

# Advanced Audio Coding Encoder Library

MPEG-2 and MPEG-4 AAC Low-Complexity,

MPEG-4 High-Efficiency AAC v2

MPEG-4 Enhanced Low Delay AAC

encoder

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

## **Disclaimer**

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. Product and corporate names may be trademarks or registered trademarks of other companies. They are used for explanation only, with no intent to infringe. All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Scope	1
1.2	Encoder Basics	1
<b>2</b>	<b>Library Usage</b>	<b>3</b>
2.1	API Files	3
2.2	Calling Sequence	3
2.3	Encoder Instance Allocation	4
2.4	Input/Output Arguments	5
2.4.1	Provide Buffer Descriptors	5
2.4.2	Provide Input/Output Argument Lists	6
2.5	Feed Input Buffer	6
2.6	Output Bitstream Data	6
2.7	Meta Data Configuration	7
2.8	Encoder Reconfiguration	7
2.9	Encoder Parametrization	7
2.9.1	Mandatory Encoder Parameters	8
2.9.2	Channel Mode Configuration	8
2.9.3	Audio Quality Considerations	8
2.10	Audio Channel Configuration	8
2.11	Supported Bitrates	10
2.12	Recommended Sampling Rate and Bitrate Combinations	10
2.12.1	AAC-LC, HE-AAC, HE-AACv2.	10
2.12.2	AAC-LD, AAC-ELD, AAC-ELD with SBR.	11
<b>3</b>	<b>Encoder Behaviour</b>	<b>13</b>
3.1	Bandwidth	13
3.2	Frame Sizes & Bit Reservoir	13
3.2.1	Estimating Average Frame Sizes	14

---

3.3	Encoder Tools . . . . .	14
<b>4</b>	<b>Command-line Usage</b>	<b>15</b>
4.1	Arguments . . . . .	15
4.1.1	Mandatory Arguments . . . . .	15
4.1.2	Optional Arguments . . . . .	15
<b>5</b>	<b>Class Index</b>	<b>19</b>
5.1	Class List . . . . .	19
<b>6</b>	<b>File Index</b>	<b>21</b>
6.1	File List . . . . .	21
<b>7</b>	<b>Class Documentation</b>	<b>23</b>
7.1	AACENC_BufDesc Struct Reference . . . . .	23
7.1.1	Detailed Description . . . . .	23
7.1.2	Member Data Documentation . . . . .	23
7.1.2.1	bufElSizes . . . . .	23
7.1.2.2	bufferIdentifiers . . . . .	23
7.1.2.3	bufs . . . . .	23
7.1.2.4	bufSizes . . . . .	24
7.1.2.5	numBufs . . . . .	24
7.2	AACENC_InArgs Struct Reference . . . . .	24
7.2.1	Detailed Description . . . . .	24
7.2.2	Member Data Documentation . . . . .	24
7.2.2.1	numAncBytes . . . . .	24
7.2.2.2	numInSamples . . . . .	24
7.3	AACENC_InfoStruct Struct Reference . . . . .	25
7.3.1	Detailed Description . . . . .	25
7.3.2	Member Data Documentation . . . . .	25
7.3.2.1	confBuf . . . . .	25
7.3.2.2	confSize . . . . .	25
7.3.2.3	encoderDelay . . . . .	25
7.3.2.4	frameLength . . . . .	25
7.3.2.5	inBufFillLevel . . . . .	25
7.3.2.6	inputChannels . . . . .	26
7.3.2.7	maxAncBytes . . . . .	26
7.3.2.8	maxOutBufBytes . . . . .	26

---

---

7.4	AACENC_MetaData Struct Reference	26
7.4.1	Detailed Description	26
7.4.2	Member Data Documentation	26
7.4.2.1	centerMixLevel	26
7.4.2.2	comp_profile	27
7.4.2.3	comp_TargetRefLevel	27
7.4.2.4	dolbySurroundMode	27
7.4.2.5	drc_profile	27
7.4.2.6	drc_TargetRefLevel	27
7.4.2.7	ETSI_DmxLvl_present	27
7.4.2.8	PCE_mixdown_idx_present	27
7.4.2.9	prog_ref_level	27
7.4.2.10	prog_ref_level_present	27
7.4.2.11	surroundMixLevel	28
7.5	AACENC_OutArgs Struct Reference	28
7.5.1	Detailed Description	28
7.5.2	Member Data Documentation	28
7.5.2.1	numAncBytes	28
7.5.2.2	numInSamples	28
7.5.2.3	numOutBytes	28
<b>8</b>	<b>File Documentation</b>	<b>29</b>
8.1	aacenc_lib.h File Reference	29
8.1.1	Detailed Description	32
8.1.2	Typedef Documentation	32
8.1.2.1	HANDLE_AACENCODER	32
8.1.3	Enumeration Type Documentation	32
8.1.3.1	AACENC_BufferIdentifier	32
8.1.3.2	AACENC_CTRLFLAGS	33
8.1.3.3	AACENC_ERROR	33
8.1.3.4	AACENC_METADATA_DRC_PROFILE	34
8.1.3.5	AACENC_PARAM	34
8.1.4	Function Documentation	36
8.1.4.1	aacEncClose	36
8.1.4.2	aacEncEncode	36
8.1.4.3	aacEncGetLibInfo	37
8.1.4.4	aacEncInfo	38

---

8.1.4.5	aacEncoder_GetParam . . . . .	38
8.1.4.6	aacEncoder_SetParam . . . . .	38
8.1.4.7	aacEncOpen . . . . .	39
8.2	main.cpp File Reference . . . . .	40
8.2.1	Detailed Description . . . . .	40
8.2.2	Function Documentation . . . . .	40
8.2.2.1	main . . . . .	40

---

# Chapter 1

## Introduction

### 1.1 Scope

This document describes the high-level interface and usage of the ISO/MPEG-2/4 AAC Encoder library developed by the Fraunhofer Institute for Integrated Circuits (IIS).

The library implements encoding on the basis of the MPEG-2 and MPEG-4 AAC Low-Complexity standard, and depending on the library's configuration, MPEG-4 High-Efficiency AAC v2 and/or AAC-ELD standard.

All references to SBR (Spectral Band Replication) are only applicable to HE-AAC or 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 Encoder 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, subparts 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 MPEG-2 or MPEG-4 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 Files

All API header files are located in the folder /include of the release package. All header files are provided for usage in C/C++ programs. The AAC encoder library API functions are located at [aacenc\\_lib.h](#).

In binary releases the encoder core resides in statically linkable libraries called for example libAACenc.a/libFDK.a (LINUX) or FDK\_fastaacLib.lib (MS Visual C++) for the plain AAC-LC core encoder and libSBRenc.a (LINUX) or FDK\_sbrEncLib.lib (MS Visual C++) for the SBR (Spectral Band Replication) and PS (Parametric Stereo) modules.

### 2.2 Calling Sequence

For encoding of ISO/MPEG-2/4 AAC 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.

1. Call [aacEncOpen\(\)](#) to allocate encoder instance with required [configuration](#).

```
HANDLE_AACENCODER hAacEncoder = NULL; /* encoder handle */
if ( (ErrorStatus = aacEncOpen(&hAacEncoder,0,0)) != AACENC_OK ) {
```

2. Call [aacEncoder\\_SetParam\(\)](#) for each parameter to be set. AOT, samplingrate, channelMode, bitrate and transport type are [mandatory](#).

```
ErrorStatus = aacEncoder_SetParam(hAacEncoder, parameter, value);
```

3. Call [aacEncEncode\(\)](#) with NULL parameters to [initialize](#) encoder instance with present parameter set.

```
ErrorStatus = aacEncEncode(hAacEncoder, NULL, NULL, NULL, NULL);
```

4. Call [aacEncInfo\(\)](#) to retrieve a configuration data block to be transmitted out of band. This is required when using RFC3640 or RFC3016 like transport.

```
AACENC_InfoStruct encInfo;
ErrorStatus = aacEncInfo(hAacEncoder, &encInfo);
```

## 5. Encode input audio data in loop.

```
do
{
```

Feed [input buffer](#) with new audio data and provide input/output [arguments](#) to `aacEncEncode()`.

```
    ErrorStatus = aacEncEncode(hAacEncoder,
                              &inBufDesc,
                              &outBufDesc,
                              &inargs,
                              &outargs);
```

Write [output data](#) to file or audio device.

```
    } while (ErrorStatus==AACENC_OK);
```

6. Call `aacEncClose()` and destroy encoder instance.

```
    aacEncClose(&hAacEncoder);
```

## 2.3 Encoder Instance Allocation

The assignment of the `aacEncOpen()` function is very flexible and can be used in the following way.

- If the amount of memory consumption is not an issue, the encoder instance can be allocated for the maximum number of possible audio channels (for example 6 or 8) with the full functional range supported by the library. This is the default open procedure for the AAC encoder if memory consumption does not need to be minimized.

```
    aacEncOpen(&hAacEncoder, 0, 0)
```

- If the required MPEG-4 AOTs do not call for the full functional range of the library, encoder modules can be allocated selectively.

AAC	SBR	PS	MD	FLAGS	value
X	-	-	-	(0x01)	0x01
X	X	-	-	(0x01 0x02)	0x03
X	X	X	-	(0x01 0x02 0x04)	0x07
X	-	-	X	(0x01  0x10)	0x11
X	X	-	X	(0x01 0x02  0x10)	0x13
X	X	X	X	(0x01 0x02 0x04 0x10)	0x17

- AAC: Allocate AAC Core Encoder module.
- SBR: Allocate Spectral Band Replication module.
- PS: Allocate Parametric Stereo module.
- MD: Allocate Meta Data module within AAC encoder.

```
    aacEncOpen(&hAacEncoder, value, 0)
```

- Specifying the maximum number of channels to be supported in the encoder instance can be done as follows.
  - For example allocate an encoder instance which supports 2 channels for all supported AOTs. The library itself may be capable of encoding up to 6 or 8 channels but in this example only 2 channel encoding is required and thus only buffers for 2 channels are allocated to save data memory.

```
aacEncOpen (&hAacEncoder, 0, 2)
```

- Additionally the maximum number of supported channels in the SBR module can be denoted separately.

In this example the encoder instance provides a maximum of 6 channels out of which up to 2 channels support SBR. This encoder instance can produce for example 5.1 channel AAC-LC streams or stereo HE-AAC (v2) streams. HE-AAC 5.1 multi channel is not possible since only 2 out of 6 channels support SBR, which saves data memory.

```
aacEncOpen (&hAacEncoder, 0, 6 | (2<<8))
```

## 2.4 Input/Output Arguments

### 2.4.1 Provide Buffer Descriptors

In the present encoder API, the input and output buffers are described with [buffer descriptors](#). This mechanism allows a flexible handling of input and output buffers without impact to the actual encoding call. Optional buffers are necessary e.g. for ancillary data, meta data input or additional output buffers describing superframing data in DAB+ or DRM+.

At least one input buffer for audio input data and one output buffer for bitstream data must be allocated. The input buffer size can be a user defined multiple of the number of input channels. PCM input data will be copied from the user defined PCM buffer to an internal input buffer and so input data can be less than one AAC audio frame. The output buffer size should be 6144 bits per channel excluding the LFE channel. If the output data does not fit into the provided buffer, an AACENC\_ERROR will be returned by [aacEncEncode\(\)](#).

```
static INT_PCM      inputBuffer[8*2048];
static UCHAR        ancillaryBuffer[50];
static AACENC_MetaData metaDataSetup;
static UCHAR        outputBuffer[8192];
```

All input and output buffer must be clustered in input and output buffer arrays.

```
static void* inBuffer[]      = { inputBuffer, ancillaryBuffer, &metaDataSetup }
;
static INT   inBufferIds[]   = { IN_AUDIO_DATA, IN AncillaryData,
    IN_METADATA_SETUP };
static INT   inBufferSize[] = { sizeof(inputBuffer), sizeof(ancillaryBuffer),
    sizeof(metaDataSetup) };
static INT   inBufferElSize[] = { sizeof(INT_PCM), sizeof(UCHAR),
    sizeof(AACENC_MetaData) };

static void* outBuffer[]    = { outputBuffer };
static INT   outBufferIds[] = { OUT_BITSTREAM_DATA };
static INT   outBufferSize[] = { sizeof(outputBuffer) };
static INT   outBufferElSize[] = { sizeof(UCHAR) };
```

Allocate buffer descriptors

```
AACENC_BufDesc inBufDesc;
AACENC_BufDesc outBufDesc;
```

Initialize input buffer descriptor

```
inBufDesc.numBufs      = sizeof(inBuffer)/sizeof(void*);
inBufDesc.buFs         = (void*)&inBuffer;
inBufDesc.bufferIdentifiers = inBufferIds;
```

```

inBufDesc.bufSizes      = inBufferSize;
inBufDesc.bufElSizes    = inBufferElSize;

```

### Initialize output buffer descriptor

```

outBufDesc.numBufs      = sizeof(outBuffer)/sizeof(void*);
outBufDesc.bufs         = (void*)&outBuffer;
outBufDesc.bufferIdentifiers = outBufferIds;
outBufDesc.bufSizes     = outBufferSize;
outBufDesc.bufElSizes   = outBufferElSize;

```

## 2.4.2 Provide Input/Output Argument Lists

The input and output arguments of an [aacEncEncode\(\)](#) call are described in argument structures.

```

AACENC_InArgs    inargs;
AACENC_OutArgs   outargs;

```

## 2.5 Feed Input Buffer

The input buffer should be handled as a modulo buffer. New audio data in the form of pulse-code-modulated samples (PCM) must be read from external and be fed to the input buffer depending on its fill level. The required sample bitrate (represented by the data type INT\_PCM which is 16, 24 or 32 bits wide) is fixed and depends on library configuration (usually 16 bit).

```

    inargs.numInSamples += WAV_InputRead ( wavIn,
                                           &inputBuffer[inargs.numInSamples],
                                           FDKmin(encInfo.inputChannels*encIn
                                           sizeof(inputBuffer) /
                                           sizeof(INT_PCM)-inargs.
numInSamples),
                                           SAMPLE_BITS
                                           );

```

After the encoder's internal buffer is fed with incoming audio samples, and [aacEncEncode\(\)](#) processed the new input data, update/move remaining samples in input buffer, simulating a modulo buffer:

```

    if (outargs.numInSamples>0) {
        FDKmemmove( inputBuffer,
                    &inputBuffer[outargs.numInSamples],
                    sizeof(INT_PCM)*(inargs.numInSamples-outargs.
numInSamples) );
        inargs.numInSamples -= outargs.numInSamples;
    }

```

## 2.6 Output Bitstream Data

If any AAC bitstream data is available, write it to output file or device. This can be done once the following condition is true:

---

```
if (outargs.numOutBytes>0) {  
  
} /* (outBytes>0) */
```

If you use file I/O then for example call `mpegFileWrite_Write()` from the library `libMpegFileWrite`

```
mpegFileWrite_Write(hMpegFile, outputBuffer, outargs.numOutBytes)  
;
```

## 2.7 Meta Data Configuration

If the present library is configured with Metadata support, it is possible to insert meta data side info into the generated audio bitstream while encoding.

To work with meta data the encoder instance has to be [allocated](#) with meta data support. The meta data mode must be configured with the [AACENC\\_METADATA\\_MODE](#) parameter and [aacEncoder\\_SetParam\(\)](#) function.

```
aacEncoder_SetParam(hAacEncoder, AACENC_METADATA_MODE, 0-2);
```

This configuration indicates how to embed meta data into bitstream. Either no insertion, MPEG or ETSI style. The meta data itself must be specified within the meta data setup structure [AACENC\\_MetaData](#).

Changing one of the [AACENC\\_MetaData](#) setup parameters can be achieved from outside the library within [IN\\_METADATA\\_SETUP](#) input buffer. There is no need to supply meta data setup structure every frame. If there is no new meta setup data available, the encoder uses the previous setup or the default configuration in initial state.

In general the audio compressor and limiter within the encoder library can be configured with the [AACENC\\_METADATA\\_DRC\\_PROFILE](#) parameter [AACENC\\_MetaData::drc\\_profile](#) and [AACENC\\_MetaData::comp\\_profile](#).

## 2.8 Encoder Reconfiguration

The encoder library allows reconfiguration of the encoder instance with new settings continuously between encoding frames. Each parameter to be changed must be set with a single [aacEncoder\\_SetParam\(\)](#) call. The internal status of each parameter can be retrieved with an [aacEncoder\\_GetParam\(\)](#) call.

There is no stand-alone reconfiguration function available. When parameters were modified from outside the library, an internal control mechanism triggers the necessary reconfiguration process which will be applied at the beginning of the following [aacEncEncode\(\)](#) call. This state can be observed from external via the [AACENC\\_INIT\\_STATUS](#) and [aacEncoder\\_GetParam\(\)](#) function. The reconfiguration process can also be applied immediately when all parameters of an [aacEncEncode\(\)](#) call are NULL with a valid encoder handle.

The internal reconfiguration process can be controlled from extern with the following access.

```
aacEncoder_SetParam(hAacEncoder, AACENC_CONTROL_STATE, AACENC_CTRLFLAGS);
```

## 2.9 Encoder Parametrization

All parameters listed in [AACENC\\_PARAM](#) can be modified within an encoder instance.

---

### 2.9.1 Mandatory Encoder Parameters

The following parameters must be specified when the encoder instance is initialized.

```
aacEncoder_SetParam(hAacEncoder, AACENC_AOT, value);
aacEncoder_SetParam(hAacEncoder, AACENC_BITRATE, value);
aacEncoder_SetParam(hAacEncoder, AACENC_SAMPLERATE, value);
aacEncoder_SetParam(hAacEncoder, AACENC_CHANNELMODE, value);
```

Beyond that is an internal auto mode which preinitializes the [AACENC\\_BITRATE](#) parameter if the parameter was not set from extern. The bitrate depends on the number of effective channels and sampling rate and is determined as follows.

```
AAC-LC (AOT_AAC_LC): 1.5 bits per sample
HE-AAC (AOT_SBR): 0.625 bits per sample
HE-AAC v2 (AOT_PS): 0.5 bits per sample
```

### 2.9.2 Channel Mode Configuration

The input audio data is described with the [AACENC\\_CHANNELMODE](#) parameter in the [aacEncoder\\_SetParam\(\)](#) call. It is not possible to use the encoder instance with a 'number of input channels' argument. Instead, the channelMode must be set as follows.

```
aacEncoder_SetParam(hAacEncoder, AACENC_CHANNELMODE, value);
```

The parameter is specified in CHANNEL\_MODE and can be mapped from the number of input channels in the following way.

```
CHANNEL_MODE chMode = MODE_INVALID;

switch (nChannels) {
    case 1: chMode = MODE_1;           break;
    case 2: chMode = MODE_2;           break;
    case 3: chMode = MODE_1_2;         break;
    case 4: chMode = MODE_1_2_1;       break;
    case 5: chMode = MODE_1_2_2;       break;
    case 6: chMode = MODE_1_2_2_1;     break;
    default:
        chMode = MODE_INVALID;
}
return chMode;
```

### 2.9.3 Audio Quality Considerations

The default encoder configuration is suggested to be used. Encoder tools such as TNS and PNS are activated by default and are internally controlled (see [Encoder Tools](#)).

There is an additional quality parameter called [AACENC\\_AFTERBURNER](#). In the default configuration this quality switch is deactivated because it would cause a workload increase which might be significant. If workload is not an issue in the application we recommended to activate this feature.

```
aacEncoder_SetParam(hAacEncoder, AACENC_AFTERBURNER, 1);
```

## 2.10 Audio Channel Configuration

The MPEG standard refers often to the so-called Channel Configuration. This Channel Configuration is used for a fixed Channel Mapping. The configurations 1-7 are predefined in MPEG standard and used

---

for implicit signalling within the encoded bitstream. For user defined Configurations the Channel Configuration is set to 0 and the Channel Mapping must be explicitly described with an appropriate Program Config Element. The present Encoder implementation does not allow the user to configure this Channel Configuration from extern. The Encoder implementation supports fixed Channel Modes which are mapped to Channel Configuration as follow.

ChannelMode	ChCfg	front_El	side_El	back_El	lfe_El
MODE_1	1	SCE			
MODE_2	2	CPE			
MODE_1_2	3	SCE, CPE			
MODE_1_2_1	4	SCE, CPE		SCE	
MODE_1_2_2	5	SCE, CPE		CPE	
MODE_1_2_2_1	6	SCE, CPE		CPE	LFE

- SCE: Single Channel Element.
- CPE: Channel Pair.
- SCE: Low Frequency Element.

Moreover, the Table describes all fixed Channel Elements for each Channel Mode which are assigned to a speaker arrangement. The arrangement includes front, side, back and lfe Audio Channel Elements.

This mapping of Audio Channel Elements is defined in MPEG standard for Channel Config 1-7. The Channel assignment for MODE\_1\_1, MODE\_2\_2 and MODE\_2\_1 is used from the ARIB standard. All other configurations are defined as suggested in MPEG.

In case of Channel Config 0 or writing matrix mixdown coefficients, the encoder enables the writing of Program Config Element itself as described in encPCE. The configuration used in Program Config Element refers to the denoted Table.

Beside the Channel Element assignment the Channel Modes are responsible for audio input data channel mapping. The Channel Mapping of the audio data depends on the selected [AACENC\\_CHANNELORDER](#) which can be MPEG or WAV like order.

Following Table describes the complete channel mapping for both Channel Order configurations.

ChannelMode	MPEG-Channelorder							WAV-Channelorder						
MODE_1	0							0						
MODE_2	0	1						0	1					
MODE_1_2	0	1	2					2	0	1				
MODE_1_2_1	0	1	2	3				2	0	1	3			
MODE_1_2_2	0	1	2	3	4			2	0	1	3	4		
MODE_1_2_2_1	0	1	2	3	4	5		2	0	1	4	5	3	

The denoted mapping is important for correct audio channel assignment when using MPEG or WAV ordering. The incoming audio channels are distributed MPEG like starting at the front channels and ending at the back channels. The distribution is used as described in Table concerning Channel Config and fix channel elements. Please see the following example for clarification.

Example: MODE\_1\_2\_2\_1 - WAV-Channelorder 5.1

Input Channel	Coder Channel
2 (front center)	0 (SCE channel)
0 (left center)	1 (1st of 1st CPE)
1 (right center)	2 (2nd of 1st CPE)
4 (left surround)	3 (1st of 2nd CPE)
5 (right surround)	4 (2nd of 2nd CPE)
3 (LFE)	5 (LFE)

## 2.11 Supported Bitrates

The FDK AAC Encoder provides a wide range of supported bitrates. The minimum and maximum allowed bitrate depends on the Audio Object Type. For AAC-LC the minimum bitrate is the bitrate that is required to write the most basic and minimal valid bitstream. It consists of the bitstream format header information and other static/mandatory information within the AAC payload. The maximum AAC framesize allowed by the MPEG-4 standard determines the maximum allowed bitrate for AAC-LC. For HE-AAC and HE-AAC v2 a library internal look-up table is used.

A good working point in terms of audio quality, sampling rate and bitrate, is at 1 to 1.5 bits/audio sample for AAC-LC, 0.625 bits/audio sample for HE-AAC and 0.5 bits/audio sample for HE-AAC v2. For example for one channel with a sampling frequency of 48 kHz, the range from 48 kbit/s to 72 kbit/s achieves reasonable audio quality for AAC-LC.

For HE-AAC and HE-AAC v2 the lowest possible audio input sampling frequency is 16 kHz because then the AAC-LC core encoder operates in dual rate mode at its lowest possible sampling frequency, which is 8 kHz. HE-AAC v2 requires stereo input audio data.

Please note that in HE-AAC or HE-AAC v2 mode the encoder supports much higher bitrates than are appropriate for HE-AAC or HE-AAC v2. For example, at a bitrate of more than 64 kbit/s for a stereo audio signal at 44.1 kHz it usually makes sense to use AAC-LC, which will produce better audio quality at that bitrate than HE-AAC or HE-AAC v2.

## 2.12 Recommended Sampling Rate and Bitrate Combinations

The following table provides an overview of recommended encoder configuration parameters which we determined by virtue of numerous listening tests.

### 2.12.1 AAC-LC, HE-AAC, HE-AACv2.

Audio Object Type	Bit Rate Range [bit/s]	Supported Sampling Rates [kHz]	Preferred Sampl. Rate [kHz]	No. of Chan.
AAC LC + SBR + PS	8000 - 11999	22.05, 24.00	24.00	2
AAC LC + SBR + PS	12000 - 17999	32.00, 44.10, 48.00	32.00	2
AAC LC + SBR + PS	18000 - 39999	32.00, 44.10, 48.00	44.10	2
AAC LC + SBR + PS	40000 - 56000	32.00, 44.10, 48.00	48.00	2
AAC LC + SBR	8000 - 11999	22.05, 24.00	24.00	1
AAC LC + SBR	12000 - 17999	32.00, 44.10, 48.00	32.00	1
AAC LC + SBR	18000 - 39999	32.00, 44.10, 48.00	44.10	1
AAC LC + SBR	40000 - 56000	32.00, 44.10, 48.00	48.00	1
AAC LC + SBR	16000 - 27999	32.00, 44.10, 48.00	32.00	2
AAC LC + SBR	28000 - 63999	32.00, 44.10, 48.00	44.10	2
AAC LC + SBR	64000 - 128000	32.00, 44.10, 48.00	48.00	2
AAC LC + SBR	64000 - 69999	32.00, 44.10, 48.00	32.00	5, 5.1
AAC LC + SBR	70000 - 159999	32.00, 44.10, 48.00	44.10	5, 5.1
AAC LC + SBR	160000 - 319999	32.00, 44.10, 48.00	48.00	5, 5.1
AAC LC + SBR	320000 - 640000	64.00, 88.20, 96.00	96.00	5, 5.1
AAC LC	8000 - 15999	11.025, 12.00, 16.00	12.00	1
AAC LC	16000 - 23999	16.00	16.00	1
AAC LC	24000 - 31999	16.00, 22.05, 24.00	24.00	1
AAC LC	32000 - 55999	32.00	32.00	1



AAC LC	56000 - 160000	32.00, 44.10, 48.00	44.10	1
AAC LC	160001 - 288000	48.00	48.00	1
-----				
AAC LC	16000 - 23999	11.025, 12.00, 16.00	12.00	2
AAC LC	24000 - 31999	16.00	16.00	2
AAC LC	32000 - 39999	16.00, 22.05, 24.00	22.05	2
AAC LC	40000 - 95999	32.00	32.00	2
AAC LC	96000 - 111999	32.00, 44.10, 48.00	32.00	2
AAC LC	112000 - 320001	32.00, 44.10, 48.00	44.10	2
AAC LC	320002 - 576000	48.00	48.00	2
-----				
AAC LC	160000 - 239999	32.00	32.00	5, 5.1
AAC LC	240000 - 279999	32.00, 44.10, 48.00	32.00	5, 5.1
AAC LC	280000 - 800000	32.00, 44.10, 48.00	44.10	5, 5.1
-----				

### 2.12.2 AAC-LD, AAC-ELD, AAC-ELD with SBR.

Audio Object Type	Bit Rate Range [bit/s]	Supported Sampling Rates [kHz]	Preferred Sampl. Rate [kHz]	No. of Chan.
ELD + SBR	16000 - 24999	32.00 - 44.10	32.00	1
ELD + SBR	25000 - 31999	32.00 - 48.00	32.00	1
ELD + SBR	32000 - 64000	32.00 - 48.00	48.00	1
-----				
ELD + SBR	32000 - 51999	32.00 - 48.00	44.10	2
ELD + SBR	52000 - 128000	32.00 - 48.00	48.00	2
-----				
ELD + SBR	72000 - 192000	44.10 - 48.00	48.00	3
-----				
ELD + SBR	96000 - 256000	44.10 - 48.00	48.00	4
-----				
ELD + SBR	120000 - 320000	44.10 - 48.00	48.00	5
-----				
LD, ELD	16000 - 19999	16.00 - 24.00	16.00	1
LD, ELD	20000 - 39999	16.00 - 32.00	24.00	1
LD, ELD	40000 - 49999	22.05 - 32.00	32.00	1
LD, ELD	50000 - 61999	24.00 - 44.10	32.00	1
LD, ELD	62000 - 84999	32.00 - 48.00	44.10	1
LD, ELD	85000 - 192000	44.10 - 48.00	48.00	1
-----				
LD, ELD	64000 - 75999	24.00 - 32.00	32.00	2
LD, ELD	76000 - 97999	24.00 - 44.10	32.00	2
LD, ELD	98000 - 135999	32.00 - 48.00	44.10	2
LD, ELD	136000 - 384000	44.10 - 48.00	48.00	2
-----				
LD, ELD	96000 - 113999	24.00 - 32.00	32.00	3
LD, ELD	114000 - 146999	24.00 - 44.10	32.00	3
LD, ELD	147000 - 203999	32.00 - 48.00	44.10	3
LD, ELD	204000 - 576000	44.10 - 48.00	48.00	3
-----				
LD, ELD	128000 - 151999	24.00 - 32.00	32.00	4
LD, ELD	152000 - 195999	24.00 - 44.10	32.00	4
LD, ELD	196000 - 271999	32.00 - 48.00	44.10	4
LD, ELD	272000 - 768000	44.10 - 48.00	48.00	4
-----				
LD, ELD	160000 - 189999	24.00 - 32.00	32.00	5
LD, ELD	190000 - 244999	24.00 - 44.10	32.00	5
LD, ELD	245000 - 339999	32.00 - 48.00	44.10	5
LD, ELD	340000 - 960000	44.10 - 48.00	48.00	5
-----				



## Chapter 3

# Encoder Behaviour

### 3.1 Bandwidth

The FDK AAC encoder usually does not use the full frequency range of the input signal, but restricts the bandwidth according to certain library-internal settings. They can be changed in the table "band-WidthTable" in the file bandwidth.cpp (if available), or via command-line argument "-w" (see chapter [Command-line Usage](#)).

However it is not recommended to change these settings, because they are based on numerous listening tests and careful tweaks to ensure the best overall encoding quality.

Theoretically a signal of for example 48 kHz can contain frequencies up to 24 kHz, but to use this full range in an audio encoder usually does not make sense. Usually the encoder has a very limited amount of bits to spend (typically 128 kbit/s for stereo 48 kHz content) and to allow full range bandwidth would waste a lot of these bits for frequencies the human ear is hardly able to perceive anyway, if at all. Hence it is wise to use the available bits for the really important frequency range and just skip the rest. At lower bitrates (e. g.  $\leq 80$  kbit/s for stereo 48 kHz content) the encoder will choose an even smaller bandwidth, because an encoded signal with smaller bandwidth and hence less artifacts sounds better than a signal with higher bandwidth but then more coding artefacts across all frequencies. These artefacts would occur if small bitrates and high bandwidths are chosen because the available bits are just not enough to encode all frequencies well.

Unfortunately some people evaluate encoding quality based on possible bandwidth as well, but it is a two-sided sword considering the trade-off described above.

Another aspect is workload consumption. The higher the allowed bandwidth, the more frequency lines have to be processed, which in turn increases the workload.

### 3.2 Frame Sizes & Bit Reservoir

For AAC there is a difference between constant bit rate and constant frame length due to the so-called bit reservoir technique, which allows the encoder to use less bits in an AAC frame for those audio signal sections which are easy to encode, and then spend them at a later point in time for more complex audio sections. The extent to which this "bit exchange" is done is limited to allow for reliable and relatively low delay real time streaming. Over a longer period in time the bitrate will be constant in the AAC constant bitrate mode, e.g. for ISDN transmission. This means that in AAC each bitstream frame will in general have a different length in bytes but over time it will reach the target bitrate. One could also make an MPEG compliant AAC encoder which always produces constant length packages for each AAC frame, but the

audio quality would be considerably worse since the bit reservoir technique would have to be switched off completely. A higher bit rate would have to be used to get the same audio quality as with an enabled bit reservoir.

The maximum AAC frame length, regardless of the available bit reservoir, is defined as 6144 bits per channel.

For mp3 by the way, the same bit reservoir technique exists, but there each bit stream frame has a constant length for a given bit rate (ignoring the padding byte). In mp3 there is a so-called "back pointer" which tells the decoder which bits belong to the current mp3 frame - and in general some or many bits have been transmitted in an earlier mp3 frame. Basically this leads to the same "bit exchange between mp3 frames" as in AAC but with virtually constant length frames.

This variable frame length at "constant bit rate" is not something special in this Fraunhofer IIS AAC encoder. AAC has been designed in that way.

### 3.2.1 Estimating Average Frame Sizes

A HE-AAC v1 or v2 audio frame contains 2048 PCM samples per channel (there is also one mode with 1920 samples per channel but this is only for special purposes such as DAB+ digital radio).

The number of HE-AAC frames  $N\_FRAMES$  per second at 44.1 kHz is:

$$N\_FRAMES = 44100/2048 = 21.5332$$

At a bit rate of 8 kbps the average number of bits per frame  $N\_BITS\_PER\_FRAME$  is:

$$N\_BITS\_PER\_FRAME = 8000/21.5332 = 371.52$$

which is about 46.44 bytes per encoded frame.

At a bit rate of 32 kbps, which is quite high for single channel HE-AAC v1, it is:

$$N\_BITS\_PER\_FRAME = 32000/21.5332 = 1486$$

which is about 185.76 bytes per encoded frame.

These bits/frame figures are average figures where each AAC frame generally has a different size in bytes. To calculate the same for AAC-LC just use 1024 instead of 2048 PCM samples per frame and channel. For AAC-LD/ELD it is either 480 or 512 PCM samples per frame and channel.

## 3.3 Encoder Tools

The AAC encoder supports TNS, PNS, MS, Intensity and activates these tools depending on the audio signal and the encoder configuration (i.e. bitrate or AOT). It is not required to configure these tools manually.

PNS improves encoding quality only for certain bitrates. Therefore it makes sense to activate PNS only for these bitrates and save the processing power required for PNS (about 10 % of the encoder) when using other bitrates. This is done automatically inside the encoder library. PNS is disabled inside the encoder library if an MPEG-2 AOT is chosen since PNS is an MPEG-4 AAC feature.

If SBR is activated, the encoder automatically deactivates PNS internally. If TNS is disabled but PNS is allowed, the encoder deactivates PNS calculation internally.

---

# Chapter 4

## Command-line Usage

In [main.cpp](#) there are two implementations of [main\(\)](#) and depending on the define `ARCH_WA_NOCMDLINE` either one of those is activated or visible to the compiler respectively. Defining `ARCH_WA_NOCMDLINE` in [main.cpp](#) provides a workaround (WA) for those architectures where there is no command-line available. Also it provides the possibility to run several command-line calls automatically by specifying them in a text file.

So if you define `ARCH_WA_NOCMDLINE` then the entry point of the program becomes:

```
int main() {
    return IIS_ProcessCmdList( BATCH_FILE, &process_file);
}
```

`IIS_ProcessCmdList()` parses each line found in the file `BATCH_FILE` and then feeds it to `process_file()`.

### 4.1 Arguments

The example encoder implementation accepts the following command-line arguments. Some of the options listed here might not be available depending on encoder library configuration. For more information on how to configure the encoder see [AACENC\\_PARAM](#).

```
%s [options] -if infile -of outfile
```

#### 4.1.1 Mandatory Arguments

```
-if [infile]
    Accepted format is RIFF/WAVE. The number of channels and WAV bitdepth depends on
    encoder configuration. The following audio input sampling frequencies are supported
    by the encoder: 8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz,
    44.1 kHz, 48 kHz, 64 kHz, 88.2 kHz, 96 kHz. The number of supported audio channels
    can differ and depends on the provided encoder configuration.

-of [outfile]
    Output bitstream file
```

#### 4.1.2 Optional Arguments

```
-a [Audio Object Type]
```

2: MPEG-4 AAC Low Complexity (AAC-LC)  
 5: MPEG-4 AAC High Efficiency AAC (HE-AAC) using Spectral Band Replication (SBR)  
 29: MPEG-4 AAC High Efficiency AAC Version 2 (HE-AAC v2) using SBR and Parametric Stereo (PS)

For MPEG-4 LD/ELD bitstreams use following AOTs.

23: MPEG-4 AAC Low-Delay.  
 39: MPEG-4 AAC Enhanced Low-Delay.

For MPEG-2 backwards-compatible bitstreams use virtual AOTs

129: MPEG-2 AAC LC  
 132: MPEG-2 AAC + SBR  
 156: MPEG-2 AAC + SBR + PS

-b [bitrate]  
 Encoding bitrate in bits per second. The default value depends on the Audio Object Type (128000 for AAC-LC, 56000 for HE-AAC and HE-AAC v2).

-m [bitrate mode]  
 0: Constant bitrate (CBR, default); reduced bitreservoir at LD/ELD configuration  
 8: LD/ELD constant bitrate with full bitreservoir

-Z [lowdelay SBR]  
 0: no ELD-SBR  
 1: ELD-SBR enabled

-t [bitstream format]  
 1: ADIF  
 2: ADTS  
 6: LATM MCP=1  
 7: LATM MCP=0  
 8: LATM MCP=1 within RAW PACKETS  
 9: LATM MCP=0 within RAW PACKETS  
 10: LOAS/LATM (LATM within LOAS)

-w [bandwidth]  
 0: Determine best bandwidth internally (default)  
 1 to fs/2: Frequency bandwidth in Hertz (AAC-LC only)  
 See chapter \ref BEHAVIOUR\_BANDWIDTH for details.

-q [afterburner]  
 0: Off (default)  
 1: On

-f [framelength]  
 Encoder core frame length.  
 1024: Default configuration.  
 960: Optional length, e.g. used in DAB+ or DRM30/DRM+.  
 512: Optional length in LD/ELD configuration.  
 480: Default LD/ELD configuration.

-z [ancillary datarate]  
 Bitrate in bits per second.

-c [ADTS CRC check]  
 0: Off  
 1: On

-i [SBR signaling]  
 0: Implicit backward compatible signaling. (default)  
 1: Explicit SBR and implicit PS signaling.  
 2: Explicit hierarchical signaling.

-hp [header period]  
 Configure header frame period.

-n [subframes]

---

---

Number of sub frames in a transport frame (default = 1).  
1-2: LOAS/LATM  
1-4: ADTS

-sf [frame number]  
Start decoding at specific frame number.

-lf [frame number]  
Stop decoding at specific frame number.

-v  
Verbose output

-anc [ancillary\_filename]  
Ancillary data filename.

Following optional parameters are available if Metadata support is enabled within the encoder library.

-M [Metadata transport]  
0: Embed no Metadata (default)  
Embed MPEG-DRC  
1: Generate DRC using Profile: None  
2: Generate DRC using Profile: FilmStandard  
3: Generate DRC using Profile: FilmLight  
4: Generate DRC using Profile: MusicStandard  
5: Generate DRC using Profile: MusicLight  
6: Generate DRC using Profile: Speech  
Embed MPEG-DRC + ETSI AncData  
8: Generate DRC using Profile: None  
9: Generate DRC using Profile: FilmStandard  
10: Generate DRC using Profile: FilmLight  
11: Generate DRC using Profile: MusicStandard  
12: Generate DRC using Profile: MusicLight  
13: Generate DRC using Profile: Speech

-D [Dialog level]  
Dialog level in dB (-31...-1 dB)  
Valid configuration: 1, 2, ..., 31

-X [Center downmix level] (ETSI) as table index.  
0: 0.0 dB  
1: -1.5 dB  
2: -3.0 dB  
3: -4.5 dB  
4: -6.0 dB  
5: -7.5 dB  
6: -9.0 dB  
7: -Inf dB

-Y [Surround downmix level] (ETSI)-> 0...7, see center downmix level.

---





# Chapter 5

## Class Index

### 5.1 Class List

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

<a href="#">AACENC_BufDesc</a> . . . . .	23
<a href="#">AACENC_InArgs</a> . . . . .	24
<a href="#">AACENC_InfoStruct</a> . . . . .	25
<a href="#">AACENC_MetaData</a> . . . . .	26
<a href="#">AACENC_OutArgs</a> . . . . .	28



# Chapter 6

## File Index

### 6.1 File List

Here is a list of all files with brief descriptions:

<a href="#">aacenc_lib.h</a> (FDK AAC Encoder library interface header file ) . . . . .	29
<a href="#">main.cpp</a> . . . . .	40



# Chapter 7

## Class Documentation

### 7.1 AACENC\_BufDesc Struct Reference

```
#include <aacenc_lib.h>
```

#### Public Attributes

- INT [numBufs](#)
- void \*\* [bufs](#)
- INT \* [bufferIdentifiers](#)
- INT \* [bufSizes](#)
- INT \* [bufElSizes](#)

#### 7.1.1 Detailed Description

Describes the input and output buffers for an [aacEncEncode\(\)](#) call.

#### 7.1.2 Member Data Documentation

##### 7.1.2.1 INT\* AACENC\_BufDesc::bufElSizes

Size of each buffer element in bytes.

Referenced by [main\(\)](#).

##### 7.1.2.2 INT\* AACENC\_BufDesc::bufferIdentifiers

Identifier of each buffer element. See [AACENC\\_BufferIdentifier](#).

Referenced by [main\(\)](#).

##### 7.1.2.3 void\*\* AACENC\_BufDesc::bufs

Pointer to vector containing buffer addresses.

Referenced by `main()`.

#### 7.1.2.4 `INT* AACENC_BufDesc::bufSizes`

Size of each buffer in 8-bit bytes.

Referenced by `main()`.

#### 7.1.2.5 `INT AACENC_BufDesc::numBufs`

Number of buffers.

Referenced by `main()`.

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

- [aacenc\\_lib.h](#)

## 7.2 AACENC\_InArgs Struct Reference

```
#include <aacenc_lib.h>
```

### Public Attributes

- `INT numInSamples`
- `INT numAncBytes`

### 7.2.1 Detailed Description

Defines the input arguments for an `aacEncEncode()` call.

### 7.2.2 Member Data Documentation

#### 7.2.2.1 `INT AACENC_InArgs::numAncBytes`

Number of ancillary data bytes to be encoded.

Referenced by `main()`.

#### 7.2.2.2 `INT AACENC_InArgs::numInSamples`

Number of valid input audio samples (multiple of input channels).

Referenced by `main()`.

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

- [aacenc\\_lib.h](#)
-

## 7.3 AACENC\_InfoStruct Struct Reference

```
#include <aacenc_lib.h>
```

### Public Attributes

- UINT [maxOutBufBytes](#)
- UINT [maxAncBytes](#)
- UINT [inBufFillLevel](#)
- UINT [inputChannels](#)
- UINT [frameLength](#)
- UINT [encoderDelay](#)
- UCHAR [confBuf](#) [64]
- UINT [confSize](#)

### 7.3.1 Detailed Description

Provides some info about the encoder configuration.

### 7.3.2 Member Data Documentation

#### 7.3.2.1 UCHAR AACENC\_InfoStruct::confBuf[64]

Configuration buffer in binary format as an AudioSpecificConfig or StreamMuxConfig according to the selected transport type.

Referenced by `main()`.

#### 7.3.2.2 UINT AACENC\_InfoStruct::confSize

Number of valid bytes in `confBuf`.

Referenced by `main()`.

#### 7.3.2.3 UINT AACENC\_InfoStruct::encoderDelay

Codec delay in PCM samples/channel. Depends on `frameLength` and AOT. Does not include framing delay for filling up encoder PCM input buffer.

#### 7.3.2.4 UINT AACENC\_InfoStruct::frameLength

Amount of input audio samples consumed each frame per channel, depending on audio object type configuration.

Referenced by `main()`.

#### 7.3.2.5 UINT AACENC\_InfoStruct::inBufFillLevel

Internal input buffer fill level in samples per channel. This parameter will automatically be cleared if `samplingrate` or `channel(Mode/Order)` changes.

---

### 7.3.2.6 `UINT AACENC_InfoStruct::inputChannels`

Number of input channels expected in encoding process.

Referenced by `main()`.

### 7.3.2.7 `UINT AACENC_InfoStruct::maxAncBytes`

Maximum number of ancillary data bytes which can be inserted into bitstream within one frame.

### 7.3.2.8 `UINT AACENC_InfoStruct::maxOutBufBytes`

Maximum number of encoder bitstream bytes within one frame. Size depends on maximum number of supported channels in encoder instance. For superframing (as used for example in DAB+), size has to be a multiple accordingly.

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

- [aacenc\\_lib.h](#)

## 7.4 AACENC\_MetaData Struct Reference

```
#include <aacenc_lib.h>
```

### Public Attributes

- [AACENC\\_METADATA\\_DRC\\_PROFILE drc\\_profile](#)
- [AACENC\\_METADATA\\_DRC\\_PROFILE comp\\_profile](#)
- [INT drc\\_TargetRefLevel](#)
- [INT comp\\_TargetRefLevel](#)
- [INT prog\\_ref\\_level\\_present](#)
- [INT prog\\_ref\\_level](#)
- [UCHAR PCE\\_mixdown\\_idx\\_present](#)
- [UCHAR ETSI\\_DmxLvl\\_present](#)
- [SCHAR centerMixLevel](#)
- [SCHAR surroundMixLevel](#)
- [UCHAR dolbySurroundMode](#)

### 7.4.1 Detailed Description

Meta Data setup structure.

### 7.4.2 Member Data Documentation

#### 7.4.2.1 `SCHAR AACENC_MetaData::centerMixLevel`

Center downmix level (0..7, according to table)

---



**7.4.2.2 AACENC\_METADATA\_DRC\_PROFILE AACENC\_MetaData::comp\_profile**

ETSI heavy compression profile. See [AACENC\\_METADATA\\_DRC\\_PROFILE](#).

**7.4.2.3 INT AACENC\_MetaData::comp\_TargetRefLevel**

Adjust limiter to avoid overload. Scaled with 16 bit.  $x \cdot 2^{16}$ .

**7.4.2.4 UCHAR AACENC\_MetaData::dolbySurroundMode**

Indication for Dolby Surround Encoding Mode.

- 0: Dolby Surround mode not indicated
- 1: 2-ch audio part is not Dolby surround encoded
- 2: 2-ch audio part is Dolby surround encoded

**7.4.2.5 AACENC\_METADATA\_DRC\_PROFILE AACENC\_MetaData::drc\_profile**

MPEG DRC compression profile. See [AACENC\\_METADATA\\_DRC\\_PROFILE](#).

**7.4.2.6 INT AACENC\_MetaData::drc\_TargetRefLevel**

Used to define expected level to: Scaled with 16 bit.  $x \cdot 2^{16}$ .

**7.4.2.7 UCHAR AACENC\_MetaData::ETSI\_DmxLvl\_present**

Flag, if dmx-lvl should be written in ETSI-ancData

**7.4.2.8 UCHAR AACENC\_MetaData::PCE\_mixdown\_idx\_present**

Flag, if dmx-idx should be written in programme config element

**7.4.2.9 INT AACENC\_MetaData::prog\_ref\_level**

Programme Reference Level = Dialogue Level: -31.75dB .. 0 dB ; stepsize: 0.25dB Scaled with 16 bit.  $x \cdot 2^{16}$ .

**7.4.2.10 INT AACENC\_MetaData::prog\_ref\_level\_present**

Flag, if prog\_ref\_level is present

---

#### 7.4.2.11 SCHAR AACENC\_MetaData::surroundMixLevel

Surround downmix level (0...7, according to table)

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

- [aacenc\\_lib.h](#)

## 7.5 AACENC\_OutArgs Struct Reference

```
#include <aacenc_lib.h>
```

### Public Attributes

- INT [numOutBytes](#)
- INT [numInSamples](#)
- INT [numAncBytes](#)

### 7.5.1 Detailed Description

Defines the output arguments for an [aacEncEncode\(\)](#) call.

### 7.5.2 Member Data Documentation

#### 7.5.2.1 INT AACENC\_OutArgs::numAncBytes

Number of ancillary data bytes consumed by the encoder.

Referenced by [main\(\)](#).

#### 7.5.2.2 INT AACENC\_OutArgs::numInSamples

Number of input audio samples consumed by the encoder.

Referenced by [main\(\)](#).

#### 7.5.2.3 INT AACENC\_OutArgs::numOutBytes

Number of valid bitstream bytes generated during [aacEncEncode\(\)](#).

Referenced by [main\(\)](#).

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

- [aacenc\\_lib.h](#)
-

# Chapter 8

## File Documentation

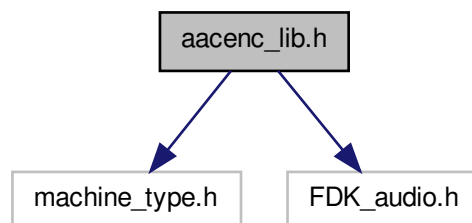
### 8.1 aacenc\_lib.h File Reference

FDK AAC Encoder library interface header file.

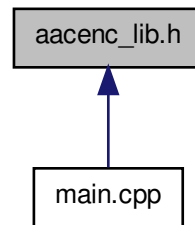
```
#include "machine_type.h"
```

```
#include "FDK_audio.h"
```

Include dependency graph for aacenc\_lib.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [AACENC\\_InfoStruct](#)
- struct [AACENC\\_BufDesc](#)
- struct [AACENC\\_InArgs](#)
- struct [AACENC\\_OutArgs](#)
- struct [AACENC\\_MetaData](#)

## Typedefs

- typedef struct AACENCODER \* [HANDLE\\_AACENCODER](#)

## Enumerations

- enum [AACENC\\_ERROR](#) {  
    [AACENC\\_OK](#) = 0x0000,  
    [AACENC\\_INVALID\\_HANDLE](#) = 0x0020,  
    [AACENC\\_MEMORY\\_ERROR](#) = 0x0021,  
    [AACENC\\_UNSUPPORTED\\_PARAMETER](#) = 0x0022,  
    [AACENC\\_INVALID\\_CONFIG](#) = 0x0023,  
    [AACENC\\_INIT\\_ERROR](#) = 0x0040,  
    [AACENC\\_INIT\\_AAC\\_ERROR](#) = 0x0041,  
    [AACENC\\_INIT\\_SBR\\_ERROR](#) = 0x0042,  
    [AACENC\\_INIT\\_TP\\_ERROR](#) = 0x0043,  
    [AACENC\\_INIT\\_META\\_ERROR](#) = 0x0044,  
    [AACENC\\_ENCODE\\_ERROR](#) = 0x0060,  
    [AACENC\\_ENCODE\\_EOF](#) = 0x0080 }  
}
-

- enum AACENC\_BufferIdentifier {  
  IN\_AUDIO\_DATA = 0,  
  IN\_ANCILLRY\_DATA = 1,  
  IN\_METADATA\_SETUP = 2,  
  OUT\_BITSTREAM\_DATA = 3,  
  OUT\_AU\_SIZES = 4 }
- enum AACENC\_METADATA\_DRC\_PROFILE {  
  AACENC\_METADATA\_DRC\_NONE = 0,  
  AACENC\_METADATA\_DRC\_FILMSTANDARD = 1,  
  AACENC\_METADATA\_DRC\_FILMLIGHT = 2,  
  AACENC\_METADATA\_DRC\_MUSICSTANDARD = 3,  
  AACENC\_METADATA\_DRC\_MUSICLIGHT = 4,  
  AACENC\_METADATA\_DRC\_SPEECH = 5 }
- enum AACENC\_CTRLFLAGS {  
  AACENC\_INIT\_NONE = 0x0000,  
  AACENC\_INIT\_CONFIG = 0x0001,  
  AACENC\_INIT\_STATES = 0x0002,  
  AACENC\_INIT\_TRANSPORT = 0x1000,  
  AACENC\_RESET\_INBUFFER = 0x2000,  
  AACENC\_INIT\_ALL = 0xFFFF }
- enum AACENC\_PARAM {  
  AACENC\_AOT = 0x0100,  
  AACENC\_BITRATE = 0x0101,  
  AACENC\_BITRATEMODE = 0x0102,  
  AACENC\_SAMPLERATE = 0x0103,  
  AACENC\_SBR\_MODE = 0x0104,  
  AACENC\_GRANULE\_LENGTH = 0x0105,  
  AACENC\_CHANNELMODE = 0x0106,  
  AACENC\_CHANNELORDER = 0x0107,  
  AACENC\_AFTERBURNER = 0x0200,  
  AACENC\_BANDWIDTH = 0x0203,  
  AACENC\_TRANSMUX = 0x0300,  
  AACENC\_HEADER\_PERIOD = 0x0301,  
  AACENC\_SIGNALING\_MODE = 0x0302,  
  AACENC\_TPSUBFRAMES = 0x0303,  
  AACENC\_PROTECTION = 0x0306,  
  AACENC\_ANCILLARY\_BITRATE = 0x0500,  
  AACENC\_METADATA\_MODE = 0x0600,  
  AACENC\_CONTROL\_STATE = 0xFF00,  
  AACENC\_NONE = 0xFFFF }

*AAC encoder setting parameters.*

---

## Functions

- [AACENC\\_ERROR aacEncOpen](#) ([HANDLE\\_AACENCODER](#) \*phAacEncoder, const UINT encModules, const UINT maxChannels)  
*Open an instance of the encoder.*
- [AACENC\\_ERROR aacEncClose](#) ([HANDLE\\_AACENCODER](#) \*phAacEncoder)  
*Close the encoder instance.*
- [AACENC\\_ERROR aacEncEncode](#) (const [HANDLE\\_AACENCODER](#) hAacEncoder, const [AACENC\\_BufDesc](#) \*inBufDesc, const [AACENC\\_BufDesc](#) \*outBufDesc, const [AACENC\\_InArgs](#) \*inargs, [AACENC\\_OutArgs](#) \*outargs)  
*Encode audio data.*
- [AACENC\\_ERROR aacEncInfo](#) (const [HANDLE\\_AACENCODER](#) hAacEncoder, [AACENC\\_InfoStruct](#) \*pInfo)  
*Acquire info about present encoder instance.*
- [AACENC\\_ERROR aacEncoder\\_SetParam](#) (const [HANDLE\\_AACENCODER](#) hAacEncoder, const [AACENC\\_PARAM](#) param, const UINT value)  
*Set one single AAC encoder parameter.*
- [UINT aacEncoder\\_GetParam](#) (const [HANDLE\\_AACENCODER](#) hAacEncoder, const [AACENC\\_PARAM](#) param)  
*Get one single AAC encoder parameter.*
- [AACENC\\_ERROR aacEncGetLibInfo](#) ([LIB\\_INFO](#) \*info)  
*Get information about encoder library build.*

### 8.1.1 Detailed Description

FDK AAC Encoder library interface header file.

### 8.1.2 Typedef Documentation

#### 8.1.2.1 typedef struct AACENCODER\* HANDLE\_AACENCODER

AAC encoder handle.

### 8.1.3 Enumeration Type Documentation

#### 8.1.3.1 enum AACENC\_BufferIdentifier

AAC encoder buffer descriptors identifier. This identifier are used within buffer descriptors [AACENC\\_BufDesc::bufferIdentifiers](#).

#### Enumerator:

[IN\\_AUDIO\\_DATA](#) Audio input buffer, interleaved INT\_PCM samples.

---

*IN Ancillary* *DATA* Ancillary data to be embedded into bitstream.

*IN Metadata* *SETUP* Setup structure for embedding meta data.

*OUT Bitstream* *DATA* Buffer holds bitstream output data.

*OUT AU* *SIZES* Buffer contains sizes of each access unit. This information is necessary for super-framing.

### 8.1.3.2 enum AACENC\_CTRLFLAGS

AAC encoder control flags.

In interaction with the [AACENC\\_CONTROL\\_STATE](#) parameter it is possible to get information about the internal initialization process. It is also possible to overwrite the internal state from extern when necessary.

#### Enumerator:

*AACENC\_INIT\_NONE* Do not trigger initialization.

*AACENC\_INIT\_CONFIG* Initialize all encoder modules configuration.

*AACENC\_INIT\_STATES* Reset all encoder modules history buffer.

*AACENC\_INIT\_TRANSPORT* Initialize transport lib with new parameters.

*AACENC\_RESET\_INBUFFER* Reset fill level of internal input buffer.

*AACENC\_INIT\_ALL* Initialize all.

### 8.1.3.3 enum AACENC\_ERROR

AAC encoder error codes.

#### Enumerator:

*AACENC\_OK* No error happened. All fine.

*AACENC\_INVALID\_HANDLE* Handle passed to function call was invalid.

*AACENC\_MEMORY\_ERROR* Memory allocation failed.

*AACENC\_UNSUPPORTED\_PARAMETER* Parameter not available.

*AACENC\_INVALID\_CONFIG* Configuration not provided.

*AACENC\_INIT\_ERROR* General initialization error.

*AACENC\_INIT\_AAC\_ERROR* AAC library initialization error.

*AACENC\_INIT\_SBR\_ERROR* SBR library initialization error.

*AACENC\_INIT\_TP\_ERROR* Transport library initialization error.

*AACENC\_INIT\_META\_ERROR* Meta data library initialization error.

*AACENC\_ENCODE\_ERROR* The encoding process was interrupted by an unexpected error.

*AACENC\_ENCODE\_EOF* End of file reached.

---

### 8.1.3.4 enum AACENC\_METADATA\_DRC\_PROFILE

Meta Data Compression Profiles.

#### Enumerator:

**AACENC\_METADATA\_DRC\_NONE** None.  
**AACENC\_METADATA\_DRC\_FILMSTANDARD** Film standard.  
**AACENC\_METADATA\_DRC\_FILMLIGHT** Film light.  
**AACENC\_METADATA\_DRC\_MUSICSTANDARD** Music standard.  
**AACENC\_METADATA\_DRC\_MUSICLIGHT** Music light.  
**AACENC\_METADATA\_DRC\_SPEECH** Speech.

### 8.1.3.5 enum AACENC\_PARAM

AAC encoder setting parameters.

Use [aacEncoder\\_SetParam\(\)](#) function to configure, or use [aacEncoder\\_GetParam\(\)](#) function to read the internal status of the following parameters.

#### Enumerator:

**AACENC\_AOT** Audio object type. See AUDIO\_OBJECT\_TYPE in FDK\_audio.h.

- 2: MPEG-4 AAC Low Complexity.
- 5: MPEG-4 AAC Low Complexity with Spectral Band Replication (HE-AAC).
- 29: MPEG-4 AAC Low Complexity with Spectral Band Replication and Parametric Stereo (HE-AAC v2). This configuration can be used only with stereo input audio data.
- 23: MPEG-4 AAC Low-Delay.
- 39: MPEG-4 AAC Enhanced Low-Delay. Since there is no AUDIO\_OBJECT\_TYPE for ELD in combination with SBR defined, enable SBR explicitly by [AACENC\\_SBR\\_MODE](#) parameter.
- 129: MPEG-2 AAC Low Complexity.
- 132: MPEG-2 AAC Low Complexity with Spectral Band Replication (HE-AAC).
- 156: MPEG-2 AAC Low Complexity with Spectral Band Replication and Parametric Stereo (HE-AAC v2). This configuration can be used only with stereo input audio data.

**AACENC\_BITRATE** Total encoder bitrate. This parameter is mandatory and interacts with [AACENC\\_BITRATEMODE](#).

- CBR: Bitrate in bits/second. See [Supported Bitrates](#) for details.

**AACENC\_BITRATEMODE** Bitrate mode. Configuration can be different kind of bitrate configurations:

- 0: Constant bitrate, use bitrate according to [AACENC\\_BITRATE](#). (default) Within none LD/ELD AUDIO\_OBJECT\_TYPE, the CBR mode makes use of full allowed bitreservoir. In contrast, at Low-Delay AUDIO\_OBJECT\_TYPE the bitreservoir is kept very small.
- 8: LD/ELD full bitreservoir for packet based transmission.

**AACENC\_SAMPLERATE** Audio input data sampling rate. Encoder supports following sampling rates: 8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100, 48000, 64000, 88200, 96000

**AACENC\_SBR\_MODE** Configure SBR independently of the chosen Audio Object Type AUDIO\_OBJECT\_TYPE:. This parameter is only available for ELD.

---



- 0: Disable Spectral Band Replication.
- 1: Enable Spectral Band Replication.

**AACENC\_GRANULE\_LENGTH** Core encoder (AAC) audio frame length in samples:

- 1024: Default configuration.
- 512: Default LD/ELD configuration.
- 480: Optional length in LD/ELD configuration.

**AACENC\_CHANNELMODE** Set explicit channel mode. Channel mode must match with number of input channels.

- 1-6: MPEG channel modes supported, see CHANNEL\_MODE in FDK\_audio.h.

**AACENC\_CHANNELORDER** Input audio data channel ordering scheme:

- 0: MPEG channel ordering (e. g. 5.1: C, L, R, SL, SR, LFE). (default)
- 1: WAVE file format channel ordering (e. g. 5.1: L, R, C, LFE, SL, SR).

**AACENC\_AFTERBURNER** This parameter controls the use of the afterburner feature. The afterburner is a type of analysis by synthesis algorithm which increases the audio quality but also the required processing power. It is recommended to always activate this if additional memory consumption and processing power consumption is not a problem. If increased MHz and memory consumption are an issue then the MHz and memory cost of this optional module need to be evaluated against the improvement in audio quality on a case by case basis.

- 0: Disable afterburner (default).
- 1: Enable afterburner.

**AACENC\_BANDWIDTH** Core encoder audio bandwidth:

- 0: Determine bandwidth internally (default, see chapter [Bandwidth](#)).
- 1 to fs/2: Frequency bandwidth in Hertz. (Experts only, better do not touch this value to avoid degraded audio quality)

**AACENC\_TRANSMUX** Transport type to be used. See TRANSPORT\_TYPE in FDK\_audio.h. Following types can be configured in encoder library:

- 0: raw access units
- 1: ADIF bitstream format
- 2: ADTS bitstream format
- 6: Audio Mux Elements (LATM) with muxConfigPresent = 1
- 7: Audio Mux Elements (LATM) with muxConfigPresent = 0, out of band StreamMuxConfig
- 10: Audio Sync Stream (LOAS)

**AACENC\_HEADER\_PERIOD** Frame count period for sending in-band configuration buffers within LATM/LOAS transport layer. Additionally this parameter configures the PCE repetition period in raw\_data\_block(). See encPCE.

- 0xFF: auto-mode default 10 for TT\_MP4\_ADTS, TT\_MP4\_LOAS and TT\_MP4\_LATM\_MCP1, otherwise 0.
- n: Frame count period.

**AACENC\_SIGNALING\_MODE** Signaling mode of the extension AOT:

- 0: Implicit backward compatible signaling. (default)
  - 1: Explicit SBR and implicit PS signaling.
  - 2: Explicit hierarchical signaling.
-

The use of backward-compatible implicit signaling is recommended if the user specifically aims at preserving compatibility with decoders only capable of decoding AAC-LC. Otherwise use non-backward-compatible explicit signaling. Bitstream formats ADTS and ADIF can only do implicit signaling.

**AACENC\_TPSUBFRAMES** Number of sub frames in a transport frame for LOAS/LATM or ADTS (default 1).

- ADTS: Maximum number of sub frames restricted to 4.
- LOAS/LATM: Maximum number of sub frames restricted to 2.

**AACENC\_PROTECTION** Configure protection in transport layer:

- 0: No protection. (default)
- 1: CRC active for ADTS bitstream format.

**AACENC\_ANCILLARY\_BITRATE** Constant ancillary data bitrate in bits/second.

- 0: Either no ancillary data or insert exact number of bytes, denoted via input parameter, numAncBytes in [AACENC\\_InArgs](#).
- else: Insert ancillary data with specified bitrate.

**AACENC\_METADATA\_MODE** Configure Meta Data. See [AACENC\\_MetaData](#) for further details:

- 0: Do not embed any metadata.
- 1: Embed MPEG defined metadata only.
- 2: Embed all metadata.

**AACENC\_CONTROL\_STATE** There is an automatic process which internally reconfigures the encoder instance when a configuration parameter changed or an error occurred. This parameter allows overwriting or getting the control status of this process. See [AACENC\\_CTRLFLAGS](#).

**AACENC\_NONE** -----

## 8.1.4 Function Documentation

### 8.1.4.1 AACENC\_ERROR aacEncClose ( HANDLE\_AACENCODER \* *phAacEncoder* )

Close the encoder instance.

Deallocate encoder instance and free whole memory.

#### Parameters

*phAacEncoder* Pointer to the encoder handle to be deallocated.

#### Returns

- AACENC\_OK, on success.
- AACENC\_INVALID\_HANDLE, on failure.

Referenced by main().

### 8.1.4.2 AACENC\_ERROR aacEncEncode ( const HANDLE\_AACENCODER *hAacEncoder*, const AACENC\_BufDesc \* *inBufDesc*, const AACENC\_BufDesc \* *outBufDesc*, const AACENC\_InArgs \* *inargs*, AACENC\_OutArgs \* *outargs* )

Encode audio data.

This function is mainly for encoding audio data. In addition the function can be used for an encoder (re)configuration process.

- PCM input data will be retrieved from external input buffer until the fill level allows encoding a single frame. This functionality allows an external buffer with reduced size in comparison to the AAC or HE-AAC audio frame length.
- If the value of the input samples argument is zero, just internal reinitialization will be applied if it is requested.
- At the end of a file the flushing process can be triggered via setting the value of the input samples argument to -1. The encoder delay lines are fully flushed when the encoder returns no valid bitstream data [AACENC\\_OutArgs::numOutBytes](#). Furthermore the end of file is signaled by the return value AACENC\_ENCODE\_EOF.
- If an error occurred in the previous frame or any of the encoder parameters changed, an internal reinitialization process will be applied before encoding the incoming audio samples.
- The function can also be used for an independent reconfiguration process without encoding. The first parameter has to be a valid encoder handle and all other parameters can be set to NULL.
- If the size of the external bitbuffer in outBufDesc is not sufficient for writing the whole bitstream, an internal error will be the return value and a reconfiguration will be triggered.

### Parameters

*hAacEncoder* A valid AAC encoder handle.

*inBufDesc* Input buffer descriptor, see [AACENC\\_BufDesc](#):

- At least one input buffer with audio data is expected.
- Optionally a second input buffer with ancillary data can be fed.

*outBufDesc* Output buffer descriptor, see [AACENC\\_BufDesc](#):

- Provide one output buffer for the encoded bitstream.

*inargs* Input arguments, see [AACENC\\_InArgs](#).

*outargs* Output arguments, [AACENC\\_OutArgs](#).

### Returns

- AACENC\_OK, on success.
- AACENC\_INVALID\_HANDLE, AACENC\_ENCODE\_ERROR, on failure in encoding process.
- AACENC\_INVALID\_CONFIG, AACENC\_INIT\_ERROR, AACENC\_INIT\_AAC\_ERROR, AACENC\_INIT\_SBR\_ERROR, AACENC\_INIT\_TP\_ERROR, AACENC\_INIT\_META\_ERROR, on failure in encoder initialization.
- AACENC\_ENCODE\_EOF, when flushing fully concluded.

Referenced by main().

#### 8.1.4.3 AACENC\_ERROR aacEncGetLibInfo ( LIB\_INFO \* info )

Get information about encoder library build.

Fill a given LIB\_INFO structure with library version information.

### Parameters

*info* Pointer to an allocated LIB\_INFO struct.

---

**Returns**

- AACENC\_OK, on success.
- AACENC\_INVALID\_HANDLE, AACENC\_INIT\_ERROR, on failure.

Referenced by main().

#### 8.1.4.4 AACENC\_ERROR aacEncInfo ( const HANDLE\_AACENCODER *hAacEncoder*, AACENC\_InfoStruct \* *pInfo* )

Acquire info about present encoder instance.

This function retrieves information of the encoder configuration. In addition to informative internal states, a configuration data block of the current encoder settings will be returned. The format is either Audio Specific Config in case of Raw Packets transport format or StreamMuxConfig in case of LOAS/LATM transport format. The configuration data block is binary coded as specified in ISO/IEC 14496-3 (MPEG-4 audio), to be used directly for MPEG-4 File Format or RFC3016 or RFC3640 applications.

**Parameters**

- hAacEncoder* A valid AAC encoder handle.
- pInfo* Pointer to [AACENC\\_InfoStruct](#). Filled on return.

**Returns**

- AACENC\_OK, on succes.
- AACENC\_INIT\_ERROR, on failure.

Referenced by main().

#### 8.1.4.5 UINT aacEncoder\_GetParam ( const HANDLE\_AACENCODER *hAacEncoder*, const AACENC\_PARAM *param* )

Get one single AAC encoder parameter.

This function is the complement to [aacEncoder\\_SetParam\(\)](#). After encoder reinitialization with user defined settings, the internal status can be obtained of each parameter, specified with [AACENC\\_PARAM](#).

**Parameters**

- hAacEncoder* A valid AAC encoder handle.
- param* Parameter to be returned. See [AACENC\\_PARAM](#).

**Returns**

Internal configuration value of specified parameter [AACENC\\_PARAM](#).

Referenced by main().

#### 8.1.4.6 AACENC\_ERROR aacEncoder\_SetParam ( const HANDLE\_AACENCODER *hAacEncoder*, const AACENC\_PARAM *param*, const UINT *value* )

Set one single AAC encoder parameter.

---

This function allows configuration of all encoder parameters specified in [AACENC\\_PARAM](#). Each parameter must be set with a separate function call. An internal validation of the configuration value range will be done and an internal reconfiguration will be signaled. The actual configuration adoption is part of the subsequent [aacEncEncode\(\)](#) call.

#### Parameters

- hAacEncoder* A valid AAC encoder handle.
- param* Parameter to be set. See [AACENC\\_PARAM](#).
- value* Parameter value. See parameter description in [AACENC\\_PARAM](#).

#### Returns

- AACENC\_OK, on success.
- AACENC\_INVALID\_HANDLE, AACENC\_UNSUPPORTED\_PARAMETER, AACENC\_INVALID\_CONFIG, on failure.

Referenced by [main\(\)](#).

#### 8.1.4.7 AACENC\_ERROR aacEncOpen ( HANDLE\_AACENCODER \* phAacEncoder, const UINT encModules, const UINT maxChannels )

Open an instance of the encoder.

Allocate memory for an encoder instance with a functional range denoted by the function parameters. Preinitialize encoder instance with default configuration.

#### Parameters

- phAacEncoder* A pointer to an encoder handle. Initialized on return.
- encModules* Specify encoder modules to be supported in this encoder instance:
  - 0x0: Allocate memory for all available encoder modules.
  - else: Select memory allocation regarding encoder modules. Following flags are possible and can be combined.
    - 0x01: AAC module.
    - 0x02: SBR module.
    - 0x04: PS module.
    - 0x10: Metadata module.
    - example: (0x01|0x02|0x04|0x10) allocates all modules and is equivalent to default configuration denoted by 0x0.
- maxChannels* Number of channels to be allocated. This parameter can be used in different ways:
  - 0: Allocate maximum number of AAC and SBR channels as supported by the library.
  - nChannels: Use same maximum number of channels for allocating memory in AAC and SBR module.
  - nChannels | (nSbrCh<<8): Number of SBR channels can be different to AAC channels to save data memory.

#### Returns

- AACENC\_OK, on success.
- AACENC\_INVALID\_HANDLE, AACENC\_MEMORY\_ERROR, AACENC\_INVALID\_CONFIG, on failure.

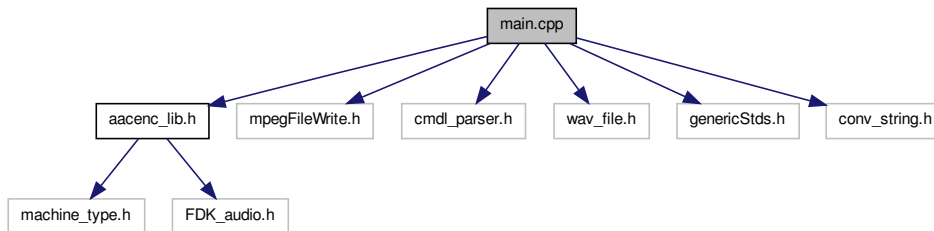
Referenced by [main\(\)](#).

---

## 8.2 main.cpp File Reference

```
#include "aacenc_lib.h"
#include "mpegFileWrite.h"
#include "cmdl_parser.h"
#include "wav_file.h"
#include "genericStds.h"
#include "conv_string.h"
```

Include dependency graph for main.cpp:



### Functions

- int [main](#) (int ac, char \*av[ ])

#### 8.2.1 Detailed Description

An example of how to use the FDK AAC Encoder API. See chapter [Calling Sequence](#) for details.

#### 8.2.2 Function Documentation

##### 8.2.2.1 int main ( int ac, char \* av[ ] )

References AACENC\_AFTERBURNER, AACENC\_ANCILLARY\_BITRATE, AACENC\_AOT, AACENC\_BANDWIDTH, AACENC\_BITRATE, AACENC\_BITRATEMODE, AACENC\_CHANNELMODE, AACENC\_CHANNELORDER, AACENC\_ENCODE\_EOF, AACENC\_ENCODE\_ERROR, AACENC\_GRANULE\_LENGTH, AACENC\_HEADER\_PERIOD, AACENC\_OK, AACENC\_PROTECTION, AACENC\_SAMPLERATE, AACENC\_SBR\_MODE, AACENC\_SIGNALING\_MODE, AACENC\_TPSUBFRAMES, AACENC\_TRANSMUX, aacEncClose(), aacEncEncode(), aacEncGetLibInfo(), aacEncInfo(), aacEncoder\_GetParam(), aacEncoder\_SetParam(), aacEncOpen(), AACENC\_BufDesc::bufEISizes, AACENC\_BufDesc::bufferIdentifiers, AACENC\_BufDesc::bufs, AACENC\_BufDesc::bufSizes, AACENC\_InfoStruct::confBuf, AACENC\_InfoStruct::confSize, AACENC\_InfoStruct::frameLength, AACENC\_InfoStruct::inputChannels, AACENC\_OutArgs::numAncBytes, AACENC\_InArgs::numAncBytes, AACENC\_BufDesc::numBufs, AACENC\_OutArgs::numInSamples, AACENC\_InArgs::numInSamples, and AACENC\_OutArgs::numOutBytes.

---

# Index

AACENC\_AFTERTURNER  
aacenc\_lib.h, 35

AACENC\_ANCELLARY\_BITRATE  
aacenc\_lib.h, 36

AACENC\_AOT  
aacenc\_lib.h, 34

AACENC\_BANDWIDTH  
aacenc\_lib.h, 35

AACENC\_BITRATE  
aacenc\_lib.h, 34

AACENC\_BITRATEMODE  
aacenc\_lib.h, 34

AACENC\_CHANNELMODE  
aacenc\_lib.h, 35

AACENC\_CHANNELORDER  
aacenc\_lib.h, 35

AACENC\_CONTROL\_STATE  
aacenc\_lib.h, 36

AACENC\_ENCODE\_EOF  
aacenc\_lib.h, 33

AACENC\_ENCODE\_ERROR  
aacenc\_lib.h, 33

AACENC\_GRANULE\_LENGTH  
aacenc\_lib.h, 35

AACENC\_HEADER\_PERIOD  
aacenc\_lib.h, 35

AACENC\_INIT\_AAC\_ERROR  
aacenc\_lib.h, 33

AACENC\_INIT\_ALL  
aacenc\_lib.h, 33

AACENC\_INIT\_CONFIG  
aacenc\_lib.h, 33

AACENC\_INIT\_ERROR  
aacenc\_lib.h, 33

AACENC\_INIT\_META\_ERROR  
aacenc\_lib.h, 33

AACENC\_INIT\_NONE  
aacenc\_lib.h, 33

AACENC\_INIT\_SBR\_ERROR  
aacenc\_lib.h, 33

AACENC\_INIT\_STATES  
aacenc\_lib.h, 33

AACENC\_INIT\_TP\_ERROR  
aacenc\_lib.h, 33

AACENC\_INIT\_TRANSPORT  
aacenc\_lib.h, 33

AACENC\_INVALID\_CONFIG  
aacenc\_lib.h, 33

AACENC\_INVALID\_HANDLE  
aacenc\_lib.h, 33

aacenc\_lib.h  
AACENC\_AFTERTURNER, 35  
AACENC\_ANCELLARY\_BITRATE, 36  
AACENC\_AOT, 34  
AACENC\_BANDWIDTH, 35  
AACENC\_BITRATE, 34  
AACENC\_BITRATEMODE, 34  
AACENC\_CHANNELMODE, 35  
AACENC\_CHANNELORDER, 35  
AACENC\_CONTROL\_STATE, 36  
AACENC\_ENCODE\_EOF, 33  
AACENC\_ENCODE\_ERROR, 33  
AACENC\_GRANULE\_LENGTH, 35  
AACENC\_HEADER\_PERIOD, 35  
AACENC\_INIT\_AAC\_ERROR, 33  
AACENC\_INIT\_ALL, 33  
AACENC\_INIT\_CONFIG, 33  
AACENC\_INIT\_ERROR, 33  
AACENC\_INIT\_META\_ERROR, 33  
AACENC\_INIT\_NONE, 33  
AACENC\_INIT\_SBR\_ERROR, 33  
AACENC\_INIT\_STATES, 33  
AACENC\_INIT\_TP\_ERROR, 33  
AACENC\_INIT\_TRANSPORT, 33  
AACENC\_INVALID\_CONFIG, 33  
AACENC\_INVALID\_HANDLE, 33  
AACENC\_MEMORY\_ERROR, 33  
AACENC\_METADATA\_DRC\_FILMLIGHT,  
34  
AACENC\_METADATA\_DRC\_-  
FILMSTANDARD, 34  
AACENC\_METADATA\_DRC\_-  
MUSICLIGHT, 34  
AACENC\_METADATA\_DRC\_-  
MUSICSTANDARD, 34  
AACENC\_METADATA\_DRC\_NONE, 34  
AACENC\_METADATA\_DRC\_SPEECH, 34  
AACENC\_METADATA\_MODE, 36  
AACENC\_NONE, 36  
AACENC\_OK, 33

- AACENC\_PROTECTION, 36
  - AACENC\_RESET\_INBUFFER, 33
  - AACENC\_SAMPLERATE, 34
  - AACENC\_SBR\_MODE, 34
  - AACENC\_SIGNALING\_MODE, 35
  - AACENC\_TPSUBFRAMES, 36
  - AACENC\_TRANSMUX, 35
  - AACENC\_UNSUPPORTED\_PARAMETER, 33
  - IN\_ANCILLRY\_DATA, 32
  - IN\_AUDIO\_DATA, 32
  - IN\_METADATA\_SETUP, 33
  - OUT\_AU\_SIZES, 33
  - OUT\_BITSTREAM\_DATA, 33
  - AACENC\_MEMORY\_ERROR
    - aacenc\_lib.h, 33
  - AACENC\_METADATA\_DRC\_FILMLIGHT
    - aacenc\_lib.h, 34
  - AACENC\_METADATA\_DRC\_FILMSTANDARD
    - aacenc\_lib.h, 34
  - AACENC\_METADATA\_DRC\_MUSICLIGHT
    - aacenc\_lib.h, 34
  - AACENC\_METADATA\_DRC\_-
    - MUSICSTANDARD
    - aacenc\_lib.h, 34
  - AACENC\_METADATA\_DRC\_NONE
    - aacenc\_lib.h, 34
  - AACENC\_METADATA\_DRC\_SPEECH
    - aacenc\_lib.h, 34
  - AACENC\_METADATA\_MODE
    - aacenc\_lib.h, 36
  - AACENC\_NONE
    - aacenc\_lib.h, 36
  - AACENC\_OK
    - aacenc\_lib.h, 33
  - AACENC\_PROTECTION
    - aacenc\_lib.h, 36
  - AACENC\_RESET\_INBUFFER
    - aacenc\_lib.h, 33
  - AACENC\_SAMPLERATE
    - aacenc\_lib.h, 34
  - AACENC\_SBR\_MODE
    - aacenc\_lib.h, 34
  - AACENC\_SIGNALING\_MODE
    - aacenc\_lib.h, 35
  - AACENC\_TPSUBFRAMES
    - aacenc\_lib.h, 36
  - AACENC\_TRANSMUX
    - aacenc\_lib.h, 35
  - AACENC\_UNSUPPORTED\_PARAMETER
    - aacenc\_lib.h, 33
  - AACENC\_BufDesc, 23
    - bufEISizes, 23
    - bufferIdentifiers, 23
    - bufs, 23
    - bufSizes, 24
    - numBufs, 24
  - AACENC\_BufferIdentifier
    - aacenc\_lib.h, 32
  - AACENC\_CTRLFLAGS
    - aacenc\_lib.h, 33
  - AACENC\_ERROR
    - aacenc\_lib.h, 33
  - AACENC\_InArgs, 24
    - numAncBytes, 24
    - numInSamples, 24
  - AACENC\_InfoStruct, 25
    - confBuf, 25
    - confSize, 25
    - encoderDelay, 25
    - frameLength, 25
    - inBufFillLevel, 25
    - inputChannels, 25
    - maxAncBytes, 26
    - maxOutBufBytes, 26
  - aacenc\_lib.h, 29
    - AACENC\_BufferIdentifier, 32
    - AACENC\_CTRLFLAGS, 33
    - AACENC\_ERROR, 33
    - AACENC\_METADATA\_DRC\_PROFILE, 33
    - AACENC\_PARAM, 34
    - aacEncClose, 36
    - aacEncEncode, 36
    - aacEncGetLibInfo, 37
    - aacEncInfo, 38
    - aacEncoder\_GetParam, 38
    - aacEncoder\_SetParam, 38
    - aacEncOpen, 39
    - HANDLE\_AACENCODER, 32
  - AACENC\_MetaData, 26
    - centerMixLevel, 26
    - comp\_profile, 26
    - comp\_TargetRefLevel, 27
    - dolbySurroundMode, 27
    - drc\_profile, 27
    - drc\_TargetRefLevel, 27
    - ETSI\_DmxLvl\_present, 27
    - PCE\_mixdown\_idx\_present, 27
    - prog\_ref\_level, 27
    - prog\_ref\_level\_present, 27
    - surroundMixLevel, 27
  - AACENC\_METADATA\_DRC\_PROFILE
    - aacenc\_lib.h, 33
  - AACENC\_OutArgs, 28
    - numAncBytes, 28
    - numInSamples, 28
    - numOutBytes, 28
  - AACENC\_PARAM
-



- aacenc\_lib.h, 34
  - aacEncClose
    - aacenc\_lib.h, 36
  - aacEncEncode
    - aacenc\_lib.h, 36
  - aacEncGetLibInfo
    - aacenc\_lib.h, 37
  - aacEncInfo
    - aacenc\_lib.h, 38
  - aacEncoder\_GetParam
    - aacenc\_lib.h, 38
  - aacEncoder\_SetParam
    - aacenc\_lib.h, 38
  - aacEncOpen
    - aacenc\_lib.h, 39
  
  - bufElSizes
    - AACENC\_BufDesc, 23
  - bufferIdentifiers
    - AACENC\_BufDesc, 23
  - bufs
    - AACENC\_BufDesc, 23
  - bufSizes
    - AACENC\_BufDesc, 24
  
  - centerMixLevel
    - AACENC\_MetaData, 26
  - comp\_profile
    - AACENC\_MetaData, 26
  - comp\_TargetRefLevel
    - AACENC\_MetaData, 27
  - confBuf
    - AACENC\_InfoStruct, 25
  - confSize
    - AACENC\_InfoStruct, 25
  
  - dolbySurroundMode
    - AACENC\_MetaData, 27
  - drc\_profile
    - AACENC\_MetaData, 27
  - drc\_TargetRefLevel
    - AACENC\_MetaData, 27
  
  - encoderDelay
    - AACENC\_InfoStruct, 25
  - ETSI\_DmxLvl\_present
    - AACENC\_MetaData, 27
  
  - frameLength
    - AACENC\_InfoStruct, 25
  
  - HANDLE\_AACENCODER
    - aacenc\_lib.h, 32
  
  - IN\_ANCILLRY\_DATA
    - aacenc\_lib.h, 32
  - IN\_AUDIO\_DATA
    - aacenc\_lib.h, 32
  - IN\_METADATA\_SETUP
    - aacenc\_lib.h, 33
  - inBufFillLevel
    - AACENC\_InfoStruct, 25
  - inputChannels
    - AACENC\_InfoStruct, 25
  
  - main
    - main.cpp, 40
  - main.cpp, 40
    - main, 40
  - maxAncBytes
    - AACENC\_InfoStruct, 26
  - maxOutBufBytes
    - AACENC\_InfoStruct, 26
  
  - numAncBytes
    - AACENC\_InArgs, 24
    - AACENC\_OutArgs, 28
  - numBufs
    - AACENC\_BufDesc, 24
  - numInSamples
    - AACENC\_InArgs, 24
    - AACENC\_OutArgs, 28
  - numOutBytes
    - AACENC\_OutArgs, 28
  
  - OUT\_AU\_SIZES
    - aacenc\_lib.h, 33
  - OUT\_BITSTREAM\_DATA
    - aacenc\_lib.h, 33
  
  - PCE\_mixdown\_idx\_present
    - AACENC\_MetaData, 27
  - prog\_ref\_level
    - AACENC\_MetaData, 27
  - prog\_ref\_level\_present
    - AACENC\_MetaData, 27
  
  - surroundMixLevel
    - AACENC\_MetaData, 27
-