, 128*N . Static attribute.
u32CodecChnCnt	The number of channels supported codec. Reserved, unused.
u32ChnCnt	The number of channels supported, the maximum number of channels that are actually enabled. Values: 1, 2, 4, 8, 16. (The input supports up to MI_AUDIO_MAX_CHN_NUM channels, and the output supports up to 2 channels)
MI_AUDIO_I2sConfig _t stI2sConfig;	Set I2S work properties

- Note

None.

- Related data types and interfaces

MI_AI_SetPubAttr

3.14. MI_AI_ChnParam_t

- Description

Define the channel parameter structure.

- Definition

```
typedef struct MI_AI_ChnParam_s
{
    MI_AI_ChnGainConfig_t stChnGain;
    MI_U32 u32Reserved;
} MI_AI_ChnParam_t
```

- Member

Member Name	Description
stChnGain	AI channel gain setting structure
u32Reserved	Reserved, not used.

- Note

None.

- Related data types and interfaces

MI_AI_SetChnParam
MI_AI_GetChnParam

3.15. MI_AUDIO_Frame_t

- Description

Define the audio frame structure.

- Definition

```
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;
```

- Member

Member Name	Description
eBitwidth	Audio sampling accuracy
eSoundmode	Audio channel mode.
pVirAddr[MI_AUDIO_MAX_CHN_NUM]	Audio frame data virtual address.
u64TimeStamp	Audio frame timestamp. With μs unit.
u32Seq	Audio frame number.
u32Len	Audio frame length. In bytes.
u32PoolId[2]	Audio frame buffer pool ID.
apSrcPcmVirAddr[MI_AUDIO_MAX_CHN_NUM]	Audio original pcm frame data virtual address.
u32SrcPcmLen	Audio original pcm frame length in bytes.

- Note
  - u32Len (audio frame length) refers to the data length of a whole buffer.
  - Mono data are directly stored, the number of sampling points is u32PtNumPerFrm, the length is u32Len; stereo data are stored by left and right channel interleaving storage method, with data format of L,R,L,R,L,R... , the sampling point is u32PtNumPerFrm, and the length is u32Len. When sound mode is queue mode, data are stored in channel order. The data length of each channel is u32Len/number of channels, and the sampling points are u32PtNumPerFrm. The data format is:
    Chn0, Chn0, Chn0, Chn0, Chn0, Chn0, Chn0...
    Chn1, Chn1, Chn1, Chn1, Chn1, Chn1, Chn1...
    Chn2, Chn2, Chn2, Chn2, Chn2, Chn2, Chn2...
    Chn3, Chn3, Chn3, Chn3, Chn3, Chn3, Chn3...
    ...
  - When no algorithm is turned on, pVirAddr is equivalent to apSrcPcmVirAddr, and u32Len is equivalent to u32SrcPcmLen. When the algorithm is turned on, pVirAddr returns the data processed by the algorithm, and apSrcPcmVirAddr returns the original PCM data collected by Audio input.

- Related data types and interfaces
  None.

3.16. MI_AUDIO_AecFrame_t

- Description
  Define the audio echo cancellation reference frame information structure.

- Definition

```
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;
```

- Member

Member Name	Description
stRefFrame	The echo cancels the reference frame structure.
bValid	The reference frame is valid ranges: TRUE: The reference frame is valid. FALSE: The reference frame is invalid. If this is invalid, this reference frame cannot be used for echo cancellation.

- Note
  None.

- Related data types and interfaces
  None.

3.17. MI_AUDIO_SaveFileInfo_t

- Description

  Define the audio save file function configuration information structure.

- Definition

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

- Member

Member Name	Description
bCfg	Configure the enable switch.
szFilePath	Audio file save path
u32FileSize	File size, ranging from [1,10240] KB .

- Note

  None.

- Related data types and interfaces

  MI_AI_SaveFile

3.18. MI_AI_VqeConfig_t

- Description

  Define an audio input sound quality enhancement configuration information structure.

- Definition

```
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_AnrConfig_t stAnrCfg;
    MI_AUDIO_AgcConfig_t stAgcCfg;
    MI_AUDIO_EqConfig_t stEqCfg;
}MI_AI_VqeConfig_t;
```

- Member

Member Name	Description
bHpfOpen	Whether the high-pass filtering function is enabled or not.
bAecOpen	Whether the echo cancellation function enables the flag.
bAnrOpen	Whether the voice noise reduction function is enabled.
bAgcOpen	Whether the automatic gain control function enables the flag
bEqOpen	Whether the equalizer function is enabled
u32ChnNum	Channel number of data
s32WorkSampleRate	Working sampling frequency. This parameter is the working sampling rate of the internal function algorithm. Value range: 8KHz/16KHz. The default is 8KHz.
s32FrameSample	The frame length of VQE, that is, the number of sampling points. Only support 128.
stHpfCfg	High-pass filtering function related configuration information.
stAecCfg	Echo cancellation function related configuration information.
stAnrCfg	Configuration information related to the voice noise reduction function.
stAgcCfg	Automatic gain control related configuration information.
stEqCfg	Equalizer related configuration information.

- Note

None.

- Related data types and interfaces

None.

3.19. MI_AUDIO_HpfConfig_t

- Description

Define an audio high-pass filtering function configuration information structure.

- Definition

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

- Member

Member Name	Description
eMode	Audio algorithm operating mode.
eHpfFreq	High pass filter cutoff frequency selection. 80: The cutoff frequency is 80 Hz; 120: The cutoff frequency is 120 Hz; 150: The cutoff frequency is 150 Hz. The default value is 150.

- Note

None.

- Related data types and interfaces

MI_AI_VqeConfig_t

3.20. MI_AUDIO_HpfFreq_e

- Description

Define the audio high pass filter cutoff frequency.

- Definition

```
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;
```

- Member

Member Name	Description
E_MI_AUDIO_HPF_FREQ_80	The cutoff frequency is 80 Hz.
E_MI_AUDIO_HPF_FREQ_120	The cutoff frequency is 120 Hz.
E_MI_AUDIO_HPF_FREQ_150	The cutoff frequency is 150 Hz.

- Note

The default configuration is 150Hz

- Related data types and interfaces

MI_AI_VqeConfig_t

3.21. MI_AI_AecConfig_t

- Description

Define an audio echo cancellation configuration information structure.

- Definition

```
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;
```

- Member

Member Name	Description
bComfortNoiseEnable	Whether to add noise. 0: not added; 1: Add.
s16DelaySample	The number of samples at the sampling point is delayed. Only valid when AEC is stereo processing, the default value is 0.
u32AecSupfreq	The echo cancellation protection frequency range, the latter data must be greater than or equal to the previous data. For example: u32AecSupfreq [0] = 10, then: u32AecSupfreq [1] must be greater than or equal to 10. The maximum frequency corresponding to the current sampling rate is divided into 128 copies on average, and u32AecSupfreq represents the number of copies constituting a frequency band. For example, suppose the current sampling rate is 16K and the corresponding maximum frequency is 8K. Each copy is 63Hz (8000 / 128 ≈ 62.5Hz). In case we use the recommended values {4,6,36, 49,50,51}, the protection frequency range will be {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, 3125~3178.5Hz, 3178.5Hz~8000Hz}. Range [1,128]; step size 1 Recommended values: {4,6,36,49,50,51}
u32AecSupIntensity	Echo cancellation protection, the smaller the value, the stronger the protection effect. Correspond to secondary parameter u32AecSupfreq. u32AecSupIntensity [0] corresponds to 0 ~ u32AecSupfreq [0], u32AecSupIntensity [1] corresponds to u32AecSupfreq [0] ~ u32AecSupfreq [1], and so on. Range [0,15]; step size 1 Recommended values {5,4,4,5,10,10,10}

- Note

None.

- Related data types and interfaces
  MI_AI_VqeConfig_t

3.22. MI_AUDIO_AnrConfig_t

- Description
  Define the audio voice noise reduction function configuration information structure.

- Definition

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

- Member

Member Name	Description
eMode	Audio algorithm operating mode. Note: ANR mode will affect the function of AGC to some extent.
u32NrIntensity	Noise reduction strength configuration, the larger the configuration value, the higher the noise reduction, but also the loss / damage of the detail sound. Range [0,30] ; step size 1 The default value is 20.
u32NrSmoothLevel	Smoothness, the larger the value, the smoother Range [0,10]; step size 1 The default value is 10.
eNrSpeed	Noise convergence speed, low speed, medium speed, high speed The default value is medium speed.

- Note
  In the case where both ANR and AGC are enabled, if ANR is set to user mode, AGC will process the audio data in frequency domain, and then evaluate the speech signal and make corresponding boost and cut. When ANR is set to default/music mode, AGC will process the audio data in time domain and boost and cut the data in full frequency band.

- Related data types and interfaces
  MI_AI_VqeConfig_t

3.23. MI_AUDIO_NrSpeed_e

- Description
  Define noise convergence speed

- Definition

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

- Member

Member Name	Description
E_MI_AUDIO_NR_SPEED_LOW	Low speed.
E_MI_AUDIO_NR_SPEED_MID	Medium speed.
E_MI_AUDIO_NR_SPEED_HIGH	High speed.

- Note

None.

- Related data types and interfaces

MI_AO_VqeConfig_t

3.24. MI_AUDIO_AgcConfig_t

- Description

Define an audio automatic gain control configuration information structure.

- Definition

```
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;
```

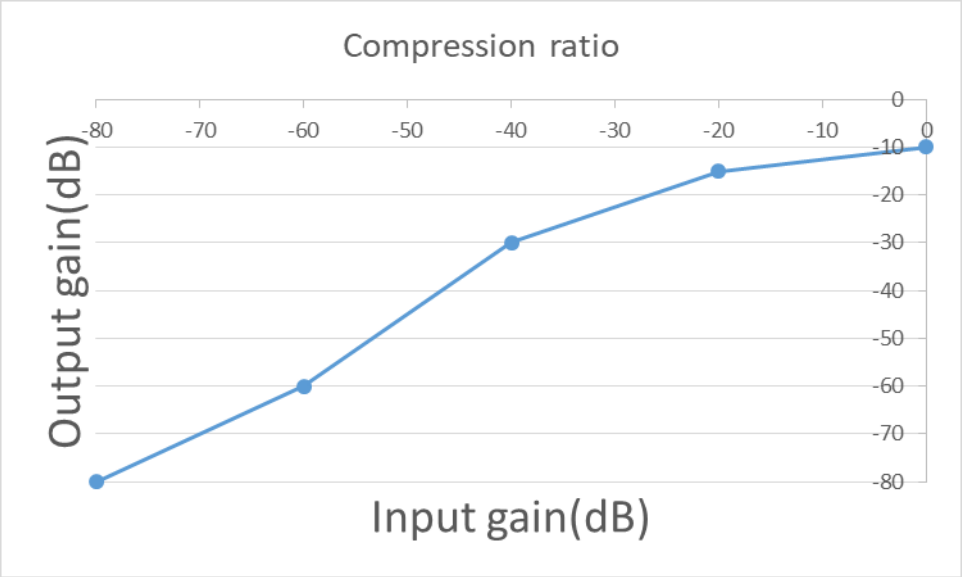
- Member

Member Name	Description
eMode	Audio algorithm operating mode.
stAgcGainInfo	Define the maximum, minimum, and initial values of the AGC gain
u32DropGainMax	Maximum gain decrease, preventing output saturation Range [0,60]; step size 1 The default value is 55.

Member Name	Description
u32AttackTime	Gain fall time interval length, 1 unit in 16 milliseconds Range [1,20]; step size 1 The default value is 0.
u32ReleaseTime	Gain rise time interval length, 1 unit in 16 milliseconds Range [1, 20] ; step size 1 The default value is 0.
s16Compression_ratio_input[5]	Input RMS energy. This parameter should be used together with the compression ratio output. The gain curve consists of multiple turning points with different slopes. Value range [-80,0]; step size 1
s16Compression_ratio_output[5]	Output target RMS energy related input energy. The gain curve consists of multiple turning points with different slopes. Value range [-80,0]; step size 1
s32TargetLevelDb	Target level, processed maximum level threshold Range [-80,0] dB; step size 1 The default value is 0.
s32NoiseGateDb	Noise floor Range [-80,0]; step size 1 Note: When the value is -80, the noise floor will not work. The default value is -55.
u32NoiseGateAttenuationDb	The percentage of attenuation of the input source when the noise floor value is effective Range [0,100]; step size 1 The default value is 0.

➢ Note

- In the case where both ANR and AGC are enabled, if ANR is set to user mode, AGC will process the audio data in frequency domain, and then evaluate the speech signal and make corresponding boost and cut. When ANR is set to default/music mode, AGC will process the audio data in time domain and boost and cut the data in full frequency band.
- S16Compression_ratio_input and s16Compression_ratio_output should be set according to the desired gain curve.
- As shown in the line graph below, the input gain of -80~0dB is divided into four slopes. The first section is -80db ~ -60db. The original gain is maintained within this range and the slope is 1. The second section is -60db ~ -40db, in which the gain needs to be slightly increased and the slope is 1.5. The third section is -40db ~ -20db and the slope in this range is 1.25. The fourth section is -20db ~0dB and the slope in this range is 0.25. Set s16Compression_ratio_input and s16Compression_ratio_output according to the turning point of the curve. If not so many sections of the curve are needed, then fill in 0 for the unwanted parts of the array.



- Related data types and interfaces
  MI_AI_VqeConfig_t

3.25. AgcGainInfo_t

- Description
  AGC gain value

- Definition

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

- Member

Member Name	Description
s32GainMax	Maximum gain Range [0,60]; step size 1 The default value is 15.
s32GainMin	Minimum gain Range [-20,30]; step size 1 The default value is 0.
s32GainInit	Initial gain Range [-20,60]; step size 1 The default value is 0.

- Note
  None.

- Related data types and interfaces
  MI_AO_VqeConfig_t

3.26. MI_AUDIO_EqConfig_t

- Description
  Define the audio equalizer function configuration information structure.

- Definition

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

- Member

Member Name	Description
eMode	Audio algorithm operating mode.
s16EqGainDb[129]	Equalizer gain adjustment value It divides the frequency range of the current sampling rate into 129 parts for adjustment. Range [-50,20]; step size 1 The default value is 0. For example, if the current sampling rate is 16K and the corresponding maximum frequency is 8K, then the frequency range of a single adjustment is 62Hz (8000/129≈62Hz), and 0-8k is divided into {0-1 * 62Hz, 1-2 * 62Hz, 2-3 * 62Hz... , 128-129 * 62Hz} = {0-62Hz, 62-124Hz, 124-186Hz..., 7938-8000Hz}, each segment corresponding to a gain value.

- Note
  None.

- Related data types and interfaces
  MI_AO_VqeConfig_t

3.27. MI_AI_AencConfig_t

- Description
  Define the audio encoding function configuration information structure.

- Definition

```
typedef struct MI_AI_AencConfig_s
{
    MI_AUDIO_AencType_e    eAencType;
    union
    {
        MI_AUDIO_AencG711Config_t    stAencG711Cfg;
        MI_AUDIO_AnecG726Config_t    stAencG726Cfg;
    };
}MI_AI_AencConfig_t;
```

- Member

Member Name	Description
eAencType	Audio encoding type.
stAencG711Cfg	G711 code related configuration information.
stAencG726Cfg	G726 code related configuration information.

- Note

None.

- Related data types and interfaces

MI_AI_SetAencAttr

3.28. MI_AUDIO_AencG711Config_t

- Description

Define the audio encoding function configuration information structure.

- Definition

```
typedef struct MI_AUDIO_AencG711Config_s
{
    MI_AUDIO_SampleRate_e eSamplerate;
    MI_AUDIO_SoundMode_e eSoundmode;
}MI_AUDIO_AencG711Config_t;
```

- Member

Member Name	Description
eSamplerate	Audio sample rate
eSoundmode	Audio channel mode.

- Note

None.

- Related data types and interfaces

MI_AI_SetAencAttr

3.29. MI_AUDIO_AencG726Config_t

- Description

  Define the audio encoding function configuration information structure.

- Definition

```
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;
```

- Member

Member Name	Description
eSamplerate	Audio sample rate.
eSoundmode	Audio channel mode.
eG726Mode	G726 working mode.

- Note

  None.

- Related data types and interfaces

  MI_AI_SetAencAttr

3.30. MI_AUDIO_AlgorithmMode_e

- Description

  Define the operating mode of the audio algorithm.

- Definition

```
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;
```

- Member

Member Name	Description
E_MI_AUDIO_ALGORITHM_MODE_DEFAULT	Default mode. Note: When using this mode, the default parameters of the algorithm will be used
E_MI_AUDIO_ALGORITHM_MODE_USER	User mode. Note: When using this mode, the user needs to reset all parameters.

Member Name	Description
E_MI_AUDIO_ALGORITHM_MODE_MUSIC	Music mode. Note: Only ANR has this mode. When this mode is used, AGC does not perform speech enhancement processing.

- Note

  In the case where both ANR and AGC are enabled, if ANR is set to user mode, AGC will process the audio data in frequency domain, and then evaluate the speech signal and make corresponding boost and cut. When ANR is set to default/music mode, AGC will process the audio data in time domain and boost and cut the data in full frequency band.

- Related data types and interfaces

  MI_AUDIO_HpfConfig_t, MI_AUDIO_AnrConfig_t, MI_AUDIO_AgcConfig_t, MI_AUDIO_EqConfig_t

3.31. MI_AI_AedConfig_t

- Description

  Define the sound event detection function configuration information structure.

- Definition

```
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;
```

- Member

Member Name	Description
bEnableNr	Whether to enable noise reduction function
eSensitivity	Sensitivity of sound event detection function
s32OperatingPoint	Operating Point Range [-10,10]; step size 1 The default value is 0. Note：Increasing the operating point will reduce the false positive rate, and reducing the operating point will reduce the leak rate.
s32VadThresholdDb	Vad threshold Db Range [-80,0]; step size 1 The default value is -40.
s32LsdThresholdDb	Lsd threshold Db

Member Name	Description
	Range [-80,0]; step size 1 The default value is -15.

- Note

  None.

- Related data types and interfaces

  MI_AI_SetAedAttr, MI_AI_GetAedAttr

3.32. MI_AUDIO_AedSensitivity_e

- Description

  Define the sensitivity of sound event detection

- Definition

```
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;
```

- Member

Member Name	Description
E_MI_AUDIO_AED_SEN_LOW	Low sensitivity
E_MI_AUDIO_AED_SEN_MID	Medium sensitivity
E_MI_AUDIO_AED_SEN_HIGH	High sensitivity

- Note

  None.

- Related data types and interfaces

  MI_AI_AedConfig_t

3.33. MI_AI_AedResult_t

- Description

  Define the sound event detection result structure.

- Definition

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

- Member

Member Name	Description
bAcousticEventDetected	Whether an acoustic event is detected
bLoudSoundDetected	Whether a loud sound is detected

- Note

None.

- Related data types and interfaces

MI_AI_AedConfig_t

3.34. MI_AI_ChnGainConfig_t

- Description

Define the AI channel gain setting structure.

- Definition

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

- Member

Member Name	Description
bEnableGainSet	Enable gain setting
s16FrontGain	Front Gain
s16RearGain	Rear Gain

- Note

The following table describes the definition of s16frontgain and s16reargain for each device.

AI Device ID	s16FrontGain	s16RearGain
Amic	Analog gain of Amic (0 - 21)	Digital gain of Amic (-60 - 30)
Dmic	Gain of Dmic	No effect
	Pretzel series (0 - 4)	
	Macaron series (-60 - 30)	
	TAIYAKI series (0 - 4)	
	TAKOYAKI series (0 - 4)	
	Pudding series (-60 - 30)	
	Ispahan series (-60 - 30)	
I2S RX	No effect	No effect
Line in	Analog gain of Line in (0 - 7)	Digital gain of Line in (-60 - 30)

- Related data types and interfaces

MI_AI_ChnParam_t

3.35. MI_AI_SslInitAttr_t

- Description

Define the Sound source localization initialized structure.

- Definition

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

- Member

Member Name	Description
u32MicDistance	Microphone Distance (cm)
bBfMode	Whether to enable beamforming mode Note: FALSE: Mechanical rotation mode TRUE: Beamforming mode

- Note

None.

- Related data types and interfaces

MI_AI_SetSslInitAttr, MI_AI_GetSslInitAttr

3.36. MI_AI_SslConfigAttr_t

- Description

  Define the Sound source localization configured structure.

- Definition

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

- Member

Member Name	Description
s32Temperature	The temperature of environment (unit Celsius) Celsius = (5/9) × (Fahrenheit−32)
s32NoiseGateDbfs	Noise gain threshold (dB) Step size 1. Note: If power of any frame is smaller than this value, the SSL will return 0 as the direction immediately. This value should be set to a negative value, because the unit is dbfs. The default setting is -80.
s32DirectionFrameNum	SSL direction frame number Note: It will determine the output one result every frame_num frames. It should be a multiple of 50. One frame of ssl has 128 sample points. time of output a result = s32DirectionFrameNum * 128 / samplerate.

- Note

  None.

- Related data types and interfaces

  MI_AI_SetSslConfigAttr, MI_AI_GetSslConfigAttr

3.37. MI_AI_BfInitAttr_t

- Description

  Define the Beamforming initialized structure.

- Definition

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

- Member

Member Name	Description
u32MicDistance	Microphone Distance (cm)
u32ChanCnt	Channel Count Note: must be 2

- Note

None.

- Related data types and interfaces

MI_AI_SetBfInitAttr, MI_AI_GetBfInitAttr

3.38. MI_AI_BfConfigAttr_t

- Description

Define the Beamforming configured structure.

- Definition

```
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;
```

- Member

Member Name	Description
s32Temperature	The temperature of environment (unit Celsius) Celsius = (5/9) × (Fahrenheit−32)
s32NoiseGateDbfs	If the power of signal is smaller than this setting, this frame will be taken as the noise part, and used to calculate some estimation. Please be careful with the use of this parameter. If set too high, it will cause speech distortion. If set too small, it will cause BF result to be not good enough. Recommended setting: -44.
s32NoiseSupressionMode	A bigger number will remove more noise (0~15). Please be careful with the use of this parameter.

Member Name	Description
	If set too high, it will cause speech distortion. If set too small, it will cause BF result to be not good enough. Recommended setting: 8.
s32NoiseEstimation	This setting could change the noise estimation method (0 or 1) for keyword spotting preprocessing. 0 for adaptation method. 1 for average method. Recommended setting: 0 For acoustic communication, 1 is recommended.
outputGain	Output signal gain control (range: 0 ~ 1). 0.7 is the normal case. Setting as 1 will add 4db. Recommended setting: 0.7 (nothing changed).

- Note

  None.

- Related data types and interfaces

  MI_AI_SetBfConfigAttr, MI_AI_GetBfConfigAttr

4. ERROR CODE

The AI API error codes are shown in the table below:

Table 1: AI API error code

Macro Definition	Description
MI_AI_ERR_INVALID_DEVID	Invalid audio input device number
MI_AI_ERR_INVALID_CHNID	Invalid audio input channel number
MI_AI_ERR_ILLEGAL_PARAM	Invalid audio input parameter setting
MI_AI_ERR_NOT_ENABLED	Audio input device or channel is not enabled
MI_AI_ERR_NULL_PTR	Input parameter empty indicator error
MI_AI_ERR_NOT_CONFIG	Audio input device properties are not set
MI_AI_ERR_NOT_SUPPORT	Operation is not supported
MI_AI_ERR_NOT_PERM	Operation not allowed
MI_AI_ERR_NOMEM	Failed to allocate memory
MI_AI_ERR_NOBUF	Insufficient audio input buffer
MI_AI_ERR_BUF_EMPTY	Audio input buffer is empty
MI_AI_ERR_BUF_FULL	Audio input buffer is full
MI_AI_ERR_SYS_NOTREADY	Audio input system is not initialized
MI_AI_ERR_BUSY	Audio input system is busy
MI_AI_ERR_VQE_ERR	Audio input VQE algorithm failed to process
MI_AI_ERR_AENC_ERR	Audio input encoding algorithm failed to process
MI_AI_ERR_AED_ERR	Audio input AED algorithm failed to process
MI_AI_ERR_SSL_ERR	Audio input SSL algorithm failed to process
MI_AI_ERR_BF_ERR	Audio input BF algorithm failed to process