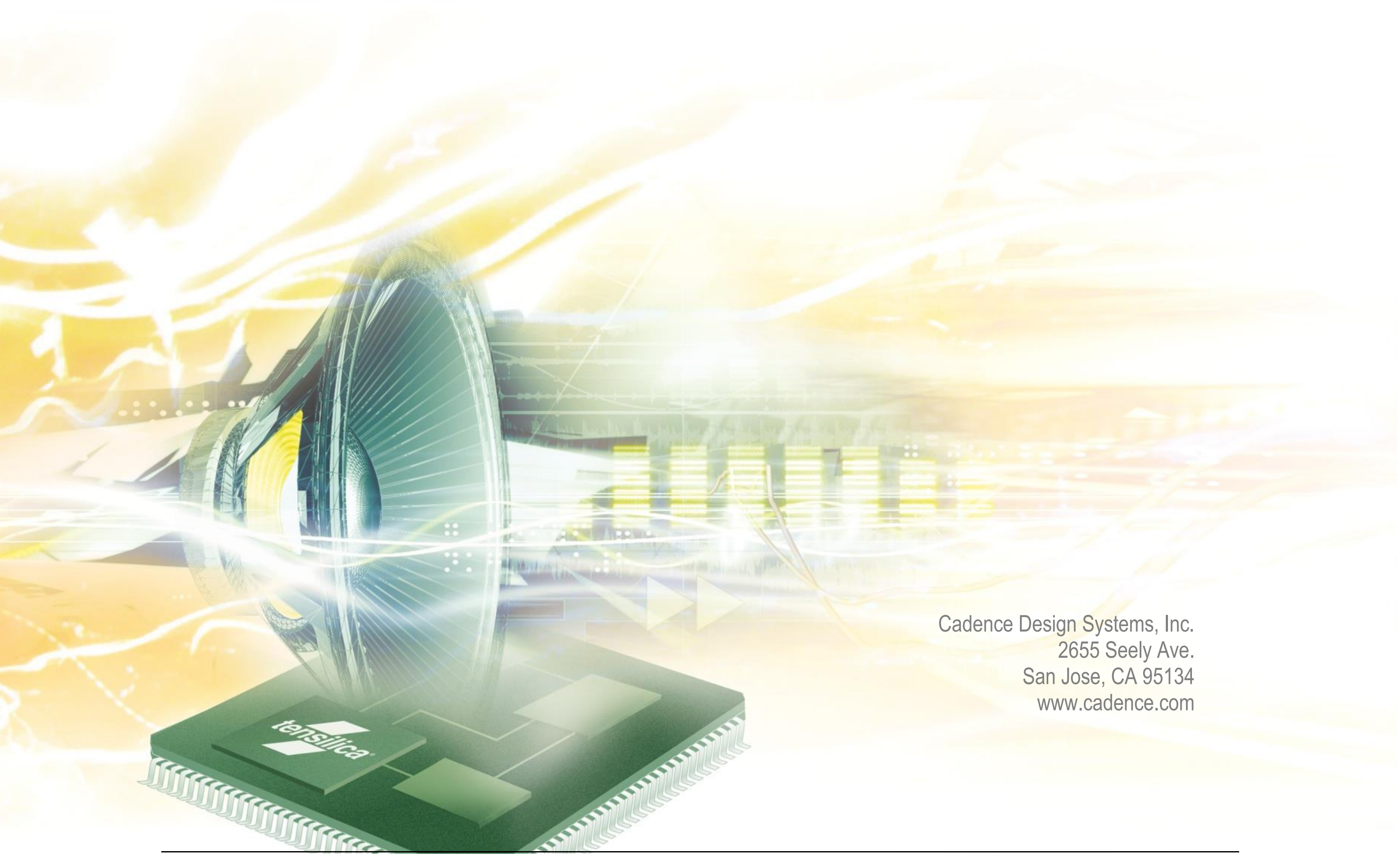
cādence®

DAB+ Decoder

Programmer's Guide

For HiFi DSPs and Fusion F1 DSP



Cadence Design Systems, Inc.
2655 Seely Ave.
San Jose, CA 95134
www.cadence.com

Version 2.8
August 2020

Contents

Figures

Tables

Document Change History

Version	Changes
2.6	■ Added APIs to make the attenuation ramp adjustable for concealment fade-out and fade-in and also to configure the number of AUs to switch from mute to fade-in ■ Added a Section 3.4 for explaining the concealment mechanism ■ Added missing documentation for **`mpeg_surr_present`** ■ Added performance data for HiFi 4 in Section 1.4.
2.7	■ Added performance data for HiFi 3z and Fusion F1 in Section 1.4.
2.8	■ Added performance data for HiFi 5 in Section 1.4.

1. Introduction to the HiFi DAB+ Decoder

The HiFi DSP DAB+ Decoder implements the HE AAC v2 audio decoder for Eureka-147 Digital Audio Broadcasting (DAB) as specified in ETSI TS 102 563 V1.1.1 [1].

For this document, HiFi DSPs include Fusion F1 DSP.

1.1 DAB+ Description

The Digital Audio Broadcasting (DAB) (or Eureka 147) standard is a digital radio technology for broadcasting radio stations designed in the 1980s. DAB is based on the MPEG-1 Layer II (MP2) audio codec. DAB+ is a non-backward-compatible upgrade of DAB released in 2007 based on the aacPlus audio codec. Compared to DAB, DAB+ provides higher audio and reception quality as well as improved bandwidth.

1.2 Document Overview

This document covers all the information required to integrate the HiFi Audio Codecs into an application. The HiFi codec libraries implement a simple API to encapsulate the complexities of the coding operations and simplify the application and system implementation. Parts of the API are common to all the HiFi codecs and these are described after the introduction. The next section then covers all the features and information particular to the HiFi DAB+ Decoder. Finally, the example test bench is described.

1.3 HiFi DAB+ Decoder Specifications

The HiFi DSP DAB+ Decoder from Cadence implements the following features:

- Cadence Audio Codec API is used
- An ETSI TS 102 563 V1.1.1 [1] compliant decoder with support for SBR and Parametric Stereo
- Audio super-frame and raw transport protocols
- Sampling frequencies: 32 and 48 kHz
- Data rates: from 8 to 192 kbps

- Mono and stereo channels
- Output PCM sample size: 16 bits or 24 bits (aligned in the 24 MSBs of a 32-bit word)
- Error concealment
- Program Associated Data (PAD) extraction
- Implicit and explicit SBR signaling
- Optional downsampling of SBR output
- Optional mono downmix for SBR processing

The decoder was tested against Dolby Laboratories' DAB+ Decoder Certification Package Version 1.0 dated 28 May 2008.

1.4 HiFi2 DAB+ Decoder Performance

The HiFi DSP DAB+ Decoders from Cadence were characterized on the HiFi 5-stage DSP. The memory usage and performance figures are provided for design reference.

- The API structure sizes returned by XA_API_CMD_GET_API_SIZE is approximately 180 bytes.
- The memory table structure size returned by XA_API_CMD_GET_MEMTABS_SIZE is approximately 150 bytes.

1.4.1 Memory

Text (Kbytes)							Data
HiFi Mini	HiFi 2	Fusion F1	HiFi 3	HiFi 3z	HiFi 4	HiFi 5	Kbytes
72.3	80.3	79.0	81.6	85.8	88.3	113.5	42.8

Runtime Memory (Kbytes)				
Persistent	Scratch	Stack	Input	Output
36.6	21.0	1.5	2.6	16.0

1.4.2 Timings

Rate kHz	Nch	Bit Rate kbps	Average CPU Load (MHz)							Stream
			HiFi Mini	HiFi 2	Fusion F1	HiFi 3	HiFi 3z	HiFi 4	HiFi 5	
48	2	128	11.8	11.6	10.4	10.7	9.1	8.6	8.6	plain AAC
48	2	48	21.0	20.9	19.0	18.0	15.8	14.3	14.5	SBR
48	2	32	21.0	20.5	18.7	17.8	15.7	14.2	14.4	PS

Note Performance specification measurements are carried on a cycle-accurate simulator assuming an ideal memory system, *i.e.*, one with zero memory wait states. This is equivalent to running with all code and data in local memories or using an infinite-size, pre-filled cache model.

Note The performance numbers for HiFi Mini/HiFi 3/HiFi 3z/HiFi 4/HiFi 5/Fusion F1 are obtained by running the test that is recompiled from the HiFi 2 source code in the HiFi Mini/HiFi 3/HiFi 3 z /HiFi 4/HiFi 5/Fusion F1 configuration. No specific optimization is done for HiFi Mini/HiFi 3/HiFi 3 z /HiFi 4/HiFi 5/Fusion F1.

2. Generic HiFi Audio Codec API

This chapter describes the API that is common to all the HiFi audio codec libraries. The API facilitates any codec that works in the overall method shown in the following diagram.

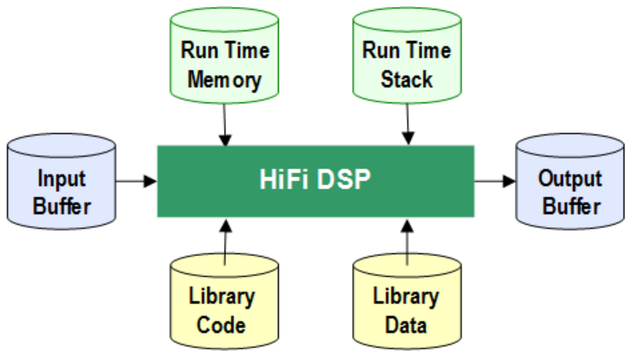


Figure 1 HiFi Audio Codec Interfaces

Section 2.1 discusses all the types of run time memory required by the codecs. There is no state information held in static memory, therefore a single thread can perform time division processing of multiple codecs. Additionally, multiple threads can perform concurrent codec processing. The API is implemented so that the application does not need to consider the codec implementation.

Through the API, the codec requests the minimum sizes required for the input and output buffers. Prior to executing the codec execution command, the codec requires that the input buffer is filled with data up to the minimum size for the input buffer. However, the codec may not consume all of the data in the input buffer. Therefore, the application must check the amount of input data consumed, copy downwards any unused portion of the input buffer, and then continue to fill the rest of the buffer with new data until the input buffer is again filled to the minimum size. The codec will produce data in the output buffer. The output data must be removed from the output buffer after the codec operation.

Applications that use these libraries should not make any assumptions about the size of the PCM “chunks” of data that each call to a codec produces or consumes. Although normally the chunks are the exact size of the underlying frame of the specified codec algorithm, they will vary between codecs and also between different operating modes of the same codec. The application should provide enough data to fill the input buffer. However, some codecs do provide information, after the initialization stage, to adjust the number of bytes of PCM data they need.

2.1 Memory Management

The HiFi audio codec API supports a flexible memory scheme and a simple interface that eases the integration into the final application. The API allows the codecs to request the required memory for their operations during run time.

The run time memory requirement consists primarily of the scratch and persistent memory. The codecs also require an input buffer and output buffer for the passing of data into and out of the codec.

2.1.1 API Object

The codec API stores its data in a small structure that is passed via a handle that is a pointer to an opaque object from the application for each API call. All state information and the memory tables that the codec requires are referenced from this structure.

2.1.2 API Memory Table

During the memory allocation, the application is prompted to allocate memory for each of the following memory areas. The reference pointer to each memory area is stored in this memory table. The reference to the table is stored in the API object.

2.1.3 Persistent Memory

This is also known as static or context memory. This is the state or history information that is maintained from one codec invocation to the next within the same thread or instance. The codecs expect that the contents of the persistent memory be unchanged by the system apart from the codec library itself for the complete lifetime of the codec operation.

2.1.4 Scratch Memory

This is the temporary buffer used by the codec for processing. The contents of this memory region should be unchanged if the actual codec execution process is active, that is, if the thread running the codec is inside any API call. This region can be used freely by the system between successive calls to the codec.

2.1.5 Input Buffer

This is the buffer used by the algorithm for accepting input data. Before the call to the codec, the input buffer needs to be completely filled with input data.

2.1.6 Output Buffer

This is the buffer in which the algorithm writes the output. This buffer must be made available for the codec before its execution call. The output buffer pointer can be changed by the application between calls to the codec. This allows the codec to write directly to the required output area. The codec will never write more data than the requested size of the output buffer.

2.2 C Language API

A single interface function is used to access the codec, with the operation specified by command codes. The actual API C call is defined per codec library and is specified in the codec-specific section. Each library has a single C API call.

The C parameter definitions for every codec library are the same and are specified in the following table.

Table 2-1 Codec API

xa_<codec>	
Description	This C API is the only access function to the audio codec.

Syntax	`XA_ERRORCODE xa_<codec>(`
	`      xa_codec_handle_t  p_xa_module_obj,`
	`      WORD32 i_cmd,`
	`      WORD32 i_idx,`
	`      pVOID  pv_value);`

Parameters	`p_xa_module_obj`
	Pointer to the opaque API structure
	
	`i_cmd`
	Command
	
	`i_idx`
	Command subtype or index
	
	`pv_value`
	Pointer to the variable used to pass in, or get out properties from the state structure

Returns	Error code based on the success or failure of the API command

The types used for the C API call are defined in the supplied header files as:

```
typedef signed int              WORD32;
typedef void                   *pVOID;
```

Each time the C API for the codec is called, a pointer to a private allocated data structure is passed as the first argument. This argument is treated as an opaque handle as there is no requirement by the application to look at the data within the structure. The size of the structure is supplied by a specific API command so that the application can allocate the required memory. Do not use `sizeof()` on the type of the opaque handle.

Some command codes are further divided into subcommands. The command and its subcommand are passed to the codec via the second and third arguments respectively.

When a value must be passed to a particular API command or an API command returns a value, the value expected or returned is passed through a pointer, which is given as the fourth argument to the C API function. In the case of passing a pointer value to the codec, the pointer is just cast to `pVOID`. It is incorrect to pass a pointer to a pointer in these cases. An example would be when the application is passing the codec a pointer to an allocated memory region.

Due to the similarities of the operations required to decode or encode audio streams, the HiFi DSP API allows the application to use a common set of procedures for each stage. By maintaining a pointer to the single API function and passing the correct API object, the same code base can be used to implement the operations required for any of the supported codecs.

2.3 Generic API Errors

The error code returned is of type `XA_ERRORCODE`, which is of type `signed int`. The format of the error codes is defined in the following table.

Table 2-2 Error Codes Format

31	30-15	14 – 11	10 – 6	5 – 0
Fatal	Reserved	Class	Codec	Sub code

The errors that can be returned from the API are subdivided into those that are fatal, which require the restarting of the entire codec, and those that are nonfatal and are provided for information to the application.

The class of an error can be API, Config, or Execution. The API errors are concerned with the incorrect use of the API. The Config errors are produced when the codec parameters are incorrect or outside the supported usage. The Execution errors are returned after a call to the main encoding or decoding process and indicate situations that have arisen due to the input data.

2.4 Commands

This section covers the commands associated with the following command sequence overview flow chart. For each stage of the flow chart there is a section that lists the required commands in the order they should occur. For individual commands, definitions, and examples refer to Section 2.6. The codecs have a common set of generic API commands that are represented by the white stages. The yellow stages are specific to each codec.

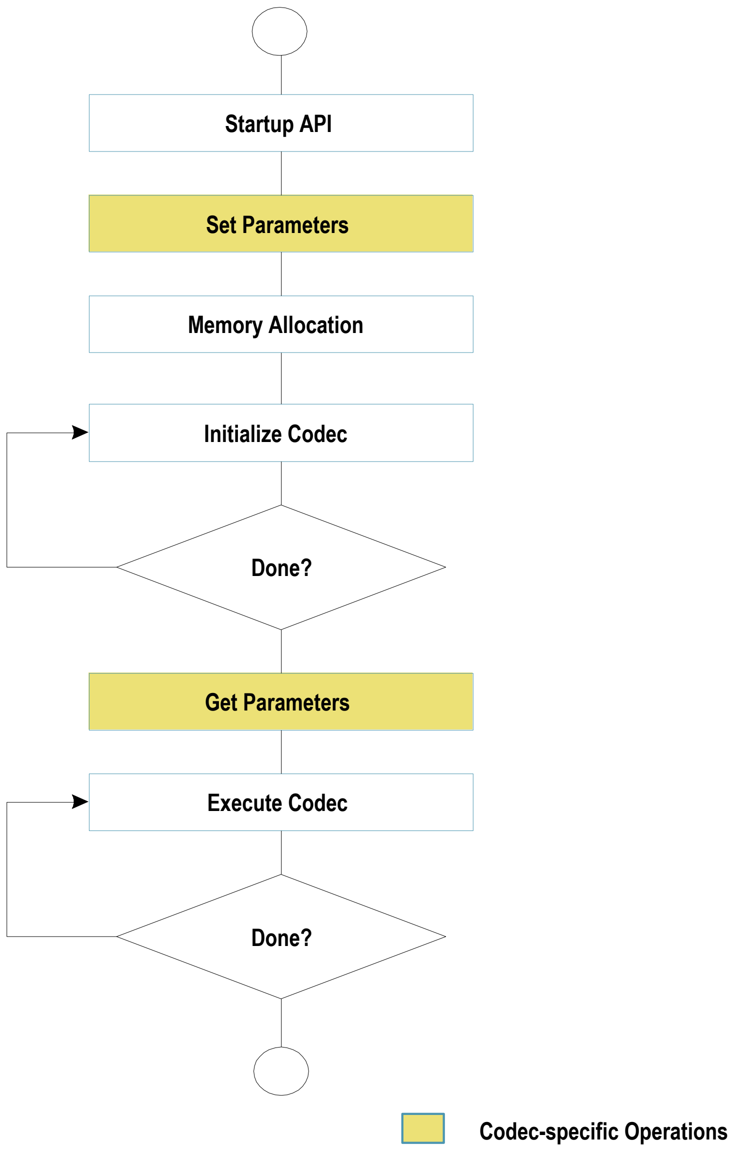


Figure 2 API Command Sequence Overview

2.4.1 Start-up API Stage

The following commands should be executed once each during start-up. The commands to get the various identification strings from the codec library are for information only and are optional. The command to get the API object size is mandatory as the real object type is hidden in the library and therefore there is no type available to use with `sizeof()`.

Table 2-3 Commands for Initialization

Command / Subcommand	Description
`XA_API_CMD_GET_LIB_ID_STRINGS` `XA_CMD_TYPE_LIB_NAME`	Get the name of the library.
`XA_API_CMD_GET_LIB_ID_STRINGS` `XA_CMD_TYPE_LIB_VERSION`	Get the version of the library.
`XA_API_CMD_GET_LIB_ID_STRINGS` `XA_CMD_TYPE_API_VERSION`	Get the version of the API.
`XA_API_CMD_GET_API_SIZE`	Get the size of the API structure.
`XA_API_CMD_INIT` `XA_CMD_TYPE_INIT_API_PRE_CONFIG_PARAMS`	Set the default values of all the configuration parameters.

2.4.2 Set Codec-Specific Parameters Stage

Refer to the specific codec section for the parameters that can be set. These parameters either control the encoding process or determine the output format of the decoder PCM data.

Table 2-4 Commands for Setting Parameters

Command / Subcommand	Description

`XA_API_CMD_SET_CONFIG_PARAM`
`XA_<codec>_CONFIG_PARAM_<param_name>`
	Set the codec-specific parameter. See the codec-specific section for parameter definitions.

2.4.3 Memory Allocation Stage

The following commands should be executed once only after all the codec-specific parameters have been set. The API is passed the pointer to the memory table structure (MEMTABS) after it is allocated by the application to the size specified. After the codec specific parameters are set, the initial codec setup is completed by performing the post-configuration portion of the initialization to determine the initial operating mode of the codec and assign sizes to the blocks of memory required for its operation. The application then requests a count of the number of memory blocks.

Table 2-5 Commands for Initial Table Allocation

Command / Subcommand	Description
`XA_API_CMD_GET_MEMTABS_SIZE`	Get the size of the memory structures to be allocated for the codec tables.
`XA_API_CMD_SET_MEMTABS_PTR`	Pass the memory structure pointer allocated for the tables.
`XA_API_CMD_INIT` `XA_CMD_TYPE_INIT_API_POST_CONFIG_PARAMS`	Calculate the required sizes for all the memory blocks based on the codec-specific parameters.
`XA_API_CMD_GET_N_MEMTABS`	Obtain the number of memory blocks required by the codec.

The following commands should then be executed in a loop to allocate the memory. The application first requests all the attributes of the memory block and then allocates it. It is important to abide by the alignment requirements. Finally, the pointer to the allocated block of memory is passed back through the API. For the input and output buffers it is not necessary to assign the correct memory at this point. The input and output buffer locations must be assigned before their first use in the EXECUTE stage. The type field refers to the memory blocks, for example input or persistent, as described in Section 2.1.

Table 2-6 Commands for Memory Allocation

Command / Subcommand	Description
`XA_API_CMD_GET_MEM_INFO_SIZE`	Get the size of the memory type being referred to by the index.
`XA_API_CMD_GET_MEM_INFO_ALIGNMENT`	Get the alignment information of the memory-type being referred to by the index.
`XA_API_CMD_GET_MEM_INFO_TYPE`	Get the type of memory being referred to by the index.
`XA_API_CMD_GET_MEM_INFO_PRIORITY`	Get the allocation priority of memory being referred to by the index.
`XA_API_CMD_SET_MEM_PTR`	Set the pointer to the memory allocated for the referred index to the input value.

2.4.4 Initialize Codec Stage

The following commands should be executed in a loop during initialization. These commands should be called until the initialization is completed as indicated by the `XA_CMD_TYPE_INIT_DONE_QUERY` command. In general, decoders can loop multiple times until the header information is found. However, encoders will perform exactly one call before they signal they are done.

There is a major difference between encoding Pulse Code Modulated (PCM) data and decoding stream data. During the initialization of a decoder, the initialization task reads the input stream to discover the parameters of the encoding. However, for an encoder there is no header information in PCM data. Even so, the encoder application is still required to perform the initialization described in this stage. However, encoders will not consume data during initialization. Furthermore, this has an implication in that some encoders provide parameters that can be used to modify the input buffer data requirements after the initialization stage. These modifications will always be a reduction in the size. The application only needs to provide the reduced amount per execution of the main codec process.

In general, the application will signal to the codec the number of bytes available in the input buffer and signal if it is the last iteration. It is not normal to hit the end of the data during initialization, but in the case of a decoder being presented with a corrupt stream it will allow a graceful termination. After the codec initialization is called, the application will ask for the number of bytes consumed. The application can also ask if the initialization is complete, it is advisable to always ask, even in the case of encoders that require only a single pass. A decoder application must keep iterating until it is complete.

Table 2-7 Commands for Initialization

Command / Subcommand	Description
`XA_API_CMD_SET_INPUT_BYTES`	Set the number of bytes available in the input buffer for initialization.
`XA_API_CMD_INPUT_OVER`	Signal to the codec the end of the bitstream.
`XA_API_CMD_INIT` `XA_CMD_TYPE_INIT_PROCESS`	Search for the valid header, does header decoding to get the parameters and initializes state and configuration structures.
`XA_API_CMD_INIT` `XA_CMD_TYPE_INIT_DONE_QUERY`	Check if the initialization process has completed.
`XA_API_CMD_GET_CURIDX_INPUT_BUF`	Get the number of input buffer bytes consumed by the last initialization.

2.4.5 Get Codec-Specific Parameters Stage

Finally, after the initialization, the codec can supply the application with information. In the case of decoders this would be the parameters it has extracted from the encoded header in the stream.

Table 2-8 Commands for Getting Parameters

Command / Subcommand	Description

`XA_API_CMD_GET_CONFIG_PARAM`
`XA_<codec>_CONFIG_PARAM_<param_name>`
	Get the value of the parameter from the codec. See the codec-specific section for parameter definitions.

2.4.6 Execute Codec Stage

The following commands should be executed continuously until the data is exhausted or the application wants to terminate the process. This is similar to the initialization stage, but includes support for the management of the output buffer. After each iteration, the application requests how much data is written to the output buffer. This amount is always limited by the size of the buffer requested during the memory block allocation. (To alter the output buffer position use `XA_API_CMD_SET_MEM_PTR` with the output buffer index.)

Table 2-9 Commands for Codec Execution

Command / Subcommand	Description
`XA_API_CMD_INPUT_OVER`	Signal the end of bitstream to the library.
`XA_API_CMD_SET_INPUT_BYTES`	Set the number of bytes available in the input buffer for the execution.
`XA_API_CMD_EXECUTE` `XA_CMD_TYPE_DO_EXECUTE`	Execute the codec thread.
`XA_API_CMD_EXECUTE` `XA_CMD_TYPE_DONE_QUERY`	Check if the end of stream has been reached.
`XA_API_CMD_GET_OUTPUT_BYTES`	Get the number of bytes output by the codec in the last frame.
`XA_API_CMD_GET_CURIDX_INPUT_BUF`	Get the number of input buffer bytes consumed by the last call to the codec.

2.5 Files Describing the API

The common include files (`include`)

- `xa_apicmd_standards.h`

  The command definitions for the generic API calls

- `xa_error_standards.h`

  The macros and definitions for all the generic errors

- `xa_memory_standards.h`

  The definitions for memory block allocation

- `xa_type_def.h`

  All the types required for the API calls

2.6 HiFi API Command Reference

In this section, the different commands are described along with their associated subcommands. The only commands missing are those specific to a single codec. The particular codec commands are generally the SET and GET commands for the operational parameters.

The commands are listed below in sections based on their primary commands type (`i_cmd`). Each section contains a table for every subcommand. In the case of no subcommands the one primary command is presented.

The commands are followed by an example C call. Along with the call there is a definition of the variable types used. This is to avoid any confusion over the type of the fourth argument. The examples are not complete C code extracts as there is no initialization of the variables before they are used.

The errors returned by the API are detailed after each of the command definitions. However, there are a few errors that are common to all the API commands; these are listed in Section 0. All the errors possible from the codec-specific commands will be defined in the codec-specific sections. Furthermore, the codec-specific sections also cover the Execution errors that occur during the initialization or execution calls to the API.

2.6.1 Common API Errors

These errors are fatal and should not be encountered during normal application operation. They signal that a serious error has occurred in the application that is calling the codec.

- 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_API_FATAL_INVALID_CMD

  `i_cmd` is not a valid command

- XA_API_FATAL_INVALID_CMD_TYPE

  `i_idx` is invalid for the specified command `(i_cmd)`

2.6.2 XA_API_CMD_GET_LIB_ID_STRINGS

Table 2-10 XA_CMD_TYPE_LIB_NAME subcommand

Subcommand	`XA_CMD_TYPE_LIB_NAME`
Description	This command obtains the name 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.

Actual Parameters	`p_xa_module_obj`
	`NULL`
	
	`i_cmd`
	`XA_API_CMD_GET_LIB_ID_STRINGS`
	
	`i_idx`
	`XA_CMD_TYPE_LIB_NAME`
	
	`pv_value`
	`process name` – Pointer to a character buffer in which the name of the library is returned

Restrictions	None

Note No codec object is required due to the name being static data in the codec library.

Example

```
char process_name[30];
res = (*api_func)(NULL,
                  XA_API_CMD_GET_LIB_ID_STRINGS,

                  XA_CMD_TYPE_LIB_NAME,

                  (pVOID) process_name);
```

Errors

- XA_API_FATAL_MEM_ALLOC

  This error is suppressed as `p_xa_module_obj` is `NULL`

- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

Table 2-11 XA_CMD_TYPE_LIB_VERSION subcommand

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.

Actual Parameters	`p_xa_module_obj`
	`NULL`
	
	`i_cmd`
	`XA_API_CMD_GET_LIB_ID_STRINGS`
	
	`i_idx`
	`XA_CMD_TYPE_LIB_VERSION`
	
	`pv_value`
	`lib_version` – Pointer to a character buffer in which the version of the library is returned

Restrictions	None

Note No codec object is required due to the version being static data in the codec library.

Example

```
char lib_version[30];
res = (*api_func)(NULL,
                  XA_API_CMD_GET_LIB_ID_STRINGS,

                  XA_CMD_TYPE_LIB_VERSION,

                  (pVOID) lib_version);
```

Errors

- XA_API_FATAL_MEM_ALLOC

  This error is suppressed as `p_xa_module_obj` is `NULL`

- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

Table 2-12 XA_CMD_TYPE_API_VERSION subcommand

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.

Actual Parameters	`p_xa_module_obj`
	`NULL`
	
	`i_cmd`
	`XA_API_CMD_GET_LIB_ID_STRINGS`
	
	`i_idx`
	`XA_CMD_TYPE_API_VERSION`
	
	`pv_value`
	`api_version` – Pointer to a character buffer in which the version of the API is returned

Restrictions	None

Note No codec object is required due to the version being static data in the codec library.

Example

```
char api_version[30];
res = (*api_func)(NULL,
                  XA_API_CMD_GET_LIB_ID_STRINGS,

                  XA_CMD_TYPE_API_VERSION,

                  (pVOID) api_version);
```

Errors

- XA_API_FATAL_MEM_ALLOC

  This error is suppressed as `p_xa_module_obj` is `NULL`

- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

2.6.3 XA_API_CMD_GET_API_SIZE

Table 2-13 XA_API_CMD_GET_API_SIZE command

Subcommand	None
Description	This command is used to obtain the size of the API structure, in order to allocate memory for the API structure. The pointer to the API size variable is passed and the API returns the size of the structure in bytes. The API structure is used for the interface and is persistent.

Actual Parameters	`p_xa_module_obj`
	`NULL`
	
	`i_cmd`
	`XA_API_CMD_GET_API_SIZE`
	
	`i_idx`
	`NULL`
	
	`pv_value`
	`&api_size` – Pointer to the API size variable

Restrictions	The application will allocate memory with an alignment of 4 bytes.

Note No codec object is required due to the size being fixed for the codec library.

Example

```
unsigned int api_size;
res = (*api_func)(NULL,
                  XA_API_CMD_GET_API_SIZE,

                  0,

                  (pVOID) &api_size);
```

Errors

- XA_API_FATAL_MEM_ALLOC

  This error is suppressed as `p_xa_module_obj` is `NULL`

- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

2.6.4 XA_API_CMD_INIT

Table 2-14 XA_CMD_TYPE_INIT_API_PRE_CONFIG_PARAMS subcommand

Subcommand	`XA_CMD_TYPE_INIT_API_PRE_CONFIG_PARAMS`
Description	This command is used to set the default value of the configuration parameters. The configuration parameters can then be altered by using one of the codec-specific parameter setting commands. Refer to the codec-specific section.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_INIT`
	
	`i_idx`
	`XA_CMD_TYPE_INIT_API_PRE_CONFIG_PARAMS`
	
	`pv_value`
	`NULL`

Restrictions	None

Example

```
res = (*api_func)(api_obj,
                  XA_API_CMD_INIT,

                  XA_CMD_TYPE_INIT_API_PRE_CONFIG_PARAMS,

                  NULL);
```

Errors

- Common API Errors

Table 2-15 XA_CMD_TYPE_INIT_API_POST_CONFIG_PARAMS subcommand

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 codec-specific parameters have been set.
Actual Parameters	p_xa_module_obj api_obj – Pointer to API structure i_cmd XA_API_CMD_INIT i_idx XA_CMD_TYPE_INIT_API_POST_CONFIG_PARAMS pv_value **NULL**
Restrictions	None

Example

```
      res = (*api_func)(api_obj,
                        XA_API_CMD_INIT,

                        XA_CMD_TYPE_INIT_API_POST_CONFIG_PARAMS,

                        NULL);
```

Errors

- Common API Errors

Table 2-16 XA_CMD_TYPE_INIT_PROCESS subcommand

Subcommand	`XA_CMD_TYPE_INIT_PROCESS`
Description	This command initializes the codec. In the case of a decoder, it searches for the valid header and performs the header decoding to get the encoded stream parameters. This command is part of the initialization loop. It must be repeatedly called until the codec signals it has finished. In the case of an encoder, the initialization of codec is performed. No output data is created during initialization.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_INIT`
	
	`i_idx`
	`XA_CMD_TYPE_INIT_PROCESS`
	
	`pv_value`
	`NULL`

Restrictions	None

Example

```
res = (*api_func)(api_obj,
                  XA_API_CMD_INIT,

                  XA_CMD_TYPE_INIT_PROCESS,

                  NULL);
```

Errors

- Common API Errors
- See the codec-specific section for execution errors

Table 2-17 XA_CMD_TYPE_INIT_DONE_QUERY subcommand

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; otherwise it is set to zero. A pointer to the flag variable is passed as an argument.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_INIT`
	
	`i_idx`
	`XA_CMD_TYPE_INIT_DONE_QUERY`
	
	`pv_value`
	`&init_done` – Pointer to a flag that indicates the completion of initialization process

Restrictions	None

Example

```
unsigned int init_done;
res = (*api_func)(api_obj,
                  XA_API_CMD_INIT,

                  XA_CMD_TYPE_INIT_DONE_QUERY,

                  (pVOID) &init_done);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

2.6.5 XA_API_CMD_GET_MEMTABS_SIZE

Table 2-18 XA_API_CMD_GET_MEMTABS_SIZE command

Subcommand	None
Description	This command is used to obtain the size of the table used to hold the memory blocks required for the codec 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 codec writes the value to the variable.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_MEMTABS_SIZE`
	
	`i_idx`
	`NULL`
	
	`pv_value`
	`&proc_mem_tabs_size` – Pointer to the memory size variable

Restrictions	The application shall allocate memory with an alignment of 4 bytes.

Example

```
unsigned int proc_mem_tabs_size;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_MEMTABS_SIZE,

                  0,

                  (pVOID) &proc_mem_tabs_size);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

2.6.6 XA_API_CMD_SET_MEMTABS_PTR

Table 2-19 XA_API_CMD_SET_MEMTABS_PTR command

Subcommand	None
Description	This command is used to set the memory structure pointer in the library to the allocated value.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_MEMTABS_PTR`
	
	`i_idx`
	`NULL`
	
	`pv_value`
	`alloc` – Allocated pointer

Restrictions	The application will allocate memory with an alignment of 4 bytes.

Example

```
int * alloc; //alloc is a pointer to the allocated memory
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_MEMTABS_PTR,

                  0,

                  (pVOID) alloc);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_API_FATAL_MEM_ALIGN

  `pv_value` is not aligned to 4 bytes

2.6.7 XA_API_CMD_GET_N_MEMTABS

Table 2-20 XA_API_CMD_GET_N_MEMTABS command

Subcommand	None
Description	This command obtains the number of memory blocks needed by the codec. 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 codec writes the value to this variable.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_N_MEMTABS`
	
	`i_idx`
	`NULL`
	
	`pv_value`
	`&n_mems` – Number of memory blocks required to be allocated

Restrictions	None

Example

```
int n_mems;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_N_MEMTABS,

                  0,

                  (pVOID) &n_mems);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

2.6.8 XA_API_CMD_GET_MEM_INFO_SIZE

Table 2-21 XA_API_CMD_GET_MEM_INFO_SIZE command

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. Note this is the actual size needed, not including any alignment packing space.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_MEM_INFO_SIZE`
	
	`i_idx`
	Index of the memory
	
	`pv_value`
	`&size` – Pointer to the memory size

Restrictions	None

Example

```
int index;
unsigned int size;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_MEM_INFO_SIZE,

                  index,

                  (pVOID) &size);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_API_FATAL_INVALID_CMD_TYPE

  `i_idx` is an invalid memory block number; valid block numbers obey the relation `0 <= i_idx < n_mems` (See XA_API_CMD_GET_N_MEMTABS)

2.6.9 XA_API_CMD_GET_MEM_INFO_ALIGNMENT

Table 2-22 XA_API_CMD_GET_MEM_INFO_ALIGNMENT command

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.
Actual Parameters	`p_xa_module_obj` `api_obj` – Pointer to API structure `i_cmd` `XA_API_CMD_GET_MEM_INFO_ALIGNMENT` `i_idx` Index of the memory `pv_value` `&alignment` – Pointer to the alignment info variable
Restrictions	None

Example

```
int index;
unsigned int alignment;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_MEM_INFO_ALIGNMENT,

                  index,

                  (pVOID) &alignment);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_API_FATAL_INVALID_CMD_TYPE

  `i_idx` is an invalid memory block number; valid block numbers obey the relation `0 <= i_idx < n_mems` (See XA_API_CMD_GET_N_MEMTABS)

2.6.10 XA_API_CMD_GET_MEM_INFO_TYPE

Table 2-23 XA_API_CMD_GET_MEM_INFO_TYPE command

Subcommand	Memory index
Description	This command gets the type of memory being referred to by the index.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_MEM_INFO_TYPE`
	
	`i_idx`
	Index of the memory
	
	`pv_value`
	`&type` – Pointer to the memory type variable

Restrictions	None

Example

```
int index;
unsigned int type;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_MEM_INFO_TYPE,

                  index,

                  (pVOID) &type);
```

Table 2-24 Memory Type Indices

Type	Description
XA_MEMTYPE_PERSIST	Persistent memory
XA_MEMTYPE_SCRATCH	Scratch memory
XA_MEMTYPE_INPUT	Input Buffer
XA_MEMTYPE_OUTPUT	Output Buffer

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_API_FATAL_INVALID_CMD_TYPE

  `i_idx` is an invalid memory block number; valid block numbers obey the relation `0 <= i_idx < n_mems` (See XA_API_CMD_GET_N_MEMTABS)

2.6.11 XA_API_CMD_GET_MEM_INFO_PRIORITY

Table 2-25 XA_API_CMD_GET_MEM_INFO_PRIORITY command

Subcommand	Memory index
Description	This command gets the allocation priority of memory being referred to by the index. (The meaning of the levels is defined on a codec-specific basis. This command returns a fixed dummy value unless the codec defines it otherwise.)

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_MEM_INFO_PRIORITY`
	
	`i_idx`
	Index of the memory
	
	`pv_value`
	`&priority` – Pointer to the memory priority variable

Restrictions	None

Example

```
int index;
unsigned int priority;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_MEM_INFO_PRIORITY,
                  index,
                  (pVOID) &priority);
```

Table 2-26 Memory Priorities

Priority	Type
0	XA_MEMPRIORITY_ANYWHERE
1	XA_MEMPRIORITY_LOWEST
2	XA_MEMPRIORITY_LOW
3	XA_MEMPRIORITY_NORM
4	XA_MEMPRIORITY_ABOVE_NORM
5	XA_MEMPRIORITY_HIGH
6	XA_MEMPRIORITY_HIGHER
7	XA_MEMPRIORITY_CRITICAL

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_API_FATAL_INVALID_CMD_TYPE

  `i_idx` is an invalid memory block number; valid block numbers obey the relation `0 <= i_idx < n_mems` (See XA_API_CMD_GET_N_MEMTABS)

2.6.12 XA_API_CMD_SET_MEM_PTR

Table 2-27 XA_API_CMD_SET_MEM_PTR Command

Subcommand	Memory index
Description	This command passes to the codec the pointer to the allocated memory. This is then stored in the memory tables structure allocated earlier. For the input and output buffers, it is legitimate to execute this command during the main codec loop.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_MEM_PTR`
	
	`i_idx`
	Index of the memory
	
	`pv_value`
	`alloc` – Pointer to the memory buffer allocated

Restrictions	The pointer must be correctly aligned to the requirements.

Example

```
int index;
void * alloc; //alloc is a pointer to the aligned memory
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_MEM_PTR,

                  index,

                  (pVOID) alloc);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_API_FATAL_INVALID_CMD_TYPE

  `i_idx` is an invalid memory block number; valid block numbers obey the relation `0 <= i_idx < n_mems` (See XA_API_CMD_GET_N_MEMTABS)
- XA_API_FATAL_MEM_ALIGN

  `pv_value` is not of the required alignment for the requested memory block

2.6.13 XA_API_CMD_INPUT_OVER

Table 2-28 XA_API_CMD_INPUT_OVER command

Subcommand	None
Description	This command tells the codec that the end of the input data has been reached. This situation can arise both in the initialization loop and the execute loop.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_INPUT_OVER`
	
	`i_idx`
	`NULL`
	
	`pv_value`
	`NULL`

Restrictions	None

Example

```
res = (*api_func)(api_obj,
                  XA_API_CMD_INPUT_OVER,

                  0,

                  NULL);
```

Errors

- Common API Errors

2.6.14 XA_API_CMD_SET_INPUT_BYTES

Table 2-29 XA_API_CMD_SET_INPUT_BYTES command

Subcommand	None
Description	This command sets the number of bytes available in the input buffer for the codec. It is used both in the initialization loop and execute loop. It is the number of valid bytes from the buffer pointer. It should be at least the minimum buffer size requested unless this is the end of the data.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_INPUT_BYTES`
	
	`i_idx`
	`NULL`
	
	`pv_value`
	`&buff_size` – Pointer to the input byte variable

Restrictions	None

Example

```
int buff_size;
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_INPUT_BYTES,

                  0,

                  (pVOID) &buff_size);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

2.6.15 XA_API_CMD_GET_CURIDX_INPUT_BUF

Table 2-30 XA_API_CMD_GET_CURIDX_INPUT_BUF command

Subcommand	None
Description	This command gets the number of input buffer bytes consumed by the codec. It is used both in the initialization loop and execute loop.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CURIDX_INPUT_BUF`
	
	`i_idx`
	`NULL`
	
	`pv_value`
	`&bytes_consumed` – Pointer to the bytes consumed variable

Restrictions	None

Example

```
int bytes_consumed;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CURIDX_INPUT_BUF,

                  0,

                  (pVOID) &bytes_consumed);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

2.6.16 XA_API_CMD_EXECUTE

Table 2-31 XA_CMD_TYPE_DO_EXECUTE subcommand

Subcommand	`XA_CMD_TYPE_DO_EXECUTE`
Description	This command executes the codec.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_EXECUTE`
	
	`i_idx`
	`XA_CMD_TYPE_DO_EXECUTE`
	
	`pv_value`
	`NULL`

Restrictions	None

Example

```
res = (*api_func)(api_obj,
                  XA_API_CMD_EXECUTE,

                  XA_CMD_TYPE_DO_EXECUTE,

                  NULL);
```

Errors

- Common API Errors
- See the codec-specific section for execution errors

Table 2-32 XA_CMD_TYPE_DONE_QUERY subcommand

Subcommand	`XA_CMD_TYPE_DONE_QUERY`
Description	This command checks to see if the end of processing has been reached. If it has, the flag value is set to 1; otherwise it is set to zero. The pointer to the flag is passed as an argument. Processing by the codec can continue for several invocations of the DO_EXECUTE command after the last input data has been passed to the codec, thus the application should not assume that the codec has finished generating all its output until so indicated by this command.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_EXECUTE`
	
	`i_idx`
	`XA_CMD_TYPE_DONE_QUERY`
	
	`pv_value`
	`&flag` – Pointer to the flag variable

Restrictions	None

Example

```
int flag;
res = (*api_func)(api_obj,
                  XA_API_CMD_EXECUTE,

                  XA_CMD_TYPE_DONE_QUERY,

                  (pVOID) &flag);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

Table 2-33 XA_CMD_TYPE_DO_RUNTIME_INIT subcommand

Subcommand	`XA_CMD_TYPE_DO_RUNTIME_INIT`
Description	This command resets the decoder's history buffers. It can be used to avoid distortions and clicks by facilitating playback ramping up and down during trick-play. The command should be issued before the application starts feeding the decoder with new data from a random place in the input stream. Note: This command is available in API version 1.14 or later.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_EXECUTE`
	
	`i_idx`
	`XA_CMD_TYPE_DO_RUNTIME_INIT`
	
	`pv_value`
	`NULL`

Restrictions	None

Example

```
res = (*api_func)(api_obj,
                  XA_API_CMD_EXECUTE,
                  XA_CMD_TYPE_DO_RUNTIME_INIT,
                  NULL);
```

Errors

- Common API Errors

2.6.17 XA_API_CMD_GET_OUTPUT_BYTES

Table 2-34 XA_API_CMD_GET_OUTPUT_BYTES command

Subcommand	None
Description	This command obtains the number of bytes output by the codec during the last execution.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_OUTPUT_BYTES`
	
	`i_idx`
	`NULL`
	
	`pv_value`
	`&out_bytes` – Pointer to the output bytes variable

Restrictions	None

Example

```
int out_bytes;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_OUTPUT_BYTES,

                  0,

                  (pVOID) &out_bytes);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

2.6.18 XA_API_CMD_GET_CONFIG_PARAM

Table 2-35 XA_CONFIG_PARAM_CUR_INPUT_STREAM_POS subcommand

Subcommand	`XA_CONFIG_PARAM_CUR_INPUT_STREAM_POS`

Description	This command reads the current input stream position, which is equal to the total number of consumed input bytes until the start of the input buffer. This running counter is set to zero at library initialization time and incremented every time the codec library consumes any bytes from the input buffer. If the application layer places a unit of input data with a byte size equal to `size` at byte offset `offset` in the input buffer, then the input stream position range for this unit may be calculated as follows:
	`start_pos = CUR_INPUT_STREAM_POS + offset`
	`end_pos   = CUR_INPUT_STREAM_POS + offset + size`

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_CONFIG_PARAM_CUR_INPUT_STREAM_POS`
	
	`pv_value`
	`&ui_cur_input_stream_pos` – Pointer to the current input stream position variable.

Restrictions	The current input stream position counter is 32-bits, therefore it will overflow and wrap-around if the input stream length is more than 2^{32}-1 bytes. This command is available in API version 1.15 or later.

Example

```
unsigned int ui_cur_input_stream_pos;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_CONFIG_PARAM_CUR_INPUT_STREAM_POS,
                  (void *) &ui_cur_input_stream_pos);
```

Errors

- Common API Errors

Table 2-36 XA_CONFIG_PARAM_GEN_INPUT_STREAM_POS subcommand

Subcommand	`XA_CONFIG_PARAM_GEN_INPUT_STREAM_POS`
Description	This command reads the input stream position of the unit (e.g., frame) corresponding to the generated (decoded or encoded) output data block. That is, if the main processing (`DO_EXECUTE`) call into the library generates any data in the output buffer, then this command reads the total number of input bytes consumed until the start of the unit that has been processed and placed into the output buffer. For example, if the application layer places a unit in the input buffer at input stream position `start_pos` (see Table 2-35), when the library generates the decoded or encoded data corresponding to this unit, it sets `GEN_INPUT_STREAM_POS` to `start_pos`.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_CONFIG_PARAM_GEN_INPUT_STREAM_POS`
	
	`pv_value`
	`&ui_gen_input_stream_pos` – Pointer to the input stream position of the generated data variable.

Restrictions	The input stream position of the generated data counter is 32-bit, therefore it will overflow and wrap-around if the input stream length is more than 2^{32}-1 bytes. This command is available in API version 1.15 or later.

Example

```
unsigned int ui_gen_input_stream_pos;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_CONFIG_PARAM_GEN_INPUT_STREAM_POS,
                  (void *) &ui_gen_input_stream_pos);
```

Errors

- Common API Errors

2.6.19 XA_API_CMD_SET_CONFIG_PARAM

Table 2-37 XA_CONFIG_PARAM_CUR_INPUT_STREAM_POS subcommand

Subcommand	`XA_CONFIG_PARAM_CUR_INPUT_STREAM_POS`
Description	This command resets the current input stream position. See Table 2-35 for details.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_CONFIG_PARAM_CUR_INPUT_STREAM_POS`
	
	`pv_value`
	`&ui_cur_input_stream_pos` – Pointer to the current input stream position variable.

Restrictions	This command is available in API version 1.15 or later.

Example

```
unsigned int ui_cur_input_stream_pos = 0;
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_CONFIG_PARAM,
                  XA_CONFIG_PARAM_CUR_INPUT_STREAM_POS,
                  (void *) &ui_cur_input_stream_pos);
```

Errors

- Common API Errors

3. HiFi DSP DAB+ Decoder

The HiFi DAB+ Decoder conforms to the generic codec API. The flow chart of the command sequence used in the example test bench is provided below.

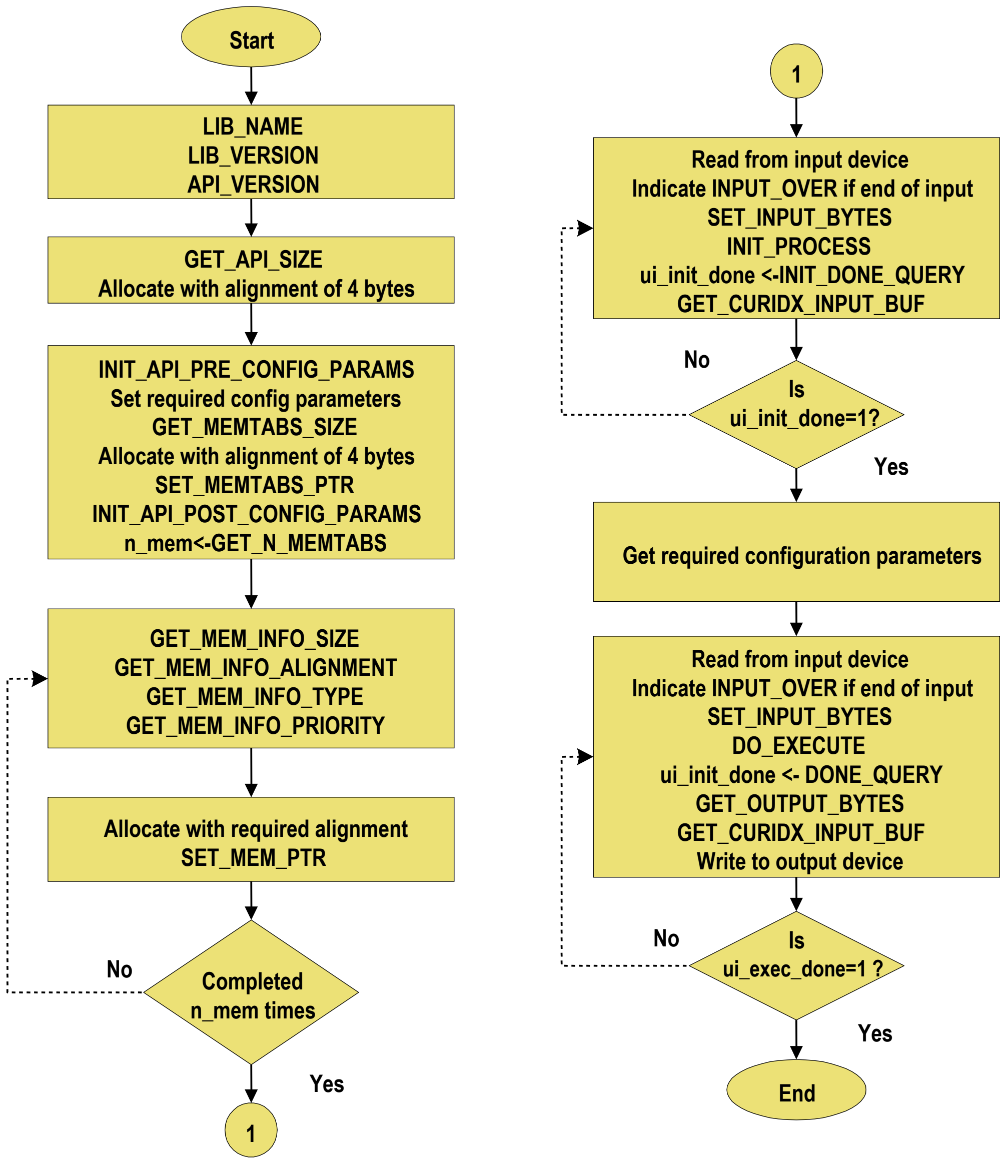


Figure 3 Flow Chart for DAB+ Decoder Integration

3.1 Files Specific to the DAB+ Decoder

The DAB+ Decoder parameter header file (`include/aac_dec`)

- `xa_ms11_ddenc_api.h`
- `xa_aac_dec_api.h`

The DAB+ decoder library (`lib`)

- `xa_dabplus_dec.a`

The DAB+ decoder API call is defined as:

```
XA_ERRORCODE xa_dabplus_dec(xa_codec_handle_t p_xa_module_obj,
                            WORD32            i_cmd,
                            WORD32            i_idx,
                            pVOID             pv_value);
```

3.2 Configuration Parameters

The HiFi DAB+ Decoder library accepts the following parameters from the user:

- **`pcm_wdsz`** – The PCM sample size in bits.
  - `16` – 16-bit PCM samples
  - `24` – 24-bit PCM samples (default). The samples are stored in the 24 MSBs of each output 32-bit word; the 8 LSBs are set to 0.
- **`bdownsample`** – SBR downsampling
  - `0` – disabled (default)
  - `1` – enabled
- **`downmix`** – Bitstream downmixing for SBR channels. Enabling this flag applies SBR processing on mono channels to reduce computational requirements of the processing. If the input to SBR processing is a channel pair, the channels are downmixed into a mono channel and then processed through the SBR tool.
  - `0` – disabled (default)
  - `1` – enabled

- **`bsformat`** – The external bitstream format can be set to one of the values below. Note that the API header file defines an enum type `XA_AACDEC_EBITSTREAM_TYPE` that can be used instead of the numerical values.
  - `8` – RAW DAB+ bitstream (no-header format). This format requires valid settings of the `externalsr` and the `raw_au_sideinfo` parameters.
  - `9` – DAB+ audio super-frame bitstream (default). This format requires a valid setting of the `externalbitrate` parameter.
- **`sbr_signaling`** – Parameter to explicitly disable or enable or set auto mode for the SBR tool.
  - `0` – Disable the SBR tool during decoding of all types of input bitstreams.
  - `1` – Enable and apply the SBR tool during decoding of all types of input bitstreams. This forces SBR processing even for plain AAC input streams.
  - `2` – Auto mode (default). Apply the SBR tool only for decoding of input bitstreams containing SBR information.
  - The `sbr_signaling` parameter controls and modifies the audio object type (`sbr_type`) returned by the library. If `sbr_signaling` is 0, then `sbr_type` is always 0 (plain AAC audio object type). If `sbr_signaling` is 1, then `sbr_type` is either 1 or 2 (indicating aacPlus object decoding).
- **`to_stereo`** – flag to specify duplication of mono signals to stereo
  - `0` – mono streams are presented as a single channel
  - `1` – mono streams are presented as 2 identical channels (default)
- **`externalsr`** – The sampling rate of basic AAC content of RAW bitstreams. Note that for RAW SBR and PS streams the sampling rate of the basic AAC content may be half the output sampling rate.
  - Default value: 48000
  - Valid values: 16000, 24000, 32000 or 48000 Hz
- **`externalbitrate`** – The data rate of the input stream. This parameter is valid only for DAB+ audio super-frame streams.
  - Default value: 48
  - Valid values: 8 kbps through 192 kbps in steps of 8 kbps
- **`raw_au_sideinfo`** – Side-info required for error concealment purposes along with each RAW DAB+ frame. This is a 16-bit value with the following contents:
  - Bit 15: CRC Error (0 – pass, 1 – fail)
  - Bits 14-0: Payload byte size

- **`fade_out_frames`** – Concealment fade-out slope length in frames/AUs. The number of consecutive invalid AUs after which the concealment switch from fade-out state to mute state (ramp-down).
  - Default value: 5
  - Valid values: 2 to 7
- **`mute_rel_frames`** – Concealment mute release length in frames/AUs. The number of consecutive valid AUs for which fade-in is suppressed after the concealment has switched to muting, to avoid annoying output of non-consecutive individual valid AUs.
  - Default value: 5
  - Valid values: 1 to 15
- **`fade_in_frames`** – Concealment fade-in slope length in frames/AUs. The number of consecutive valid AU frames after which decoder concealment switch from fade-in state to normal state (ramp up).
  - Default value: 5
  - Valid values: 2 to 7

After the initialization the configuration parameters related to the input stream and decoded audio can be obtained by getting the following parameter values:

- **`aac_format`** – The transport format for the encoded input bitstream.
  - `8` – raw DAB+
  - `9` – DAB+ audio super-frame
- **`num_channels`** – Number of decoded channels present in the output buffer.
  - Valid values: 1 or 2
- **`samp_freq`** – The output sample rate given in Hz.
- **`aac_samplerate`** – The sample rate at which the plain AAC decoder is operating.
  - When SBR is in use, this figure can be half the output sample rate.
- **`data_rate` –** This command gets the data rate of the encoded stream.
- **`acmod`** – Information about the audio coding mode of the input bitstream describing the encoded channel configuration.
  - `0` – undefined
  - `1` – mono (1/0),
  - `2` – parametric stereo
  - `3` – dual mono (1/0 + 1/0),
  - `4` – stereo (2/0),

- **sbr_type** – The decoded Audio Object Type.
  - 0 – plain AAC-LC (low complexity) object type
  - 1 – aacPlus object type containing SBR element.
  - 2 – aacPlusv2 object type containing PS object type
- **pcm_wdsz** – The output PCM bit width (16 or 24). Note that the 24-bit samples are stored in the 24 MSBs of each output 32-bit word; the 8 LSBs are set to 0.
- **pad_size** – The byte size of the PAD (Program Associated Data). This is the total of the X-PAD size (up to 196 bytes) and F-PAD size (2 bytes). The location of the PAD buffer can be obtained through the pad_ptr parameter. The PAD size will be 0 if PAD is not present in the input stream.
- **pad_ptr** – The pointer to a buffer containing the PAD. The buffer contains F-PAD followed by X-PAD. The size of the PAD buffer can be obtained through the pad_size parameter. The pointer will be NULL if PAD is not present in the input stream.
- **mpeg_surr_present** – Presence of MPEG surround in current decoded DAB+ super-frame when **bsformat** is DAB+ audio super-frame bitstream. If the decoder is fed with a DAB+ audio super-frame stream that has MPEG surround content, then it decodes the mono/stereo portion of AAC and discards the MPEG surround content.
  - 0 – Not present
  - 1 – Present.
- **fade_out_frames** – Concealment fade-out slope length in frames/AUs. This API returns the last value set by the application. If the application has not set a value, it returns the default.
  - Valid values: 2 to 7
- **mute_rel_frames** – Concealment mute release length in frames/AUS. This API returns the last value set by the application. If the application has not set a value, it returns the default.
  - Valid values: 1 to 15
- **fade_in_frames** – Concealment fade-in slope length in frames/AU. This API returns the last value set by the application. If the application has not set a value, it returns the default.
  - Valid values: 2 to 7

3.3 HiFi DAB+ Decoder-Specific Commands

These are the commands unique to the HiFi DAB+ decoder. They are listed in sections based on their primary commands type (i_cmd). Each section contains a table for every subcommand. In the case of no subcommands the one primary command is presented.

3.3.1 Initialization and Execution Errors

These errors can result from the initialization or execution API calls:

- XA_AACDEC_EXECUTE_NONFATAL_INSUFFICIENT_FRAME_DATA

  The input buffer has insufficient data for initialization or execution. The application needs to feed more data into the input buffer.

- XA_AACDEC_EXECUTE_NONFATAL_RUNTIME_INIT_RAMP_DOWN

  This non-fatal status code may be returned after a DO_EXECUTE call following a RUNTIME_INIT command for SBR streams. The output ramp down of SBR decoding extends to two frames and this status code indicates that the ramp down has not finished yet.

- XA_AACDEC_EXECUTE_NONFATAL_DABPLUS_HEADER_NOT_FOUND

  This error occurs at initialization time when the library does not find a valid header in the input buffer. The library will keep looking for a valid header during the subsequent INIT_PROCESS calls.

- XA_AACDEC_EXECUTE_NONFATAL_PARSING_ERROR

  Unable to parse the input frame. Frame skipped.

- XA_AACDEC_CONFIG_NONFATAL_PARTIAL_CHANROUTING

  This is a warning that the application has specified an incomplete channel routing. The partial routing will be used for decoding.

- XA_AACDEC_CONFIG_FATAL_INVALID_PCM_WDSZ

  Invalid PCM bit width. The bit width must be 16 or 24.

- XA_AACDEC_CONFIG_FATAL_INVALID_EXTERNALBITRATE

  Invalid data rate specified for DAB+ stream.

- XA_AACDEC_CONFIG_NONFATAL_DATA_RATE_NOT_SET

  This indicates that the data rate is not set. It is not available till the first few (10 to 15) frames are decoded. It is also not available in case of raw bitstreams.

- XA_AACDEC_CONFIG_NONFATAL_INVALID_GEN_STRM_POS

  The input stream position of the decoded data is invalid. This may occur if the decoder has not generated any output data, or a runtime-init ramp-down frame is active, or the current input stream position is reset by the application.

- XA_AACDEC_CONFIG_FATAL_INVALID_CONCEALMENT_PARAM

  The fade_in_frames or fade_out_frames or mute_rel_frames value is out of range.

The following fatal errors require the codec to be completely re-initialized and presented with an alternative bitstream.

- XA_AACDEC_EXECUTE_FATAL_UNSUPPORTED_FORMAT

  This indicates that the initialization routine is unable to determine the format, or is not able to find the sync-word.

- XA_AACDEC_EXECUTE_FATAL_PARSING_ERROR

  The transport bitstream presented is not in the specified format.

- XA_AACDEC_EXECUTE_FATAL_RAW_FRAME_PARSE_ERROR

  An error was encountered in trying to decoding the raw bitstream.

- XA_AACDEC_EXECUTE_FATAL_BAD_INPUT_FAILURE

  An error was encountered in decoding the raw bitstream due to sync word loss or corrupt input.

- XA_AACDEC_EXECUTE_FATAL_ERROR_IN_CHANROUTING

  This error indicates that the channel routing is erroneous – either channels present in the input bitstream are not routed to the output due to incomplete channel routing, or two or more channels are routed to the same sample offset in the output buffer

3.3.2 XA_API_CMD_SET_CONFIG_PARAM

Table 3-1 XA_AACDEC_CONFIG_PARAM_PCM_WDSZ subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_PCM_WDSZ`
Description	This command sets the output PCM sample bit width to 16 or 24. Each 24-bit sample is aligned in the 24 MSBs of a 32-bit word in the output buffer.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_PCM_WDSZ`
	
	`pv_value`
	`&pcm_wdsz` – Pointer to the width of the PCM sample variable

Restrictions	Valid values: 16 or 24 (default)

Example

```
int pcm_wdsz = 16;
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_PCM_WDSZ,
                  (void *) &pcm_wdsz);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_FATAL_INVALID_PCM_WDSZ

  The specified PCM bit width is invalid

Table 3-2 XA_AACDEC_CONFIG_PARAM_BDOWNSAMPLE subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_BDOWNSAMPLE`
Description	This command sets the SBR downsample flag.

Actual Parameters	`P_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_BDOWNSAMPLE`
	
	`pv_value`
	`&bdownsample` – Pointer to the SBR downsample flag variable

Restrictions	Valid values: 0 or 1.

Example

```
int bdownsample = 1;
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_BDOWNSAMPLE,
                  (void *) &bdownsample);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_FATAL_INVALID_BDOWNSAMPLE

  Value not valid

Table 3-3 XA_AACDEC_CONFIG_PARAM_BBITSTREAMDOWNMIX subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_BBITSTREAMDOWNMIX`
Description	This command sets the downmix flag. The flag is used to configure the SBR tool to perform SBR processing on downmixed mono channel, instead of channel pair. The mono channel is derived by downmixing the input channel pair.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_BBITSTREAMDOWNMIX`
	
	`pv_value`
	`&downmix` – Pointer to the downmix flag variable

Restrictions	Valid values: 0 or 1.

Example

```
int downmix = 1;
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_BBITSTREAMDOWNMIX,
                  (void *) &downmix);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_FATAL_INVALID_BBITSTREAMDOWNMIX

  Value not valid

Table 3-4 XA_AACDEC_CONFIG_PARAM_EXTERNALSAMPLINGRATE subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_EXTERNALSAMPLINGRATE`
Description	This command sets the sampling rate of the basic AAC stream in case of RAW bitstreams. Note that for RAW SBR streams the sampling rate of the basic AAC content may be half the output sampling rate.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_EXTERNALSAMPLINGRATE`
	
	`pv_value`
	`&externalsr` – Pointer to the external sample rate variable

Restrictions	Valid values: 16000, 24000, 32000, 48000. Default: 48000

Example

```
int externalsr = 32000;
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_EXTERNALSAMPLINGRATE,
                  (void *) &externalsr);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_FATAL_INVALID_EXTERNALSAMPLINGRATE

  Value not valid

Table 3-5 XA_AACDEC_CONFIG_PARAM_EXTERNALBITRATE subcommand

Subcommand	XA_AACDEC_CONFIG_PARAM_EXTERNALBITRATE
Description	This command sets the data rate of the input stream. This parameter is valid only for DAB+ audio super-frame streams.
Actual Parameters	p_xa_module_obj api_obj – Pointer to API structure i_cmd XA_API_CMD_SET_CONFIG_PARAM i_idx XA_AACDEC_CONFIG_PARAM_EXTERNALBITRATE pv_value &externalbitrate – Pointer to the external bitrate variable
Restrictions	Valid values: 8 through 192 in steps of 8 kbps. Default: 48 kbps

Example

```
int externalbitrate = 192;
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_EXTERNALBITRATE,
                  (void *) &externalbitrate);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  pv_value is NULL
- XA_AACDEC_CONFIG_FATAL_INVALID_EXTERNALBITRATE

  Value not valid

Table 3-6 XA_AACDEC_CONFIG_PARAM_EXTERNALBSFORMAT subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_EXTERNALBSFORMAT`
Description	This command sets the external bitstream type. The default format is DAB+ audio super-frame.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_EXTERNALBSFORMAT`
	
	`pv_value`
	`&bsformat` – Pointer to the bitstream format variable

Restrictions	Valid values: 8 (RAW DAB+) and 9 (DAB+ audio super-frame). The default is 9.

Example

```
int bsformat = 8 /* RAW DAB+ */;
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_EXTERNALBSFORMAT,
                  (void *) &bsformat);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_FATAL_INVALID_EXTERNALBSFORMAT

  Value not valid

Table 3-7 XA_AACDEC_CONFIG_PARAM_RAW_AU_SIDEINFO subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_RAW_AU_SIDEINFO`
Description	This command sets the side-info for each RAW DAB+ frame. For details, see Section **Error! Reference source not found.**.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_RAW_AU_SIDEINFO`
	
	`pv_value`
	`&raw_au_sideinfo` – Pointer to the side-info variable

Restrictions	None

Example

```
short raw_au_sideinfo =
            (0 /* 0 = CRC OK */ << 15) | payload_size;

res = (*api_func)(api_obj,
                  XA_API_CMD_SET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_RAW_AU_SIDEINFO,
                  (void *) &raw_au_sideinfo);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

Table 3-8 XA_AACDEC_CONFIG_PARAM_OUTNCHANS subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_OUTNCHANS`
Description	This command sets the maximum number of decoded channels present in the output buffer. If a channel is not present in the encoded input stream, the corresponding sample value is set to zero in the output buffer.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_OUTNCHANS`
	
	`pv_value`
	`&outnchans` – Pointer to the number of output channels variable

Restrictions	Valid value: 2

Example

```
int outnchans = 2;
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_OUTNCHANS,
                  (void *)&outnchans);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_FATAL_INVALID_OUTNCHANS

  Value not valid

Table 3-9 XA_AACDEC_CONFIG_PARAM_SBR_SIGNALING subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_SBR_SIGNALING`
Description	This command enables/disables the SBR processing or sets it in auto mode for using the SBR tool in decoding of the input streams. For details, see Section **Error! Reference source not found.**.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_SBR_SIGNALING`
	
	`pv_value`
	`&sbr_signaling` – Pointer to the flag

Restrictions	Valid values are 0, 1 or 2 (default).

Example

```
int sbr_signaling = 1;
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_SBR_SIGNALING,
                  (void *)&sbr_signaling);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_FATAL_INVALID_SBR_SIGNALING

  Value not valid

Table 3-10 XA_AACDEC_CONFIG_PARAM_TO_STEREO subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_TO_STEREO`
Description	This command enables or disables interleaving of mono to stereo in the output buffer. If enabled, the mono signal is replicated in two (stereo) output channels.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_TO_STEREO`
	
	`pv_value`
	`&to_stereo` – Pointer to the stereo conversion flag variable

Restrictions	0 to disable; 1 to enable (default).

Example

```
int to_stereo = 0;
res = (*api_func)(api_obj,
                  XA_API_CMD_SET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_TO_STEREO,
                  (void *) &to_stereo);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_FATAL_INVALID_TO_STEREO

  Value not valid

Table 3-11 XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_OUT_FRAMES subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_OUT_FRAMES`
Description	This command sets the fade-out slope length in frames/AUs (fade_out_frames) required for concealment. Concealment is explained in Section 3.4.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_OUT_FRAMES`
	
	`pv_value`
	`&fade_out_frames` – Pointer to the fade-out frame length variable

Restrictions	Valid values: 2 to 7. Default value is 5

Example

```
int fade_out_frames = 4;
res = (*api_func)(api_obj,
      XA_API_CMD_SET_CONFIG_PARAM,
      XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_OUT_FRAMES,
      (void *) &fade_out_frames);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_FATAL_INVALID_CONCEALMENT_PARAM

  Value not valid

Table 3-12 XA_AACDEC_CONFIG_PARAM_CONCEALMENT_MUTE_RELEASE_FRAMES subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_CONCEALMENT_MUTE_RELEASE_FRAMES`
Description	This command sets the number of consecutive valid frames/AUs (mute_release_frames) after which concealment state changes from mute to fade-in. Concealment is explained in Section 3.4.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_CONCEALMENT_MUTE_RELEASE_FRAMES`
	
	`pv_value`
	`&mute_rel_frames` – Pointer to the mute release frame length variable

Restrictions	Valid values: 1 to 15. Default value is 5.

Example

```
int mute_rel_frames = 2;
res = (*api_func)(api_obj,
        XA_API_CMD_SET_CONFIG_PARAM,
        XA_AACDEC_CONFIG_PARAM_CONCEALMENT_MUTE_RELEASE_FRAMES,
        (void *) &mute_rel_frames);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_FATAL_INVALID_CONCEALMENT_PARAM

  Value not valid

Table 3-13 XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_IN_FRAMES subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_FADE_IN_FRAMES`
Description	This command sets the fade-in slope length in frames/AUs (fade_in_frames) required for concealment. Concealment is explained in Section 3.4.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_SET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_IN_FRAMES`
	
	`pv_value`
	`&fade_in_frames` – Pointer to the fade-in frame length variable

Restrictions	Valid values: 2 to 7. Default value is 5

Example

```
int fade_in_frames = 3;
res = (*api_func)(api_obj,
             XA_API_CMD_SET_CONFIG_PARAM,
             XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_IN_FRAMES,
             (void *) &fade_in_frames);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_FATAL_INVALID_CONCEALMENT_PARAM

  Value not valid

3.3.3 XA_API_CMD_GET_CONFIG_PARAM

Table 3-14 XA_AACDEC_CONFIG_PARAM_AAC_FORMAT subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_AAC_FORMAT`
Description	This command gets the transport format of the input stream. Value 8 indicates raw DAB+ format. Value 9 indicates DAB+ audio super-frame format.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_AAC_FORMAT`
	
	`pv_value`
	`&aac_format` – Pointer to the AAC format variable

Restrictions	None

Example

```
int aac_format;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_AAC_FORMAT,
                  (void *)&aac_format);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_NONFATAL_PARAMS_NOT_SET

  This parameter is not available until after the initialization stage of the codec is complete. This value is read from the encoded stream header information.

Table 3-15 XA_AACDEC_CONFIG_PARAM_SBR_TYPE subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_SBR_TYPE`
Description	This command gets the decoded AAC audio object information. 0 means plain AAC object and 1 means aacPlus v1 object and 2 means aacPlus v2 (Parametric Stereo) object.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_SBR_TYPE`
	
	`pv_value`
	`&sbr_type` – Pointer to the SBR type variable

Restrictions	None

Example

```
int sbr_type;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_SBR_TYPE,
                  (void *) &sbr_type);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_NONFATAL_PARAMS_NOT_SET

  This parameter is not available until after the initialization stage of the codec is complete. This value is read from the encoded stream header information.

Table 3-16 XA_AACDEC_CONFIG_PARAM_SAMP_FREQ subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_SAMP_FREQ`
Description	This command gets the output sample rate

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_SAMP_FREQ`
	
	`pv_value`
	`&samp_freq` – Pointer to the output sample rate variable

Restrictions	None

Example

```
int samp_freq;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_SAMP_FREQ,
                  (void *) &samp_freq);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_NONFATAL_PARAMS_NOT_SET

  This parameter is not available until after the initialization stage of the codec is complete. This value is read from the encoded stream header information.

Table 3-17 XA_AACDEC_CONFIG_PARAM_AAC_SAMPLERATE subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_AAC_SAMPLERATE`
Description	This command gets the AAC sample rate.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_AAC_SAMPLERATE`
	
	`pv_value`
	`&aac_samplerate` – Pointer to the AAC sample rate variable

Restrictions	None

Example

```
int aac_samplerate;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_AAC_SAMPLERATE,
                  (void *) &aac_samplerate);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_NONFATAL_PARAMS_NOT_SET

  This parameter is not available until after the initialization stage of the codec is complete. This value is read from the encoded stream header information.

Table 3-18 XA_AACDEC_CONFIG_PARAM_DATA_RATE subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_DATA_RATE`
Description	This command gets the data rate of the encoded stream. This value is available only after decoding a few (typically 10 to 15) frames of data.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_DATA_RATE`
	
	`pv_value`
	`&data_rate` – Pointer to the input data rate variable

Restrictions	None

Example

```
int data_rate;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_DATA_RATE,
                  (void *) &data_rate);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_NONFATAL_DATA_RATE_NOT_SET

  This parameter is not available. This value is not available until the first few frames (typically 10 to 15) are decoded. The value is not available for raw bitstream format.

Table 3-19 XA_AACDEC_CONFIG_PARAM_ACMOD subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_ACMOD`
Description	This command gets the audio coding mode by identifying the channels encoded in the input bitstream. Details are described in Section 3.2.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_ACMOD`
	
	`pv_value`
	`&acmod` – Pointer to the audio coding mode variable

Restrictions	None

Example

```
int acmod;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_ACMOD,
                  (void *) &acmod);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`
- XA_AACDEC_CONFIG_NONFATAL_PARAMS_NOT_SET

  This parameter is not available until after the initialization stage of the codec is complete. This value is read from the encoded stream header information.

Table 3-20 XA_AACDEC_CONFIG_PARAM_NUM_CHANNELS subcommand

Subcommand	XA_AACDEC_CONFIG_PARAM_NUM_CHANNELS
Description	This command gets the output number of channels.
Actual Parameters	p_xa_module_obj api_obj – Pointer to API structure i_cmd XA_API_CMD_GET_CONFIG_PARAM i_idx XA_AACDEC_CONFIG_PARAM_NUM_CHANNELS pv_value &num_channels – Pointer to the output number of channels variable
Restrictions	None

Example

```
int num_channels;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_NUM_CHANNELS,
                  (void *) &num_channels);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  pv_value is NULL
- XA_AACDEC_CONFIG_NONFATAL_PARAMS_NOT_SET

  This parameter is not available until after the initialization stage of the codec is complete. This value is read from the encoded stream header information.

Table 3-21 XA_AACDEC_CONFIG_PARAM_PCM_WDSZ subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_PCM_WDSZ`
Description	This command gets the output bit width – 16 or 24. Each 24-bit sample is aligned in the 24 MSBs of a 32-bit word in the output buffer.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API sstructure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_PCM_WDSZ`
	
	`pv_value`
	`&pcm_wdsz` – Pointer to the width of the PCM sample variable

Restrictions	None

Example

```
int pcm_wdsz;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_PCM_WDSZ,
                  (void *) &pcm_wdsz);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

Table 3-22 XA_AACDEC_CONFIG_PARAM_PAD_SIZE subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_PAD_SIZE`
Description	This command gets the Program Associated Data (PAD) byte size. This size is the total of the X-PAD size (up to 196 bytes) and the F-PAD size (2 bytes),as defined in Section 5.4 of ETSI TS 102 563 [1]. The location of the PAD buffer can be obtained through the `XA_AACDEC_CONFIG_PARAM_PAD_PTR` subcommand. The PAD size will be 0 if PAD is not present in the input bitstream.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_PAD_SIZE`
	
	`pv_value`
	`&pad_size` – Pointer to the PAD size variable

Restrictions	None

Example

```
int pad_size;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_PAD_SIZE,
                  (void *) &pad_size);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

Table 3-23 XA_AACDEC_CONFIG_PARAM_PAD_PTR subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_PAD_PTR`
Description	This command gets the Program Associated Data (PAD) buffer location. The buffer contains X-PAD followed by F-PAD as defined in Section 5.4 of ETSI TS 102 563 [1]. The size of the PAD buffer can be obtained through the `XA_AACDEC_CONFIG_PARAM_PAD_SIZE` subcommand. The PAD pointer will be NULL if PAD is not present in the input stream.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_PAD_PTR`
	
	`pv_value`
	`&pad_ptr` – Pointer to the PAD buffer pointer

Restrictions	None

Example

```
char *pad_ptr;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_PAD_PTR,
                  (void *) &pad_ptr);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

Table 3-24 XA_AACDEC_CONFIG_PARAM_DRC_EXT_PRESENT subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_DRC_EXT_PRESENT`
Description	This command returns a value that indicates if DRC data is present or absent in dynamic_range_info () in the extension payload (contained in fill_element) for the current frame.(Refer to Section 5.4.2 of ETSI TS 102 563 [1] for details.)

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_DRC_EXT_PRESENT`
	
	`pv_value`
	`&drc_ext_flag` – Pointer to a variable that indicates the presence of DRC data in the current frame. Valid values are either 1 (present) or 0 (absent)

Restrictions	None

Example

```
int drc_ext_flag;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_DRC_EXT_PRESENT,
                  (void *) &drc_ext_flag);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

Table 3-25 XA_AACDEC_CONFIG_PARAM_MPEGSURR_PRESENT subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_MPEGSURR_PRESENT`
Description	This command gets the MPEG surround configuration for the current super frame. 0 – MPEG surround is not present 1 – MPEG surround with 5.1 output channels is present (Refer to Section 5.2 of ETSI TS 102 563 V1.1.1 [1] for more details.)

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_MPEGSURR_PRESENT`
	
	`pv_value`
	`&mpeg_surr_present` – Pointer to the MPEG surround presence indicator variable.

Restrictions	• MPEG surround decoding is not supported. • This parameter is valid only for DAB+ audio super-frame streams.

Example

```
int mpeg_surr_present;
res = (*api_func)(api_obj,
                  XA_API_CMD_GET_CONFIG_PARAM,
                  XA_AACDEC_CONFIG_PARAM_MPEGSURR_PRESENT,
                  (void *) &mpeg_surr_present);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

Table 3-26 XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_OUT_FRAMES subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_OUT_FRAMES`
Description	This command gets the fade-out slope length in frames/AUs set by the application. If the application did not set, it returns the default value. Concealment is explained in Section 3.4.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_OUT_FRAMES`
	
	`pv_value`
	`&fade_out_frames` – Pointer to the fade-out frame length variable

Restrictions	None

Example

```
int fade_out_frames;
res = (*api_func)(api_obj,
          XA_API_CMD_GET_CONFIG_PARAM,
          XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_OUT_FRAMES,
                    (void *) &fade_out_frames);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

Table 3-27 XA_AACDEC_CONFIG_PARAM_CONCEALMENT_MUTE_RELEASE_FRAMES subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_CONCEALMENT_MUTE_RELEASE_FRAMES`
Description	This command gets the mute release length in frames/AUs set by the application. If the application did not set, it returns the default value. Concealment is explained in Section 3.4.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_CONCEALMENT_MUTE_RELEASE_FRAMES`
	
	`pv_value`
	`&mute_rel_frames` – Pointer to the mute release frame length variable

Restrictions	None

Example

```
int mute_rel_frames;
res = (*api_func)(api_obj,
      XA_API_CMD_GET_CONFIG_PARAM,
      XA_AACDEC_CONFIG_PARAM_CONCEALMENT_MUTE_RELEASE_FRAMES,
                  (void *) &mute_rel_frames);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

Table 3-28 XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_IN_FRAMES subcommand

Subcommand	`XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_IN_FRAMES`
Description	This command gets the fade-in slope length in frames/AUs set by the application. If the application did not set, it returns the default value. Concealment is explained in Section 3.4.

Actual Parameters	`p_xa_module_obj`
	`api_obj` – Pointer to API structure
	
	`i_cmd`
	`XA_API_CMD_GET_CONFIG_PARAM`
	
	`i_idx`
	`XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_IN_FRAMES`
	
	`pv_value`
	`&fade_in_frames` – Pointer to the fade-in frame length variable

Restrictions	None

Example

```
int fade_in_frames;
res = (*api_func)(api_obj,
      XA_API_CMD_GET_CONFIG_PARAM,
      XA_AACDEC_CONFIG_PARAM_CONCEALMENT_FADE_IN_FRAMES,
                   (void *) &fade_in_frames);
```

Errors

- Common API Errors
- XA_API_FATAL_MEM_ALLOC

  `pv_value` is `NULL`

3.4 Concealment Mechanism

The AAC core decoder includes a concealment function that increases the delay of the decoder by one AU.

In case a single AU is corrupted, the decoder interpolates between the preceding and the following valid AUs to create the spectral data for the missing AU. The interpolation is explained in section A1.1.1 of ETSI TS 102 563 V1.1.1 [1]

If multiple AUs are corrupted, the concealment implements first a fade out based on slightly modified spectral values from the last valid AU. The decoder allows the application to select the number of AUs for the attenuation ramp of fade out (fade_out_frames). If the decoder recovers from the error condition within fade_out_frames, the decoding becomes normal immediately.

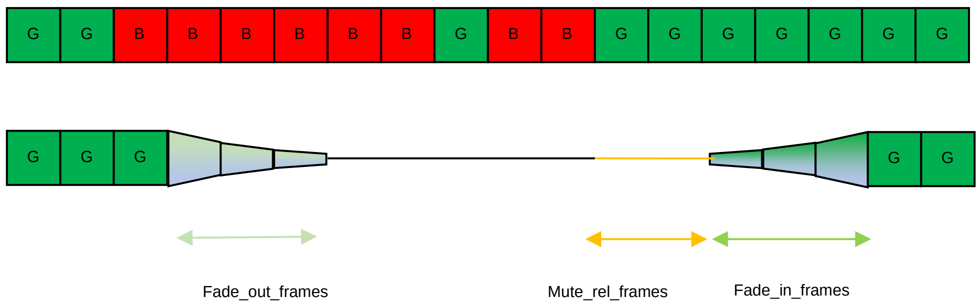
If the error condition persists beyond *fade_out_frames*, the concealment switches to muting, which means the complete spectrum will be set to 0. When the decoder recovers from the error condition, it will still continue being in the mute state for a certain number of frames to avoid annoying output of non-consecutive individual valid AUs (mute_rel_frames). The application can also control the mute_rel_frames.

If the valid AUs continue beyond mute_rel_frames, the decoder fades in the output frames for a certain number of frames specified by the application. (*fade_in_frame*). Once the number of valid AU exceeds *fade_in_frame,* the decoding becomes normal.

If any invalid frame is received within fade_in_frames, the decoder triggers the fadeout and repeats this concealment process again.

Note	The concealment algorithm selects the attenuation factor to achieve smooth attenuation ramp for both fade out and fade in based on the selection of *fade_out_frames* and *fade_in_frames* respectively. Refer the relevant configuration sub-commands for range and default.

The following example demonstrates the concealment process involving fade-out, mute-release and fade out.



4. Example Test Bench

The supplied test bench consists of the following files:

- Test bench source files (found in `test/src`)
  - `xa_aac_dec_error_handler.c`
  - `xa_aac_dec_sample_testbench.c`
- Makefile to build the executable (`test/build`)
  - `makefile_testbench_sample`
- Sample parameter file to run the test bench (`test/build`)
  - `paramfilesimple.txt`

4.1 Making an Executable

To build the application:

1. Go to `test/build`.
2. In the console, enter:

```
xt-make -f makefile_testbench_sample clean dabplus
```

This will build the decoder example test bench `xa_dabplus_dec_test`.

4.2 Usage

The sample application executable can be run with command-line options or with a parameter file. The command-line usage is as follows:

```
      xt-run xa_dabplus_dec_test -ifile:<infile> -ofile:<outfile>
                    [-sfile:<side-info-file>]
                    [-b<bsformat>]
                    [-d<bdownsample>]
                    [-f<to_stereo>]
                    [-m<downmix>]
                    [-p<extsr>]
                    [-r<extbr>]
                    [-w<pcm_wdsz>]
                    [-x<sbr_signaling>]
                    [-cfo<fade_out_frames>]
                    [-cmr<mute_rel_frames>]
                    [-cfi<fade_in_frames>]
```

Where:

<infile>	Name of the AAC input file
<outfile>	Name of the output ".wav" file
<side-info-file>	Name of the side-info file in `rawdabplus` mode
<bsformat>	Bitstream format (`dabplus or rawdabplusr`)
<bdownsample>	Enables (1) or disables (0) downsampled SBR mode (disabled by default)
<to_stereo>	Enables (1) or disables (0) duplication of mono output to interleaved stereo (enabled by default)
<downmix>	Enables (1) or disables (0) SBR mono downmix (disabled by default).
<extsr>	The sample rate of a RAW bitstream (`-p 32000`)
<extbr>	The bit rate of the DAB+ audio super-frame stream (`-r128`).
<pcm_wdsz>	Output PCM word size – 16 or 24 (default).
<sbr_signaling>	Disable (0), enable (1) or turn SBR processing in auto mode (2) (auto by default).
<fade_out_frames>	Concealment fade-out slope length in frames/AUs. Valid values: 2 to 7. Default: 5.

`<mute_rel_frames>`	Concealment mute release length in frames/AUs. Valid values: 1 to 15. Default: 5.
`<fade_in_frames>`	Concealment fade-in slope length in frames/AUs. Valid values: 2 to 7. Default: 5.

Refer to the parameter definitions in Section 3.2 for a full description of their usage. Note that the space between the option name and the option value is optional.

If no command line arguments are given, the application reads the commands from the parameter file `paramfilesimple.txt`.

Following is the syntax for writing the `paramfilesimple.txt` file:

```
@Start

@Input_path <path to be appended to all input files>
@Output_path <path to be appended to all output files>
<command line 1>
<command line 2>
....
@Stop
```

The DAB+ Decoder can be run for multiple test files using the different command lines. The syntax for command lines in the parameter file is the same as the syntax for specifying options on the command line to the test bench program.

Note All the @<*command*>s should be at the first column of a line except the @New_line command.

Note All the @<*command*>s are case sensitive. If the command line in the parameter file has to be broken to two parts on two different lines use the @New_line command.

For example:

`<command line part 1>` `@New_line`

`<command line part 2>``.`

Note Blank lines will be ignored.

Note Individual lines can be commented out using "//" at the beginning of the line.

5. References

[1] *ETSI TS 102 563 V1.1.1 (2007-02) Digital Audio Broadcasting (DAB); Transport of Advanced Audio Coding (AAC) Audio.*