Fraunhofer
IIS

Advanced Audio Coding Decoder Library

MPEG-2 and MPEG-4

AAC Low-Complexity (AAC-LC),

High-Efficiency AAC v2 (HE-AAC v2),

AAC Low-Delay (AAC-LD), and

AAC Enhanced Low-Delay (AAC-ELD)

decoder

Fraunhofer Institut Integrierte Schaltungen IIS,
Fraunhofer Institute for Integrated Circuits IIS
http://www.iis.fraunhofer.de/amm

Revision 2.5.7 , August 15, 2013

Contents

Chapter 1

Introduction

1.1 Scope

This document describes the high-level interface and usage of the ISO/MPEG-2/4 AAC Decoder library developed by the Fraunhofer Institute for Integrated Circuits (IIS). Depending on the library configuration, it implements decoding of AAC-LC (Low-Complexity), HE-AAC (High-Efficiency AAC, v1 and v2), AAC-LD (Low-Delay) and AAC-ELD (Enhanced Low-Delay).

All references to SBR (Spectral Band Replication) are only applicable to HE-AAC and AAC-ELD versions of the library. All references to PS (Parametric Stereo) are only applicable to HE-AAC v2 versions of the library.

1.2 Decoder Basics

This document can only give a rough overview about the ISO/MPEG-2 and ISO/MPEG-4 AAC audio coding standard. To understand all the terms in this document, you are encouraged to read the following documents.

- ISO/IEC 13818-7 (MPEG-2 AAC), which defines the syntax of MPEG-2 AAC audio bitstreams.
- ISO/IEC 14496-3 (MPEG-4 AAC, subpart 1 and 4), which defines the syntax of MPEG-4 AAC audio bitstreams.
- Lutzky, Schuller, Gayer, Krämer, Wabnik, "A guideline to audio codec delay", 116th AES Convention, May 8, 2004

MPEG Advanced Audio Coding is based on a time-to-frequency mapping of the signal. The signal is partitioned into overlapping portions and transformed into frequency domain. The spectral components are then quantized and coded.

An MPEG2 or MPEG4 AAC audio bitstream is composed of frames. Contrary to MPEG-1/2 Layer-3 (mp3), the length of individual frames is not restricted to a fixed number of bytes, but can take on any length between 1 and 768 bytes.

Chapter 2

Library Usage

2.1 API Description

All API header files are located in the folder /include of the release package. They are described in detail in this document. All header files are provided for usage in C/C++ programs. The AAC decoder library API functions are located at aacdecoder_lib.h.

In binary releases the decoder core resides in statically linkable libraries called for example libAACdec.a, (Linux) or FDK_aacDec_lib (Microsoft Visual C++).

2.2 Calling Sequence

For decoding of ISO/MPEG-2/4 AAC or HE-AAC v2 bitstreams the following sequence is mandatory. Input read and output write functions as well as the corresponding open and close functions are left out, since they may be implemented differently according to the user's specific requirements. The example implementation in main.cpp uses file-based input/output, and in such case call mpegFileRead_Open() to open an input file and to allocate memory for the required structures, and the corresponding mpegFileRead_Close() to close opened files and to de-allocate associated structures. mpegFileRead_Open() tries to detect the bitstream format and in case of MPEG-4 file format or Raw Packets file format (a Fraunhofer IIS proprietary format) reads the Audio Specific Config data (ASC). An unsuccessful attempt to recognize the bitstream format requires the user to provide this information manually (see CommandLineUsage). For any other bitstream formats that are usually applicable in streaming applications, the decoder itself will try to synchronize and parse the given bitstream fragment using the FDK transport library. Hence, for streaming applications (without file access) this step is not necessary.

1. Call aacDecoder_Open() to open and retrieve a handle to a new AAC decoder instance.

   ```
   aacDecoderInfo = aacDecoder_Open(mpegFileRead_GetTransportType(hDataSrc), nrOfL
       ayers);
   ```

2. If out-of-band config data (Audio Specific Config (ASC) or Stream Mux Config (SMC)) is available, call aacDecoder_ConfigRaw() to pass it to the decoder and before the decoding process starts. If this data is not available in advance, the decoder will get it from the bitstream and configure itself while decoding with aacDecoder_DecodeFrame().

3. Begin decoding loop.

```
do {
```

4. Read data from bitstream file or stream into a client-supplied input buffer ("inBuffer" in main.cpp). If it is very small like just 4, aacDecoder_DecodeFrame() will repeatedly return AAC_DEC_NOT_ENOUGH_BITS until enough bits were fed by aacDecoder_Fill(). Only read data when this buffer has completely been processed and is then empty. For file-based input execute mpegFileRead_Read() or any other implementation with similar functionality.

5. Call aacDecoder_Fill() to fill the decoder's internal bitstream input buffer with the client-supplied external bitstream input buffer.

```
aacDecoder_Fill(aacDecoderInfo, inBuffer, bytesRead, bytesValid);
```

6. Call aacDecoder_DecodeFrame() which writes decoded PCM audio data to a client-supplied buffer. It is the client's responsibility to allocate a buffer which is large enough to hold this output data.

```
ErrorStatus = aacDecoder_DecodeFrame(aacDecoderInfo, TimeData, OUT_BUF_SIZE,
  flags);
```

   If the bitstream's configuration (number of channels, sample rate, frame size) is not known in advance, you may call aacDecoder_GetStreamInfo() to retrieve a structure containing this information and then initialize an audio output device. In the example main.cpp, if the number of channels or the sample rate has changed since program start or since the previously decoded frame, the audio output device will be re-initialized. If WAVE file output is chosen, a new WAVE file for each new configuration will be created.

7. Repeat steps 5 to 7 until no data to decode is available anymore, or if an error occured.

8. Call aacDecoder_Close() to de-allocate all AAC decoder and transport layer structures.

2.3 Buffer System

There are three main buffers in an AAC decoder application. One external input buffer to hold bitstream data from file I/O or elsewhere, one decoder-internal input buffer, and one to hold the decoded output PCM sample data, whereas this output buffer may overlap with the external input buffer.

The external input buffer is set in the example framework main.cpp and its size is defined by IN_BUF_SIZE. You may freely choose different sizes here. To feed the data to the decoder-internal input buffer, use the function aacDecoder_Fill(). This function returns important information about how many bytes in the external input buffer have not yet been copied into the internal input buffer (variable bytesValid). Once the external buffer has been fully copied, it can be re-filled again. In case you want to re-fill it when there are still unprocessed bytes (bytesValid is unequal 0), you would have to additionally perform a memcpy(), so that just means unnecessary computational overhead and therefore we recommend to re-fill the buffer only when bytesValid is 0.

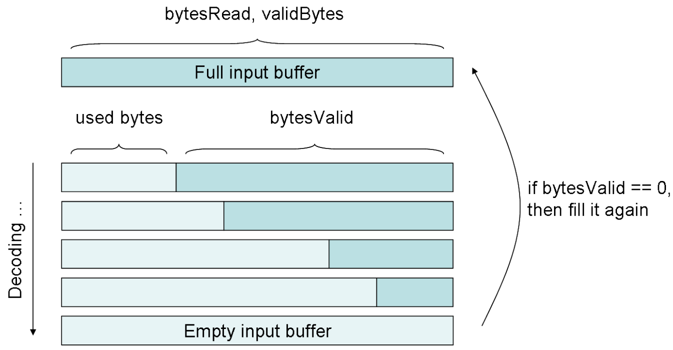


Figure 2.1: Lifecycle of the external input buffer

The size of the decoder-internal input buffer is set in tpdec_lib.h (see define TRANSPORTDEC_INBUF_-SIZE). You may choose a smaller size under the following considerations:

- each input channel requires 768 bytes
- the whole buffer must be of size 2^n

So for example a stereo decoder:

$$TRANSPORTDEC\_INBUF\_SIZE = 2 * 768 = 1536 => 2048$$

tpdec_lib.h and TRANSPORTDEC_INBUF_SIZE are not part of the decoder's library interface. Therefore only source-code clients may change this setting. If you received a library release, please ask us and we can change this in order to meet your memory requirements.

Chapter 3

Decoder audio output

3.1 Obtaining channel mapping information

The decoded audio output format is indicated by a set of variables of the CStreamInfo structure. While the members sampleRate, frameSize and numChannels might be quite self explaining, pChannelType and pChannelIndices might require some more detailed explanation.

These two arrays indicate what is each output channel supposed to be. Both array have CStreamInfo::numChannels cells. Each cell of pChannelType indicates the channel type, described in the enum AUDIO_CHANNEL_TYPE defined in FDK_audio.h. The cells of pChannelIndices indicate the sub index among the channels starting with 0 among all channels of the same audio channel type.

The indexing scheme is the same as for MPEG-2/4. Thus indices are counted upwards starting from the front direction (thus a center channel if any, will always be index 0). Then the indices count up, starting always with the left side, pairwise from front toward back. For detailed explanation, please refer to ISO/IEC 13818-7:2005(E), chapter 8.5.3.2.

In case a Program Config is included in the audio configuration, the channel mapping described within it will be adopted.

In case of MPEG-D Surround the channel mapping will follow the same criteria described in ISO/IEC 13818-7:2005(E), but adding corresponding top channels to the channel types front, side and back, in order to avoid any loss of information.

3.2 Changing the audio output format

The channel interleaving scheme and the actual channel order can be changed at runtime through the parameters AAC_PCM_OUTPUT_INTERLEAVED and AAC_PCM_OUTPUT_CHANNEL_MAPPING. See the description of those parameters and the decoder library function aacDecoder_SetParam() for more detail.

3.3 Channel mapping examples

The following examples illustrate the location of individual audio samples in the audio buffer that is passed to aacDecoder_DecodeFrame() and the expected data in the CStreamInfo structure which can be obtained by calling aacDecoder_GetStreamInfo().

3.3.1 Stereo

In case of AAC_PCM_OUTPUT_INTERLEAVED set to 0 and AAC_PCM_OUTPUT_CHANNEL_-MAPPING set to 1, a AAC-LC bit stream which has channelConfiguration = 2 in its audio specific config would lead to the following values in CStreamInfo:

CStreamInfo::numChannels = 2

CStreamInfo::pChannelType = { ACT_FRONT, ACT_FRONT }

CStreamInfo::pChannelIndices = { 0, 1 }

Since AAC_PCM_OUTPUT_INTERLEAVED is set to 0, the audio channels will be located as contiguous blocks in the output buffer as follows:

```
<left sample 0>  <left sample 1>  <left sample 2>  ... <left sample N>
<right sample 0> <right sample 1> <right sample 2> ... <right sample N>
```

Where N equals to CStreamInfo::frameSize .

3.3.2 Surround 5.1

In case of AAC_PCM_OUTPUT_INTERLEAVED set to 1 and AAC_PCM_OUTPUT_CHANNEL_-MAPPING set to 1, a AAC-LC bit stream which has channelConfiguration = 6 in its audio specific config, would lead to the following values in CStreamInfo:

CStreamInfo::numChannels = 6

CStreamInfo::pChannelType = { ACT_FRONT, ACT_FRONT, ACT_FRONT, ACT_LFE, ACT_BACK, ACT_BACK }

CStreamInfo::pChannelIndices = { 1, 2, 0, 0, 0, 1 }

Since AAC_PCM_OUTPUT_CHANNEL_MAPPING is 1, WAV file channel ordering will be used. For a 5.1 channel scheme, thus the channels would be: front left, front right, center, LFE, surround left, surround right. Thus the third channel is the center channel, receiving the index 0. The other front channels are front left, front right being placed as first and second channels with indices 1 and 2 correspondingly. There is only one LFE, placed as the fourth channel and index 0. Finally both surround channels get the type definition ACT_BACK, and the indices 0 and 1.

Since AAC_PCM_OUTPUT_INTERLEAVED is set to 1, the audio channels will be placed in the output buffer as follows:

```
<front left sample 0> <front right sample 0>
<center sample 0> <LFE sample 0>
<surround left sample 0> <surround right sample 0>

<front left sample 1> <front right sample 1>
<center sample 1> <LFE sample 1>
<surround left sample 1> <surround right sample 1>

...

<front left sample N> <front right sample N>
<center sample N> <LFE sample N>
<surround left sample N> <surround right sample N>
```

Where N equals to CStreamInfo::frameSize .

3.3.3 ARIB coding mode 2/1

In case of AAC_PCM_OUTPUT_INTERLEAVED set to 1 and AAC_PCM_OUTPUT_CHANNEL_-MAPPING set to 1, in case of a ARIB bit stream using coding mode 2/1 as described in ARIB STD-B32 Part 2 Version 2.1-E1, page 61, would lead to the following values in CStreamInfo:

CStreamInfo::numChannels = 3

CStreamInfo::pChannelType = { ACT_FRONT, ACT_FRONT,:: ACT_BACK }

CStreamInfo::pChannelIndices = { 0, 1, 0 }

The audio channels will be placed as follows in the audio output buffer:

```
<front left sample 0> <front right sample 0>  <mid surround sample 0>

<front left sample 1> <front right sample 1> <mid surround sample 1>

...

<front left sample N> <front right sample N> <mid surround sample N>

Where N equals to CStreamInfo::frameSize .
```

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Class Documentation

6.1 CStreamInfo Struct Reference

This structure gives information about the currently decoded audio data. All fields are read-only.

```
#include <aacdecoder_lib.h>
```

Public Attributes

- INT sampleRate
- INT frameSize
- INT numChannels
- AUDIO_CHANNEL_TYPE ∗ pChannelType
- UCHAR ∗ pChannelIndices
- INT aacSampleRate
- INT profile
- AUDIO_OBJECT_TYPE aot
- INT channelConfig
- INT bitRate
- INT aacSamplesPerFrame
- INT aacNumChannels
- AUDIO_OBJECT_TYPE extAot
- INT extSamplingRate
- UINT flags
- SCHAR epConfig
- INT numLostAccessUnits
- UINT numTotalBytes
- UINT numBadBytes
- UINT numTotalAccessUnits
- UINT numBadAccessUnits

6.1.1 Detailed Description

This structure gives information about the currently decoded audio data. All fields are read-only.

6.1.2 Member Data Documentation

6.1.2.1 INT CStreamInfo::aacNumChannels

The number of audio channels after AAC core processing (before PS or MPS processing). CAUTION: This are not the final number of output channels!

6.1.2.2 INT CStreamInfo::aacSampleRate

sampling rate in Hz without SBR (from configuration info).

6.1.2.3 INT CStreamInfo::aacSamplesPerFrame

Samples per frame for the AAC core (from ASC).

1024 or 960 for AAC-LC

512 or 480 for AAC-LD and AAC-ELD

6.1.2.4 AUDIO_OBJECT_TYPE CStreamInfo::aot

Audio Object Type (from ASC): is set to the appropriate value for MPEG-2 bitstreams (e. g. 2 for AAC-LC).

6.1.2.5 INT CStreamInfo::bitRate

Instantaneous bit rate.

6.1.2.6 INT CStreamInfo::channelConfig

Channel configuration (0: PCE defined, 1: mono, 2: stereo, ...

6.1.2.7 SCHAR CStreamInfo::epConfig

epConfig level (from ASC): only level 0 supported, -1 means no ER (e. g. AOT=2, MPEG-2 AAC, etc.)

6.1.2.8 AUDIO_OBJECT_TYPE CStreamInfo::extAot

Extension Audio Object Type (from ASC)

6.1.2.9 INT CStreamInfo::extSamplingRate

Extension sampling rate in Hz (from ASC)

6.1.2.10 UINT CStreamInfo::flags

Copy if internal flags. Only to be written by the decoder, and only to be read externally.

6.1.2.11 INT CStreamInfo::frameSize

The frame size of the decoded PCM audio signal.

1024 or 960 for AAC-LC

2048 or 1920 for HE-AAC (v2)

512 or 480 for AAC-LD and AAC-ELD

6.1.2.12 UINT CStreamInfo::numBadAccessUnits

This is the number of total access units that were considered with errors from numTotalBytes.

6.1.2.13 UINT CStreamInfo::numBadBytes

This is the number of total bytes that were considered with errors from numTotalBytes.

6.1.2.14 INT CStreamInfo::numChannels

The number of output audio channels in the decoded and interleaved PCM audio signal.

6.1.2.15 INT CStreamInfo::numLostAccessUnits

This integer will reflect the estimated amount of lost access units in case aacDecoder_DecodeFrame() returns AAC_DEC_TRANSPORT_SYNC_ERROR. It will be < 0 if the estimation failed.

6.1.2.16 UINT CStreamInfo::numTotalAccessUnits

This is the number of total access units that have passed through the decoder.

6.1.2.17 UINT CStreamInfo::numTotalBytes

This is the number of total bytes that have passed through the decoder.

6.1.2.18 UCHAR* CStreamInfo::pChannelIndices

Audio channel index for each output audio channel. See ISO/IEC 13818-7:2005(E), 8.5.3.2 Explicit channel mapping using a program_config_element()

6.1.2.19 AUDIO_CHANNEL_TYPE* CStreamInfo::pChannelType

Audio channel type of each output audio channel.

6.1.2.20 INT CStreamInfo::profile

MPEG-2 profile (from file header) (-1: not applicable (e. g. MPEG-4)).

6.1.2.21 INT CStreamInfo::sampleRate

The samplerate in Hz of the fully decoded PCM audio signal (after SBR processing).

The documentation for this struct was generated from the following file:

- aacdecoder_lib.h

Chapter 7

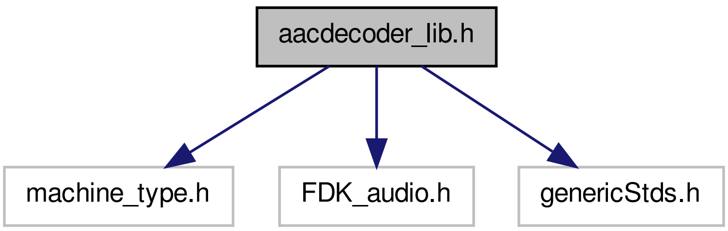
File Documentation

7.1 aacdecoder_lib.h File Reference

FDK AAC decoder library interface header file.

```
#include "machine_type.h"
#include "FDK_audio.h"
#include "genericStds.h"
```

Include dependency graph for aacdecoder_lib.h:



Classes

- struct CStreamInfo
  This structure gives information about the currently decoded audio data. All fields are read-only.

Defines

- #define IS_INIT_ERROR(err) ((((err)>=aac_dec_init_error_start) && ((err)<=aac_dec_init_-error_end)) ? 1 : 0)

- #define IS_DECODE_ERROR(err) ((((err)>=aac_dec_decode_error_start) && ((err)<=aac_dec_-decode_error_end)) ? 1 : 0)
- #define IS_OUTPUT_VALID(err) (((err) == AAC_DEC_OK) || IS_DECODE_ERROR(err))
- #define AACDEC_CONCEAL 1
- #define AACDEC_FLUSH 2
- #define AACDEC_INTR 4
- #define AACDEC_CLRHIST 8

Typedefs

- typedef struct AAC_DECODER_INSTANCE * HANDLE_AACDECODER

Enumerations

- enum AAC_DECODER_ERROR {
  AAC_DEC_OK = 0x0000,
  AAC_DEC_OUT_OF_MEMORY = 0x0002,
  AAC_DEC_UNKNOWN = 0x0005,
  aac_dec_sync_error_start = 0x1000,
  AAC_DEC_TRANSPORT_SYNC_ERROR = 0x1001,
  AAC_DEC_NOT_ENOUGH_BITS = 0x1002,
  aac_dec_sync_error_end = 0x1FFF,
  aac_dec_init_error_start = 0x2000,
  AAC_DEC_INVALID_HANDLE = 0x2001,
  AAC_DEC_UNSUPPORTED_AOT = 0x2002,
  AAC_DEC_UNSUPPORTED_FORMAT = 0x2003,
  AAC_DEC_UNSUPPORTED_ER_FORMAT = 0x2004,
  AAC_DEC_UNSUPPORTED_EPCONFIG = 0x2005,
  AAC_DEC_UNSUPPORTED_MULTILAYER = 0x2006,
  AAC_DEC_UNSUPPORTED_CHANNELCONFIG = 0x2007,
  AAC_DEC_UNSUPPORTED_SAMPLINGRATE = 0x2008,
  AAC_DEC_INVALID_SBR_CONFIG = 0x2009,
  AAC_DEC_SET_PARAM_FAIL = 0x200A,
  AAC_DEC_NEED_TO_RESTART = 0x200B,
  aac_dec_init_error_end = 0x2FFF,
  aac_dec_decode_error_start = 0x4000,
  AAC_DEC_TRANSPORT_ERROR = 0x4001,
  AAC_DEC_PARSE_ERROR = 0x4002,
  AAC_DEC_UNSUPPORTED_EXTENSION_PAYLOAD = 0x4003,
  AAC_DEC_DECODE_FRAME_ERROR = 0x4004,
  AAC_DEC_CRC_ERROR = 0x4005,
  AAC_DEC_INVALID_CODE_BOOK = 0x4006,
  AAC_DEC_UNSUPPORTED_PREDICTION = 0x4007,

AAC_DEC_UNSUPPORTED_CCE = 0x4008,
AAC_DEC_UNSUPPORTED_LFE = 0x4009,
AAC_DEC_UNSUPPORTED_GAIN_CONTROL_DATA = 0x400A,
AAC_DEC_UNSUPPORTED_SBA = 0x400B,
AAC_DEC_TNS_READ_ERROR = 0x400C,
AAC_DEC_RVLC_ERROR = 0x400D,
aac_dec_decode_error_end = 0x4FFF,
aac_dec_anc_data_error_start = 0x8000,
AAC_DEC_ANC_DATA_ERROR = 0x8001,
AAC_DEC_TOO_SMALL_ANC_BUFFER = 0x8002,
AAC_DEC_TOO_MANY_ANC_ELEMENTS = 0x8003,
aac_dec_anc_data_error_end = 0x8FFF }

AAC decoder error codes.

- enum AACDEC_PARAM {
AAC_PCM_OUTPUT_INTERLEAVED = 0x0000,
AAC_PCM_DUAL_CHANNEL_OUTPUT_MODE = 0x0002,
AAC_PCM_OUTPUT_CHANNEL_MAPPING = 0x0003,
AAC_PCM_LIMITER_ENABLE = 0x0004,
AAC_PCM_LIMITER_ATTACK_TIME = 0x0005,
AAC_PCM_LIMITER_RELEAS_TIME = 0x0006,
AAC_PCM_MIN_OUTPUT_CHANNELS = 0x0011,
AAC_PCM_MAX_OUTPUT_CHANNELS = 0x0012,
AAC_CONCEAL_METHOD = 0x0100,
AAC_DRC_BOOST_FACTOR = 0x0200,
AAC_DRC_ATTENUATION_FACTOR = 0x0201,
AAC_DRC_REFERENCE_LEVEL = 0x0202,
AAC_DRC_HEAVY_COMPRESSION = 0x0203,
AAC_QMF_LOWPOWER = 0x0300,
AAC_MPEGS_ENABLE = 0x0500,
AAC_TPDEC_CLEAR_BUFFER = 0x0603 }

AAC decoder setting parameters.

Functions

- LINKSPEC_H AAC_DECODER_ERROR aacDecoder_AncDataInit (HANDLE_AACDECODER self, UCHAR ∗buffer, int size)

  Initialize ancillary data buffer.

- LINKSPEC_H AAC_DECODER_ERROR aacDecoder_AncDataGet (HANDLE_AACDECODER self, int index, UCHAR ∗∗ptr, int ∗size)

  Get one ancillary data element.

- LINKSPEC_H AAC_DECODER_ERROR aacDecoder_SetParam (const HANDLE_-AACDECODER self, const AACDEC_PARAM param, const INT value)

  Set one single decoder parameter.

- LINKSPEC_H AAC_DECODER_ERROR aacDecoder_GetFreeBytes (const HANDLE_-AACDECODER self, UINT ∗pFreeBytes)

  Get free bytes inside decoder internal buffer.

- LINKSPEC_H HANDLE_AACDECODER aacDecoder_Open (TRANSPORT_TYPE transportFmt, UINT nrOfLayers)

  Open an AAC decoder instance.

- LINKSPEC_H AAC_DECODER_ERROR aacDecoder_ConfigRaw (HANDLE_AACDECODER self, UCHAR ∗conf[ ], const UINT length[ ])

  Explicitly configure the decoder by passing a raw AudioSpecificConfig (ASC) or a StreamMuxConfig (SMC), contained in a binary buffer. This is required for MPEG-4 and Raw Packets file format bitstreams as well as for LATM bitstreams with no in-band SMC. If the transport format is LATM with or without LOAS, configuration is assumed to be an SMC, for all other file formats an ASC.

- LINKSPEC_H AAC_DECODER_ERROR aacDecoder_Fill (HANDLE_AACDECODER self, UCHAR ∗pBuffer[ ], const UINT bufferSize[ ], UINT ∗bytesValid)

  Fill AAC decoder's internal input buffer with bitstream data from the external input buffer. The function only copies such data as long as the decoder-internal input buffer is not full. So it grabs whatever it can from pBuffer and returns information (bytesValid) so that at a subsequent call of aacDecoder_Fill(), the right position in pBuffer can be determined to grab the next data.

- LINKSPEC_H AAC_DECODER_ERROR aacDecoder_DecodeFrame (HANDLE_-AACDECODER self, INT_PCM ∗pTimeData, const INT timeDataSize, const UINT flags)

  Decode one audio frame.

- LINKSPEC_H void aacDecoder_Close (HANDLE_AACDECODER self)

  De-allocate all resources of an AAC decoder instance.

- LINKSPEC_H CStreamInfo ∗ aacDecoder_GetStreamInfo (HANDLE_AACDECODER self)

  Get CStreamInfo handle from decoder.

- LINKSPEC_H INT aacDecoder_GetLibInfo (LIB_INFO ∗info)

  Get decoder library info.

7.1.1 Detailed Description

FDK AAC decoder library interface header file.

7.1.2 Define Documentation

7.1.2.1 #define AACDEC_CLRHIST 8

Flag for aacDecoder_DecodeFrame(): Clear all signal delay lines and history buffers. Caution: This can cause discontinuities in the output signal.

7.1.2.2 #define AACDEC_CONCEAL 1

Flag for aacDecoder_DecodeFrame(): do not consider new input data. Do concealment.

7.1.2.3 #define AACDEC_FLUSH 2

Flag for aacDecoder_DecodeFrame(): Do not consider new input data. Flush filterbanks (output delayed audio).

7.1.2.4 #define AACDEC_INTR 4

Flag for aacDecoder_DecodeFrame(): Signal an input bit stream data discontinuity. Resync any internals as necessary.

7.1.2.5 #define IS_DECODE_ERROR(*err*) ((((err)>=aac_dec_decode_error_start) && ((err)<=aac_dec_decode_error_end)) ? 1 : 0)

Macro to identify decode errors.

7.1.2.6 #define IS_INIT_ERROR(*err*) ((((err)>=aac_dec_init_error_start) && ((err)<=aac_dec_init_error_end)) ? 1 : 0)

Macro to identify initialization errors.

7.1.2.7 #define IS_OUTPUT_VALID(*err*) (((err) == AAC_DEC_OK) || IS_DECODE_ERROR(err))

Macro to identify if the audio output buffer contains valid samples after calling aacDecoder_DecodeFrame().

7.1.3 Typedef Documentation

7.1.3.1 typedef struct AAC_DECODER_INSTANCE* HANDLE_AACDECODER

7.1.4 Enumeration Type Documentation

7.1.4.1 enum AAC_DECODER_ERROR

AAC decoder error codes.

Enumerator:

AAC_DEC_OK No error occured. Output buffer is valid and error free.

AAC_DEC_OUT_OF_MEMORY Heap returned NULL pointer. Output buffer is invalid.

AAC_DEC_UNKNOWN Error condition is of unknown reason, or from a another module. Output buffer is invalid.

aac_dec_sync_error_start

AAC_DEC_TRANSPORT_SYNC_ERROR The transport decoder had syncronisation problems. Do not exit decoding. Just feed new bitstream data.

AAC_DEC_NOT_ENOUGH_BITS The input buffer ran out of bits.

aac_dec_sync_error_end

aac_dec_init_error_start

AAC_DEC_INVALID_HANDLE The handle passed to the function call was invalid (NULL).

AAC_DEC_UNSUPPORTED_AOT The AOT found in the configuration is not supported.

AAC_DEC_UNSUPPORTED_FORMAT The bitstream format is not supported.

AAC_DEC_UNSUPPORTED_ER_FORMAT The error resilience tool format is not supported.

AAC_DEC_UNSUPPORTED_EPCONFIG The error protection format is not supported.

AAC_DEC_UNSUPPORTED_MULTILAYER More than one layer for AAC scalable is not supported.

AAC_DEC_UNSUPPORTED_CHANNELCONFIG The channel configuration (either number or arrangement) is not supported.

AAC_DEC_UNSUPPORTED_SAMPLINGRATE The sample rate specified in the configuration is not supported.

AAC_DEC_INVALID_SBR_CONFIG The SBR configuration is not supported.

AAC_DEC_SET_PARAM_FAIL The parameter could not be set. Either the value was out of range or the parameter does not exist.

AAC_DEC_NEED_TO_RESTART The decoder needs to be restarted, since the requiered configuration change cannot be performed.

aac_dec_init_error_end

aac_dec_decode_error_start

AAC_DEC_TRANSPORT_ERROR The transport decoder encountered an unexpected error.

AAC_DEC_PARSE_ERROR Error while parsing the bitstream. Most probably it is corrupted, or the system crashed.

AAC_DEC_UNSUPPORTED_EXTENSION_PAYLOAD Error while parsing the extension payload of the bitstream. The extension payload type found is not supported.

AAC_DEC_DECODE_FRAME_ERROR The parsed bitstream value is out of range. Most probably the bitstream is corrupt, or the system crashed.

AAC_DEC_CRC_ERROR The embedded CRC did not match.

AAC_DEC_INVALID_CODE_BOOK An invalid codebook was signalled. Most probably the bitstream is corrupt, or the system crashed.

AAC_DEC_UNSUPPORTED_PREDICTION Predictor found, but not supported in the AAC Low Complexity profile. Most probably the bitstream is corrupt, or has a wrong format.

AAC_DEC_UNSUPPORTED_CCE A CCE element was found which is not supported. Most probably the bitstream is corrupt, or has a wrong format.

AAC_DEC_UNSUPPORTED_LFE A LFE element was found which is not supported. Most probably the bitstream is corrupt, or has a wrong format.

AAC_DEC_UNSUPPORTED_GAIN_CONTROL_DATA Gain control data found but not supported. Most probably the bitstream is corrupt, or has a wrong format.

AAC_DEC_UNSUPPORTED_SBA SBA found, but currently not supported in the BSAC profile.

AAC_DEC_TNS_READ_ERROR Error while reading TNS data. Most probably the bitstream is corrupt or the system crashed.

AAC_DEC_RVLC_ERROR Error while decoding error resillient data.

aac_dec_decode_error_end

aac_dec_anc_data_error_start

AAC_DEC_ANC_DATA_ERROR Non severe error concerning the ancillary data handling.

AAC_DEC_TOO_SMALL_ANC_BUFFER The registered ancillary data buffer is too small to receive the parsed data.

AAC_DEC_TOO_MANY_ANC_ELEMENTS More than the allowed number of ancillary data elements should be written to buffer.

aac_dec_anc_data_error_end

7.1.4.2 enum AACDEC_PARAM

AAC decoder setting parameters.

Enumerator:

AAC_PCM_OUTPUT_INTERLEAVED PCM output mode (1: interleaved (default); 0: not interleaved).

AAC_PCM_DUAL_CHANNEL_OUTPUT_MODE Defines how the decoder processes two channel signals:

0: Leave both signals as they are (default).

1: Create a dual mono output signal from channel 1.

2: Create a dual mono output signal from channel 2.

3: Create a dual mono output signal by mixing both channels (L' = R' = 0.5∗Ch1 + 0.5∗Ch2).

AAC_PCM_OUTPUT_CHANNEL_MAPPING Output buffer channel ordering. 0: MPEG PCE style order, 1: WAV file channel order (default).

AAC_PCM_LIMITER_ENABLE Enable signal level limiting.

-1: Auto-config. Enable limiter for all non-lowdelay configurations by default.

0: Disable limiter in general.

1: Enable limiter always. It is recommended to call the decoder with a AACDEC_CLRHIST flag to reset all states when the limiter switch is changed explicitly.

AAC_PCM_LIMITER_ATTACK_TIME Signal level limiting attack time in ms. Default confguration is 15 ms. Adjustable range from 1 ms to 15 ms.

AAC_PCM_LIMITER_RELEAS_TIME Signal level limiting release time in ms. Default configuration is 50 ms. Adjustable time must be larger than 0 ms.

AAC_PCM_MIN_OUTPUT_CHANNELS Minimum number of PCM output channels. If higher than the number of encoded audio channels, a simple channel extension is applied.

-1, 0: Disable channel extenstion feature. The decoder output contains the same number of channels as the encoded bitstream.

1: This value is currently needed only together with the mix-down feature. See AAC_PCM_MAX_OUTPUT_CHANNELS and note 2 below.

2: Encoded mono signals will be duplicated to achieve a 2/0/0.0 channel output configuration.

6: The decoder trys to reorder encoded signals with less than six channels to achieve a 3/0/2.1 channel output signal. Missing channels will be filled with a zero signal. If reordering is not possible the empty channels will simply be appended. Only available if instance is configured to support multichannel output.

8: The decoder trys to reorder encoded signals with less than eight channels to achieve a 3/0/4.1 channel output signal. Missing channels will be filled with a zero signal. If reordering is not possible the empty channels will simply be appended. Only available if instance is configured to support multichannel output.

NOTE:

1. The channel signalling (CStreamInfo::pChannelType and CStreamInfo::pChannelIndices) will not be modified. Added empty channels will be signalled with channel type AUDIO_-CHANNEL_TYPE::ACT_NONE.

2. If the parameter value is greater than that of AAC_PCM_MAX_OUTPUT_CHANNELS both will be set to the same value.

3. This parameter does not affect MPEG Surround processing.

AAC_PCM_MAX_OUTPUT_CHANNELS Maximum number of PCM output channels. If lower than the number of encoded audio channels, downmixing is applied accordingly. If dedicated metadata is available in the stream it will be used to achieve better mixing results.

-1, 0: Disable downmixing feature. The decoder output contains the same number of channels as the encoded bitstream.

1: All encoded audio configurations with more than one channel will be mixed down to one mono output signal.

2: The decoder performs a stereo mix-down if the number encoded audio channels is greater than two.

6: If the number of encoded audio channels is greater than six the decoder performs a mix-down to meet the target output configuration of 3/0/2.1 channels. Only available if instance is configured to support multichannel output.

8: This value is currently needed only together with the channel extension feature. See AAC_-PCM_MIN_OUTPUT_CHANNELS and note 2 below. Only available if instance is configured to support multichannel output.

NOTE:

1. Down-mixing of any seven or eight channel configuration not defined in ISO/IEC 14496-3 PDAM 4 is not supported by this software version.

2. If the parameter value is greater than zero but smaller than AAC_PCM_MIN_OUTPUT_-CHANNELS both will be set to same value.

3. The operating mode of the MPEG Surround module will be set accordingly.

4. Setting this param with any value will disable the binaural processing of the MPEG Surround module (AAC_MPEGS_BINAURAL_ENABLE=0).

AAC_CONCEAL_METHOD Error concealment: Processing method.

0: Spectral muting.

1: Noise substitution (see CONCEAL_NOISE).

2: Energy interpolation (adds additional signal delay of one frame, see CONCEAL_INTER).

AAC_DRC_BOOST_FACTOR Dynamic Range Control: Scaling factor for boosting gain values. Defines how the boosting DRC factors (conveyed in the bitstream) will be applied to the decoded signal. The valid values range from 0 (don't apply boost factors) to 127 (fully apply all boosting factors).

AAC_DRC_ATTENUATION_FACTOR Dynamic Range Control: Scaling factor for attenuating gain values. Same as AAC_DRC_BOOST_FACTOR but for attenuating DRC factors.

AAC_DRC_REFERENCE_LEVEL Dynamic Range Control: Target reference level. Defines the level below full-scale (quantized in steps of 0.25dB) to which the output audio signal will be normalized to by the DRC module. The valid values range from 0 (full-scale) to 127 (31.75 dB below full-scale). The value smaller than 0 switches off normalization.

AAC_DRC_HEAVY_COMPRESSION Dynamic Range Control: En-/Disable DVB specific heavy compression (aka RF mode). If set to 1, the decoder will apply the compression values from the DVB specific ancillary data field. At the same time the MPEG-4 Dynamic Range Control tool will be disabled. By default heavy compression is disabled.

AAC_QMF_LOWPOWER Quadrature Mirror Filter (QMF) Bank processing mode.
-1: Use internal default. Implies MPEG Surround partially complex accordingly.
0: Use complex QMF data mode.
1: Use real (low power) QMF data mode.

AAC_MPEGS_ENABLE MPEG Surround: Allow/Disable decoding of MPS content. Available only for decoders with MPEG Surround support.

AAC_TPDEC_CLEAR_BUFFER Clear internal bit stream buffer of transport layers. The decoder will start decoding at new data passed after this event and any previous data is discarded.

7.1.5 Function Documentation

7.1.5.1 LINKSPEC_H AAC_DECODER_ERROR aacDecoder_AncDataGet (HANDLE_AACDECODER *self*, int *index*, UCHAR ** *ptr*, int * *size*)

Get one ancillary data element.

Parameters

self AAC decoder handle.
index Index of the ancillary data element to get.
ptr Pointer to a buffer receiving a pointer to the requested ancillary data element.
size Pointer to a buffer receiving the length of the requested ancillary data element.

Returns

Error code.

7.1.5.2 LINKSPEC_H AAC_DECODER_ERROR aacDecoder_AncDataInit (HANDLE_AACDECODER *self*, UCHAR * *buffer*, int *size*)

Initialize ancillary data buffer.

Parameters

self AAC decoder handle.
buffer Pointer to (external) ancillary data buffer.
size Size of the buffer pointed to by buffer.

Returns

Error code.

7.1.5.3 LINKSPEC_H void aacDecoder_Close (HANDLE_AACDECODER *self*)

De-allocate all resources of an AAC decoder instance.

Parameters

self AAC decoder handle.

Returns

void

7.1.5.4 LINKSPEC_H AAC_DECODER_ERROR aacDecoder_ConfigRaw (HANDLE_AACDECODER *self,* UCHAR ∗ *conf[ ],* const UINT *length[ ]*)

Explicitly configure the decoder by passing a raw AudioSpecificConfig (ASC) or a StreamMuxConfig (SMC), contained in a binary buffer. This is required for MPEG-4 and Raw Packets file format bitstreams as well as for LATM bitstreams with no in-band SMC. If the transport format is LATM with or without LOAS, configuration is assumed to be an SMC, for all other file formats an ASC.

Parameters

self AAC decoder handle.

conf Pointer to an unsigned char buffer containing the binary configuration buffer (either ASC or SMC).

length Length of the configuration buffer in bytes.

Returns

Error code.

7.1.5.5 LINKSPEC_H AAC_DECODER_ERROR aacDecoder_DecodeFrame (HANDLE_AACDECODER *self,* INT_PCM ∗ *pTimeData,* const INT *timeDataSize,* const UINT *flags*)

Decode one audio frame.

Parameters

self AAC decoder handle.

pTimeData Pointer to external output buffer where the decoded PCM samples will be stored into.

flags Bit field with flags for the decoder:

(flags & AACDEC_CONCEAL) == 1: Do concealment.

(flags & AACDEC_FLUSH) == 2: Discard input data. Flush filter banks (output delayed audio).

(flags & AACDEC_INTR) == 4: Input data is discontinuous. Resynchronize any internals as necessary.

Returns

Error code.

7.1.5.6 LINKSPEC_H AAC_DECODER_ERROR aacDecoder_Fill (HANDLE_AACDECODER *self,* UCHAR ∗ *pBuffer[ ],* const UINT *bufferSize[ ],* UINT ∗ *bytesValid*)

Fill AAC decoder's internal input buffer with bitstream data from the external input buffer. The function only copies such data as long as the decoder-internal input buffer is not full. So it grabs whatever it can from pBuffer and returns information (bytesValid) so that at a subsequent call of aacDecoder_Fill(), the right position in pBuffer can be determined to grab the next data.

Parameters

self AAC decoder handle.

pBuffer Pointer to external input buffer.

bufferSize Size of external input buffer. This argument is required because decoder-internally we need the information to calculate the offset to pBuffer, where the next available data is, which is then fed into the decoder-internal buffer (as much as possible). Our example framework implementation fills the buffer at pBuffer again, once it contains no available valid bytes anymore (meaning bytesValid equal 0).

bytesValid Number of bitstream bytes in the external bitstream buffer that have not yet been copied into the decoder's internal bitstream buffer by calling this function. The value is updated according to the amount of newly copied bytes.

Returns

Error code.

7.1.5.7 LINKSPEC_H AAC_DECODER_ERROR aacDecoder_GetFreeBytes (const HANDLE_AACDECODER *self,* UINT ∗ *pFreeBytes*)

Get free bytes inside decoder internal buffer.

Parameters

self Handle of AAC decoder instance

pFreeBytes Pointer to variable receving amount of free bytes inside decoder internal buffer

Returns

Error code

7.1.5.8 LINKSPEC_H INT aacDecoder_GetLibInfo (LIB_INFO ∗ *info*)

Get decoder library info.

Parameters

info Pointer to an allocated LIB_INFO structure.

Returns

0 on success

7.1.5.9 LINKSPEC_H CStreamInfo∗ aacDecoder_GetStreamInfo (HANDLE_AACDECODER *self*)

Get CStreamInfo handle from decoder.

Parameters

self AAC decoder handle.

Returns

Reference to requested CStreamInfo.

7.1.5.10 LINKSPEC_H HANDLE_AACDECODER aacDecoder_Open (TRANSPORT_TYPE *transportFmt,* UINT *nrOfLayers*)

Open an AAC decoder instance.

Parameters

transportFmt The transport type to be used

Returns

AAC decoder handle

7.1.5.11 LINKSPEC_H AAC_DECODER_ERROR aacDecoder_SetParam (const HANDLE_AACDECODER *self,* const AACDEC_PARAM *param,* const INT *value*)

Set one single decoder parameter.

Parameters

self AAC decoder handle.

param Parameter to be set.

value Parameter value.

Returns

Error code.

Index