- Table of Contents -

[1. Overview 3](#_Toc8203723)

[1.1 Specifications Outline 3](#_Toc8203724)

[1.2 Configuration 5](#_Toc8203725)

[2. Software Specifications 9](#_Toc8203726)

[2.1 API specifications 9](#_Toc8203727)

[2.2 Command 10](#_Toc8203728)

[2.2.1 Command list 11](#_Toc8203729)

[2.2.2 Detail of Command Specifications 21](#_Toc8203730)

[2.3 Structures 94](#_Toc8203731)

[2.3.1 XARelrdr type structure 95](#_Toc8203732)

[2.3.2 XARelcap type structure 97](#_Toc8203733)

[2.4 Memory Specifications 98](#_Toc8203734)

[2.4.1 Persistent Area 98](#_Toc8203735)

[2.4.2 Stack Area 99](#_Toc8203736)

[2.4.3 Heap Area 99](#_Toc8203737)

[2.4.4 Input Buffer 100](#_Toc8203738)

[2.4.5 Output Buffer 100](#_Toc8203739)

[2.5 Error Processing 102](#_Toc8203740)

[2.5.1 Error codes 102](#_Toc8203741)

[3. Processing Flow 109](#_Toc8203742)

[4. Appendix 111](#_Toc8203743)

[4.1 CTU (Channel transfer unit) 111](#_Toc8203744)

[4.2 MIX (Mixing) 112](#_Toc8203745)

[4.3 16-bit monaural 114](#_Toc8203746)

[4.4 Conversion functions 114](#_Toc8203747)

[4.5 State transformation 115](#_Toc8203748)

- List of Figures -

[Figure 1‑1 Example of the ADSP System Configuration for renderer function 5](#_Toc8203710)

[Figure 1‑2 Example of the ADSP System Configuration uses CTU/MIX for renderer function 6](#_Toc8203711)

[Figure 1‑3 Example of the ADSP System Configuration for capture function 7](#_Toc8203712)

[Figure 2‑1 Command sequence overview 10](file:///D:\02_RSS2_Middleware1\03_Working\02_repositories\s492\trunk\documents\40_FD\40_RDR\en\04_Body_for_Android.docx#_Toc8203713)

[Figure 2‑2 PCM 16-bit Data Access (Little Endian Mode) 101](#_Toc8203714)

[Figure 2‑3 PCM 24-bit Data Access (Little Endian Mode) 101](#_Toc8203715)

[Figure 2‑4 Output Formats 101](#_Toc8203716)

[Figure 3‑1 Example of the Application Processing Flow 110](#_Toc8203717)

[Figure 4‑1 Block diagram of CTU 111](#_Toc8203718)

[Figure 4‑2 Example of using CTU/MIX for 3 streams to SSI0 and another stream to SSI3 112](#_Toc8203719)

[Figure 4‑3 Example of using CTU/MIX to play 8 streams to SSI0 and SSI3. 113](#_Toc8203720)

[Figure 4‑4 Data format of 16-bit monaural in SW and HW 114](#_Toc8203721)

[Figure 4‑5 State transformation in the plugin 115](#_Toc8203722)

- List of Tables -

[Table 1‑1 Basic Specification 3](#_Toc8203685)

[Table 1‑2 Supported Specifications for capture function 3](#_Toc8203686)

[Table 1‑3 Supported Specifications for renderer function 3](#_Toc8203687)

[Table 1‑4 Memory Size Requirements 4](#_Toc8203688)

[Table 1‑5 Version Information 4](#_Toc8203689)

[Table 2‑1 API Functions of renderer 9](#_Toc8203690)

[Table 2‑2 API Functions of capture 9](#_Toc8203691)

[Table 2‑3 List of supported none supported command, subcommand 11](#_Toc8203692)

[Table 2‑4 List of Initialization Commands 13](#_Toc8203693)

[Table 2‑5 List of Set Commands for renderer 14](#_Toc8203694)

[Table 2‑6 List of Set Commands for capture 15](#_Toc8203695)

[Table 2‑7 List of Memory allocation Commands 16](#_Toc8203696)

[Table 2‑8 List of initialize commands 17](#_Toc8203697)

[Table 2‑9 List of Get commands for renderer 18](#_Toc8203698)

[Table 2‑10 List of Get commands for capture 19](#_Toc8203699)

[Table 2‑11 List of execution commands 20](#_Toc8203700)

[Table 2‑12 Structures 94](#_Toc8203701)

[Table 2‑13 XARelrdr type structure information 96](#_Toc8203702)

[Table 2‑14 XARelcap type structure information 97](#_Toc8203703)

[Table 2‑15 Persistent Area Description 98](#_Toc8203704)

[Table 2‑16 Stack Area Description 99](#_Toc8203705)

[Table 2‑17 Input Buffer Description 100](#_Toc8203706)

[Table 2‑18 Output Buffer Description 100](#_Toc8203707)

[Table 2‑19 Error Codes for Renderer 102](#_Toc8203708)

[Table 2‑20 Error Codes for Capture 106](#_Toc8203709)

# Overview

This section provides an overview of the Renderer plugin. It contains renderer and capture function.

## Specifications Outline

Renderer function plays the audio signal based on the parameter that was set.

Capture function capture/record the audio signal based on the parameter that was set.

Table 1‑1 Basic Specification

| Item | Description |
| --- | --- |
| DSP | Cadence Design Systems, Inc. HiFi2 |
| Compiler | Xtensa C and C++ Compiler (version 12.0.4) |
| Endian | Little Endian |

Table 1‑2 Supported Specifications for capture function

| Item | Description |
| --- | --- |
| Input data format | 16-bit/24-bit linear PCM (fixed point) |
| Output data format | 16-bit/24-bit linear PCM (fixed point) |
| Sampling frequency (Hz) supported | 48000 / 44100 / 32000 |
| Number of channels supported | Max 2 channels |
| Reentrant | Supported |
| Volume control | Support volume update during plugin execution |
| Other | - |
| Restrictions | - |

Table 1‑3 Supported Specifications for renderer function

| Item | Description |
| --- | --- |
| Input data format | 16-bit/24-bit linear PCM (fixed point) |
| Output data format | 16-bit/24-bit linear PCM (fixed point) |
| Sampling frequency (Hz) supported | 48000 / 44100 / 32000 |
| Input data channel | Monaural, stereo, 4 channels, 6 channels, 8 channels |
| Output data channel | Max 2 channels |
| Reentrant | Supported |
| Volume control | Support volume update during plugin execution. |
| Other | - |
| Restrictions | - |

[Note] In case of channel transfer from input monaural to output stereo, the sound will output to both two channels.

Table 1‑4 Memory Size Requirements

| Memory type | Location | Memory area name | | Size (in bytes) | |
| --- | --- | --- | --- | --- | --- |
| Instruction | ROM | Instruction area | | 78393 | |
| Data | Constant table area | |
| Other area(Depended on the compiler) | |
| RAM  Capture | Software work area | | 1756 | |
| Area breakdown | Persistent area | Size breakdown | 1692 |
| Scratch area | 0 |
| Built-in descriptor area | 64 |
| User work area | | 9576 | |
| Area breakdown | Input buffer | Size breakdown | 0 |
| Output buffer | 8192 |
| Structure | 1384 |
| Stack area | | 1168 | |
| Other area(Depended on the compiler) | | 0 | |
| RAM  Renderer | Software work area | | 1748 | |
| Area breakdown | Persistent area | Size breakdown | 1684 |
| Scratch area | 0 |
| Built-in descriptor area | 64 |
| User work area | | 9616 | |
| Area breakdown | Input buffer | Size breakdown | 8192 |
| Output buffer | 0 |
| Structure | 1424 |
| Stack area | | 1440 | |
| Other area(Depended on the compiler) | | 0 | |

[Note] Area whose location is shown as ROM in the location column can be included in RAM or ROM.

[Note] Area whose location is shown as RAM in the location column can be included in RAM only.

[Note] Built-in is a memory area to allocate descriptor memory, which need in the DMAC transfer type of plugin.

Table 1‑5 Version Information

| Item | Description |
| --- | --- |
| Library Version information | Version 1.0.4 |
| API Version information | Version 1.0.0 |

## Configuration

Figure 1‑1 shows an example of the ADSP system configuration which uses renderer function.

ADSP Reference Plugin

HiFi2 Codec Library

ADSP Framework

Stream

(Compressed Audio data)

HiFi2 Renderer

PCM data

PCM data

DAC

Speaker

SRC

PCM data

DVC

SCU

SSI

CTU

MIX

Figure 1‑1 Example of the ADSP System Configuration for renderer function

Figure 1‑2 show an example of using channel converter and mixer function to mix 4 streams from 4 Renderer plugins.

ADSP Reference Plugin

ADSP Reference Plugin

ADSP Reference Plugin

ADSP Reference Plugin

HiFi2 Renderer

DAC

Speaker

SRC

DVC

SCU

SSI

CTU

MIX

HiFi2 Renderer

HiFi2 Renderer

HiFi2 Renderer

SRC

CTU

SRC

CTU

SRC

CTU

HiFi2 Codec Library

ADSP Framework

Stream

(Compressed Audio data)

PCM data

PCM data

Figure 1‑2 Example of the ADSP System Configuration uses CTU/MIX for renderer function

Figure 1‑3 shows an example of the ADSP system configuration which uses capture function.

ADSP Reference Plugin

HiFi2 Codec Library

ADSP Framework

Stream

(Compressed Audio data)

HiFi2 Capture

PCM data

PCM data

ADC

Microphone

PCM data

DVC

SRC

SCU

SSI

Figure 1‑3 Example of the ADSP System Configuration for capture function

1. Stream (Compressed Audio data)

Compressed Audio data is a linear PCM data sample compressed according to the compression Audio specifications. For these specifications, depends on HiFi2 Codec Library.

1. HiFi2 Codec Library

It is Codec Library for HiFi2. It decodes the compression Audio in renderer case and encodes in capture case. The user should procure to suit the target system.

1. ADSP Framework

It controls ADSP Plugin. It is software provided separately as Framework.

1. HiFi2 Renderer (ADSP Reference Plugin)

It performs output handling of PCM data to other Audio device. It is this software set up as ADSP Reference Plugin.

1. HiFi2 Capture (ADSP Reference Plugin)

It performs input handling of PCM data from other Audio device. It is this software set up as ADSP Reference Plugin.

1. PCM data

16-bit/24-bit linear PCM data which is a processing by this software.

1. SCU

It performs sampling rate converters (SRC), channel transfer (CTU), mixing (MIX), and volume control (DVC).

1. SRC

It performs sampling rate conversion function.

1. CTU

It performs channel transfer unit function such as down-mixing and splitter functions.

1. MIX

It is used for mixing (adding) streams from two to four audio stream sources into a single stream. It also support the volume control (gain level) for each input stream.

1. DVC

It performs mute and volume control functions.

1. SSI

Send or receive audio data interfacing with a variety devices of offering I2C format.

1. ADC

The ADC converts an analog signal into 16-bit linear PCM data.

# Software Specifications

## API specifications

Because one interface function accesses the procedure that was appointed by a command in renderer plugin, it is used.

In renderer case

Table 2‑1 API Functions of renderer

|  |  |
| --- | --- |
| xa\_rel\_rdr | |
| Description | This API is the only access function to the renderer. |
| Syntax | XA\_ERRORCODE xa\_rel\_rdr(  xa\_codec\_handle\_t p\_xa\_module\_obj,  WORD32 i\_cmd,  WORD32 i\_idx,  pVOID pv\_value); |
| Parameters | p\_xa\_module\_obj : Pointer to opaque API structure.  i\_cmd : Command. (defined in the supplied header files as)  i\_idx : Command subtype or index. (defined in the supplied header files as)  pv\_value : Pointer to the variable used to pass in, or get out properties, from state structure. |
| Returns | Error Code based on the success or failure of API command (defined in the supplied header files as) |

In capture case

Table 2‑2 API Functions of capture

|  |  |
| --- | --- |
| xa\_rel\_cap | |
| Description | This API is the only access function to the capture. |
| Syntax | XA\_ERRORCODE xa\_rel\_cap(  xa\_codec\_handle\_t p\_xa\_module\_obj,  WORD32 i\_cmd,  WORD32 i\_idx,  pVOID pv\_value); |
| Parameters | p\_xa\_module\_obj : Pointer to opaque API structure.  i\_cmd : Command. (defined in the supplied header files as)  i\_idx : Command subtype or index. (defined in the supplied header files as)  pv\_value : Pointer to the variable used to pass in, or get out properties, from state structure. |
| Returns | Error Code based on the success or failure of API command (defined in the supplied header files as) |

## Command

Using API functions of the Table 2‑1 and Table 2‑2, it performs each processing by a combination of Command/Subcommand.

Start-up API

Parameters Setting

Memory Allocation

Initialization

Done?

Parameters Getting

Plugin Executing

Done?

Figure 2‑1 Command sequence overview

### Command list

Below table presents commands used in renderer and capture case.

Table 2‑3 List of supported none supported command, subcommand

|  |  |  |  |
| --- | --- | --- | --- |
| Command | Sub command | Ren | Cap |
| XA\_API\_CMD\_GET\_LIB\_ID\_STRINGS | XA\_CMD\_TYPE\_LIB\_VERSION | Y | Y |
| XA\_CMD\_TYPE\_API\_VERSION | Y | Y |
| XA\_API\_CMD\_GET\_API\_SIZE | - | Y | Y |
| XA\_API\_CMD\_INIT | XA\_CMD\_TYPE\_INIT\_API\_PRE\_CONFIG\_PARAMS | Y | Y |
| XA\_CMD\_TYPE\_INIT\_API\_POST\_CONFIG\_PARAMS | Y | Y |
| XA\_CMD\_TYPE\_INIT\_PROCESS | Y | Y |
| XA\_CMD\_TYPE\_INIT\_DONE\_QUERY | Y | Y |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | XA\_CAP\_CONFIG\_PARAM\_PCM\_WIDTH | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_CHANNELS | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_SAMPLE\_RATE | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_FRAME\_SIZE | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_INPUT1 | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL1 | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_INPUT2 | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL2 | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_VOLUME\_RATE | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_STATE | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_RING\_NUM | N | Y |
| XA\_RDR\_CONFIG\_PARAM\_PCM\_WIDTH | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_CHANNELS | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_SAMPLE\_RATE | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_FRAME\_SIZE | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT1 | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL1 | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT2 | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL2 | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_VOLUME\_RATE | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_CHANNELS | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_MIX\_CONTROL | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_STATE | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_RING\_NUM | Y | N |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | XA\_CAP\_CONFIG\_PARAM\_PCM\_WIDTH | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_CHANNELS | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_SAMPLE\_RATE | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_FRAME\_SIZE | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_INPUT1 | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL1 | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_INPUT2 | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL2 | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_VOLUME\_RATE | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_STATE | N | Y |
| XA\_CAP\_CONFIG\_PARAM\_RING\_NUM | N | Y |
| XA\_RDR\_CONFIG\_PARAM\_PCM\_WIDTH | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_CHANNELS | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_SAMPLE\_RATE | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_FRAME\_SIZE | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT1 | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL1 | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT2 | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL2 | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_VOLUME\_RATE | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_CHANNELS | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_MIX\_CONTROL | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_STATE | Y | N |
| XA\_RDR\_CONFIG\_PARAM\_RING\_NUM | Y | N |
| XA\_API\_CMD\_GET\_MEMTABS\_SIZE | - | Y | Y |
| XA\_API\_CMD\_SET\_MEMTABS\_PTR | - | Y | Y |
| XA\_API\_CMD\_GET\_N\_MEMTABS | - | Y | Y |
| XA\_API\_CMD\_GET\_MEM\_INFO\_SIZE | - | Y | Y |
| XA\_API\_CMD\_GET\_MEM\_INFO\_ALIGNMENT | - | Y | Y |
| XA\_API\_CMD\_GET\_MEM\_INFO\_TYPE | - | Y | Y |
| XA\_API\_CMD\_SET\_MEM\_PTR | - | Y | Y |
| XA\_API\_CMD\_SET\_INPUT\_BYTES | - | Y | N |
| XA\_API\_CMD\_INPUT\_OVER | - | N | Y |
| XA\_API\_CMD\_GET\_CURIDX\_INPUT\_BUF | - | Y | N |
| XA\_API\_CMD\_EXECUTE | XA\_CMD\_TYPE\_DO\_EXECUTE | N | Y |
| XA\_CMD\_TYPE\_DONE\_QUERY | Y | N |
| XA\_API\_CMD\_GET\_OUTPUT\_BYTES | - | N | Y |

Ren: Renderer Cap: Capture

Y: command or sub command is used by renderer or capture.

N: command or sub command is not used by renderer or capture. They are presented in detailed command part to refer. Actually, they are not implemented.

-: None sub command

#### Start-up API

Table 2‑4 List of Initialization Commands

| upper stage: Command / lower step : Subcommand | | Description |
| --- | --- | --- |
| 1 | XA\_API\_CMD\_GET\_LIB\_ID\_STRINGS | Get the version of the library |
| XA\_CMD\_TYPE\_LIB\_VERSION |
| 2 | XA\_API\_CMD\_GET\_LIB\_ID\_STRINGS | Get the version of the API |
| XA\_CMD\_TYPE\_API\_VERSION |
| 3 | XA\_API\_CMD\_GET\_API\_SIZE | Get the size of the API structure |
| Null |
| 4 | XA\_API\_CMD\_INIT | Set the default values of all the configuration parameters |
| XA\_CMD\_TYPE\_INIT\_API\_PRE\_CONFIG\_PARAMS |

#### Setting parameters

Table 2‑5 List of Set Commands for renderer

| upper stage : Command / lower step : Subcommand | | Description |
| --- | --- | --- |
| 1 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the input PCM sample bit width to 16 or 24 bits |
| XA\_RDR\_CONFIG\_PARAM\_PCM\_WIDTH |
| 2 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the input PCM channels (support monaural, stereo, 4 channels, 6 channels, 8 channels) |
| XA\_RDR\_CONFIG\_PARAM\_CHANNELS |
| 3 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the input PCM sampling frequency (supported 32000/44100/48000Hz) |
| XA\_RDR\_CONFIG\_PARAM\_SAMPLE\_RATE |
| 4 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the input/output frame size |
| XA\_RDR\_CONFIG\_PARAM\_FRAME\_SIZE |
| 5 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the output destination Audio 1st devicefor Renderer |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT1 |
| 6 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set ADMA channel number usage for Audio device 1st (supported Audio-DMAC, Audio-DMAC-pp) |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL1 |
| 7 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the output destination Audio 2nd device for Renderer |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT2 |
| 8 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set ADMA channel number usage for Audio device 2nd (supported Audio-DMAC, Audio-DMAC-pp) |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL2 |
| 9 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the output PCM sampling frequency (supported 32000/44100/48000Hz) |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE |
| 10 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the output PCM volume rate compare with input PCM (supported from 0 – 8 times) |
| XA\_RDR\_CONFIG\_PARAM\_VOLUME\_RATE |
| 11 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the output PCM channels (support maximum 2 channels) |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_CHANNELS |
| 12 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the MIX module usage for Renderer (value 1/0 to used/unused mix function) |
| XA\_RDR\_CONFIG\_PARAM\_MIX\_CONTROL |
| 13 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the operation state of Renderer plugin (RUN/PAUSE/IDLE/RESET) |
| XA\_RDR\_CONFIG\_PARAM\_STATE |
| 14 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the total number of ring buffers to use |
| XA\_RDR\_CONFIG\_PARAM\_RING\_NUM |

Table 2‑6 List of Set Commands for capture

| upper stage : Command / lower step : Subcommand | | Description |
| --- | --- | --- |
| 1 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the input PCM sample bit width to 16 or 24 bits |
| XA\_CAP\_CONFIG\_PARAM\_PCM\_WIDTH |
| 2 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the input PCM channels (Maximum is 2 channels) |
| XA\_CAP\_CONFIG\_PARAM\_CHANNELS |
| 3 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the input PCM sampling frequency (supported 32000/44100/48000Hz) |
| XA\_CAP\_CONFIG\_PARAM\_SAMPLE\_RATE |
| 4 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the input/output frame size |
| XA\_CAP\_CONFIG\_PARAM\_FRAME\_SIZE |
| 5 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the input source Audio 1st device for Capture |
| XA\_CAP\_CONFIG\_PARAM\_INPUT1 |
| 6 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set ADMA channel number usage for Audio device 1st (supported Audio-DMAC, Audio-DMAC-pp) |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL1 |
| 7 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the input source Audio 2nd device for Capture |
| XA\_CAP\_CONFIG\_PARAM\_INPUT2 |
| 8 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set ADMA channel number usage for Audio device 2nd (supported Audio-DMAC, Audio-DMAC-pp) |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL2 |
| 9 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the output PCM sampling frequency (supported 32000/44100/48000Hz) |
| XA\_CAP\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE |
| 10 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the output PCM volume rate compare with input PCM (supported from 0 – 8 times) |
| XA\_CAP\_CONFIG\_PARAM\_VOLUME\_RATE |
| 11 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the operation state of Capture plugin (RUN/PAUSE/IDLE/RESET) |
| XA\_CAP\_CONFIG\_PARAM\_STATE |
| 12 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the total number of ring buffers to use |
| XA\_CAP\_CONFIG\_PARAM\_RING\_NUM |

#### Memory allocation

Table 2‑7 List of Memory allocation Commands

| upper stage : Command / lower step : Subcommand | | Description |
| --- | --- | --- |
| 1 | XA\_API\_CMD\_GET\_MEMTABS\_SIZE | Get the size of the memory structures to be allocated for the plugin tables |
| Null |
| 2 | XA\_API\_CMD\_SET\_MEMTABS\_PTR | Pass the memory structure pointer allocated for the tables |
| Null |
| 3 | XA\_API\_CMD\_INIT | Calculate the required sizes for all the memory blocks based on the setting specific parameters |
| XA\_CMD\_TYPE\_INIT\_API\_POST\_CONFIG\_PARAMS |
| 4 | XA\_API\_CMD\_GET\_N\_MEMTABS | Obtain the number of memory blocks required by plugin |
| Null |
| 5 | XA\_API\_CMD\_GET\_MEM\_INFO\_SIZE | Get the size of the memory type being referred to by the index |
| Index |
| 6 | XA\_API\_CMD\_GET\_MEM\_INFO\_ALIGNMENT | Get the alignment information of the memory type being referred to by the index |
| Index |
| 7 | XA\_API\_CMD\_GET\_MEM\_INFO\_TYPE | Get the type of memory being referred to by the index |
| Index |
| 8 | XA\_API\_CMD\_SET\_MEM\_PTR | Set the pointer to the memory allocated for the referred index to the input value |
| Index |

#### Initialize plugin

Table 2‑8 List of initialize commands

| upper stage : Command / lower step : Subcommand | | Description |
| --- | --- | --- |
| 1 | XA\_API\_CMD\_SET\_INPUT\_BYTES | Set the number of bytes available in the input buffer |
| Null |
| 2 | XA\_API\_CMD\_INPUT\_OVER | Signal to the plugin the end of the bit stream in renderer case |
| Null |
| 3 | XA\_API\_CMD\_INIT | Setup for the HW operation, and initialize state and configuration structure |
| XA\_CMD\_TYPE\_INIT\_PROCESS |
| 4 | XA\_API\_CMD\_INIT | Check if the initialization process has completed |
| XA\_CMD\_TYPE\_INIT\_DONE\_QUERY |
| 5 | XA\_API\_CMD\_GET\_CURIDX\_INPUT\_BUF | Get the number of input buffer bytes consumed |
| Null |

#### Getting parameters

Table 2‑9 List of Get commands for renderer

| upper stage : Command / lower step : Subcommand | | Description |
| --- | --- | --- |
| 1 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM sample bit width to 16 or 24 bits |
| XA\_RDR\_CONFIG\_PARAM\_PCM\_WIDTH |
| 2 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the input PCM channels (support monaural, stereo, 4 channels, 6 channels, 8 channels) |
| XA\_RDR\_CONFIG\_PARAM\_CHANNELS |
| 3 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM sampling frequency (supported 32000/44100/48000Hz) |
| XA\_RDR\_CONFIG\_PARAM\_SAMPLE\_RATE |
| 4 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM frame size |
| XA\_RDR\_CONFIG\_PARAM\_FRAME\_SIZE |
| 5 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get Renderer output destination Audio 1st device |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT1 |
| 6 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get ADMA channel number usage for Audio 1st device |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL1 |
| 7 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get Renderer output destination Audio 2nd device |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT2 |
| 8 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get ADMA channel number usage for Audio 2nd device |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL2 |
| 9 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM sampling frequency |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE |
| 10 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM volume rate compare with input PCM |
| XA\_RDR\_CONFIG\_PARAM\_VOLUME\_RATE |
| 11 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM channels (support maximum 2 channels) |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_CHANNELS |
| 12 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the MIX module usage of Renderer (value 1/0 to used/unused mix function) |
| XA\_RDR\_CONFIG\_PARAM\_MIX\_CONTROL |
| 13 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the operation state of Renderer plugin (RUN/PAUSE/IDLE/RESET) |
| XA\_RDR\_CONFIG\_PARAM\_STATE |
| 14 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the total number of ring buffers |
| XA\_RDR\_CONFIG\_PARAM\_RING\_NUM |

Table 2‑10 List of Get commands for capture

| upper stage : Command / lower step : Subcommand | | Description |
| --- | --- | --- |
| 1 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM sample bit width to 16 or 24 bits |
| XA\_CAP\_CONFIG\_PARAM\_PCM\_WIDTH |
| 2 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM channels (Maximum is 2 channels) |
| XA\_CAP\_CONFIG\_PARAM\_CHANNELS |
| 3 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM sampling frequency (supported 32000/44100/48000Hz) |
| XA\_CAP\_CONFIG\_PARAM\_SAMPLE\_RATE |
| 4 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM frame size |
| XA\_CAP\_CONFIG\_PARAM\_FRAME\_SIZE |
| 5 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get Capture input source Audio 1st device |
| XA\_CAP\_CONFIG\_PARAM\_INPUT1 |
| 6 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get ADMA channel number usage for Audio device 1st |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL1 |
| 7 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get Capture input destination Audio 2nd device |
| XA\_CAP\_CONFIG\_PARAM\_INPUT2 |
| 8 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get ADMA channel number usage for Audio 2nd device |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL2 |
| 9 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM sampling frequency |
| XA\_CAP\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE |
| 10 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the output PCM volume rate compare with input PCM |
| XA\_CAP\_CONFIG\_PARAM\_VOLUME\_RATE |
| 11 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the operation state of Capture plugin (RUN/PAUSE/IDLE/RESET) |
| XA\_CAP\_CONFIG\_PARAM\_STATE |
| 12 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the total number of ring buffers |
| XA\_CAP\_CONFIG\_PARAM\_RING\_NUM |

#### Execution

Table 2‑11 List of execution commands

| upper stage : Command / lower step : Subcommand | | Description |
| --- | --- | --- |
| 1 | XA\_API\_CMD\_INPUT\_OVER | Stop capture in capture case |
| Null |
| 2 | XA\_API\_CMD\_SET\_INPUT\_BYTES | Set the number of bytes available in the input buffer |
| Null |
| 3 | XA\_API\_CMD\_EXECUTE | Execute the capture |
| XA\_CMD\_TYPE\_DO\_EXECUTE |
| 4 | XA\_API\_CMD\_EXECUTE | Check if the execution process has completed |
| XA\_CMD\_TYPE\_DONE\_QUERY |
| 5 | XA\_API\_CMD\_GET\_OUTPUT\_BYTES | Get the number of bytes output by the plugin in the last frame |
| Null |
| 6 | XA\_API\_CMD\_GET\_CURIDX\_INPUT\_BUF | Get the number of input buffer bytes consumed |
| Null |

### Detail of Command Specifications

The next sections describe this library command functions by using the description format below.

|  |  |
| --- | --- |
| Subcommand | Name of subcommand |
| Description | Outlines the function |
| Arguments | Describes the arguments for the function |
| Return value | Return values of function |
| Restrictions | Provides information such as precautions in using the function |

[Note] This syntax format complies with ANSI-C.

#### XA\_API\_CMD\_GET\_LIB\_ID\_STRINGS command

FD\_PLG\_RDR\_001

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CMD\_TYPE\_LIB\_VERSION | |
| Description | This command obtains the version of the library in the form of a string. The maximum length of the string that the library will provide is 30 bytes. Therefore the application shall pass a pointer to a buffer of a minimum size of 30 bytes. This command is optional. | |
| Arguments | p\_xa\_module\_obj | |
| NULL | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_LIB\_ID\_STRINGS | |
| i\_idx | |
| XA\_CMD\_TYPE\_LIB\_VERSION | |
| pv\_value | |
| Pointer to a character buffer in which the version of the library is returned | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | pv\_value is NULL. |
| Restrictions | - | |

Example

char lib\_version[30];

res = (\*api\_func)(NULL,

XA\_API\_CMD\_GET\_LIB\_ID\_STRINGS,

XA\_CMD\_TYPE\_LIB\_VERSION,

(pVOID) lib\_version);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CMD\_TYPE\_API\_VERSION | |
| Description | This command obtains the version of the API in the form of a string. The maximum length of the string that the library will provide is 30 bytes. Therefore the application shall pass a pointer to a buffer of a minimum size of 30 bytes. This command is optional. | |
| Arguments | p\_xa\_module\_obj | |
| NULL | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_LIB\_ID\_STRINGS | |
| i\_idx | |
| XA\_CMD\_TYPE\_API\_VERSION | |
| pv\_value | |
| Pointer to a character buffer in which the version of the API is returned | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | pv\_value is NULL. |
| Restrictions | - | |

[Covers: RD\_011, RD\_012]

Example

char api\_version[30];

res = (\*api\_func)(NULL,

XA\_API\_CMD\_GET\_LIB\_ID\_STRINGS,

XA\_CMD\_TYPE\_API\_VERSION,

(pVOID) api\_version);

#### XA\_API\_CMD\_GET\_API\_SIZE command

FD\_PLG\_RDR\_002

|  |  |  |
| --- | --- | --- |
| Subcommand | (None) | |
| Description | This command is used to obtain the size of the API structure, in order to allocate memory for the API structure. | |
| Arguments | p\_xa\_module\_obj | |
| NULL | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_API\_SIZE | |
| i\_idx | |
| Null | |
| pv\_value | |
| Pointer to API size variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | pv\_value is NULL. |
| Restrictions | The application shall allocate memory with an alignment of 4 bytes. | |

[Covers: RD\_011, RD\_012]

Example

WORD32 api\_size;

res = (\*api\_func)(api\_obj,

XA\_CMD\_TYPE\_API\_SIZE,

0,

&api\_size);

#### XA\_API\_CMD\_INIT command

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CMD\_TYPE\_INIT\_API\_PRE\_CONFIG\_PARAMS | |
| Description | This command is used to set the default value of the configuration parameters. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_INIT | |
| i\_idx | |
| XA\_CMD\_TYPE\_INIT\_API\_PRE\_CONFIG\_PARAMS | |
| pv\_value | |
| NULL | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| Restrictions | - | |

FD\_PLG\_RDR\_003

[Covers: RD\_011, RD\_012]

Example

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_INIT,

XA\_CMD\_TYPE\_INIT\_API\_PRE\_CONFIG\_PARAMS,

NULL);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CMD\_TYPE\_INIT\_API\_POST\_CONFIG\_PARAMS | |
| Description | This command is used to calculate the sizes of all the memory blocks required by the application. It should occur after the plugin specific parameters have been set. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_INIT | |
| i\_idx | |
| XA\_CMD\_TYPE\_INIT\_API\_POST\_CONFIG\_PARAMS | |
| pv\_value | |
| NULL | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE  (in Capture case)  XA\_RDR\_CONFIG\_FATAL\_STATE  (in Renderer case) | Incorrect sequence call (i.e. call before set memory table step) |
| XA\_CAP\_EXEC\_FATAL\_INTERNAL  (in Capture case)  XA\_RDR\_EXEC\_FATAL\_INTERNAL  (in Renderer case) | Invalid connection device setting path  (i.e. setting SRC or SSI module for both device1 and device2) |
| Restrictions | - | |

FD\_PLG\_RDR\_004

[Covers: RD\_011, RD\_012]

Example

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_INIT,

XA\_CMD\_TYPE\_INIT\_API\_POST\_CONFIG\_PARAMS,

NULL);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CMD\_TYPE\_INIT\_PROCESS | |
| Description | Setup for the HW operation, and initialize state and configuration structure. No output data is created during initialization. In this state, plugin will check all hardware modules. If a module is busy, plugin will try to establish connection with next available one. If all modules are busy, this command will return error code and stop immediately.  In case of using mix function with a different plugin [(\*)](#_MIX_(Mixing)), if user set different data format (PCM width, output sampling rate, output channel), this command will return error and stop immediately. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_INIT | |
| i\_idx | |
| XA\_CMD\_TYPE\_INIT\_PROCESS | |
| pv\_value | |
| NULL | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_EXEC\_FATAL\_STATE  (in Capture case)  (XA\_RDR\_EXEC\_FATAL\_STATE  (in Renderer case) | Incorrect sequence call (i.e. called before post-configuration step, or run post-configuration without persistent buffer allocation, or without valid shared buffers, built-in descriptor memory allocation (in case of ADMAC used)) |
| XA\_CAP\_EXEC\_FATAL\_HW  (in Capture case)  XA\_RDR\_EXEC\_FATAL\_HW  (in Renderer case) | All hardware resource are not available. |
| XA\_RDR\_EXEC\_FATAL\_FORMAT\_MISMATCH  (only Renderer) | The configured output format is different between this plugin and the plugin it is intended to mix with. |
| Restrictions | - | |

FD\_PLG\_RDR\_005

[Covers: RD\_011, RD\_012]

Example

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_INIT,

XA\_CMD\_TYPE\_INIT\_PROCESS,

NULL);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CMD\_TYPE\_INIT\_DONE\_QUERY | |
| Description | This command checks to see if the initialization process has completed. If it has, the flag value is set to 1; else, it is set to zero. A pointer to the flag variable is passed as an argument. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_INIT | |
| i\_idx | |
| XA\_CMD\_TYPE\_INIT\_DONE\_QUERY | |
| pv\_value | |
| Pointer to flag that indicates the completion of initialization process | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_EXEC\_FATAL\_STATE  (in Capture case)  (XA\_RDR\_EXEC\_FATAL\_STATE  (in Renderer case) | Incorrect sequence call  (i.e. call before post-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_006

[Covers: RD\_011, RD\_012]

Example

WORD32 done;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_INIT,

XA\_CMD\_TYPE\_INIT\_DONE\_QUERY,

&done);

#### XA\_API\_CMD\_GET\_MEMTABS\_SIZE command

FD\_PLG\_RDR\_007

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | This command is used to obtain the size of the table used to hold the memory blocks required for the plugin operation. The API returns the total size of the required table. A pointer to the size variable is sent with this API command and the plugin writes the value to the variable. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_MEMTABS\_SIZE | |
| i\_idx | |
| Null | |
| pv\_value | |
| Pointer to memory size variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE (in Capture case)  XA\_RDR\_CONFIG\_FATAL\_STATE (in Renderer case) | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

[Covers: RD\_011, RD\_012]

Example

WORD32 memtab\_size;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_MEMTABS\_SIZE,

0,

&memtab\_size);

#### XA\_API\_CMD\_SET\_MEMTABS\_PTR command

FD\_PLG\_RDR\_008

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | This command is used to set the memory structure pointer in the library to the allocated value. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_MEMTABS\_PTR | |
| i\_idx | |
| Null | |
| pv\_value | |
| Allocated pointer | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj or pv\_value is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE (in Capture case)  XA\_RDR\_CONFIG\_FATAL\_STATE (in Renderer case) | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

[Covers: RD\_011, RD\_012]

Example

pVOID memtab\_ptr;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_MEMTABS\_PTR,

0,

memtab\_ptr);

#### XA\_API\_CMD\_GET\_N\_MEMTABS command

FD\_PLG\_RDR\_009

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | This command is used to obtain the number of memory blocks needed by the plugin. This value is used as the iteration counter for the allocation of the memory blocks. A pointer to each memory block will be placed in the previously allocated memory tables. The pointer to the variable is passed to the API and the plugin writes the value to this variable. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_N\_MEMTABS | |
| i\_idx | |
| Null | |
| pv\_value | |
| Pointer to variable of number of memory blocks required to be allocated | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| Restrictions | - | |

[Covers: RD\_011, RD\_012]

Example

WORD32 n\_memtab;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_N\_MEMTABS,

0,

&n\_memtab);

#### XA\_API\_CMD\_GET\_MEM\_INFO\_SIZE command

FD\_PLG\_RDR\_010

|  |  |  |
| --- | --- | --- |
| Subcommand | Memory index | |
| Description | This command obtains the size of the memory type being referred to by the index. The size in bytes is returned in the variable pointed to by the final argument. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_MEM\_INFO\_SIZE | |
| i\_idx | |
| Index of the memory  0 - Persistent Area  1 - Input Buffer (in Renderer case)  1 - Output Buffer (in Capture case)]  2 - Built-in memory | |
| pv\_value | |
| Pointer to memory size | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned 4 bytes |
| XA\_API\_FATAL\_INVALID\_CMD\_TYPE | FIFO is used when setting for built-in memory |
| XA\_CAP\_CONFIG\_FATAL\_STATE  (in Capture case)  XA\_RDR\_CONFIG\_FATAL\_STATE  (in Renderer case) | Incorrect sequence call  (i.e. call before post-configuration step) |
| Restrictions | The index of built-in memory is only valid when 1st DMA device is ADMAC. | |

[Covers: RD\_011, RD\_012]

WORD32 mem\_size;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_MEM\_INFO\_SIZE,

index,

&mem\_size);

#### XA\_API\_CMD\_GET\_MEM\_INFO\_ALIGNMENT command

FD\_PLG\_RDR\_011

|  |  |  |
| --- | --- | --- |
| Subcommand | Memory index | |
| Description | This command gets the alignment information of the memory-type being referred to by the index. The alignment required in bytes is returned to the application. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_MEM\_INFO\_ALIGNMENT | |
| i\_idx | |
| Index of the memory  0 - Persistent Area  1 - Input Buffer (in Renderer case)  1 - Output Buffer (in Capture case)  2 - Built-in memory | |
| pv\_value | |
| Pointer to the alignment info variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned 4 bytes |
| XA\_API\_FATAL\_INVALID\_CMD\_TYPE | FIFO is used when setting for built-in memory |
| XA\_CAP\_CONFIG\_FATAL\_STATE  (in Capture case)  XA\_RDR\_CONFIG\_FATAL\_STATE  (in Renderer case) | Incorrect sequence call  (i.e. call before post-configuration step) |
| Restrictions | The index of built-in memory is only valid when 1st DMA device is ADMAC. | |

[Covers: RD\_011, RD\_012]

Example

WORD32 mem\_align;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_MEM\_INFO\_ALIGNMENT,

index,

&mem\_align);

#### XA\_API\_CMD\_GET\_MEM\_INFO\_TYPE command

FD\_PLG\_RDR\_012

|  |  |  |
| --- | --- | --- |
| Subcommand | Memory index | |
| Description | This command gets the alignment information of the memory-type being referred to by the index. The alignment required in bytes is returned to the application. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_MEM\_INFO\_TYPE | |
| i\_idx | |
| Index of the memory  0 - Persistent Area  1 - Input Buffer (in renderer case)  1 - Output Buffer (in capture case)  2 - Built-in memory | |
| pv\_value | |
| Pointer to the memory type variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned 4 bytes |
| XA\_API\_FATAL\_INVALID\_CMD\_TYPE | FIFO is used when setting for built-in memory |
| XA\_CAP\_CONFIG\_FATAL\_STATE  (in Capture case)  XA\_RDR\_CONFIG\_FATAL\_STATE  (in Renderer case) | Incorrect sequence call  (i.e. call before post-configuration step) |
| Restrictions | The index of built-in memory is only valid when 1st DMA device is ADMAC. | |

[Covers: RD\_011, RD\_012]

Example

WORD32 mem\_type;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_MEM\_INFO\_TYPE,

index,

&mem\_type);

#### XA\_API\_CMD\_SET\_MEM\_PTR command

FD\_PLG\_RDR\_013

|  |  |  |
| --- | --- | --- |
| Subcommand | Memory index | |
| Description | This command passes to the plugin the pointer to the allocated memory. This is then stored in the memory tables structure allocated earlier. For the input and output buffers, it maps all data buffers to shared memory/local memory and data will be transferred in an order from the first to the last index of shared buffers. It is legitimate to execute this command during the main plugin loop. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_MEM\_PTR | |
| i\_idx | |
| Index of the memory  0 - Persistent Area  1 - Input Buffer ( renderer case)  1 - Output Buffer ( capture case)  2 - Built-in memory | |
| pv\_value | |
| Pointer to the memory block | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes.  pv\_value is not aligned to required alignment for the requested memory block. |
| XA\_API\_FATAL\_INVALID\_CMD\_TYPE | FIFO is used when setting for built-in memory |
| XA\_CAP\_CONFIG\_FATAL\_STATE  (in Capture case)  XA\_RDR\_CONFIG\_FATAL\_STATE  (in Renderer case) | Incorrect sequence call  (i.e. call before post-configuration step) |
| Restrictions | The pointer must be correctly aligned to the requirements.  The index of built-in memory is only valid when 1st DMA device is ADMAC. | |

[Covers: RD\_011, RD\_012]

Example

pVOID addr;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_MEM\_PTR,

index,

addr);

#### XA\_API\_CMD\_INPUT\_OVER command

FD\_PLG\_RDR\_014

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | This command is used to tell the plugin that the input signal is over. The execution or initialization step will continue in loop until it all the remaining input data is processed. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_INPUT\_OVER | |
| i\_idx | |
| Null | |
| pv\_value | |
| NULL | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_EXEC\_FATAL\_STATE  (only for Capture) | Incorrect sequence call  (i.e. call before initialization step – init process) |
| Restrictions | - | |

[Covers: RD\_011, RD\_012]

Example

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_INPUT\_OVER,

0,

NULL);

#### XA\_API\_CMD\_SET\_INPUT\_BYTES command

FD\_PLG\_RDR\_015

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | In capture this command will do nothing. The purpose of this command is filled the full list of standard API.  In renderer this command will set number of bytes available in the input buffer. If renderer’s operation state is reset, it will re-setup and start HW modules and change the operation state to run. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_INPUT\_BYTES | |
| i\_idx | |
| Null | |
| pv\_value | |
| Pointer to the input byte variable (Any value is OK with Capture case) | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes in renderer case. |
| XA\_RDR\_EXEC\_FATAL\_INPUT  (only for Renderer) | Invalid input buffer size (i.e. minus buffer size or buffer size is not align with sample size) |
| XA\_RDR\_EXEC\_FATAL\_STATE  (only for Renderer) | Input memory is not allocated before or incorrect sequence call |
| Restrictions | - | |

[Covers: RD\_011, RD\_012]

Example

WORD32 filled;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_INPUT\_BYTES,

0,

&filled);

#### XA\_API\_CMD\_GET\_CURIDX\_INPUT\_BUF command

FD\_PLG\_RDR\_016

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | In Capture, this command will return value 0 each time it’s called.  In Renderer, this command will return number of input buffer bytes consumed. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CURIDX\_INPUT\_BUF | |
| i\_idx | |
| Null | |
| pv\_value | |
| Pointer to number variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_EXEC\_FATAL\_STATE  (only for Renderer) | Input memory is not allocated before. |
| Restrictions | - | |

[Covers: RD\_011, RD\_012]

Example

WORD32 consumed;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CURIDX\_INPUT\_BUF,

0,

&consumed);

#### XA\_API\_CMD\_EXECUTE command

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CMD\_TYPE\_DO\_EXECUTE | |
| Description | This command executes Capture. If capture’s operation state is reset, it will re-setup and start HW modules and change the operation state to run. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_EXECUTE | |
| i\_idx | |
| XA\_CMD\_TYPE\_DO\_EXECUTE | |
| pv\_value | |
| NULL | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_EXEC\_FATAL\_STATE  (only implement in Capture case) | Incorrect sequence call  (i.e. call before initialization step)  or output memory is not allocated before. |
| Restrictions | - | |

FD\_PLG\_RDR\_017

[Covers: RD\_011, RD\_012]

Example

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_EXECUTE,

XA\_CMD\_TYPE\_DO\_EXECUTE,

NULL);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CMD\_TYPE\_DONE\_QUERY | |
| Description | This command checks to see if the end of processing has been reached. If it is, the flag value is set to 1; else, it is set to zero. The pointer to the flag is passed as an argument. Processing by the plugin can continue for several invocations of the DO\_EXECUTE command after the last input data has been passed to the plugin, so the application should not assume that the plugin has finished generating all its output until so indicated by this command. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_EXECUTE | |
| i\_idx | |
| XA\_CMD\_TYPE\_DONE\_QUERY | |
| pv\_value | |
| Pointer to the flag variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_RDR\_EXEC\_FATAL\_STATE  (only implement in Renderer) | Incorrect sequence call (i.e. call before initialization step) |
| Restrictions | - | |

FD\_PLG\_RDR\_018

[Covers: RD\_011, RD\_012]

Example

WORD32 done;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_EXECUTE,

XA\_CMD\_TYPE\_DONE\_QUERY,

&done);

#### XA\_API\_CMD\_GET\_OUTPUT\_BYTES command

FD\_PLG\_RDR\_019

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | This command obtains the number of bytes output by the plugin during the last execution. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_OUTPUT\_BYTES | |
| i\_idx | |
| Null | |
| pv\_value | |
| Pointer to the flag variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj or pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_EXEC\_FATAL\_STATE  (only Capture) | Incorrect sequence call (i.e. call before initialization step) or output memory is not allocated before. |
| Restrictions | - | |

[Covers: RD\_011, RD\_012]

Example

WORD32 produced;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_OUTPUT\_BYTES,

0,

&produced);

#### XA\_API\_CMD\_SET\_CONFIG\_PARAM command

2.2.2.16.1 Set command for Renderer

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_PCM\_WIDTH | |
| Description | Set the PCM sample bit width to 16 or 24 bits | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_PCM\_WIDTH | |
| pv\_value | |
| Pointer to the sample bit width variable  Valid value: 16 or 24  Default value: 16 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_ERR\_PCM\_WIDTH | PCM sample width size is not valid. |
| XA\_RDR\_CONFIG\_FATAL\_ERR\_MONO\_24BIT | Setting is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_020

[Covers: RD\_011]

Example

WORD32 pcm\_width;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_PCM\_WIDTH,

&pcm\_width);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_CHANNELS | |
| Description | Set the input PCM channels number | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_CHANNELS | |
| pv\_value | |
| Pointer to the input PCM channels variable  Valid value: 1 (monaural), 2 (stereo), 4 (4 channels), 6 (6 channels), 8 (8 channels)  Default value: 2 (stereo) | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_ERR\_CHANNELS | Input PCM channels are invalid. |
| XA\_RDR\_CONFIG\_FATAL\_ERR\_MONO\_24BIT | Setting is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_021

[Covers: RD\_011]

Example

WORD32 ch;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_CHANNELS,

&ch);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_SAMPLE\_RATE | |
| Description | Set the PCM sampling frequency | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_SAMPLE\_RATE | |
| pv\_value | |
| Pointer to the input sampling frequency variable  Valid value: 32,000 / 44,100 / 48,000 Hz  Default value: 44,100 Hz | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_ERR\_SAMPLE\_RATE | Input PCM sampling frequency is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_022

[Covers: RD\_011]

Example

WORD32 sample\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_SAMPLE\_RATE,

&sample\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_FRAME\_SIZE | |
| Description | Set the PCM frame size | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_FRAME\_SIZE | |
| pv\_value | |
| Pointer to the input frame size variable  Valid value: frame size is power of two value  Default value: 1024 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_ERR\_FRAME\_SIZE | PCM frame size value is not the power of two. |
| Restrictions | - | |

FD\_PLG\_RDR\_023

[Covers: RD\_011]

Example

WORD32 frame\_size;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_FRAME\_SIZE,

&frame\_size);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_OUTPUT1 | |
| Description | Set 1st output destination device for Renderer | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT1 | |
| pv\_value | |
| Pointer to the output device index  Default value: 0 (use SSI0) | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_ERR\_SOURCE | The output source is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_024

[Covers: RD\_011]

[Note] Value range of output source relates to the PDMA and ADMAC source destination enum. It is presented below:

SSI module index: SSI\_MODULE\_MIN (0) <= output\_source <= SSI\_MODULE\_MAX (97)

SCU SRC module index: SCU\_SRC\_INPUT\_MODULE\_MIN (110) <= output\_source <= SCU\_SRC\_INPUT\_MODULE\_MAX (119)

Other index: reserved, not used.

The output source information is necessary. Default value is 140 means that it is fault if output source is not set by user.

Example

WORD32 output\_source;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_OUTPUT1,

&output\_source);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL1 | |
| Description | Set ADMA channel number usage for 1st Audio device. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL1 | |
| pv\_value | |
| Pointer to the Audio-DMAC / Audio-DMAC-peripheral-peripheral channels number  Valid value:  ADMAC\_CH[29-60] : Use Audio-DMAC to transfer  ADMACPP\_CH[00-28] : Use Audio-DMAC-pp extended to transfer  Default value: 0 - ADMACPP\_CH00 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_ERR\_DMACHANNEL | DMA channel is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_025

[Covers: RD\_011]

Example

WORD32 dma\_channel;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL1,

&dma\_channel);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_OUTPUT2 | |
| Description | Set 2nd output destination device for Renderer | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure. | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT2 | |
| pv\_value | |
| Pointer to the output device  Default value: 140 (the 2nd output device is not used) | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_ERR\_SOURCE | Output source is invalid |
| Restrictions | - | |

FD\_PLG\_RDR\_026

[Covers: RD\_011]

[Note] Value range of output source relates to the PDMA and ADMAC source destination enum. It is presented below:

SSI module index: SSI\_MODULE\_MIN (0) <= output\_source <= SSI\_MODULE\_MAX (97)

SCU SRC module index: SCU\_SRC\_INPUT\_MODULE\_MIN (110) <= output\_source <= SCU\_SRC\_INPUT\_MODULE\_MAX (119)

Other index: reserved, not used.

The output source information is necessary. Default value is 140 means that it is fault if output source is not set by user.

Example

WORD32 output\_source;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_OUTPUT2,

&output\_source);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL2 | |
| Description | Set ADMA channel number usage for 2nd Audio device. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL2 | |
| pv\_value | |
| Pointer to the Audio-DMAC / Audio-DMAC-peripheral-peripheral channels number  Valid value:  ADMAC\_CH[29-60] : Use Audio-DMAC to transfer  ADMACPP\_CH[00-28] : Use Audio-DMAC-pp extended to transfer  Default value: 1 - ADMACPP\_CH01 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_ERR\_DMACHANNEL | DMA channel is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_027

[Covers: RD\_011]

Example

WORD32 dma\_channel;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL2,

&dma\_channel);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | |
| Description | Set output sample rate in Sampling Rate Converter (SRC) of Sampling Rate Converter Unit (SCU). If this setting value is different from input sample rate of PCM, SRC connection will be enabled even without setting connection device path. And the connection will automatically use the available Audio-DMAC channel. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | |
| pv\_value | |
| Pointer to the output sampling frequency variable  Valid value: 32,000 / 44,100 / 48,000 (Hz)  Default value: 44,100 Hz | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_ERR\_SAMPLE\_RATE | Output sample rate is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_028

[Covers: RD\_011]

Example:

WORD32 sample\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE,

&sample\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_VOLUME\_RATE | |
| Description | Set the output PCM volume rate in Digital Volume and Mute Function (DVC), and Mixer (MIX) (if mix function is used) of Sampling Rate Converter Unit (SCU). Any set value except for 0xFFFF FFFF (disable DVC) enables DVC of SCU module and the connection will be established even without setting connection path. This command can be set during plugin execution to update the output volume rate. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_VOLUME\_RATE | |
| pv\_value | |
| Pointer to the volume ratio number (using Fix-point Q3.20)  Valid value:  0xFFFF FFFF : disable DVC module  [0, 0x7F FFFF] : setting volume rate value  Default value: 0xFFFF FFFF | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_VOLUME\_RATE | Volume rate value is invalid. |
| Restrictions | To use volume update function, DVC must be enabled in advance (before Renderer goes to post-configuration state). | |

FD\_PLG\_RDR\_029

[Covers: RD\_011]

Example:

WORD32 vol\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_VOLUME\_RATE,

&vol\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_OUT\_CHANNELS | |
| Description | Set the output PCM channels number. If this setting is difference between input and output channel, the CTU module will be enabled to perform channel transfer. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_CHANNELS | |
| pv\_value | |
| Pointer to the output PCM channels variable  Valid value: 1 (monaural), 2 (stereo)  Default value: 2 (stereo) | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_ERR\_CHANNELS | Output PCM channels is invalid. |
| XA\_RDR\_CONFIG\_FATAL\_ERR\_MONO\_24BIT | Setting is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_030

[Covers: RD\_011]

Example

WORD32 ch\_out;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_OUT\_CHANNELS,

&ch\_out);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_MIX\_CONTROL | |
| Description | Set the MIX module usage for Renderer | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_MIX\_CONTROL | |
| pv\_value | |
| Pointer to the MIX control flag variable  Valid value: 0 (not use mix function), 1 (use mix function)  Default value: 0 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_RDR\_CONFIG\_NONFATAL\_MIX\_CONTROL | The setting value is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_031

[Covers: RD\_011]

Example

WORD32 mix\_ctl;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_MIX\_CONTROL,

&mix\_ctl);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_STATE | |
| Description | Set the operation state for Renderer plugin. Renderer can change from RUN to PAUSE, from RUN to IDLE, from PAUSE to IDLE, from PAUSE to RUN, from RUN to RESET and from PAUSE to RESET, RESET to IDLE during plugin execution. This command only has effect when Renderer is in the execution state. Otherwise, it will do nothing and return no error. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_STATE | |
| pv\_value | |
| Pointer to the expected state of plugin  Valid value:  0 - XA\_RDR\_STATE\_RUN: execute data to output  1 - XA\_RDR\_STATE\_IDLE: stop and destroy plugin  2 - XA\_RDR\_STATE\_PAUSE: pause the executing data to output  3 – XA\_RDR\_STATE\_RESET: reset the plugin  Default value: 0 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step) |
| XA\_RDR\_EXEC\_FATAL\_HW | Cannot reset the plugin |
| XA\_RDR\_CONFIG\_NONFATAL\_OPERATION | The setting value is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_032

[Covers: RD\_011]

Example

WORD32 state;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_STATE,

&state);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_RING\_NUM | |
| Description | Set the total number of ring buffers to use | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_RING\_NUM | |
| pv\_value | |
| Pointer to the total number of ring buffers  Valid value: 2, 4  Default value: 2 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step or after post-configuration step) |
| XA\_RDR\_CONFIG\_FATAL\_RING\_NUM | Ring num value is invalid |
| Restrictions | - | |

FD\_PLG\_RDR\_071

[Covers: RD\_015]

Example

WORD32 ring\_num;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_RING\_NUM,

&ring\_num);

2.2.2.16.2 Set command for capture

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_PCM\_WIDTH | |
| Description | Set the PCM sample bit width to 16 or 24 bits | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_PCM\_WIDTH | |
| pv\_value | |
| Pointer to the sample bit width variable  Valid value: 16 or 24  Default value: 16 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_CAP\_CONFIG\_NONFATAL\_ERR\_PCM\_WIDTH | PCM sample width size is not valid. |
| XA\_CAP\_CONFIG\_FATAL\_ERR\_MONO\_24BIT | Setting is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_033

[Covers: RD\_012]

Example

WORD32 pcm\_width;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_PCM\_WIDTH,

&pcm\_width);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_CHANNELS | |
| Description | Set the PCM channels number | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_CHANNELS | |
| pv\_value | |
| Pointer to the input channels variable  Valid value: 1 (monaural), 2 (stereo)  Default value: 2 (stereo) | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_CAP\_CONFIG\_NONFATAL\_ERR\_CHANNELS | PCM input channels is invalid. |
| XA\_CAP\_CONFIG\_FATAL\_ERR\_MONO\_24BIT | Setting is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_034

[Covers: RD\_012]

Example

WORD32 ch;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_CHANNELS,

&ch);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_SAMPLE\_RATE | |
| Description | Set the PCM sampling frequency | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_SAMPLE\_RATE | |
| pv\_value | |
| Pointer to the input sampling frequency variable  Valid value: 32,000 / 44,100 / 48,000 (Hz)  Default value: 44,100 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_CAP\_CONFIG\_NONFATAL\_ERR\_SAMPLE\_RATE | PCM input sampling frequency is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_035

[Covers: RD\_012]

Example

WORD32 sample\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_SAMPLE\_RATE,

&sample\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_FRAME\_SIZE | |
| Description | Set the PCM frame size | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_FRAME\_SIZE | |
| pv\_value | |
| Pointer to the input frame size variable  Valid value: frame size is power of two value  Default value: 1024 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_CAP\_CONFIG\_NONFATAL\_ERR\_FRAME\_SIZE | PCM frame size value is not the power of two. |
| Restrictions | - | |

FD\_PLG\_RDR\_036

[Covers: RD\_012]

Example

WORD32 frame\_size;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_FRAME\_SIZE,

&frame\_size);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_INPUT1 | |
| Description | Set 1st input device for Capture | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_INPUT1 | |
| pv\_value | |
| Pointer to the input source value  Default value: 10 (SSI1) | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_CAP\_CONFIG\_NONFATAL\_ERR\_SOURCE | PCM input source is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_037

[Covers: RD\_012]

[Note] Value range of input source relates to the PDMA and ADMAC source destination enum. It is presented below:

SSI module index: SSI\_MODULE\_MIN (0) <= output\_source <= SSI\_MODULE\_MAX (97)

SCU SRC module index: SCU\_SRC\_INPUT\_MODULE\_MIN (110) <= output\_source <= SCU\_SRC\_INPUT\_MODULE\_MAX (119)

Other index: reserved, not used.

The output source information is necessary. Default value is 140 means that it is fault if output source is not set by user.

Example

WORD32 input\_source;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_INPUT1,

&input\_source);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL1 | |
| Description | Set ADMA channel number usage for 1st Audio device | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL1 | |
| pv\_value | |
| Pointer to the ADMA channels number  Valid value:  ADMAC\_CH[29-60] : Use Audio-DMAC to transfer  ADMACPP\_CH[0-28] : Use Audio-DMAC-pp extended to transfer  Default value: ADMACPP\_CH10 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_CAP\_CONFIG\_NONFATAL\_ERR\_DMACHANNEL | PCM DMA channel is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_038

[Covers: RD\_012]

Example

WORD32 dma\_channel;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL1,

&dma\_channel);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_INPUT2 | |
| Description | Set 2nd input device for Capture | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_INPUT2 | |
| pv\_value | |
| Pointer to the input source value  Default value: 140 (the 2nd input device is not used) | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_CAP\_CONFIG\_NONFATAL\_ERR\_SOURCE | PCM input source is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_039

[Covers: RD\_012]

[Note] Value range of input source relates to the PDMA and ADMAC source destination enum. It is presented below:

SSI module index: SSI\_MODULE\_MIN (0) <= output\_source <= SSI\_MODULE\_MAX (97)

SCU SRC module index: SCU\_SRC\_INPUT\_MODULE\_MIN (110) <= output\_source <= SCU\_SRC\_INPUT\_MODULE\_MAX (119)

Other index: reserved, not used.

The output source information is necessary. Default value is 140 means that it is fault if output source is not set by user.

Example

WORD32 input\_source;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_INPUT2,

&input\_source);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL2 | |
| Description | Set ADMA channel number usage for 2nd Audio device | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL2 | |
| pv\_value | |
| Pointer to the ADMA channels number  Valid value:  ADMAC\_CH[0-31] : Audio-DMAC usage channel 0 -31  ADMACPP\_CH[0-28] : Audio-DMAC-pp extended usage channel 0-28  Default value: ADMACPP\_CH11 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_CAP\_CONFIG\_NONFATAL\_ERR\_DMACHANNEL | PCM DMA channel is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_040

[Covers: RD\_012]

Example

WORD32 dma\_channel;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL2,

&dma\_channel);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | |
| Description | Set output sample rate in Sampling Rate Converter (SRC) of Sampling Rate Converter Unit (SCU). If this setting value is different from input sample rate of PCM, SRC connection will be enabled even without setting connection device path. And the connection will automatically use the available Audio-DMAC channel. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | |
| pv\_value | |
| Pointer to the output sampling frequency variable  Valid value: 32,000 / 44,100 / 48,000 (Hz)  Default value: 44,100 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step and after post-configuration step) |
| XA\_CAP\_CONFIG\_NONFATAL\_ERR\_SAMPLE\_RATE | PCM output sample rate is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_041

[Covers: RD\_012]

Example:

WORD32 sample\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE,

&sample\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_VOLUME\_RATE | |
| Description | Set the output PCM volume rate in Digital Volume and Mute Function (DVC) of Sampling Rate Converter Unit (SCU). Any set value other than 0xFFFF FFFF (disable DVC) enables DVC of SCU module and the connection will be established even without setting connection path. This command can be set during plugin execution to update the output volume rate. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_VOLUME\_RATE | |
| pv\_value | |
| Pointer to the volume ratio number (using Fix-point Q3.20)  Valid value:  0xFFFF FFFF : disable DVC module  [0, 0x7F FFFF] : setting volume rate value  Default value: 0xFFFF FFFF | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step) |
| XA\_CAP\_CONFIG\_NONFATAL\_VOLUME\_RATE | PCM volume rate value is invalid. |
| Restrictions | To use volume update function, DVC must be enabled in advance (before Capture goes to post-configuration state). | |

FD\_PLG\_RDR\_042

[Covers: RD\_012]

Example:

WORD32 vol\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_VOLUME\_RATE,

&vol\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_STATE | |
| Description | Set the operation state for Capture plugin. Capture can change from RUN to PAUSE, from RUN to IDLE, from PAUSE to IDLE, from PAUSE to RUN, from RUN to RESET, from PAUSE to RESET, from RESET to IDLE during plugin execution. This command only has effect when Capture is in the execution state. Otherwise, it will do nothing and return no error. | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_STATE | |
| pv\_value | |
| Pointer to the expected state of plugin  Valid value:  0 - XA\_CAP\_STATE\_RUN: execute data to output  1 - XA\_CAP\_STATE\_IDLE: stop and destroy plugin  2 - XA\_CAP\_STATE\_PAUSE: pause the executing data to output  3 - XA\_CAP\_STATE\_RESET: reset the plugin  Default value: 0 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step) |
| XA\_CAP\_EXEC\_FATAL\_HW | Cannot reset the plugin |
| XA\_CAP\_CONFIG\_NONFATAL\_OPERATION | The setting value is invalid. |
| Restrictions | - | |

FD\_PLG\_RDR\_043

[Covers: RD\_012]

Example

WORD32 state;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_STATE,

&state);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_RING\_NUM | |
| Description | Set total number of ring buffers to use | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_SET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_RING\_NUM | |
| pv\_value | |
| Pointer to the total number of ring buffers  Valid value: 2, 4  Default value: 2 | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step or after post-configuration step) |
| XA\_CAP\_CONFIG\_FATAL\_RING\_NUM | Ring num value is invalid |
| Restrictions | - | |

FD\_PLG\_RDR\_068

[Covers: RD\_015]

Example

WORD32 ring\_num;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_SET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_RING\_NUM,

&ring\_num);

#### XA\_API\_CMD\_GET\_CONFIG\_PARAM command

2.2.2.17.1 Get command for renderer

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_PCM\_WIDTH | |
| Description | Get the PCM sample bit width settings | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_PCM\_WIDTH | |
| pv\_value | |
| Pointer to the sample bit width variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_044

[Covers: RD\_011]

Example

WORD32 pcm\_width;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_PCM\_WIDTH,

&pcm\_width);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_CHANNELS | |
| Description | Get the input PCM channels number setting | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_CHANNELS | |
| pv\_value | |
| Pointer to channels variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_045

[Covers: RD\_011]

Example

WORD32 ch;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_CHANNELS,

&ch);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_SAMPLE\_RATE | |
| Description | Get the PCM sampling frequency setting | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_SAMPLE\_RATE | |
| pv\_value | |
| Pointer to sampling frequency variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_046

[Covers: RD\_011]

Example

WORD32 sample\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_SAMPLE\_RATE,

&sample\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_FRAME\_SIZE | |
| Description | Get the PCM frame size setting | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_FRAME\_SIZE | |
| pv\_value | |
| Pointer to frame size variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_067

[Covers: RD\_011]

Example

WORD32 frame\_size;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_FRAME\_SIZE,

&frame\_size);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_OUTPUT1 | |
| Description | Get 1st output destination device for Renderer info | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT1 | |
| pv\_value | |
| Pointer to the 1st output destination device value | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_047

[Covers: RD\_011]

Example

WORD32 output\_source;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_OUTPUT1,

&output\_source);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL1 | |
| Description | Get ADMA channel number usage for 1st Audio device info | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL1 | |
| pv\_value | |
| Pointer to the 1st Audio-DMAC / Audio-DMAC-peripheral-peripheral channel variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_048

[Covers: RD\_011]

Example

WORD32 dma\_channel;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL1,

&dma\_channel);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_OUTPUT2 | |
| Description | Get 2nd output destination device for Renderer info | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_OUTPUT2 | |
| pv\_value | |
| Pointer to the 2nd output destination device value | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_049

[Covers: RD\_011]

Example

WORD32 output\_source;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_OUTPUT2,

&output\_source);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL2 | |
| Description | Get ADMA channel number usage for 2nd Audio device info | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL2 | |
| pv\_value | |
| Pointer to the 2nd Audio-DMAC / Audio-DMAC-peripheral-peripheral channel variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_050

[Covers: RD\_011]

Example

WORD32 dma\_channel;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_DMACHANNEL2,

&dma\_channel);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | |
| Description | Get output sample rate setting | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | |
| pv\_value | |
| Pointer to the output sampling frequency variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_051

[Covers: RD\_011]

Example:

WORD32 sample\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE,

&sample\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_VOLUME\_RATE | |
| Description | Get the output PCM volume rate setting value | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_VOLUME\_RATE | |
| pv\_value | |
| Pointer to the volume ratio number (using Fix-point Q3.20) | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_052

[Covers: RD\_011]

Example:

WORD32 vol\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_VOLUME\_RATE,

&vol\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_OUT\_CHANNELS | |
| Description | Get the output PCM channels number setting | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_OUT\_CHANNELS | |
| pv\_value | |
| Pointer to output channels variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_053

[Covers: RD\_011]

Example

WORD32 ch\_out;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_OUT\_CHANNELS,

&ch\_out);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_MIX\_CONTROL | |
| Description | Get the MIX module usage of Renderer | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_MIX\_CONTROL | |
| pv\_value | |
| Pointer to Mix control flag variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_054

[Covers: RD\_011]

Example

WORD32 mix\_ctl;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_MIX\_CONTROL,

&mix\_ctl);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_STATE | |
| Description | Get the current operation state of Renderer plugin | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_STATE | |
| pv\_value | |
| Pointer to the current operation state of plugin | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_055

[Covers: RD\_011]

Example

WORD32 cur\_state;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_STATE,

&cur\_state);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_RDR\_CONFIG\_PARAM\_RING\_NUM | |
| Description | Get the total number of ring buffers | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_RDR\_CONFIG\_PARAM\_RING\_NUM | |
| pv\_value | |
| Pointer to the total number of ring buffers | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_RDR\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_070

[Covers: RD\_015]

Example

WORD32 ring\_num;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_RDR\_CONFIG\_PARAM\_RING\_NUM,

&ring\_num);

2.2.2.17.2 Get command for capture

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_PCM\_WIDTH | |
| Description | Get the PCM sample bit width settings | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_PCM\_WIDTH | |
| pv\_value | |
| Pointer to the sample bit width variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. called before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_056

[Covers: RD\_012]

Example

WORD32 pcm\_width;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_PCM\_WIDTH,

&pcm\_width);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_CHANNELS | |
| Description | Get the PCM channels number setting | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_CHANNELS | |
| pv\_value | |
| Pointer to channels variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_057

[Covers: RD\_012]

Example

WORD32 ch;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_CHANNELS,

&ch);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_SAMPLE\_RATE | |
| Description | Get the PCM sampling frequency setting | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_SAMPLE\_RATE | |
| pv\_value | |
| Pointer to sampling frequency variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_058

[Covers: RD\_012]

Example

WORD32 sample\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_SAMPLE\_RATE,

&sample\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_FRAME\_SIZE | |
| Description | Get the PCM frame size setting | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_FRAME\_SIZE | |
| pv\_value | |
| Pointer to frame size variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_059

[Covers: RD\_012]

Example

WORD32 frame\_size;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_FRAME\_SIZE,

&frame\_size);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_INPUT1 | |
| Description | Get 1st input source device for Capture info | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_INPUT1 | |
| pv\_value | |
| Pointer to the 1st input device value | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_060

[Covers: RD\_012]

Example

WORD32 input\_source;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_INPUT1,

&input\_source);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL1 | |
| Description | Get ADMA channel number usage for 1st Audio device info | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL1 | |
| pv\_value | |
| Pointer to the 1st Audio-DMAC channel | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_061

[Covers: RD\_012]

Example

WORD32 dma\_channel;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL1,

&dma\_channel);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_INPUT2 | |
| Description | Get 2nd input source device for Capture info | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_INPUT2 | |
| pv\_value | |
| Pointer to the 2nd input source device value | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_062

[Covers: RD\_012]

Example

WORD32 input\_source;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_INPUT2,

&input\_source);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL2 | |
| Description | Get ADMA channel number usage for 2nd Audio device info | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL2 | |
| pv\_value | |
| Pointer to the 2nd Audio-DMAC channel | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_063

[Covers: RD\_012]

Example

WORD32 dma\_channel;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_DMACHANNEL2,

&dma\_channel);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | |
| Description | Get output sample rate setting value | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE | |
| pv\_value | |
| Pointer to the output sampling frequency variable | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_064

[Covers: RD\_012]

Example:

WORD32 sample\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_OUT\_SAMPLE\_RATE,

&sample\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_VOLUME\_RATE | |
| Description | Get the output PCM volume rate setting value | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_VOLUME\_RATE | |
| pv\_value | |
| Pointer to the volume ratio number (using Fix-point Q3.20) | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_065

[Covers: RD\_012]

Example

WORD32 vol\_rate;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_VOLUME\_RATE,

&vol\_rate);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_STATE | |
| Description | Get the current operation state of Capture plugin | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_STATE | |
| pv\_value | |
| Pointer to the current operation state of plugin | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_066

[Covers: RD\_012]

Example

WORD32 cur\_state;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_STATE,

&cur\_state);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CAP\_CONFIG\_PARAM\_RING\_NUM | |
| Description | Get the total number of ring buffers | |
| Arguments | p\_xa\_module\_obj | |
| Pointer to API Structure | |
| i\_cmd | |
| XA\_API\_CMD\_GET\_CONFIG\_PARAM | |
| i\_idx | |
| XA\_CAP\_CONFIG\_PARAM\_RING\_NUM | |
| pv\_value | |
| Pointer to the total number of ring buffers | |
| Return value | XA\_NO\_ERROR | Normally ends. |
| XA\_API\_FATAL\_MEM\_ALLOC | p\_xa\_module\_obj / pv\_value is NULL. |
| XA\_API\_FATAL\_MEM\_ALIGN | p\_xa\_module\_obj is not aligned to 4 bytes. |
| XA\_CAP\_CONFIG\_FATAL\_STATE | Incorrect sequence call  (i.e. call before pre-configuration step) |
| Restrictions | - | |

FD\_PLG\_RDR\_069

[Covers: RD\_015]

Example

WORD32 ring\_num;

res = (\*api\_func)(api\_obj,

XA\_API\_CMD\_GET\_CONFIG\_PARAM,

XA\_CAP\_CONFIG\_PARAM\_RING\_NUM,

&ring\_num);

## Structures

Table 2‑12 lists the structures for this software. The user should reserve areas required for these structures. For detailed specifications of these input structures, refer to Section 2.3.1.

Table 2‑12 Structures

| Structure name | Size (bytes) | Outline |
| --- | --- | --- |
| XARelrdr | 1424 (bytes) | API’s structure to stores the information of API |
| XARelcap | 1384 (bytes) | API’s structure to stores the information of API |

### XARelrdr type structure

The XARelrdr type structure is the work area used by the renderer of Renderer plugin. When using this plugin, secure the area with the application program. It’s not necessary to refer to this area because it only contains the internal variables and working buffers of the plugin. Make sure not to change the value of this area with the application program.

Table 2‑13 XARelrdr type structure information

| Member name | Outline |
| --- | --- |
| pVOID pMem\_tabs | Memory table controller |
| UWORD32 ring\_num | Number of ring buffer |
| UWORD32 ring\_size | Total size of ring-buffer in sample |
| UWORD32 buffer\_size | Total size of ring-buffer in bytes |
| UWORD32 sample\_size | Size of PCM sample |
| UWORD32 read\_idx | FIFO reading position |
| volatile UWORD32 buf\_idx | Index of the current buffer |
| volatile UWORD32 last\_buf\_idx | Index of the previous buffer |
| UWORD32 filled | Number of samples present in the buffer |
| UWORD32 state | Component state |
| UWORD32 consumed | Number of samples consumed to framework |
| XosEvent relrdr\_event | Renderer polling event |
| XosThread relrdr\_thread | Renderer polling thread |
| UWORD32 frame\_size | Number of sample in each frame |
| UWORD32 in\_channels | Number of input channels |
| UWORD32 out\_channels | Number of output channels |
| UWORD32 in\_rate | Input sampling rate |
| UWORD32 out\_rate | Output sampling rate |
| UWORD32 pcm\_width | Sample width |
| UWORD32 stage\_flag | Present current stage used in ADMAC |
| UWORD32 stage\_size | Stage memory block size when using DMAC |
| UWORD32 stage\_num | Stage number when using ADMAC |
| UWORD32 trans\_num | Transfer number time when using ADMAC |
| relren\_Device dev[2] | Output device info array after setting param |
| relren\_Enable\_Module enable\_module | Module list is used |
| WORD32 volume | Volume value |
| UWORD32 mix\_ctrl | Mix control value |
| UWORD32 operation\_state | Operation state of plugin |
| SSIU\_SSI\_MODULE ssi\_module | SSI module information |
| WORD32 first\_run | Flag indicating whether the current stream is the first stream |
| SRC\_START\_MODE start\_mode | Start mode when using SRC |

### XARelcap type structure

The XARelcap type structure is the work area used by the capture of Renderer plugin. When using this plugin, secure the area with the application program. It’s not necessary to refer to this area because it only contains the internal variables and working buffers of the plugin. Make sure not to change the value of this area with the application program.

Table 2‑14 XARelcap type structure information

| Member name | Outline |
| --- | --- |
| pVOID pMem\_tabs | Memory table controller |
| UWORD32 sample\_size | Size of PCM sample |
| UWORD32 state | Component state |
| UWORD32 transfered | Number of samples have been transferred |
| XosEvent relcap\_event | Capture polling event |
| XosThread relcap\_thread | Capture polling thread |
| UWORD32 frame\_size | Number of sample in each frame |
| UWORD32 channels | Number of channels |
| UWORD32 in\_rate | Input sampling rate |
| UWORD32 out\_rate | Output sample rate |
| UWORD32 pcm\_width | Sample width |
| UWORD32 ring\_num | Number of ring buffer |
| UWORD32 stage\_flag | Present current stage used in ADMAC |
| UWORD32 stage\_size | Stage memory block size when using DMAC |
| UWORD32 stage\_num | Stage number when using DMAC |
| UWORD32 trans\_num | Transfer number time when using DMAC |
| UWORD32 ring\_size | Total size of ring-buffer in samples |
| UWORD32 buffer\_size | Total size of ring-buffer in bytes |
| relcap\_Device dev[2] | Output device info array after set param |
| relcap\_Enable\_module enable\_module | Module list is used |
| WORD32 volume | Volume value |
| SSIU\_SSI\_MODULE ssi\_module | SSI module information |
| UWORD32 operation\_state | Operation state of plugin |
| SRC\_START\_MODE start\_mode | Start mode when using SRC |
| UWORD32 buf\_idx | Index of the current buffer |

## Memory Specifications

This section describes the memory areas used by this software.

### Persistent Area

Table 2‑15 Persistent Area Description

| Item | Area which always holds values when this software is used.  If the user manipulates this area after initialization, the correct execution of this software is not ensured. |
| --- | --- |
| Symbol name | - (freely defined by the user) |
| Size | Obtain the actually required size with 2.2.2.7 |
| Area reservation | The user should reserve this area. |
| Allocation | This area is included in RAM. |
| Alignment | Align this area on a 4-byte boundary. |

### Stack Area

Table 2‑16 Stack Area Description

| Item | Stack area used by this software |
| --- | --- |
| Symbol name | - (freely defined by the user) |
| Size | Obtain the actually required size with 2.2.2.7 |
| Area reservation | The user should reserve this area.  To use this software, reserve a software stack area which exceeds the size above. |
| Allocation | This area is included in RAM. |
| Alignment | - |

### Heap Area

This software does not use a heap area.

### Input Buffer

Input buffer only is used in the renderer case.

Table 2‑17 Input Buffer Description

| Item | Area which stores inputs from this software.  The input buffer contains 16-bit/24-bit linear PCM data (hereinafter called PCM data).  If the user manipulates this area during rendering processing, the normal execution of the program cannot be ensured. |
| --- | --- |
| Symbol name | - (freely defined by the user) |
| Size | Please secure more than size with 2.2.2.7 (a multiple of 2.2.2.7). |
| Area reservation | The user should reserve this area.  The user can freely use this area after the rendering of one block. |
| Allocation | This area is included in RAM. |
| Alignment | Align this area on a 4-byte boundary. |

### Output Buffer

Output buffer only is used in the capture case.

Table 2‑18 Output Buffer Description

| Item | Area which stores outputs from this software.  The output buffer contains 16-bit/24-bit linear PCM data (hereinafter called PCM data).  If the user manipulates this area during rendering processing, the normal execution of the program cannot be ensured. |
| --- | --- |
| Symbol name | - (freely defined by the user) |
| Size | Please secure more than size with 2.2.2.7 (a multiple of 2.2.2.7). |
| Area reservation | The user should reserve this area.  The user can freely use this area after the rendering of one block. |
| Allocation | This area is included in RAM. |
| Alignment | Align this area on a 4-byte boundary. |

* + - * 1. Input/ Output data storage method

Data is input/ output in the formats as shown in Figure 2‑4 (consecutive buffers are specified for the channels). The input/output buffer (memory) stores data in 2-byte (16-bit) units. The byte order for accessing the buffer is little endian (see Figure 2‑2).

0x12

16 bit PCM Data

Offset [byte]

Access Image

0x34

0x12

0x34

MSB

LSB

+0

+1

Figure 2‑2 PCM 16-bit Data Access (Little Endian Mode)

0x00

24 bit PCM Data

Offset [byte]

Access Image

0x12

0x34

0x56

MSB

LSB

+1

+2

0x34

0x00

+0

0x56

0x12

+3

Figure 2‑3 PCM 24-bit Data Access (Little Endian Mode)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Output PCM Data | |  |
| MSB |  |  | LSB |
|  | Channel-0 Data0 | |  |
| +1 | Channel-1 Data0 | |  |
| +2 | Channel-0 Data1 | |  |
| +3 | Channel-1 Data1 | |  |
|  | … | |  |
| +1022 | Channel-0 Data511 | |  |
| +1023 | Channel-1 Data511 | |  |
| +1024 | Channel-0 Data512 | |  |
| +1025 | Channel-1 Data512 | |  |
|  | … | |  |
| +2046 | Channel-0 Data1023 | |  |
| +2047 | Channel-1 Data1023 | |  |
| [Sample] | | | |
| Stereo Output Format | | | |

Figure 2‑4 Output Formats

## Error Processing

This software's functions return the error codes listed in Table 2‑20.

### Error codes

Below are the error codes for this software.

Table 2‑19 Error Codes for Renderer

| Error code (32bit) | Value | Description |
| --- | --- | --- |
| [1]  XA\_NO\_ERROR | 0x00000000 | The processing results are normal.  The process has terminated normally. |
| [2]  XA\_API\_FATAL\_MEM\_ALLOC | 0xFFFF8000 | Abnormality has occurred, which disables process continuation. An address of API structure was specified at the argument is NULL, the program execution is incorrect.  Because it becomes the common API error, please check the correct procedure. |
| [3]  XA\_API\_FATAL\_MEM\_ALIGN | 0xFFFF8001 | Abnormality has occurred, which disables process continuation. An address of API structure was specified at the argument does not 4 byte align. Because it becomes the common API error, please check the correct procedure. |
| [4]  XA\_API\_FATAL\_INVALID\_CMD | 0xFFFF8002 | Abnormality has occurred, which disables process continuation. The command was specified at the argument does not support. Because it becomes the common API error, please check the correct procedure. |
| [5]  XA\_API\_FATAL\_INVALID\_CMD\_TYPE | 0xFFFF8003 | Abnormality has occurred, which disables process continuation. The subcommand was specified at the argument does not support. Because it becomes the common API error, please check the correct procedure. |
| [6]  XA\_RDR\_CONFIG\_FATAL\_STATE | 0xFFFF8881 | Abnormality has occurred, which disables process continuation. The subcommand was specified at the argument does not support. Because it becomes the common API error, please check the correct procedure. |
| [7]  XA\_RDR\_CONFIG\_FATAL\_ERR\_MONO\_24BIT | 0xFFFF8883 | It is an error for invalid setting, 24 bit mono channel case does not supported. |
| [8]  XA\_RDR\_EXEC\_FATAL\_STATE | 0xFFFF9081 | Abnormality has occurred, which disables process continuation. The subcommand was specified at the argument does not support. Because it becomes the common API error, please check the correct procedure. |
| [7]  XA\_RDR\_EXEC\_FATAL\_INPUT | 0xFFFF9082 | Abnormality has occurred, which disables process continuation. The input size is not align with sample size. Because it becomes the common API error, please check the correct size of input buffers. |
| [9]  XA\_RDR\_EXEC\_FATAL\_INTERNAL | 0xFFFF9083 | Abnormality has occurred, which disables process continuation. Some of setting becomes incorrect after combination. Because it becomes the common API error, please check the correct parameters. |
| [10]  XA\_RDR\_EXEC\_FATAL\_HW | 0xFFFF9080 | Abnormality has occurred, which disables process continuation. Hardware modules are not available... Because it becomes the common API error, please check the correct parameters and make sure the resource is validity. |
| [11]  XA\_RDR\_CONFIG\_NONFATAL\_ERR\_PCM\_WIDTH | 0x00000880 | It is an error for renderer specifications out of the range.  The PCM width was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16) |
| [12]  XA\_RDR\_CONFIG\_NONFATAL\_ERR\_CHANNELS | 0x00000881 | It is an error for renderer specifications out of the range.  The channel numbers was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16) |
| [13]  XA\_RDR\_CONFIG\_NONFATAL\_ERR\_SAMPLE\_RATE | 0x00000882 | It is an error for renderer specifications out of the range.  The sample rate was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16) |
| [14]  XA\_RDR\_CONFIG\_NONFATAL\_ERR\_FRAME\_SIZE | 0x00000883 | It is an error for renderer specifications out of the range.  The Input buffer size was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16) |
| [15]  XA\_RDR\_CONFIG\_NONFATAL\_ERR\_SOURCE | 0x00000884 | It is an error for renderer specifications out of the range.  The input buffer size was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16 |
| [16]  XA\_RDR\_CONFIG\_NONFATAL\_ERR\_DMACHANNEL | 0x00000885 | It is an error for renderer specifications out of the range.  The input buffer size was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16) |
| [17]  XA\_RDR\_CONFIG\_NONFATAL\_VOLUME\_RATE | 0x00000886 | It is an error for renderer specification out of range.  The volume rate value was specified at the argument does not support. |
| [18]  XA\_RDR\_CONFIG\_NONFATAL\_MIX\_CONTROL | 0x00000887 | It is an error for renderer specification out of range.  The mix control value was specified at the argument does not support. |
| [19] XA\_RDR\_EXEC\_FATAL\_FORMAT\_MISMATCH | 0xFFFF9084 | It is an error related to the mismatching data format between 2 output streams which are going to mix together. Please make sure these streams have the same format (PCM width, output sampling rate, output channel). |
| [20] XA\_RDR\_CONFIG\_NONFATAL\_OPERATION | 0x00000888 | It is an error for renderer specification out of range.  The setting state was specified at the argument does not support. |
| [21] XA\_RDR\_CONFIG\_FATAL\_RING\_NUM | 0xFFFF8884 | It is an error for renderer specification out of range regarding the total number of ring buffers to be used. |
| [22] | Others | Reserved |

Table 2‑20 Error Codes for Capture

| Error code (32bit) | Value | Description |
| --- | --- | --- |
| [1]  XA\_NO\_ERROR | 0x00000000 | The processing results are normal.  The process has terminated normally. |
| [2]  XA\_API\_FATAL\_MEM\_ALLOC | 0xFFFF8000 | Abnormality has occurred, which disables process continuation. An address of API structure was specified at the argument is NULL, the program execution is incorrect.  Because it becomes the common API error, please check the correct procedure. |
| [3]  XA\_API\_FATAL\_MEM\_ALIGN | 0xFFFF8001 | Abnormality has occurred, which disables process continuation. An address of API structure was specified at the argument does not 4 byte align. Because it becomes the common API error, please check the correct procedure. |
| [4]  XA\_API\_FATAL\_INVALID\_CMD | 0xFFFF8002 | Abnormality has occurred, which disables process continuation. The command was specified at the argument does not support. Because it becomes the common API error, please check the correct procedure. |
| [5]  XA\_API\_FATAL\_INVALID\_CMD\_TYPE | 0xFFFF8003 | Abnormality has occurred, which disables process continuation. The subcommand was specified at the argument does not support. Because it becomes the common API error, please check the correct procedure. |
| [6]  XA\_CAP\_CONFIG\_FATAL\_STATE | 0xFFFF88C0 | Abnormality has occurred, which disables process continuation. The subcommand was specified at the argument does not support. Because it becomes the common API error, please check the correct procedure. |
| [7] XA\_CAP\_CONFIG\_FATAL\_ERR\_MONO\_24BIT | 0xFFFF88C2 | It is an error for invalid setting, 24 bit mono channel case does not supported. |
| [8]  XA\_CAP\_EXEC\_FATAL\_STATE | 0xFFFF90C0 | Abnormality has occurred, which disables process continuation. The subcommand was specified at the argument does not support. Because it becomes the common API error, please check the correct procedure. |
| [9]  XA\_CAP\_EXEC\_FATAL\_INTERNAL | 0xFFFF90C2 | Abnormality has occurred, which disables process continuation. Some of setting becomes incorrect after combination (hardware modules are not available...). Because it becomes the common API error, please check the correct parameters. |
| [10]  XA\_CAP\_EXEC\_FATAL\_HW | 0xFFFF90C1 | Abnormality has occurred, which disables process continuation. Hardware modules are not available... Because it becomes the common API error, please make sure the resource is validity. |
| [11]  XA\_CAP\_CONFIG\_NONFATAL\_ERR\_PCM\_WIDTH | 0x000008C0 | It is an error for Capture specifications out of the range.  The output buffer size was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16) |
| [12]  XA\_CAP\_CONFIG\_NONFATAL\_ERR\_CHANNELS | 0x000008C1 | It is an error for Capture specifications out of the range.  The output buffer size was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16) |
| [13]  XA\_CAP\_CONFIG\_NONFATAL\_ERR\_SAMPLE\_RATE | 0x000008C2 | It is an error for Capture specifications out of the range.  The output buffer size was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16) |
| [14]  XA\_CAP\_CONFIG\_NONFATAL\_ERR\_FRAME\_SIZE | 0x000008C3 | It is an error for Capture specifications out of the range.  The output buffer size was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16) |
| [15]  XA\_CAP\_CONFIG\_NONFATAL\_ERR\_SOURCE | 0x000008C4 | It is an error for Capture specifications out of the range.  The output buffer size was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16) |
| [16]  XA\_CAP\_CONFIG\_NONFATAL\_ERR\_DMACHANNEL | 0x000008C5 | It is an error for Capture specifications out of the range.  The output buffer size was specified at the argument does not support. Please set an appropriate value.(Refer to 2.2.2.16) |
| [17]  XA\_CAP\_CONFIG\_NONFATAL\_VOLUME\_RATE | 0x000008C6 | It is an error for Capture specification out of range.  The volume rate value was specified at the argument does not support. |
| [18] XA\_CAP\_CONFIG\_NONFATAL\_OPERATION | 0x000008C7 | It is an error for Capture specification out of range.  The setting state was specified at the argument does not support. |
| [19] XA\_CAP\_CONFIG\_FATAL\_RING\_NUM | 0xFFFF88C4 | It is an error for capture specification out of range regarding the total number of ring buffers to be used. |
| [20] | Others | Reserved |

# Processing Flow

Figure 3‑1 shows a flow diagram of processing performed by an application which uses this software. It applies for both case: renderer and capture.

The basic steps executed by the framework are shaded. The steps defined by the user framework are white. Design the process to suit the target system.

Initialize Processing

END

START

Read data from input. Indicate INPUT\_OVER if end of input.

Process data buffer

Transfer data to output

Set the I/O buffer

(Table 2-7 No.8)

Initialization

Error

Yes

No

Short of buffer size?

Set memory blocks

Get configuration parameters.

Startup API

Set the number of channels and PCM width, sample rate

Invalid parameters?

Yes

Get memory blocks info

Set the default parameters (API).

Get the size of the API structure.

No

Get version info.

Parameters Getting

Memory Allocation

Renderer/Capture Executing

Parameters Setting

Figure 3‑1 Example of the Application Processing Flow

# Appendix

This section will explain more detail about the configuration to using CTU, MIX module to perform channel transfer and mixing functions.

## CTU (Channel transfer unit)

CTU implements the channel transfer unit function. It converts PCM format from “x” channel (for input PCM of Renderer plugin) to “y” channel (expected output channel for speaker).

*(x = monaural, stereo, 4 channels, 6 channels, 8 channels; y = monaural, stereo)*

The setting of input, output channel can be referred in XA\_API\_CMD\_SET\_CONFIG\_PARAM command.

To use CTU functions, the input channel’s and output channel’s values must be different, if not, it implies that CTU module is not used, except when Renderer is using MIX function. Because CTU and MIX are electrically connected as Figure 4‑1.

There are two discrete CTU modules available. And each module includes four sub-modules. The output of these sub-modules internally connects a MIX module (see Figure 4‑1).

So, if you play two PCMs (from 2 Renderers) and mix them together, two sub-modules of CTU0 will be enabled, while CTU1 module is still available. But if you use CTU function in playing two PCMs (from 2 Renderers) separately to two different outputs, then both CTU0 and CTU1 are enabled. Note that, the index of CTU will be selected inside Renderer plugins automatically.

Figure 4‑1 shows all available of CTU modules.

CTU01

MIX0

CTU00

CTU02

CTU03

PCM data

PCM data

PCM data

PCM data

PCM data

CTU0

CTU11

MIX1

CTU10

CTU12

CTU13

PCM data

PCM data

PCM data

PCM data

PCM data

CTU1

Figure 4‑1 Block diagram of CTU

## MIX (Mixing)

MIX implements the mixing (adding) two to four streams from Renderer plugins into a single stream, which will be output to a speaker.

The setting of plugin to use MIX functions can be referred to XA\_API\_CMD\_SET\_CONFIG\_PARAM command.

As Figure 4‑1, there are two MIX modules available. And each one supports maximum of 4 inputs.

The below conditions must be assured when mixing PCMs output from Renderers.

* + - * 1. Set mix control flag
* Plugin has to set the mix control flag to assure that this stream want to mix with others.
  + - * 1. Same output destination
* All plugins have to set the same output SSI module index. Otherwise, the plugin will play independently as PCM 3 in Figure 4‑2.
  + - * 1. Input mixing available
* A Mix module supports maximum of 4 streams. So when the 5th stream performs mixing (condition (2)), the busy error of hardware will be returned.
  + - * 1. Consistent PCMs format
* The output PCM format (output channels, output sampling rate, and PCM width) has to have same configurations. Otherwise, this plugin will return an error and stop immediately.

Below are figures that present some use cases in using CTU/MIX modules in ADSP Renderer plugin.

DAC

SRC4

DVC0

SCU

SSI0

CTU01

MIX0

 Renderer

SRC3

CTU00

SRC0

CTU02

Speaker

 Renderer

 Renderer

PCM 0

PCM 1

PCM 2

 Renderer

PCM 3

DAC

SSI3

Speaker

SRC2

DVC1

Figure 4‑2 Example of using CTU/MIX for 3 streams to SSI0 and another stream to SSI3

DAC

SRC4

DVC0

SCU

SSI0

CTU01

MIX0

 Renderer

SRC3

CTU00

SRC0

CTU02

SRC2

CTU03

Speaker

 Renderer

 Renderer

 Renderer

PCM 0

PCM 1

PCM 2

PCM 3

DAC

SRC9

DVC1

SSI3

CTU11

MIX1

 Renderer

SRC6

CTU10

SRC1

CTU12

SRC5

CTU13

Speaker

 Renderer

 Renderer

 Renderer

PCM 4

PCM 5

PCM 6

PCM 7

Figure 4‑3 Example of using CTU/MIX to play 8 streams to SSI0 and SSI3.

About the volume control feature, MIX supports to control the decibel (gain level) of each input stream. From that, the output volume will be controlled by both DVC and MIX (Figure 4‑3). On the other hand, by setting MIX, user can set different volume rates for four Renderer plugins when mixing. The output sound is a combination of the inputs and their expected volumes.

## 16-bit monaural

0x12

16 bit PCM Data in SW:

16-bit PCM Data in HW:

Padded

MSB

LSB

0x12

Figure 4‑4 Data format of 16-bit monaural in SW and HW

For 16-bit monaural case, it is a difference in data size that it is in SW and HW. Therefore, it is impossible to convert between SW and HW. Then, only FIFO is used for the 16-bit monaural case. If the 1st device is assigned ADMAC to transfer data, the plugin itself automatically changes it to ADMACpp to use FIFO for data transfer.

## Conversion functions

If the conversion function is used:

* The volume is set.
* Input sampling rate and output sampling rate are different.
* Input channel and output channel are different, or MIX function is enabled (only in Renderer case).

But XA\_RDR\_CONFIG\_PARAM\_OUTPUT2, or XA\_CAP\_CONFIG\_PARAM\_INPUT2 are not set by user. Renderer/Capture plugins will automatically enable SRC module and using ADMAC transfer type to transfer data between the plugin and Audio HWs.

## State transformation

Control by user

Control by plugin

Figure 4‑5 State transformation in the plugin

Figure above shows how state is transformed in the plugin. When the current state is RESET, the plugin itself changes the state internally to RUN in the next execution.