- Table of Contents -

[1. Overview 3](#_Toc529973407)

[1.1. Overview of this document. 3](#_Toc529973408)

[1.2. The architecture of the Software and scope of this document 3](#_Toc529973409)

[1.3. Specification overview 3](#_Toc529973410)

[1.4. Memory specification 4](#_Toc529973411)

[1.5. Related documents 4](#_Toc529973412)

[2. Software Specifications 5](#_Toc529973413)

[2.1. API sepecifications 5](#_Toc529973414)

[2.2. Command list 6](#_Toc529973415)

[2.2.1. Startup API 7](#_Toc529973416)

[2.2.2. Parameters Setting 7](#_Toc529973417)

[2.2.3. Memory Allocation 8](#_Toc529973418)

[2.2.4. Initialization 8](#_Toc529973419)

[2.2.5. Parameters Getting 9](#_Toc529973420)

[2.2.6. Equalizer executing 9](#_Toc529973421)

[2.3. The list of structures 10](#_Toc529973422)

[2.4. Command function specifications 11](#_Toc529973423)

[2.4.1. XA\_API\_CMD\_GET\_LIB\_ID\_STRINGS command 11](#_Toc529973424)

[2.4.2. XA\_API\_CMD\_GET\_API\_SIZE command 13](#_Toc529973425)

[2.4.3. XA\_API\_CMD\_INIT command 14](#_Toc529973426)

[2.4.4. XA\_API\_CMD\_SET\_CONFIG\_PARAM command 18](#_Toc529973427)

[2.4.5. XA\_API\_CMD\_GET\_MEMTABS\_SIZE command 28](#_Toc529973428)

[2.4.6. XA\_API\_CMD\_SET\_MEMTABS\_PTR command 29](#_Toc529973429)

[2.4.7. XA\_API\_CMD\_GET\_N\_MEMTABS command 30](#_Toc529973430)

[2.4.8. XA\_API\_CMD\_GET\_MEM\_INFO\_SIZE command 31](#_Toc529973431)

[2.4.9. XA\_API\_CMD\_GET\_MEM\_INFO\_TYPE command 32](#_Toc529973432)

[2.4.10. XA\_API\_CMD\_GET\_MEM\_INFO\_ALIGNMENT command 33](#_Toc529973433)

[2.4.11. XA\_API\_CMD\_SET\_MEM\_PTR command 34](#_Toc529973434)

[2.4.12. XA\_API\_CMD\_GET\_CONFIG\_PARAM command 35](#_Toc529973435)

[2.4.13. XA\_API\_CMD\_SET\_INPUT\_BYTES command 45](#_Toc529973436)

[2.4.14. XA\_API\_CMD\_INPUT\_OVER command 46](#_Toc529973437)

[2.4.15. XA\_API\_CMD\_EXECUTE command 47](#_Toc529973438)

[2.4.16. XA\_API\_CMD\_GET\_OUTPUT\_BYTES command 49](#_Toc529973439)

[2.4.17. XA\_API\_CMD\_GET\_CURIDX\_INPUT\_BUF command 50](#_Toc529973440)

[2.5. Memory Specifications 51](#_Toc529973441)

[2.5.1. Persistent Area 51](#_Toc529973442)

[2.5.2. Input Buffer 52](#_Toc529973443)

[2.5.3. Output Buffer 53](#_Toc529973444)

[2.6. Structures specification 56](#_Toc529973445)

[2.6.1. API Structure 56](#_Toc529973446)

[2.6.2. Persistent Structure 56](#_Toc529973447)

[2.6.3. Equalizer settings structure 57](#_Toc529973448)

[2.6.4. Biquad filter structure 58](#_Toc529973449)

[2.7. Error processing 59](#_Toc529973450)

[3. Processing Flow 62](#_Toc529973451)

[4. Notes 64](#_Toc529973452)

[4.1. Function Call 64](#_Toc529973453)

[4.2. Other notes 64](#_Toc529973454)

[4.2.1. Allocation of memory 64](#_Toc529973455)

[4.2.2. Out of range memory access 64](#_Toc529973456)

[4.2.3. Combination with other applications 64](#_Toc529973457)

[4.2.4. Monitoring on Performance 64](#_Toc529973458)

- List of Figures -

[Figure 1‑1 The software architecture 3](#_Toc529973459)

[Figure 2‑1 API Command Sequence Overview 6](#_Toc529973460)

[Figure 2‑2 PCM 16-bit Data Access (Little Endian Mode) 54](#_Toc529973461)

[Figure 2‑3 PCM 24-bit Data Access (Little Endian Mode) 54](#_Toc529973462)

[Figure 2‑4 Input (Output) Formats 55](#_Toc529973463)

[Figure 3‑1 Application Processing Flow 63](#_Toc529973464)

- List of Tables -

[Table 1‑1 Basic Specifications 3](#_Toc529973465)

[Table 1‑2 Supported Specifications 4](#_Toc529973466)

[Table 1‑3 Memory Size Requirements 4](#_Toc529973467)

[Table 1‑4 Related documents 4](#_Toc529973468)

[Table 2‑1 API Functions of Equalizer 5](#_Toc529973469)

[Table 2‑2 Commands for Initialization 7](#_Toc529973470)

[Table 2‑3 Commands for Parameters Setting 7](#_Toc529973471)

[Table 2‑4 Commands for Initial Memory Table Allocation 8](#_Toc529973472)

[Table 2‑5 Commands for Memory Allocation 8](#_Toc529973473)

[Table 2‑6 Commands for Equalizer Initialization 8](#_Toc529973474)

[Table 2‑7 Commands for Getting Parameters 9](#_Toc529973475)

[Table 2‑8 Commands for Equalizer Execution 9](#_Toc529973476)

[Table 2‑9 Structures 10](#_Toc529973477)

[Table 2‑10 Input Buffer Description 52](#_Toc529973478)

[Table 2‑11 Output Buffer Description 53](#_Toc529973479)

# Overview

## Overview of this document.

In this chapter, overview of ADSP Equalizer plugin is explained.

## The architecture of the Software and scope of this document

The architecture of ADSP Equalizer is shown in Figure 1-1. ADSP Equalizer is an ADSP plugin which is controlled by ADSP Framework.

ARM

User Application

ADSP

Codec

DAC/ADC

ARM

Audio HW

SCU/ SSI/

ADMA

ADSP Interface

NC/EC

ADSP Driver

ADSP Framework

ADSP Reference Renderer Plugin

ADSP Reference Equalizer Plugin

User Space

Kernel Space

This document’s target is in side of red square.

Figure 1‑1 The software architecture

## Specification overview

Equalizer changes frequency characteristic of the audio signal based on the parameter that was set and performs sound quality correction. Filter processing is performed for the PCM data which were input, and coordinates ingredient of the specific frequency band.

Table 1-1 shows the basic specification and Table 1-2 shows the support specification of Equalizer.

EQZ\_FD001

Table 1‑1 Basic Specifications

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

[Covers: 96870]

EQZ\_FD002

Table 1‑2 Supported Specifications

|  |  |
| --- | --- |
| 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 |
| Filter | Direct form II(second-order IIR digital biquad filter) |
| Band number of partitions | 9 Band |
| Reentrant | Supported |
| Other | Coefficient change function |

[Covers: 82251]

## Memory specification

Table 1‑3 Memory Size Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| Memory type | Location | Memory area name | Size (in bytes) |
| Instruction | ROM | Instruction area | 16293 |
| Data | Constant table area |
| Other area( Depend on the compiler) |
| RAM | Persistent area | 596 |
| Input buffer | 8192 |
| Output buffer | 8192 |
| Scratch area | 8192 |
| Structure | 240 |
| Stack | 336 |
| 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.

## Related documents

Table 1‑4 Related documents

|  |  |  |
| --- | --- | --- |
| No. | Name | Published by |
| [1] | ADSP Framework User’s Manual | Renesas Electronics Corporation |

# Software Specifications

## API sepecifications

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

Table 2‑1 API Functions of Equalizer

|  |  |
| --- | --- |
| xa\_rel\_eqz | |
| Description | This API is the only access function to the equalizer. |
| Syntax | XA\_ERRORCODE xa\_rel\_eqz(  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 list

Using API function of the Table 2-1, it performs each processing by a combination of Command/Subcommand with overview flow chart below

Start-up API

Parameters Setting

Memory Allocation

Initialization

Done?

Parameters Getting

Equalizer Executing

Done?

Figure 2‑1 API Command Sequence Overview

### Startup API

Table 2‑2 Commands for Initialization

|  |  |  |
| --- | --- | --- |
| Upper stage : Command / lower stage :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 structure. |
| 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\_PARA  MS |

### Parameters Setting

EQZ\_FD003

Table 2‑3 Commands for Parameters Setting

|  |  |  |
| --- | --- | --- |
| Upper stage : Command / lower stage :Subcommand | | Description |
| 1 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set the center frequency of a peaking filter or transition frequency of a Bass/Treble filter for filter n. |
| XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_FC |
| 2 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set type (Peaking, Bass, Treble) for filter n. |
| XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_TYPE |
| 3 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set bandwidth for filter n. |
| XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BW |
| 4 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set gain for filter n. |
| XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_GA |
| 5 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set base gain for filter n. |
| XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BA |
| 6 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set graphic equalizer gain. |
| XA\_EQZ\_CONFIG\_PARAM\_BAND\_<n>\_GCOEF\_GA |
| 7 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set PCM data width. |
| XA\_EQZ\_CONFIG\_PARAM\_PCM\_WIDTH |
| 8 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set channel numbers. |
| XA\_EQZ\_CONFIG\_PARAM\_CH |
| 9 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Set sampling frequency. |
| XA\_EQZ\_CONFIG\_PARAM\_FS |
| 10 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Select equalizer type. |
| XA\_EQZ\_CONFIG\_PARAM\_SELECT\_EQZ\_TYPE |

[Covers: 82251]

[Note] <n> is index filter 0 to 8 for Parametric Equalizer and 0 to 4 for Graphic Equalizer. It will be specified in detailed subcommands.

### Memory Allocation

Table 2‑4 Commands for Initial Memory Table Allocation

|  |  |  |
| --- | --- | --- |
| Upper stage : Command / lower stage :Subcommand | | Description |
| 1 | XA\_API\_CMD\_GET\_MEMTABS\_SIZE | Get the size of the memory structures to be allocated for the memory 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 equalizer 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 equalizer. |
| (NULL) |

Table 2‑5 Commands for Memory Allocation

|  |  |  |
| --- | --- | --- |
| Upper stage : Command / lower stage :Subcommand | | Description |
| 1 | XA\_API\_CMD\_GET\_MEM\_INFO\_SIZE | Get the size of the each memory. |
| (NULL) |
| 2 | XA\_API\_CMD\_GET\_MEM\_INFO\_ALIGNMENT | Get the alignment information of the memory-type being referred to by the index. |
| (NULL) |
| 3 | XA\_API\_CMD\_GET\_MEM\_INFO\_TYPE | Get the type of memory being referred to by the index. |
| (NULL) |
| 4 | XA\_API\_CMD\_SET\_MEM\_PTR | Set the pointer to the memory allocated for the referred index to the input value. |
| (NULL) |

### Initialization

Table 2‑6 Commands for Equalizer Initialization

|  |  |  |
| --- | --- | --- |
| Upper stage : Command / lower stage :Subcommand | | Description |
| 1 | XA\_API\_CMD\_INPUT\_OVER | Signal the end of bit stream to the library. |
| (NULL) |
| 2 | XA\_API\_CMD\_SET\_INPUT\_BYTES | Set the number of bytes available in the input buffer for initialization. |
| (NULL) |
| 3 | XA\_API\_CMD\_INIT | Initialize state and start run-time data process |
| 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 by the last initialization. |
| (NULL) |

### Parameters Getting

Table 2‑7 Commands for Getting Parameters

|  |  |  |
| --- | --- | --- |
| Upper stage : Command / lower stage :Subcommand | | Description |
| 1 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the center frequency of a peaking filter or transition frequency of a Bass/Treble filter for filter n. |
| XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_FC |
| 2 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get type (Peaking, Bass, Treble) for filter n. |
| XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_TYPE |
| 3 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get bandwidth for filter n. |
| XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BW |
| 4 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get gain for filter n. |
| XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_GA |
| 5 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get base gain for filter n. |
| XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BA |
| 6 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get graphic equalizer gain. |
| XA\_EQZ\_CONFIG\_PARAM\_BAND\_<n>\_GCOEF\_GA |
| 7 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get PCM data width. |
| XA\_EQZ\_CONFIG\_PARAM\_PCM\_WIDTH |
| 8 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get channel numbers. |
| XA\_EQZ\_CONFIG\_PARAM\_CH |
| 9 | XA\_API\_CMD\_GET\_CONFIG\_PARAM | Get the sampling frequency of input signal. |
| XA\_EQZ\_CONFIG\_PARAM\_FS |
| 10 | XA\_API\_CMD\_SET\_CONFIG\_PARAM | Get equalizer type. |
| XA\_EQZ\_CONFIG\_PARAM\_SELECT\_EQZ\_TYPE |

### Equalizer executing

Table 2‑8 Commands for Equalizer Execution

|  |  |  |
| --- | --- | --- |
| Upper stage : Command / lower stage :Subcommand | | Description |
| 1 | XA\_API\_CMD\_INPUT\_OVER | Signal the end of bit stream to the library. |
| (NULL) |
| 2 | XA\_API\_CMD\_SET\_INPUT\_BYTES | Set the number of bytes available in the input buffer for the execution. |
| (NULL) |
| 3 | XA\_API\_CMD\_EXECUTE | This command executes the equalizer. |
| XA\_CMD\_TYPE\_DO\_EXECUTE |
| 4 | XA\_API\_CMD\_EXECUTE | Check if the end of processing has been reached. |
| XA\_CMD\_TYPE\_DONE\_QUERY |
| 5 | XA\_API\_CMD\_GET\_OUTPUT\_BYTES | Get the number of bytes output by the equalizer in the last frame. |
| (NULL) |
| 6 | XA\_API\_CMD\_GET\_CURIDX\_INPUT\_BUF | Get the number of input buffer bytes consumed by the last call to equalizer. |
| (NULL) |

## The list of structures

Table 2-9 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 2.5

Table 2‑9 Structures

|  |  |
| --- | --- |
| Structure name | Outline |
| API structure | Store the information of API. |
| Parametric Equalizer settings structure | Store a parameter to calculate a necessary filter coefficient as Parametric Equalizer. |
| Graphic Equalizer settings structure | Store a parameter to calculate a necessary filter coefficient as Graphic Equalizer. |
| Equalizer settings structure | Store the parameters necessary for equalizer. |
| Biquad filter structure | Store the parameters necessary for Biquad filter. |

## Command function specifications

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

FD\_PLG\_EQZ\_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. | |
| Parameter | 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 | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - pv\_value is NULL |

[Covers: RD\_013]

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. | |
| Parameter | 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 | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - pv\_value is NULL |

[Covers: RD\_013]

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\_EQZ\_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. | |
| Parameter | 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 | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - pv\_value is NULL |
| Restrictions | The application shall allocate memory with an alignment of 4 bytes. | |

[Covers: RD\_013]

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. | |
| Parameter | 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 | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes. |

FD\_PLG\_EQZ\_003

[Covers: RD\_013]

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 equalizer specific parameters have been set. | |
| Parameter | 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 | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_004

[Covers: RD\_013]

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 | This command initializes the equalizer. It kicks run-time initialization process. | |
| Parameter | 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 | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy.  XA\_EQZ\_EXEC\_FATAL\_STATE – if input, output and scratch memory are not allocated |

FD\_PLG\_EQZ\_005

[Covers: RD\_013]

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. | |
| Parameter | 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 | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj, pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_006

[Covers: RD\_013]

Example

WORD32 done;

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

XA\_API\_CMD\_INIT,

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

&done);

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

[Note] <n> is index filter 0, 1, 2... 8. It will be specified in detailed subcommands.

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_FC | |
| Description | This command is used to set the center frequency of a peaking filter or transition frequency of a Bass/Treble filter for filter n. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_SET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_FC |
| pv\_value | Address that stored center/ transition frequency.  Value range:   * Peaking filter: 20-20kHz (or less than Nyquist frequency) * Bass filter: 50-500Hz * Treble filter: 5k - 11kHz |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy.  XA\_EQZ\_CONFIG\_NONFATAL\_ERR\_FC- Outside range for center/ transition frequency. |

FD\_PLG\_EQZ\_007

[Covers: RD\_013]

Example

WORD32 coef\_fc;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_FC,

&coef\_fc);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_TYPE | |
| Description | This command is used to set type (Peaking, Bass, Treble, Through) for filter n. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_SET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_TYPE |
| pv\_value | Address that stored filter type.  Value range: Peaking, Bass, Treble, Through filter  Through is default filter type. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy.  XA\_EQZ\_CONFIG\_NONFATAL\_ERR\_TYPE - Outside range for filter type. |

FD\_PLG\_EQZ\_008

[Covers: RD\_013]

Example

WORD32 coef\_type;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_TYPE,

&coef\_type);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BW | |
| Description | This command is used to set bandwidth for filter n. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_SET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BW |
| pv\_value | Address that stored filter bandwidth.  Value range: to (fixed point Q5.27) |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy.  XA\_EQZ\_CONFIG\_NONFATAL\_ERR\_BW - Outside range for filter bandwidth. |

FD\_PLG\_EQZ\_009

[Covers: RD\_013]

Example

WORD32 coef\_bw;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BW,

&coef\_bw);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_GA | |
| Description | This command is used to set gain for filter n. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_SET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_GA |
| pv\_value | Address that stored filter gain.  Value range: to (fixed point Q4.28) |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy.  XA\_EQZ\_CONFIG\_NONFATAL\_ERR\_GA - Outside range for filter gain. |

FD\_PLG\_EQZ\_010

[Note] Value range: -15dB to 15dB (fixed point Q4.28)

-15dB:

15dB:

[Covers: RD\_013]

Example

WORD32 coef\_ga;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_GA,

&coef\_ga);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BA | |
| Description | This command is used to set base gain for filter n. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_SET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BA |
| pv\_value | Address that stored filter base gain.  Value range: to (fixed point Q4.28) |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy.  XA\_EQZ\_CONFIG\_NONFATAL\_ERR\_BA - Outside range for filter base gain. |

FD\_PLG\_EQZ\_011

[Covers: RD\_013]

Example

WORD32 coef\_ba;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BA,

&coef\_ba);

[Note] <m> is index filter 0, 1, 2... 4. It will be specified in detailed subcommands.

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_<m>\_GCOEF\_GA | |
| Description | This command is used to set graphic equalizer gain. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_SET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_BAND\_<n>\_GCOEF\_GA |
| pv\_value | Address that stored graphic equalizer gain.  Value range: to (fixed point Q4.28) |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy.  XA\_EQZ\_CONFIG\_NONFATAL\_ERR\_GA - Outside range for filter gain. |

FD\_PLG\_EQZ\_012

[Covers: RD\_013]

Example

WORD32 gcoef\_ga;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_GCOEF\_GA,

&gcoef\_ga);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_PCM\_WIDTH | |
| Description | This command is used to set PCM data width. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_SET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_PCM\_WIDTH |
| pv\_value | Address that stored PCM data width.  Value range: 16 bits /24 bits |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy.  XA\_EQZ\_CONFIG\_FATAL\_ERR\_PCM\_WIDTH – Outside range for PCM width. |

FD\_PLG\_EQZ\_013

[Covers: RD\_013]

Example

WORD32 pcm\_width;

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

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

XA\_EQZ\_CONFIG\_PARAM\_PCM\_WIDTH,

&pcm\_width);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_CH | |
| Description | This command is used to set channel numbers. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_SET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_CH |
| pv\_value | Address that stored channel numbers.  Value range: 1 or 2 |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy.  XA\_EQZ\_CONFIG\_FATAL\_ERR\_CH - Outside range for channel numbers. |

FD\_PLG\_EQZ\_014

[Covers: RD\_013]

Example

WORD32 ch;

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

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

XA\_EQZ\_CONFIG\_PARAM\_PARAM\_CH,

&ch);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FS | |
| Description | This command is used to set the sampling frequency of input signal. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_SET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FS |
| pv\_value | Address that stored sampling frequency.  Value range: 48kHz / 44.1kHz / 32kHz |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_INVALID\_CMD\_TYPE - i\_idx is incorrect.  XA\_EQZ\_CONFIG\_FATAL\_ERR\_FS - Outside range for sampling frequency. |

FD\_PLG\_EQZ\_015

[Covers: RD\_013]

Example

WORD32 param\_fs;

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

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

XA\_EQZ\_CONFIG\_PARAM\_PARAM\_FS,

&param\_fs);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_SELECT\_EQZ\_TYPE | |
| Description | This command is used to select Equalizer type which is Parametric Equalizer or Graphic Equalizer. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_SET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_SELECT\_EQZ\_TYPE |
| pv\_value | Address that stored index of Equalizer type.  Value range:   * Parametric Equalizer: 0 * Graphic Equalizer: 1 |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_INVALID\_CMD\_TYPE - i\_idx is incorrect.  XA\_EQZ\_CONFIG\_NONFATAL\_ERR\_SELECT\_EQZ\_TYPE – Equalizer type does not support. |

FD\_PLG\_EQZ\_016

[Covers: RD\_013]

Example

WORD32 eqz\_type;

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

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

XA\_EQZ\_CONFIG\_PARAM\_EQZ\_TYPE,

&eqz\_type);

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

FD\_PLG\_EQZ\_017

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | This command is used to obtain the size of the table used to hold memory blocks. These blocks required for the equalizer operation. The API returns the total size of the required table. | |
| Parameter | 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 | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - d is not aligned to 4 bytes  XA\_EQZ\_CONFIG\_FATAL\_STATE – Precondition is incorrect. |

[Covers: RD\_013]

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\_EQZ\_018

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | This command is used to set the memory structure pointer to the allocated value. | |
| Parameter | 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 | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d or pv\_value is not aligned to 4 bytes  XA\_EQZ\_CONFIG\_FATAL\_STATE – Precondition is incorrect. |

[Covers: RD\_013]

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\_EQZ\_019

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | This command is used to obtain the number of memory blocks the equalizer needs. 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. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_N\_MEMTABS |
| i\_idx | NULL |
| pv\_value | Number of memory blocks required to be allocated. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes. |

[Covers: RD\_013]

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\_EQZ\_020

|  |  |  |
| --- | --- | --- |
| 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. | |
| Parameter | 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  2 - Output Buffer  3 – Scratch Area |
| pv\_value | Pointer to memory size. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL. XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy.  XA\_API\_FATAL\_INVALID\_CMD\_TYPE - i\_idx is an invalid memory block number. |

[Covers: RD\_013]

Example

WORD32 mem\_size;

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

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

index,

&mem\_size);

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

FD\_PLG\_EQZ\_021

|  |  |  |
| --- | --- | --- |
| Subcommand | Memory index | |
| Description | This command gets the type of memory being referred to by the index. | |
| Parameter | 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  2 - Output Buffer  3 – Scratch Area |
| pv\_value | Pointer to memory type variable. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_API\_FATAL\_INVALID\_CMD\_TYPE - i\_idx is an invalid memory block number. |

[Covers: RD\_013]

Example

WORD32 mem\_type;

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

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

index,

&mem\_type);

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

FD\_PLG\_EQZ\_022

|  |  |  |
| --- | --- | --- |
| Subcommand | Memory index | |
| Description | This command gets the alignment information of the memory-type referred to by the index. The alignment required in bytes is returned to the application. | |
| Parameter | 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  2 - Output Buffer  3 – Scratch Area |
| pv\_value | Pointer to the alignment info variable. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_API\_FATAL\_INVALID\_CMD\_TYPE - i\_idx is an invalid memory index. |

[Covers: RD\_013]

Example

WORD32 mem\_align;

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

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

index,

&mem\_align);

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

FD\_PLG\_EQZ\_023

|  |  |  |
| --- | --- | --- |
| Subcommand | Memory index | |
| Description | This command passes to the equalizer the pointer to the allocated memory. | |
| Parameter | 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  2 - Output Buffer  3 – Scratch Area |
| pv\_value | Pointer to memory buffer allocated. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL. XA\_API\_FATAL\_INVALID\_CMD\_TYPE - i\_idx is an invalid memory block number.  XA\_API\_FATAL\_MEM\_ALIGN – d or pv\_value is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

[Covers: RD\_013]

Example

pVOID addr;

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

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

index,

addr);

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

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_PCM\_WIDTH | |
| Description | This command is used to get PCM width parameter. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_PCM\_WIDTH |
| pv\_value | Pointer to PCM width. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_024

[Covers: RD\_013]

Example

WORD32 pcm\_width;

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

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

XA\_EQZ\_CONFIG\_PARAM\_PCM\_WIDTH,

&pcm\_width);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_CH | |
| Description | This command is used to get PCM width parameter. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_CH |
| pv\_value | Pointer to channel numbers. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_025

[Covers: RD\_013]

Example

WORD32 ch;

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

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

XA\_EQZ\_CONFIG\_PARAM\_CH,

&ch);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_FC | |
| Description | This command is used to get the center frequency of a peaking filter or transition frequency of a Bass/Treble filter for filter n. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_FC |
| pv\_value | Pointer to center / transition frequency |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_026

[Covers: RD\_013]

Example

WORD32 coef\_fc;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_FC,

&coef\_fc);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_TYPE | |
| Description | This command is used to get type (Peaking, Bass, Treble, Through) for filter n. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_TYPE |
| pv\_value | Pointer to filter type. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_027

[Covers: RD\_013]

Example

WORD32 coef\_type;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_TYPE,

&coef\_type);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BW | |
| Description | This command is used to get bandwidth for filter n. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BW |
| pv\_value | Pointer to filter bandwidth. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_028

[Covers: RD\_013]

Example

WORD32 coef\_bw;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BW,

&coef\_bw);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_GA | |
| Description | This command is used to get gain for filter n. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_GA |
| pv\_value | Pointer to filter gain. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_029

[Covers: RD\_013]

Example

WORD32 coef\_ga;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_GA,

&coef\_ga);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BA | |
| Description | This command is used to get base gain for filter n. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BA |
| pv\_value | Pointer to filter base gain. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_030

[Covers: RD\_013]

Example

WORD32 coef\_ba;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_COEF\_BA,

&coef\_ba);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_BAND\_<n>\_GCOEF\_GA | |
| Description | This command is used to get graphic equalizer gain. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_BAND\_<n>\_GCOEF\_GA |
| pv\_value | Pointer to graphic equalizer gain. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_031

[Covers: RD\_013]

Example

WORD32 gcoef\_ga;

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

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

XA\_EQZ\_CONFIG\_PARAM\_FILTER\_<n>\_GCOEF\_GA,

&gcoef\_ga);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_FS | |
| Description | This command is used to get the sampling frequency. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_FS |
| pv\_value | Pointer to sampling frequency. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_032

[Covers: RD\_013]

Example

WORD32 param\_fs;

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

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

XA\_EQZ\_CONFIG\_PARAM\_PARAM\_FS,

&param\_fs);

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_EQZ\_CONFIG\_PARAM\_SELECT\_EQZ\_TYPE | |
| Description | This command is used to select Equalizer type which is Parametric Equalizer or Graphic Equalizer. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_SET\_CONFIG\_PARAM |
| i\_idx | XA\_EQZ\_CONFIG\_PARAM\_SELECT\_EQZ\_TYPE |
| pv\_value | Address that stored index of Equalizer type.  Value range:   * Parametric Equalizer: 0 * Graphic Equalizer: 1 |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_INVALID\_CMD\_TYPE - i\_idx is incorrect.  XA\_EQZ\_CONFIG\_NONFATAL\_ERR\_SELECT\_EQZ\_TYPE – Equalizer type does not support.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_CONFIG\_FATAL\_STATE – precondition does not satisfy. |

FD\_PLG\_EQZ\_033

[Covers: RD\_013]

Example

WORD32 eqz\_type;

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

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

XA\_EQZ\_CONFIG\_PARAM\_EQZ\_TYPE,

&eqz\_type);

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

FD\_PLG\_EQZ\_034

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | This command is used to set the number of bytes available in the input buffer for the execution. | |
| Parameter | 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. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - The pointer is incorrectly aligned to the requirements.  XA\_EQZ\_EXEC\_FATAL\_STATE – Precondition is incorrect.  XA\_EQZ\_EXEC\_FATAL\_INPUT – pv\_value is invalid |

[Covers: RD\_013]

Example

WORD32 filled;

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

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

0,

&filled);

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

FD\_PLG\_EQZ\_035

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | This command is used to tell the equalizer that the end of the input data has been reached. This situation can arise both in the initialization loop and the execute loop. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_INPUT\_OVER |
| i\_idx | NULL |
| pv\_value | NULL |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - The pointer is incorrectly aligned to the requirements. |

[Covers: RD\_013]

Example

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

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

0,

NULL);

### XA\_API\_CMD\_EXECUTE command

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_CMD\_TYPE\_DO\_EXECUTE | |
| Description | This command executes the equalizer. | |
| Parameter | 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 | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_EXEC\_FATAL\_STATE – Precondition is incorrect. |

FD\_PLG\_EQZ\_036

[Covers: RD\_013]

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 equalizer can continue for several invocations of the DO\_EXECUTE command after the last input data has been passed to the equalizer, so the application should not assume that the equalizer has finished generating all its output until so indicated by this command. | |
| Parameter | 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 | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN – d is not aligned to 4 bytes.  XA\_EQZ\_EXEC\_FATAL\_STATE – Precondition is incorrect. |

FD\_PLG\_EQZ\_037

[Covers: RD\_013]

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\_EQZ\_038

|  |  |  |
| --- | --- | --- |
| Subcommand | None | |
| Description | This command obtains the number of bytes output by the equalizer during the last execution. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_OUTPUT\_BYTES |
| i\_idx | NULL |
| pv\_value | Pointer to output bytes variable. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - The pointer is incorrectly aligned to the requirements.  XA\_EQZ\_EXEC\_FATAL\_STATE – Precondition is incorrect. |

[Covers: RD\_013]

Example

WORD32 produced;

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

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

0,

&produced);

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

|  |  |  |
| --- | --- | --- |
| Subcommand | XA\_API\_CMD\_GET\_CURIDX\_INPUT\_BUF | |
| Description | This command gets the number of input buffer bytes consumed by the equalizer. It is used both in the initialization loop and execute loop. | |
| Parameter | p\_xa\_module\_obj | Pointer to API Structure. |
| i\_cmd | XA\_API\_CMD\_GET\_CURIDX\_INPUT\_BUF |
| i\_idx | NULL |
| pv\_value | Pointer to bytes consumed variable. |
| Return value | Normal | XA\_NO\_ERROR |
| Error | XA\_API\_FATAL\_MEM\_ALLOC - p\_xa\_module\_obj or pv\_value or d->pMem\_tabs->pInput is NULL.  XA\_API\_FATAL\_MEM\_ALIGN - The pointer is incorrectly aligned to the requirements. |

FD\_PLG\_EQZ\_039

[Covers: RD\_013]

Example

WORD32 consumed;

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

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

0,

&consumed);

## Memory Specifications

This section describes the memory areas used by this software.

### Persistent Area

|  |  |
| --- | --- |
| 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 2.4.8. |
| Area reservation | The user should reserve this area. |
| Allocation | This area is included in RAM. |
| Alignment | Align this area on a 4-byte boundary. |

### Input Buffer

Table 2‑10 Input Buffer Description

|  |  |
| --- | --- |
| Item | Area which stores inputs to this software.  The input buffer contains 16/24 bit linear PCM data.  If the user manipulates this area during equalize processing, the normal execution of the program cannot be ensured.  [Note] This software does not support an input buffer which is a circular buffer. |
| Symbol name | - (freely defined by the user) |
| Size | Please secure more than size with 2.4.8. |
| Area reservation | The user should reserve this area.  The user can freely use this area after the equalizing of one block. |
| Allocation | This area is included in RAM. |
| Alignment | Align this area on a 4-byte boundary. |

### Output Buffer

Table 2‑11 Output Buffer Description

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

* + - * 1. Input and output data storage method

Data is in the formats as shown in Figure 2‑4(consecutive buffers are specified for the channels). The input and output buffer (memory) store 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 2‑2PCM 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)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Input (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 Input (Output) Format | | | |

Figure 2‑4 Input (Output) Formats

## Structures specification

### API Structure

Name of structure type: XARelEqz

Structure Members:

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| WORD32 | iState | State of API |
| pVOID | pMem\_tabs | Memory table controller |
| pVOID | iChannels | Number of input channels |
| WORD32 | iPcm\_width | Width of PCM data |
| WORD32 | iEqz\_type | Number of input channels |
| WORD32 | iFs | Type of Equalizer |
| WORD32 | iSample\_size | Sample size |
| WORD32 | iSubmitted | Total submitted samples |
| WORD32 | iConsumed | Current consumed samples |
| WORD32 | iProduced | Produced output samples |
| releqz\_setParametricEqualizerCoef | stEqCoef | Parametric equalizer coefficient |
| releqz\_setGraphicEqualizerCoef | stEqGCoef | Graphic equalizer coefficient |

### Persistent Structure

Name of structure type: releqz\_Persistent

Function: It stores equalizer calculation data buffer.

Structure Members:

|  |  |  |
| --- | --- | --- |
| Name | | Description |
| Equalizer | \*stEq | Equalizer calculation data buffer |

### Equalizer settings structure

Structure name: Equalizer

Function: It stores the parameters necessary for Equalizer object.

Structure Members:

|  |  |  |
| --- | --- | --- |
| Name | | Description |
| Biquad | biquad[EQ\_FILTER\_N] | Biquad filter |
| WORD32 | coef[EQ\_FILTER\_N][BQ\_COEF\_N] | Coefficients for each Biquad |
| WORD32 | blockSize | Block size of one equalizer processing |
| WORD32 | sampleStep | Sample step |

### Biquad filter structure

Structure name: Biquad

Function: It stores the parameters necessary for Biquad filter object.

Structure Members:

|  |  |  |
| --- | --- | --- |
| Name | | Description |
| WORD32 | delay[2] | Store data delay time. |
| WORD32 | \*coef | Pointer to biquad filter coefficient. |

## Error processing

|  |  |  |  |
| --- | --- | --- | --- |
| Status code | Error code (32bit) | Value | Description |
| Normal | XA\_NO\_ERROR | 0x00000000 | The processing results are normal.  The process has terminated normally. |
| Error | 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. |
| Error | 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. |
| Error | 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. |
| Error | 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. |
| Error | XA\_EQZ\_CONFIG\_FATAL\_ERR\_CH | 0xFFFF0800 | It is an error for equalizer specifications out of the range.  The number of the channels was specified at the argument does not support. Please set a right value. |
| Error | XA\_EQZ\_CONFIG\_PARAM\_ERR\_FS | 0xFFFF0801 | It is an error for equalizer specifications out of the range.  The number of the sampling frequency was specified at the argument does not support. Please set a right value. |
| Error | XA\_EQZ\_CONFIG\_PARAM\_ERR\_SIZE | 0xFFFF0802 | It is an error for equalizer specifications out of the range.  The Input or output buffer size was specified at the argument does not support. Please set an appropriate value. Refer 2.5.2 and 2.5.3. |
| Error | XA\_EQZ\_CONFIG\_PARAM\_ERR\_FC | 0x00000803 | It is an error for equalizer specifications out of the range.  The number of the center/ transition frequency was specified at the argument does not support. Please set a right value. |
| Error | XA\_EQZ\_CONFIG\_PARAM\_ERR\_GA | 0x00000804 | It is an error for equalizer specifications out of the range.  The number of the filter gain was specified at the argument does not support. Please set a right value. |
| Error | XA\_EQZ\_CONFIG\_PARAM\_ERR\_BA | 0x00000805 | It is an error for equalizer specifications out of the range.  The number of the filter base gain was specified at the argument does not support. Please set a right value. |
| Error | XA\_EQZ\_CONFIG\_PARAM\_ERR\_BW | 0x00000806 | It is an error for equalizer specifications out of the range.  The number of the filter bandwidth was specified at the argument does not support. Please set a right value. |
| Error | XA\_EQZ\_CONFIG\_PARAM\_ERR\_TYPE | 0x00000807 | It is an error for equalizer specifications out of the range.  The number of the filter type was specified at the argument does not support. Please set a right value. |
| Error | XA\_EQZ\_CONFIG\_PARAM\_ERR\_PCM\_WIDTH | 0xFFFF0808 | It is an error when setting PCM width of input data. This value was specified at the argument does not support. Please set a right value. |
| Error | XA\_EQZ\_CONFIG\_PARAM\_ERR\_OVERWRITE | 0x00000809 | When setting Parametric and Graphic Equalizer currently. The result will be overwritten. |
| Error | XA\_EQZ\_CONFIG\_PARAM\_ERR\_SELECT\_EQZ\_TYPE | 0x00000810 | It is an error when select equalizer type. This value was specified at the argument does not support. Please set a right value. |
| Error | XA\_EQZ\_CONFIG\_FATAL\_STATE | 0x00000811 | It is an error when precondition does not satisfy. |

# Processing Flow

Figure 3.1 shows a flow diagram of processing performed by an application which uses this software. These steps are grouped into 6 stages: Startup API, Parameters Setting, Memory Allocation, Initialization, Parameter Getting and Equalizer Executing.   
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.

[Note] **Set the equalizer parameter** phase performs for only Parametric Equalizer or Graphic Equalizer. If we set them currently it will return error code: (XA\_EQZ\_CONFIG\_NONFATAL\_ERR\_OVERWRITE).

Clear of the filter buffer.

END

START

Set the each Info.

(Table 2-4, 2-5)

Equalizer

Set the I/O buffer

(Table 2-6)

Initialization

Error

Yes

No

short of buffer size ?

Get the size of the each memory.

Get configuration parameters.

Startup API

Set the number of channels and PCM width

invalid channels or PCM width?

Yes

Set the number of blocks.

Set the default parameters (API).

Get the size of the API structure.

Invalid setting?

Yes

No

No

Set the equalizer parameter.

Get version info.

Parameters Getting

Memory Allocation

Equalizer Executing

Parameters Setting

Figure 3‑1 Application Processing Flow

# Notes

This section describes the notice of developing user programs.

## Function Call

User programs which call the functions in this specification should obey the calling rules of compiler.

## Other notes

### Allocation of memory

Before calling a function in this software, reserve a persistent area, an input/output buffer area, and areas for structures which should hold the arguments of functions.

### Out of range memory access

The functions in this specification never access out of allocated memory or related I/O.

### Combination with other applications

Take care not to duplicate symbol names when other applications are combined with other programs.

### Monitoring on Performance

The products embedding this Software shall observe performance of the Software periodically with Watch Dog timer or such functions in order not to damage system performance.