MI AO API

Version 2.11

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

REVISION HISTORY

Revision No.	Description	Date
2.03	• Initial release	04/12/2018
2.04	• Updated for accuracy	02/25/2019
2.05	• Added description regarding audio algorithm	03/25/2019
2.06	• Added MI_AO_SetChnParam and MI_AO_GetChnParam	03/30/2019
2.07	• Updated audio supporting sample rate	06/17/2019
2.08	• Added src gain setting API	07/04/2019
2.09	• Fixed the wrong sample rate • Updated MI_AUDIO_Frame_t structure	07/31/2019
2.10	• Updated I2s mode and Mclk • Fixed the wrong description of AGC	10/26/2019
2.11	• Updated for clarity	01/02/2020

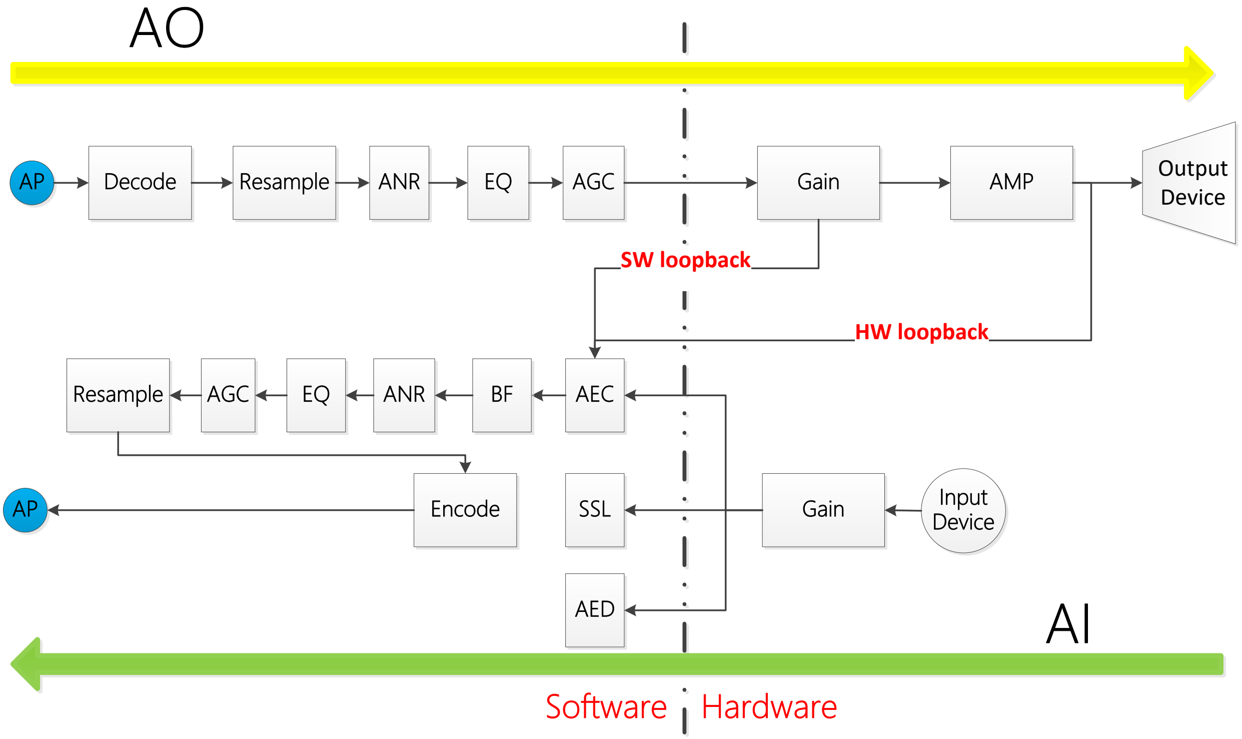
TABLE OF CONTENTS

1. SUMMARY

1.1. Module Description

Audio Output (AO) is mainly used to configure and enable Audio Output devices, send audio frame data, and perform acoustic algorithm processing. Acoustic algorithm processing mainly includes Sample Rate Conversion, Acoustic Noise Reduction, High-Pass Filtering, Equalizer, Automatic Gain Control and so on.

1.2. Flow Block Diagram



1.3. Keyword Description

- Device
  Different from Device concepts of other modules, AO Device refers to different external input devices such as Line out, I2S TX, and HDMI.
- Channel
  AO Channel refers to the number of physical channels.
- 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.
- HPF
  HPF refers to High-Pass Filtering.

2. API REFERENCE

2.1. Overview

Audio output (AO) is mainly used to enable audio output devices, send audio frames to output channels and other functions.

2.2. API List

API name	**Features**
MI_AO_SetPubAttr	Set AO device properties
MI_AO_GetPubAttr	Get AO device properties
MI_AO_Enable	Enable AO device
MI_AO_Disable	Disable AO device
MI_AO_EnableChn	Enable AO channel
MI_AO_DisableChn	Disable AO channel
MI_AO_SendFrame	Send audio frame
MI_AO_EnableReSmp	Enable AO resampling
MI_AO_DisableReSmp	Disable AO resampling.
MI_AO_PauseChn	Pause AO channel
MI_AO_ResumeChn	Restore AO channel
MI_AO_ClearChnBuf	Clear the current audio data buffer in the AO channel
MI_AO_QueryChnStat	Query the current audio data cache status in the AO channel
MI_AO_SetVolume	Set the AO device volume
MI_AO_GetVolume	Get the volume of the AO device
MI_AO_SetMute	Set the AO device mute status
MI_AO_GetMute	Get the OM device mute status
MI_AO_GetFd	Obtain the device file handle corresponding to the audio output channel number
MI_AO_ClrPubAttr	Clear AO device properties
MI_AO_SetVqeAttr	Set AO's sound quality enhancement related properties
MI_AO_GetVqeAttr	Get AO's sound quality enhancement related properties
MI_AO_EnableVqe	Enable AO's sound quality enhancements
MI_AO_DisableVqe	Disable AO's sound quality enhancements
MI_AO_SetAdecAttr	Set AO decoding function related properties

API name	Features
MI_AO_GetAdecAttr	Get the AO decoding function related properties
MI_AO_EnableAdec	Enable AO decoding.
MI_AO_DisableAdec	Disable AO decoding.
MI_AO_SetChnParam	Set the parameter of AO channel
MI_AO_GetChnParam	Get the parameter of AO channel
MI_AO_SetSrcGain	Set AO device as Acoustic Echo Cancellation's gain

2.2.1 MI_AO_SetPubAttr

- Features

Set the AO device properties.

- Syntax

MI_S32 MI_AO_SetPubAttr(MI_AUDIO_DEV AoDevId, MI_AUDIO_Attr_t *pstAttr);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
pstAttr	AO device property pointer.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note

Audio output device attributes include sampling rate, sampling accuracy, output operation mode, number of sampling points per frame, number of channels and configuration parameters of I2S. If the codec needs to be docked, these attributes should be consistent with the requirements of the codec to be docked.

Sampling rate

The sampling rate refers to the number of sampling points in one second. The higher the sampling rate is, the smaller the distortion is and the more data is processed. Generally speaking, 8K sampling rate is used for voice and 32K or above for audio. Currently, only 8/ 11.025/12/16/22.05/24/32/44.1/48kHz sampling rate is supported. If you need to dock codec, please confirm whether the docked audio codec supports the sampling rate to be set when setting.

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.

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.

Number of channels

1 for mono and 2 for stereo.

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 initialize the AO device, send a frame of audio data, and uninitialize the AO device.

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

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

MI_SYS_Init();

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

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

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

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

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

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

```

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

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

MI_SYS_Exit();
```

2.2.2 MI_AO_GetPubAttr

➢ Features

Get the AO device properties.

➢ Syntax

MI_S32 MI_AO_GetPubAttr(MI_AUDIO_DEV AoDevId, MI_AUDIO_Attr_t *pstAttr);

➢ Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
pstAttr	AO device property pointer.	Input

➢ Return Value

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

➢ Requirement

- Header files: mi_ao.h
- Library file: libmi_ao.a/libmi_ao.so

➢ Note

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

➢ Example

Refer to the example section of MI_AO_SetPubAttr.

2.2.3 MI_AO_Enable

- Features

  Enable AO devices.

- Syntax

  MI_S32 MI_AO_Enable(MI_AUDIO_DEV AoDevId);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

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

- Example

  Refer to the example section of MI_AO_SetPubAttr.

2.2.4 MI_AO_Disable

- Features

  Disable AO devices.

- Syntax

  MI_S32 MI_AO_Disable(MI_AUDIO_DEV AoDevId);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input

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

- Requirement
    - Header files: mi_ao.h
    - Library file: libmi_ao.a/libmi_ao.so

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

- Example

  Refer to the example section of MI_AO_SetPubAttr.

2.2.5 MI_AO_EnableChn

- Features

  Enable AO channel

- Syntax

  MI_S32 MI_AO_EnableChn(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio output channel number. Only channel 0 is supported.	Input

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

- Requirement
    - Header files: mi_ao.h
    - Library file: libmi_ao.a/libmi_ao.so

- Note

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

- Example

  Refer to the example section of MI_AO_SetPubAttr.

2.2.6 MI_AO_DisableChn

- Features

AI channel disabled

- Syntax

MI_S32 MI_AO_DisableChn(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio output channel number. Only channel 0 is supported.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note

Before disabling the AO channel, you need to disable the enabled Audio Algorithm on that channel.

- Example

Refer to the example section of MI_AO_SetPubAttr.

2.2.7 MI_AO_SendFrame

- Features

Send an AO audio frame.

- Syntax

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

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio output channel number. Only channel 0 is supported.	Input
pstData	Audio frame structure pointer.	Input
s32MilliSec	Timeout for getting data -1 indicates blocking mode, waiting for no data;	Input

parameter name	description	Input/Output
	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.	

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note
  - 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 error return
  - When calling this interface to send audio frames to the AO output, you must first enable the corresponding AO channel.

- Example

Refer to the example section of MI_AO_SetPubAttr.

2.2.8 MI_AO_EnableReSmp

- Features

Enable AO resampling

- Syntax

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

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio input channel number. Only channel 0 is supported.	Input
eInSampleRate	The input sample rate for audio resampling.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so libSRC_LINUX.so

- Note
  - After AO channels is enabled, call this interface to enable resampling function before binding the AO channels.

- Example

  The following code shows how to turn resampling on and off. Input data is 16K, and resample is 8K for playback.

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

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

MI_SYS_Init();

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

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

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

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

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

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

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

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

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

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

MI_SYS_Exit();

```

2.2.9 MI_AO_DisableReSmp

- Features

  Disable AO resampling

- Syntax

  MI_S32 MI_AO_DisableReSmp(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio input channel number. Only channel 0 is supported.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so libSRC_LINUX.so

- Note

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

- Example

Refer to the MI_AO_EnableReSmp example section.

2.2.10 MI_AO_PauseChn

- Features

Pause AO channel

- Syntax

```
MI_S32 MI_AO_PauseChn(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);
```

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio input channel number. Only channel 0 is supported.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note

When the AO channel is disabled, it is not allowed to call this interface to suspend the AO channel.

- Example

The following code shows how to pause and resume AO playback.

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

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

MI_SYS_Init();

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

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

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

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

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

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

ret = MI_AO_PauseChn(AoDevId, AoChn);
```

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

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

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

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

MI_SYS_Exit();
```

2.2.11 MI_AO_ResumeChn

- Features

Restore the AO channel.

- Syntax

MI_S32 MI_AO_ResumeChn (MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio input channel number. Only channel 0 is supported.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note
  - After the AO channel is paused, it can be restored by calling this interface.
  - When the AO channel is in the suspended state or enabled state, calling this interface returns success; otherwise, the call will return an error.

- Example

  Refer to the MI_AO_PauseChn example section.

2.2.12 MI_AO_ClearChnBuf

- Features

  Clear the current audio data buffer in the AO channel.

- Syntax

  MI_S32 MI_AO_ClearChnBuf (MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio input channel number. Only channel 0 is supported.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note

  This interface is called after the AO channel is successfully enabled.

- Example

  The following code shows how to get the AO channel buffer status and clear the buffer.

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

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

MI_SYS_Init();

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

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

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

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

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

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

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

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

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

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

MI_SYS_Exit();
```

2.2.13 MI_AO_QueryChnStat

- Features

Query the current audio data cache status in the AO channel.

- Syntax

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

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio output channel number. Only channel 0 is supported.	Input
pstStatus	Cache state structure pointer.	Input

- Return Value

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

- Requirement

  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note

This interface is called after the AO channel is successfully enabled.

- Example

Refer to the MI_AO_ClearChnBuf example section.

2.2.14 MI_AO_SetVolume

- Features

  Set the volume of the AO device.

- Syntax

  MI_S32 MI_AO_SetVolume(MI_AUDIO_DEV AoDevId, MI_S32 s32VolumeDb);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
s32VolumeDb	Audio device volume size in dB (-60 - 30).	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note

  This interface is called after the AO device is successfully enabled.

- Example

  The following code shows how to set and get the volume parameters of AO.

```
MI_S32 ret;
MI_AUDIO_Attr_t stAttr;
MI_AUDIO_Attr_t stGetAttr;
MI_AUDIO_DEV AoDevId = 0;
MI_AO_CHN AoChn = 0;
MI_U8 u8Buf[1024];
MI_AUDIO_Frame_t stAoSendFrame;
MI_S32 s32VolumeDb = 0;

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

MI_SYS_Init();

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

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

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

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

/* set ao volume */
ret = MI_S32 MI_AO_SetVolume(AoDevId, s32VolumeDb);
if (MI_SUCCESS != ret)
{
    printf("set volume ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}

/* get ao volume */
ret = MI_S32 MI_AO_GetVolume(AoDevId, &s32VolumeDb);
if (MI_SUCCESS != ret)
{
    printf("get volume ao dev %d err:0x%x\n", AoDevId, ret);
    return ret;
}



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

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


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

```

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

MI_SYS_Exit();
```

2.2.15 MI_AO_GetVolume

- Features

  Get the volume of the AO device.

- Syntax

  MI_S32 MI_AO_GetVolume(MI_AUDIO_DEV AoDevId, MI_S32 *ps32VolumeDb);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
ps32VolumeDb	Audio device volume size pointer	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note

  This interface is called after the AO device is successfully enabled.

- Example

  Refer to the example section of MI_AO_SetVolume.

2.2.16 MI_AO_SetMute

- Features

  Set the AO device mute status.

- Syntax

  MI_S32 MI_AO_SetMute(MI_AUDIO_DEV AoDevId, MI_BOOL bEnable);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
bEnable	Whether the audio device is muted. TRUE : Enable mute function; FALSE : Turn off the mute function.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note
  - This interface is called after the AO device is successfully enabled.

- Example

The following code shows how to get and set the mute state.

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

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

MI_SYS_Init();

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

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

```

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

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

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

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

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

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


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

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

MI_SYS_Exit();
```

2.2.17 MI_AO_GetMute

- Features

  Get the AO device mute status.

- Syntax

  MI_S32 MI_AO_GetMute(MI_AUDIO_DEV AoDevId, MI_BOOL *pbEnable);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
pbEnable	Audio device mute status pointer.	Output

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note

  This interface is called after the AO device is successfully enabled.

- Example

  Refer to the MI_AO_SetMute example section

2.2.18 MI_AO_ClrPubAttr

- Features

  Clear AO device properties.

- Syntax

  MI_S32 MI_AO_ClrPubAttr(MI_AUDIO_DEV AoDevId);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note

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

- Example

  The following code shows how to set and clear the properties of the AO device.

```
MI_S32 ret;
MI_AUDIO_Attr_t stAttr;
MI_AUDIO_DEV AoDevId = 0;

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

MI_SYS_Init();

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

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

MI_SYS_Exit();
```

2.2.19 MI_AO_SetVqeAttr

- Features

  Set the AO's sound quality enhancement related properties.

- Syntax

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

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio output channel number. Only channel 0 is supported.	Input
pstVqeConfig	Audio output sound quality enhancement configuration structure pointer	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so libAPC_LINUX.so

- Note
  - Before you enable the sound quality enhancement function, you must set the sound quality enhancement related properties of the corresponding AO channel.
  - Before setting the AO's sound quality enhancement function related properties, you must first enable the corresponding AO channel.
  - When the same AO channel sound quality enhancement does Not supported dynamic set properties, re-set the sound quality AO channel enhancement relevant attributes need to turn off the sound quality function AO channel, then set the sound quality AO channel enhancements related attributes.
  - When you set the sound quality enhancement feature, you can select some of the features that are enabled by configuring the corresponding sound quality enhancement properties.
  - All algorithms of Vqe support 8/16K, only ANR/AGC/EQ support 48K.

- Example

The following code shows how to set up and enable the sound quality enhancement.

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

MI_AUDIO_HpfConfig_t stHpfCfg = {
    .eMode = E_MI_AUDIO_ALGORITHM_MODE_USER,
    .eHpfFreq = E_MI_AUDIO_HPF_FREQ_150,
};

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

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

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

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

MI_SYS_Init();

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

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

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

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

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

stAoSetVqeConfig.bAgcOpen = TRUE;
stAoSetVqeConfig.bAnrOpen = TRUE;
stAoSetVqeConfig.bEqOpen = TRUE;
stAoSetVqeConfig.bHpfOpen = TRUE;
stAoSetVqeConfig.s32FrameSample = 128;
stAoSetVqeConfig.s32WorkSampleRate = E_MI_AUDIO_SAMPLE_RATE_8000;
memcpy(&stAoSetVqeConfig.stAgcCfg, &stAgcCfg, sizeof(MI_AUDIO_AgcConfig_t));
memcpy(&stAoSetVqeConfig.stAnrCfg, &stAnrCfg, sizeof(MI_AUDIO_AnrConfig_t));
memcpy(&stAoSetVqeConfig.stEqCfg, &stEqCfg, sizeof(MI_AUDIO_EqConfig_t));
memcpy(&stAoSetVqeConfig.stHpfCfg, &stHpfCfg, sizeof(MI_AUDIO_HpfConfig_t));

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

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

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

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

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

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

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

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

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

MI_SYS_Exit();
```

2.2.20 MI_AO_GetVqeAttr

➢ Features

Get the AO's sound quality enhancement related properties.

➢ Syntax

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

➢ Parameters

parameter name	description	Input/Output
AiDevId	Audio device number	Input
AiChn	Audio input channel number. Only channel 0 is supported.	Input
pstVqeConfig	Audio output sound quality enhancement configuration structure pointer	Output

➢ Return Value

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

➢ Requirement

- Header files: mi_ao.h
- Library file: libmi_ao.a/libmi_ao.so libAPC_LINUX.so

➢ Note

Before obtaining the sound quality enhancement related attributes, you must first set the sound quality enhancement related attributes of the corresponding AO channel.

➢ Example

Refer to the MI_AO_SetVqeAttr example section.

2.2.21 MI_AO_EnableVqe

- Features

Enable AO's sound quality enhancements.

- Syntax

MI_S32 MI_AO_EnableVqe(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio output channel number. Only channel 0 is supported.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so libAPC_LINUX.so

- Note
  - The corresponding AO channel must be enabled before enabling the sound quality enhancement.
  - When the sound quality enhancement function of the same AO channel is enabled multiple times, the return is successful.
  - After disabling the AO channel, if you re-enable the AO channel and use the sound quality enhancement feature, you need to call this interface to re-enable sound quality enhancement.
  - Vqe supports 8K 16K

- Example

Refer to the MI_AO_SetVqeAttr example section.

2.2.22 MI_AO_DisableVqe

- Features

Disable AO’s sound quality enhancements.

- Syntax

MI_S32 MI_AO_DisableVqe(MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio output channel number. Only channel 0 is supported.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so libAPC_LINUX.so

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

- Example

  Refer to the MI_AO_SetVqeAttr example section.

2.2.23 MI_AO_SetAdecAttr

- Features

  Set the AO decoding function related properties.

- Syntax

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

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio output channel number. Only channel 0 is supported.	Input
pstAdecConfig	Audio decoding configuration structure pointer	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so libg711.a libg726.a

- Note

  Before setting the properties related to the decoding function of AO, the corresponding AO channel must be enabled first.

- Example

  The following code shows how to set and enable the decoding function.

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


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

MI_SYS_Init();

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

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

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

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

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

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

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

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

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

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

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

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

/* disable ao device */
ret = MI_AO_Disable(AoDevId);
```

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

MI_SYS_Exit();
```

2.2.24 MI_AO_GetAdecAttr

➢ Features

Get the AO decoding function related properties.

➢ Syntax

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

➢ Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio output channel number. Only channel 0 is supported.	Input
pstAdecConfig	Audio decoding configuration structure pointer	Output

➢ Return Value

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

➢ Requirement

- Header files: mi_ao.h
- Library file: libmi_ao.a/libmi_ao.so libg711.a libg726.a

➢ Note

No

➢ Example

Refer to the MI_AO_SetAdecAttr example section.

2.2.25 MI_AO_EnableAdec

➢ Features

Enable AO decoding.

➢ Syntax

MI_AO_EnableAdec (MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio output channel number. Only channel 0 is supported.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so libg711.a libg726.a

- Note

  Before enabling the decoding function of AO, the decoding parameters corresponding to AO channel must be set first.

- Example

  Refer to the MI_AO_SetAdecAttr example section.

2.2.26 MI_AO_DisableAdec

- Features

  Disable AO decoding.

- Syntax

  MI_AO_DisableAdec (MI_AUDIO_DEV AoDevId, MI_AO_CHN AoChn);

- Parameters

parameter name	description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio output channel number. Only channel 0 is supported.	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so libg711.a libg726.a

- Note

  No

- Example

Refer to the MI_AO_SetAdecAttr example section.

2.2.27 MI_AO_SetChnParam

- Features

Set AO channel parameters

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio input channel number. Value range: [0, MI_AUDIO_MAX_CHN_NUM).	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_ao.a/libmi_ao.so

- Note

None.

- Example

The following code shows how to set and get channel parameters.

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


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

```

MI_SYS_Init();

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

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

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

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

memset(&stChnParam, 0x0, sizeof(stChnParam));
stChnParam.stChnGain.bEnableGainSet = TRUE;
stChnParam.stChnGain.s16Gain = 0;

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

memset(&stChnParam, 0x0, sizeof(stChnParam));

ret = MI_AO_GetChnParam(AoDevId, AoChn, &stChnParam);
if (MI_SUCCESS != ret)
{
    printf("get chn param ao dev %d chn %d err:0x%x\n", AoDevId, AoChn, ret);
    return ret;
}

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

```

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

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

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

MI_SYS_Exit();
```

2.2.28 MI_AO_GetChnParam

- Features

  Get AO channel parameters

- Syntax

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

- Parameters

Parameter Name	Description	Input/Output
AoDevId	Audio device number	Input
AoChn	Audio input channel number. Value range: [0, MI_AUDIO_MAX_CHN_NUM).	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_ao.a/libmi_ao.so

- Note

  None.

- Example

Refer to the MI_AO_SetChnParam example section.

2.2.29 MI_AO_SetSrcGain

- Features

Set AO device as Acoustic Echo Cancellation's gain.

- Syntax

MI_S32 MI_AO_SetSrcGain(MI_AUDIO_DEV AoDevId, MI_S32 s32VolumeDb);

- Parameters

Parameter Name	Description	Input/Output
AoDevId	Audio device number	Input
s32VolumeDb	Audio device volume size in dB (-60 - 30)	Input

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

- Requirement
  - Header files: mi_ao.h
  - Library file: libmi_ao.a/libmi_ao.so

- Note

This interface is called after the AO device is successfully enabled.

- Example

The following code shows how to set SRC gain.

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

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

MI_SYS_Init();

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

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

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

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

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

MI_SYS_Exit();
```

3. AO DATA TYPE

The AO 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_AO_CHN	Define the audio output 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 and output device attribute structures
MI_AO_ChnState_t	Data buffer status structure of the audio output channel
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_AO_VqeConfig_t	Defining audio output 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_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_AO_AdecConfig_t	Define the decoding function configuration information structure.
MI_AO_ChnParam_t	Define channel parameter structure
MI_AO_ChnGainConfig_t	Define channel gain structure

3.1. MI_AUDIO_DEV

- Description

  Define the audio Input/Output device number.

- Definition

  Typedef MI_S32 MI_AUDIO_DEV

- Precautions

  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
Line out	Support 2 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	Support 2 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
I2S TX	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 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.
HDMI	Not supported	Not supported	Supported	Not supported	Not supported	Not supported
HDMI + Line out	Not supported	Not supported	Supported	Not supported	Not supported	Not supported

- Related data types and interfaces

  No

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

- Precautions

  No

- Related data types and interfaces

  No

3.3. MI_AO_CHN

- Description

  Define the audio output channel.

- Definition

  typedef MI_S32 MI_AO_CHN

- Precautions

  No

- Related data types and interfaces

  No

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 sample rate
E_MI_AUDIO_SAMPLE_RATE_48000	48kHz sampling rate
E_MI_AUDIO_SAMPLE_RATE_96000	96kHz sampling rate

- Precautions

The enumeration value here does not start at 0, but is the same as the actual sample rate value.

- 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 24bit width

- Precautions

Currently the software only supports 16bit bit width.

- Related data types and interfaces

No

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

- Precautions

Whether the master mode or 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 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,
     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	For audio input only

- Precautions

For the two-channel mode, only the left channel (that is, the channel whose number is less than half the channel number u32ChnCnt in the device attribute) should be performed.

For operation, the SDK will automatically perform corresponding operations on the right channel.

- Related data types and interfaces

MI_AUDIO_Attr_t.

3.8. 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.

- Precautions

No

- Related data types and interfaces

  MI_AO_VqeConfig_t

3.9. MI_AUDIO_AdecType_e

- Description

  Define the audio decoding type.

- Definition

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

- Member

Member name	description
E_MI_AUDIO_ADEC_TYPE_G711A	G711A decoding.
E_MI_AUDIO_ADEC_TYPE_G711U	G711U decoding.
E_MI_AUDIO_ADEC_TYPE_G726	G726 decoding.

- Precautions

  No

- Related data types and interfaces

  MI_AO_AdecConfig

3.10. MI_AUDIO_G726Mode_e

- Description

  DefinitionG726 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.

- Precautions

  No

- Related data types and interfaces

  MI_AO_AdecConfig_t

3.11. 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

- Precautions

  No

- Related data types and interfaces

  MI_AUDIO_I2sConfig_t

3.12. 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

- Precautions

None.

- Related data types and interfaces

MI_AUDIO_I2sConfig_t

3.13. 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	AO synchronous AI clock Static attribute.

- Precautions

  No

- Related data types and interfaces

  MI_AUDIO_Attr_t

3.14. 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.

Member name	description
u32FrmNum	The number of cached frames. Reserved, not used.
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, not used
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 2channels)
MI_AUDIO_I2sConfig_t stI2sConfig;	Set I2S work properties

- Precautions

  No

- Related data types and interfaces

  MI_AO_SetPubAttr

3.15. MI_AO_ChnState_t

- Description

  The data buffer status structure of the audio output channel.

- Definition

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

- Member

Member name	description
u32ChnTotalNum	The total number of cache bytes for the output channel.
u32ChnFreeNum	The number of free cache bytes available.
u32ChnBusyNum	The number of cache bytes that are occupied.

- Precautions

  No

- Related data types and interfaces

  No

3.16. 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[2];
    MI_U64 u64TimeStamp;/*audio frame timestamp*/
    MI_U32 u32Seq; /*audio frame seq*/
    MI_U32 u32Len; /*data lenth per channel in frame*/
    MI_U32 au32PoolId[2];
}MI_AUDIO_Frame_t;
```

- Member

Member name	description
eBitwidth	Audio sampling accuracy
eSoundmode	Audio channel mode.
pVirAddr[2]	Audio frame data virtual address.
u64TimeStamp	Audio frame timestamp, in µs
u32Seq	Audio frame number.
u32Len	Audio frame length, in bytes.
u32PoolId[2]	Audio frame buffer pool ID.

- Precautions
  - u32Len (audio frame length) refers to the data length of a frame.
  - Mono data is directly stored, the number of sampling points is u32PtNumPerFrm, the length is u32Len. Stereo data is interleaved by left and right channels, with the length of u32Len.

- Related data types and interfaces

No

3.17. MI_AUDIO_AecFrame_t

- Description

Define an 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.

- Precautions

  No

- Related data types and interfaces

  No

3.18. 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.

- Precautions

  No

- Related data types and interfaces

  MI_AI_SaveFile

3.19. MI_AO_VqeConfig_t

- Description

  Define the audio output sound quality enhancement configuration information structure.

- Definition

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

- Member

Member name	description
bHpfOpen	Whether the high-pass filtering function is enabled or not.
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
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.
stAnrCfg	Configuration information related to the voice noise reduction function.
stAgcCfg	Automatic gain control related configuration information.
stEqCfg	Equalizer related configuration information.

- Precautions

  No

- Related data types and interfaces

  No

3.20. 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.

- Precautions

No

- Related data types and interfaces

MI_AO_VqeConfig_t

3.21. 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

3.22. 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	Degree of smoothing 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.

- Precautions

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_AO_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.

- Precautions

No

- 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;
```

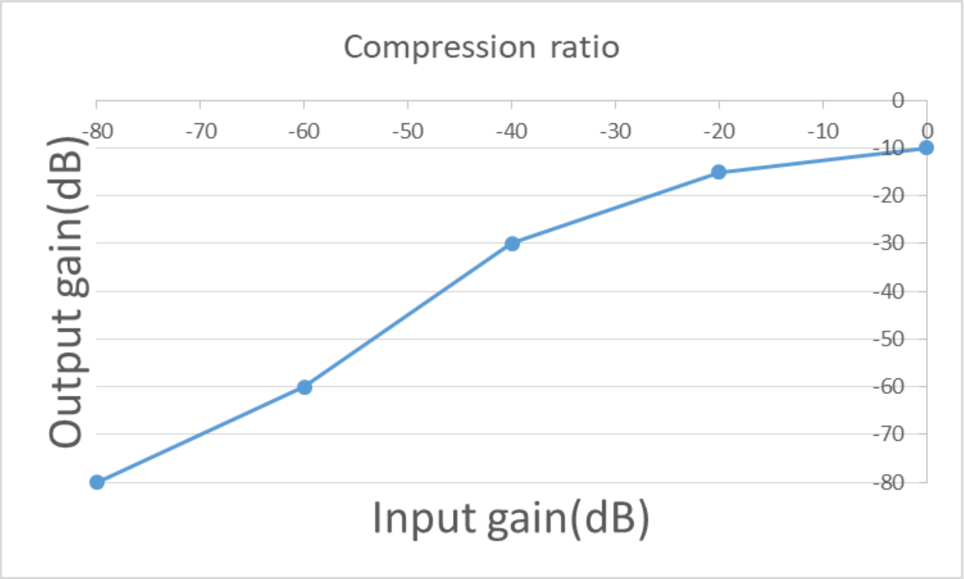
- Member

Member name	description
eMode	Audio algorithm operating mode.
stAgcGainInfo	Maximum, minimum, and initial values of the Definition AGC gain
u32DropGainMax	Maximum gain reduction to prevent 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
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.

➢ Precautions

- 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_AO_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.

- Precautions

  No

- 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. 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.

- Precautions

  No

- Related data types and interfaces

  MI_AO_VqeConfig_t

3.27. MI_AO_AdecConfig_t

- Description

  Define the decoding function configuration information structure.

- Definition

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

- Member

Member name	description
eAdecType	Audio decoding type.
stAdecG711Cfg	G711 decodes related configuration information.
stAdecG726Cfg	G726 decodes related configuration information.

- Precautions

No

- Related data types and interfaces

MI_AO_SetAdecAttr

3.28. MI_AUDIO_AdecG711Config_t

- Description

Define the G711 decoding function configuration information structure.

- Definition

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

- Member

Member name	description
eSamplerate	Audio sample rate.
eSoundmode	Audio channel mode.

- Precautions

No

- Related data types and interfaces

MI_AO_SetAdecAttr

3.29. MI_AUDIO_AdecG726Config_s

- Description

  Define the G711 decoding function configuration information structure.

- Definition

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

- Member

Member name	description
eSamplerate	Audio sample rate.
eSoundmode	Audio channel mode.
eG726Mode	G726 working mode.

- Precautions

  No

- Related data types and interfaces

  MI_AO_SetAdecAttr

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.
E_MI_AUDIO_ALGORITHM_MODE_MUSIC	Music mode.

Member Name	Description
	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_AO_ChnParam_t

- Description

  Define the channel parameter structure.

- Definition

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

- Member

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

- Note

  None.

- Related data types and interfaces

  MI_AO_SetChnParam
  MI_AO_GetChnParam

3.32. MI_AO_ChnGainConfig_t

- Description

  Define the AO channel gain setting structure.

- Definition

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

- Member

Member Name	Description
bEnableGainSet	Whether enable gain setting
s16Gain	Gain (-60 – 30 dB)

- Note

None.

- Related data types and interfaces

[MI_AO_ChnParam_t]

4. ERROR CODE

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

Table 1: AO API error codes

Macro Definition	**Description**
MI_AO_ERR_INVALID_DEVID	Invalid audio output device number
MI_AO_ERR_INVALID_CHNID	Invalid audio output channel number
MI_AO_ERR_ILLEGAL_PARAM	Invalid audio output parameter setting
MI_AO_ERR_NOT_ENABLED	Audio output device or channel is not enabled
MI_AO_ERR_NULL_PTR	Input parameter empty indicator error
MI_AO_ERR_NOT_CONFIG	Audio output device properties are not set
MI_AO_ERR_NOT_SUPPORT	Operation is not supported
MI_AO_ERR_NOT_PERM	Operation not allowed
MI_AO_ERR_NOMEM	Fail to allocate memory
MI_AO_ERR_NOBUF	Insufficient audio output buffer
MI_AO_ERR_BUF_EMPTY	Audio output buffer is empty
MI_AO_ERR_BUF_FULL	Audio output buffer is full
MI_AO_ERR_SYS_NOTREADY	Audio output system is not initialized
MI_AO_ERR_BUSY	Audio output system is busy
MI_AO_ERR_VQE_ERR	Audio output VQE algorithm failed to process
MI_AO_ERR_ADEC_ERR	Audio output decoding algorithm processing failed