

Advanced Audio Coding Encoder Library

MPEG-2 and MPEG-4,
AAC Low-Complexity (AAC-LC),
High-Efficiency AAC v2 (HE-AAC v2),
AAC Low-Delay (AAC-LD),
AAC Enhanced Low-Delay (AAC-ELD v2)
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.

© Copyright 1999-2018 Fraunhofer IIS. All rights reserved.

Contents

1	Introduction	1
1.1	Scope	1
1.2	Encoder Basics	1
2	Library Usage	3
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	6
2.8	Encoder Reconfiguration	7
2.9	Encoder Parametrization	7
2.9.1	Mandatory Encoder Parameters	7
2.9.2	Channel Mode Configuration	8
2.9.3	Bitreservoir Configuration	8
2.9.4	Variable Bitrate Mode	8
2.9.5	Audio Quality Considerations	9
2.9.6	ELD Auto Configuration Mode	9
2.9.7	Reduced Delay (Downscaled) Mode	10
2.10	Audio Channel Configuration	10
2.11	Supported Bitrates	12
2.12	Recommended Sampling Rate and Bitrate Combinations	12
2.12.1	AAC-LC, HE-AAC, HE-AACv2 in Dualrate SBR mode.	12
2.12.2	AAC-LD, AAC-ELD, AAC-ELD with SBR in Dualrate SBR mode.	13
2.12.3	AAC-ELD with SBR in Downsampled SBR mode.	14
2.12.4	AAC-ELD v2, AAC-ELD v2 with SBR.	14
3	Encoder Behaviour	17
3.1	Bandwidth	17
3.2	Frame Sizes & Bit Reservoir	17
3.2.1	Estimating Average Frame Sizes	19
3.3	Encoder Tools	19
4	Class Index	21
4.1	Class List	21
5	File Index	23
5.1	File List	23

6	Class Documentation	25
6.1	AACENC_BufDesc Struct Reference	25
6.1.1	Detailed Description	25
6.1.2	Member Data Documentation	25
	numBufs	25
	bufs	25
	bufferIdentifiers	25
	bufSizes	26
	bufEISizes	26
6.2	AACENC_InArgs Struct Reference	26
6.2.1	Detailed Description	26
6.2.2	Member Data Documentation	26
	numInSamples	26
	numAncBytes	26
6.3	AACENC_InfoStruct Struct Reference	26
6.3.1	Detailed Description	27
6.3.2	Member Data Documentation	27
	maxOutBufBytes	27
	maxAncBytes	27
	inBufFillLevel	27
	inputChannels	27
	frameLength	27
	nDelay	28
	nDelayCore	28
	confBuf	28
	confSize	28
6.4	AACENC_MetaData Struct Reference	28
6.4.1	Detailed Description	29
6.4.2	Member Data Documentation	29
	drc_profile	29
	comp_profile	29
	drc_TargetRefLevel	29
	comp_TargetRefLevel	29
	prog_ref_level_present	29
	prog_ref_level	29
	PCE_mixdown_idx_present	29
	ETSI_DmxLvl_present	30
	centerMixLevel	30
	surroundMixLevel	30
	dolbySurroundMode	30
	drcPresentationMode	30
	extAncDataEnable	30
	extDownmixLevelEnable	30
	extDownmixLevel_A	30
	extDownmixLevel_B	31
	dmxGainEnable	31
	dmxGain5	31
	dmxGain2	31
	lfeDmxEnable	31
	lfeDmxLevel	31
	ExtMetaData	31
6.5	AACENC_OutArgs Struct Reference	31
6.5.1	Detailed Description	31
6.5.2	Member Data Documentation	32

numOutBytes	32
numInSamples	32
numAncBytes	32
bitResState	32
7 File Documentation	33
7.1 aacenc.lib.h File Reference	33
7.1.1 Detailed Description	35
7.1.2 Typedef Documentation	35
HANDLE_AACENCODER	35
7.1.3 Enumeration Type Documentation	35
AACENC_ERROR	35
AACENC_BufferIdentifier	36
AACENC_METADATA_DRC_PROFILE	36
AACENC_CTRLFLAGS	37
AACENC_PARAM	37
7.1.4 Function Documentation	44
aacEncOpen()	44
aacEncClose()	45
aacEncEncode()	45
aacEncInfo()	46
aacEncoder_SetParam()	47
aacEncoder_GetParam()	47
aacEncGetLibInfo()	47
Index	51

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 in [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 uses file-based input/output.

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

```
HANDLE_AACENCODER hAacEncoder = NULL;
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;
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*encInfo.
                                     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) {
}
```

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

```
mpegFileWrite_Write(hMpegFile, outputBuffer, outargs.numOutBytes,
                   aacEncoder_GetParam(hAacEncoder, AACENC_GRANULE_LENGTH));
```

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, 0-3);
```

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 (dualrate sbr)
HE-AAC (AOT.SBR): 1.125 bits per sample (downsampled sbr)
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;
    case 7: chMode = MODE_6_1;         break;
    case 8: chMode = MODE_7_1_BACK;    break;
    default:
        chMode = MODE_INVALID;
}
return chMode;
```

2.9.3 Bitreservoir Configuration

In AAC, the default bitreservoir configuration depends on the chosen bitrate per frame and the number of effective channels. The size can be determined as below.

$$bitreservoir = nEffChannels * 6144 - (bitrate * framelength / samplerate)$$

Due to audio quality concerns it is not recommended to change the bitreservoir size to a lower value than the default setting! However, for minimizing the delay for streaming applications or for achieving a constant size of the bitstream packages in each frame, it may be necessary to change the bitreservoir size. This can be done with the [AACENC_PEAK_BITRATE](#) parameter.

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

By setting [AACENC_BITRATEMODE](#) to fixed framing, the bitreservoir is disabled. A disabled bitreservoir results in a constant size for each bitstream package. Please note that especially at lower bitrates a disabled bitreservoir can downgrade the audio quality considerably! The default bitreservoir configuration can be achieved as follows.

```
aacEncoder_SetParam(hAacEncoder, AACENC_BITRESERVOIR, -1);
```

To achieve acceptable audio quality with a reduced bitreservoir size setting at least 1000 bits per audio channel is recommended. For a multichannel audio file with 5.1 channels the bitreservoir reduced to 5000 bits results in acceptable audio quality.

2.9.4 Variable Bitrate Mode

The encoder provides various Variable Bitrate Modes that differ in audio quality and average overall bitrate. The given values are averages over time, different encoder settings and strongly depend on the type of audio signal. The VBR configurations can be adjusted via [AACENC_BITRATEMODE](#) encoder parameter.

VBR_MODE	Approx. Bitrate in kbps/channel	
	AAC-LC	AAC-LD/AC_ELD
-----+-----		
VBR_1	32 - 48	32 - 56

VBR_2		40 - 56		40 - 64
VBR_3		48 - 64		48 - 72
VBR_4		64 - 80		64 - 88
VBR_5		96 - 120		112 - 144

The bitrate ranges apply for individual audio channels. In case of multichannel configurations the average bitrate might be estimated by multiplying with the number of effective channels. This corresponds to all audio input channels exclusively the low frequency channel. At configurations which are making use of downmix modules the AAC core channels respectively downmix channels shall be considered. For [AACENC_AOT](#) which are using SBR, the average bitrate can be estimated by using the ratio of 0.5 for dualrate SBR and 0.75 for downsampled SBR configurations.

2.9.5 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, 0/1);
```

2.9.6 ELD Auto Configuration Mode

For ELD configuration a so called auto configurator is available which configures SBR and the SBR ratio by itself. The configurator is used when the encoder parameter [AACENC_SBR_MODE](#) and [AACENC_SBR_RATIO](#) are not set explicitly.

Based on sampling rate and chosen bitrate a reasonable SBR configuration will be used.

Sampling Rate [kHz]	Total Bitrate [bit/s]	No. of Chan	SBR	SBR Ratio

]min, 16[min - max	1	off	---

[16]	min - 27999	1	on	downsampled SBR
	28000 - max	1	off	---

]16 - 24]	min - 39999	1	on	downsampled SBR
	40000 - max	1	off	---

]24 - 32]	min - 27999	1	on	dualrate SBR
	28000 - 55999	1	on	downsampled SBR
	56000 - max	1	off	---

]32 - 44.1]	min - 63999	1	on	dualrate SBR
	64000 - max	1	off	---

]44.1 - 48]	min - 63999	1	on	dualrate SBR
	64000 - max	1	off	---

]min, 16[min - max	2	off	---

[16]	min - 31999	2	on	downsampled SBR
	32000 - 63999	2	on	downsampled SBR
	64000 - max	2	off	---

]16 - 24]	min - 47999	2	on	downsampled SBR
	48000 - 79999	2	on	downsampled SBR
	80000 - max	2	off	---

]24 - 32]	min - 31999	2	on	dualrate SBR
	32000 - 67999	2	on	dualrate SBR
	68000 - 95999	2	on	downsampled SBR
	96000 - max	2	off	---

]32 - 44.1]	min - 43999	2	on	dualrate SBR
	44000 - 127999	2	on	dualrate SBR
	128000 - max	2	off	---

]44.1 - 48]	min - 43999	2	on	dualrate SBR
	44000 - 127999	2	on	dualrate SBR
	128000 - max	2	off	---

2.9.7 Reduced Delay (Downscaled) Mode

The downscaled mode of AAC-ELD reduces the algorithmic delay of AAC-ELD by virtually increasing the sampling rate. When using the downscaled mode, the bitrate should be increased for keeping the same audio quality level. For common signals, the bitrate should be increased by 25% for a downscale factor of 2.

Currently, downscaling factors 2 and 4 are supported. To enable the downscaled mode in the encoder, the framelength parameter AACENC_GRANULE_LENGTH must be set accordingly to 256 or 240 for a downscale factor of 2 or 128 or 120 for a downscale factor of 4. The default values of 512 or 480 mean that no downscaling is applied.

```
aacEncoder_SetParam(hAACEncoder, AACENC_GRANULE_LENGTH, 256);
aacEncoder_SetParam(hAACEncoder, AACENC_GRANULE_LENGTH, 128);
```

Downscaled bitstreams are fully backwards compatible. However, the legacy decoder needs to support high sample rate, e.g. 96kHz. The signaled sampling rate is multiplied by the downscale factor. Although not required, downscaling should be applied when decoding downscaled bitstreams. It reduces CPU workload and the output will have the same sampling rate as the input. In an ideal configuration both encoder and decoder should run with the same downscale factor.

The following table shows approximate filter bank delays in ms for common sampling rates(sr) at framesize(fs), and downscale factor(dsf), based on this formula:

$$1000 * fs / (dsf * sr)$$

	512/2	512/4	480/2	480/4
22050	17.41	8.71	16.33	8.16
32000	12.00	6.00	11.25	5.62
44100	8.71	4.35	8.16	4.08
48000	8.00	4.00	7.50	3.75

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 and 11,12,14 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	Height	front_El	side_El	back_El	lfe_El
MODE_1	1	NORM	SCE			
MODE_2	2	NORM	CPE			
MODE_1_2	3	NORM	SCE, CPE			
MODE_1_2_1	4	NORM	SCE, CPE		SCE	
MODE_1_2_2	5	NORM	SCE, CPE		CPE	
MODE_1_2_2_1	6	NORM	SCE, CPE		CPE	LFE
MODE_1_2_2_2_1	7	NORM	SCE, CPE, CPE		CPE	LFE
MODE_6_1	11	NORM	SCE, CPE		CPE, SCE	LFE
MODE_7_1_BACK	12	NORM	SCE, CPE		CPE, CPE	LFE
MODE_7_1_TOP_FRONT	14	NORM	SCE, CPE		CPE	LFE
		TOP	CPE			
MODE_7_1_REAR_SURROUND	0	NORM	SCE, CPE		CPE, CPE	LFE
MODE_7_1_FRONT_CENTER	0	NORM	SCE, CPE, CPE		CPE	LFE

- NORM: Normal Height Layer. - TOP: Top Height Layer. - BTM: Bottom Height Layer.
- SCE: Single Channel Element. - CPE: Channel Pair. - LFE: Low Frequency Element.

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 in the normal height layer, possibly followed by front, side, and back elements in the top and bottom layer (Channel Configuration 14).

This mapping of Audio Channel Elements is defined in MPEG standard for Channel Config 1-7 and 11,12,14.

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	
MODE_1_2_2_2_1	0	1	2	3	4	5	6	7	2	6	7	0	1	4
MODE_6_1	0	1	2	3	4	5	6	2	0	1	4	5	6	3
MODE_7_1_BACK	0	1	2	3	4	5	6	7	2	0	1	6	7	4
MODE_7_1_TOP_FRONT	0	1	2	3	4	5	6	7	2	0	1	4	5	3
MODE_7_1_REAR_SURROUND	0	1	2	3	4	5	6	7	2	0	1	6	7	4
MODE_7_1_FRONT_CENTER	0	1	2	3	4	5	6	7	2	6	7	0	1	4

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 dualrate HE-AAC, 1.125 bits/audio sample for downsampled 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 in Dualrate SBR mode.

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	32.00	2
AAC LC + SBR + PS	18000 - 39999	32.00, 44.10, 48.00	44.10	2
AAC LC + SBR + PS	40000 - 64000	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	32.00	1
AAC LC + SBR	18000 - 39999	32.00, 44.10, 48.00	44.10	1
AAC LC + SBR	40000 - 64000	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

CHAPTER 2. LIBRARY USAGE

AAC LC + SBR	70000 - 239999	32.00, 44.10, 48.00	44.10	5, 5.1
AAC LC + SBR	240000 - 319999	32.00, 44.10, 48.00	48.00	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 in Dualrate SBR mode.

Unlike to HE-AAC configuration the SBR is not covered by ELD audio object type and needs to be enabled explicitly. Use [AACENC_SBR_MODE](#) to configure SBR and its samplingrate ratio with [AACENC_SBR_RATIO](#) parameter.

Audio Object Type	Bit Rate Range [bit/s]	Supported Sampling Rates [kHz]	Preferred Sampl. Rate [kHz]	No. of Chan.
ELD + SBR	18000 - 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	78000 - 160000	32.00 - 48.00	48.00	3
ELD + SBR	104000 - 212000	32.00 - 48.00	48.00	4
ELD + SBR	130000 - 246000	32.00 - 48.00	48.00	5, 5.1
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

2.12. RECOMMENDED SAMPLING RATE AND BITRATE COMBINATIONS LIBRARY USAGE

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, 5.1
LD, ELD	190000 - 244999	24.00 - 44.10	32.00	5, 5.1
LD, ELD	245000 - 339999	32.00 - 48.00	44.10	5, 5.1
LD, ELD	340000 - 960000	44.10 - 48.00	48.00	5, 5.1

2.12.3 AAC-ELD with SBR in Downsampled SBR mode.

Audio Object Type	Bit Rate Range [bit/s]	Supported Sampling Rates [kHz]	Preferred Sampl. Rate [kHz]	No. of Chan.
ELD + SBR (downsampled SBR)	18000 - 24999 25000 - 31999 32000 - 47999 48000 - 64000	16.00 - 22.05 16.00 - 24.00 22.05 - 32.00 22.05 - 48.00	22.05 24.00 32.00 32.00	1 1 1 1
ELD + SBR (downsampled SBR)	32000 - 51999 52000 - 59999 60000 - 95999 96000 - 128000	16.00 - 24.00 22.05 - 24.00 22.05 - 32.00 22.05 - 48.00	24.00 24.00 32.00 32.00	2 2 2 2
ELD + SBR (downsampled SBR)	78000 - 99999 100000 - 143999 144000 - 159999 160000 - 192000	22.05 - 24.00 22.05 - 32.00 22.05 - 48.00 32.00 - 48.00	24.00 32.00 32.00 32.00	3 3 3 3
ELD + SBR (downsampled SBR)	104000 - 149999 150000 - 191999 192000 - 211999 212000 - 256000	22.05 - 24.00 22.05 - 32.00 22.05 - 48.00 32.00 - 48.00	24.00 32.00 32.00 32.00	4 4 4 4
ELD + SBR (downsampled SBR)	130000 - 171999 172000 - 239999 240000 - 320000	22.05 - 24.00 22.05 - 32.00 32.00 - 48.00	24.00 32.00 32.00	5, 5.1 5, 5.1 5, 5.1

2.12.4 AAC-ELD v2, AAC-ELD v2 with SBR.

The ELD v2 212 configuration must be configured explicitly with [AACENC_CHANNELMODE](#) parameter according MODE_212 value. SBR can be configured separately through [AACENC_SBR_MODE](#) and [AACENC_SBR_RATIO](#) parameter. Following configurations shall apply to both framelengths 480 and 512. For ELD v2 configuration without SBR and framelength 480 the supported sampling rate is restricted to the range from 16 kHz up to 24 kHz.

Audio Object Type	Bit Rate Range [bit/s]	Supported Sampling Rates [kHz]	Preferred Sampl. Rate	No. of Chan.
-------------------	---------------------------	--------------------------------------	-----------------------------	-----------------

CHAPTER 2. LIBRARY USAGE

			[kHz]	
ELD-212	16000 - 19999	16.00 - 24.00	16.00	2
(without SBR)	20000 - 39999	16.00 - 32.00	24.00	2
	40000 - 49999	22.05 - 32.00	32.00	2
	50000 - 61999	24.00 - 44.10	32.00	2
	62000 - 84999	32.00 - 48.00	44.10	2
	85000 - 192000	44.10 - 48.00	48.00	2
ELD-212 + SBR	18000 - 20999	32.00	32.00	2
(dualrate SBR)	21000 - 25999	32.00 - 44.10	32.00	2
	26000 - 31999	32.00 - 48.00	44.10	2
	32000 - 64000	32.00 - 48.00	48.00	2
ELD-212 + SBR	18000 - 19999	16.00 - 22.05	22.05	2
(downsampled SBR)	20000 - 24999	16.00 - 24.00	22.05	2
	25000 - 31999	16.00 - 24.00	24.00	2
	32000 - 64000	24.00 - 24.00	24.00	2

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 "bandWidthTable" in the file bandwidth.cpp (if available).

The encoder API provides the `AACENC_BANDWIDTH` parameter to adjust the bandwidth explicitly.

```
aacEncoder_SetParam(hAACEncoder, AACENC_BANDWIDTH, value);
```

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. Also, the maximum bandwidth that can be set manually by the user is 20kHz or $fs/2$, whichever value is smaller.

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. ≤ 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 double-edged 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. Therefore, for AAC-ELD, the bitreservoir is limited. It varies between 500 and 4000 bits/frame, depending on the bitrate/channel.

- For a bitrate of 12kbps/channel and below, the AAC-ELD bitreservoir is 500 bits/frame.
- For a bitrate of 70kbps/channel and above, the AAC-ELD bitreservoir is 4000 bits/frame.
- Between 12kbps/channel and 70kbps/channel, the AAC-ELD bitreservoir is increased linearly.
- For AAC-LC, the bitrate is only limited by the maximum AAC frame length. It is, regardless of the available bit reservoir, defined as 6144 bits per channel.

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.

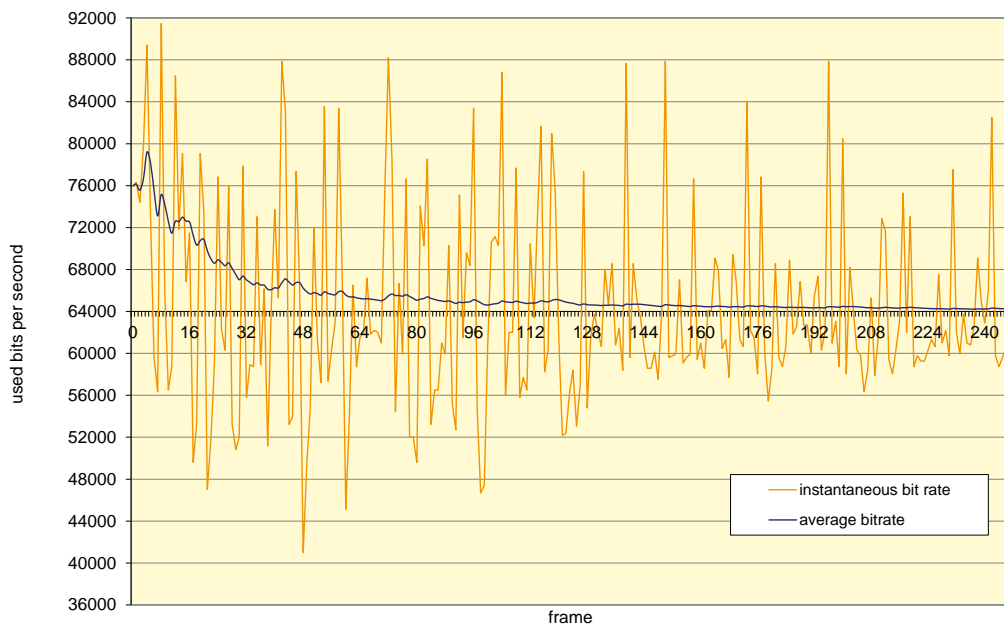


Figure 3.1: Average bitrate over time

The figure shows by means of a stereo file encoded with a constant bitrate of 64kbps, that the average bitrate converges quickly to the set bitrate. After 64 frames, the average bitrate already is 65.3 kbps. After 230 frames, the average bitrate is below 64.3 kbps.

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.

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

Class Index

4.1 Class List

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

AACENC_BufDesc	25
AACENC_InArgs	26
AACENC_InfoStruct	26
AACENC_MetaData	28
AACENC_OutArgs	31

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

[aacenc.lib.h](#)

FDK AAC Encoder library interface header file 33

Chapter 6

Class Documentation

6.1 AACENC_BufDesc Struct Reference

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

Public Attributes

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

6.1.1 Detailed Description

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

6.1.2 Member Data Documentation

numBufs

```
INT AACENC_BufDesc::numBufs
```

Number of buffers.

bufs

```
void** AACENC_BufDesc::bufs
```

Pointer to vector containing buffer addresses.

bufferIdentifiers

```
INT* AACENC_BufDesc::bufferIdentifiers
```

Identifier of each buffer element. See [AACENC_BufferIdentifier](#).

bufSizes

```
INT* AACENC_BufDesc::bufSizes
```

Size of each buffer in 8-bit bytes.

bufElSizes

```
INT* AACENC_BufDesc::bufElSizes
```

Size of each buffer element in bytes.

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

- [aacenc.lib.h](#)

6.2 AACENC_InArgs Struct Reference

```
#include <aacenc.lib.h>
```

Public Attributes

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

6.2.1 Detailed Description

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

6.2.2 Member Data Documentation

numInSamples

```
INT AACENC_InArgs::numInSamples
```

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

numAncBytes

```
INT AACENC_InArgs::numAncBytes
```

Number of ancillary data bytes to be encoded.

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

- [aacenc.lib.h](#)

6.3 AACENC_InfoStruct Struct Reference

```
#include <aacenc.lib.h>
```


Public Attributes

- `UINT maxOutBufBytes`
- `UINT maxAncBytes`
- `UINT inBufFillLevel`
- `UINT inputChannels`
- `UINT frameLength`
- `UINT nDelay`
- `UINT nDelayCore`
- `UCHAR confBuf [64]`
- `UINT confSize`

6.3.1 Detailed Description

Provides some info about the encoder configuration.

6.3.2 Member Data Documentation

maxOutBufBytes

`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.

maxAncBytes

`UINT AACENC_InfoStruct::maxAncBytes`

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

inBufFillLevel

`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.

inputChannels

`UINT AACENC_InfoStruct::inputChannels`

Number of input channels expected in encoding process.

frameLength

`UINT AACENC_InfoStruct::frameLength`

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

nDelay

```
UINT AACENC_InfoStruct::nDelay
```

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

nDelayCore

```
UINT AACENC_InfoStruct::nDelayCore
```

Codec delay in PCM samples/channel, w/o delay caused by the decoder SBR module. This delay is needed to correctly write edit lists for gapless playback. The decoder may not know how much delay is introduced by SBR, since it may not know if SBR is active at all (implicit signaling), therefore the decoder must take into account any delay caused by the SBR module.

confBuf

```
UCHAR AACENC_InfoStruct::confBuf[64]
```

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

confSize

```
UINT AACENC_InfoStruct::confSize
```

Number of valid bytes in confBuf.

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

- [aacenc.lib.h](#)

6.4 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](#)
- UCHAR [drcPresentationMode](#)
- struct {
 - UCHAR [extAncDataEnable](#)
 - UCHAR [extDownmixLevelEnable](#)
 - UCHAR [extDownmixLevel_A](#)
 - UCHAR [extDownmixLevel_B](#)
 - UCHAR [dmxGainEnable](#)

```

    INT dmxDmxGain5
    INT dmxDmxGain2
    UCHAR lfeDmxEnable
    UCHAR lfeDmxLevel
} ExtMetaData

```

6.4.1 Detailed Description

Meta Data setup structure.

6.4.2 Member Data Documentation

drc_profile

[AACENC_METADATA_DRC_PROFILE](#) AACENC_MetaData::drc_profile
 MPEG DRC compression profile. See [AACENC_METADATA_DRC_PROFILE](#).

comp_profile

[AACENC_METADATA_DRC_PROFILE](#) AACENC_MetaData::comp_profile
 ETSI heavy compression profile. See [AACENC_METADATA_DRC_PROFILE](#).

drc.TargetRefLevel

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

comp.TargetRefLevel

INT AACENC_MetaData::comp.TargetRefLevel
 Adjust limiter to avoid overload. Scaled with 16 bit. $x \cdot 2^{16}$.

prog_ref_level_present

INT AACENC_MetaData::prog_ref_level_present
 Flag, if prog_ref_level is present

prog_ref_level

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

PCE_mixdown_idx_present

UCHAR AACENC_MetaData::PCE_mixdown_idx_present
 Flag, if dmx-idx should be written in programme config element

ETSI_DmxLvl_present

UCHAR AACENC.Metadata::ETSI_DmxLvl_present
Flag, if dmx-lvl should be written in ETSI-ancData

centerMixLevel

SCHAR AACENC.Metadata::centerMixLevel
Center downmix level (0...7, according to table)

surroundMixLevel

SCHAR AACENC.Metadata::surroundMixLevel
Surround downmix level (0...7, according to table)

dolbySurroundMode

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

drcPresentationMode

UCHAR AACENC.Metadata::drcPresentationMode
Indicatin for DRC Presentation Mode.

- 0: Presentation mode not inticated
- 1: Presentation mode 1
- 2: Presentation mode 2

extAncDataEnable

UCHAR AACENC.Metadata::extAncDataEnable

extDownmixLevelEnable

UCHAR AACENC.Metadata::extDownmixLevelEnable

extDownmixLevel_A

UCHAR AACENC.Metadata::extDownmixLevel_A

extDownmixLevel_B

UCHAR AACENC_MetaData::extDownmixLevel_B

dmxGainEnable

UCHAR AACENC_MetaData::dmxGainEnable

dmxGain5

INT AACENC_MetaData::dmxGain5

dmxGain2

INT AACENC_MetaData::dmxGain2

lfeDmxEnable

UCHAR AACENC_MetaData::lfeDmxEnable

lfeDmxLevel

UCHAR AACENC_MetaData::lfeDmxLevel

ExtMetaData

struct { ... } AACENC_MetaData::ExtMetaData

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

- [aacenc.lib.h](#)

6.5 AACENC_OutArgs Struct Reference

```
#include <aacenc.lib.h>
```

Public Attributes

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

6.5.1 Detailed Description

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

6.5.2 Member Data Documentation

numOutBytes

INT AACENC_OutArgs::numOutBytes

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

numInSamples

INT AACENC_OutArgs::numInSamples

Number of input audio samples consumed by the encoder.

numAncBytes

INT AACENC_OutArgs::numAncBytes

Number of ancillary data bytes consumed by the encoder.

bitResState

INT AACENC_OutArgs::bitResState

State of the bit reservoir in bits.

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

- [aacenc.lib.h](#)

Chapter 7

File Documentation

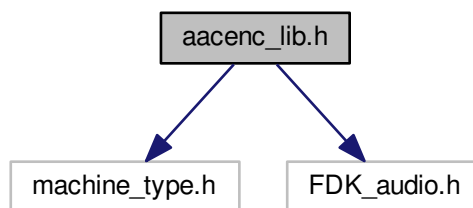
7.1 aacenc_lib.h File Reference

FDK AAC Encoder library interface header file.

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

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

Include dependency graph for aacenc_lib.h:



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_INIT_MPS_ERROR = 0x0045,
AACENC_ENCODE_ERROR = 0x0060,
AACENC_ENCODE_EOF = 0x0080 }
• enum AACENC_BufferIdentifier {
    IN_AUDIO_DATA = 0,
    IN Ancillary_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,
    AACENC_METADATA_DRC_NOT_PRESENT = 256 }
• 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_SBR_RATIO = 0x0108,
    AACENC_AFTERBURNER = 0x0200,
    AACENC_BANDWIDTH = 0x0203,
    AACENC_PEAK_BITRATE = 0x0207,
    AACENC_TRANSMUX = 0x0300,
    AACENC_HEADER_PERIOD = 0x0301,
    AACENC_SIGNALING_MODE = 0x0302,
    AACENC_TPSUBFRAMES = 0x0303,
    AACENC_AUDIOMUXVER = 0x0304,
    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.

7.1.1 Detailed Description

FDK AAC Encoder library interface header file.

7.1.2 Typedef Documentation

HANDLE_AACENCODER

```
typedef struct AACENCODER* HANDLE_AACENCODER
AAC encoder handle.
```

7.1.3 Enumeration Type Documentation

AACENC_ERROR

```
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_INIT_MPS_ERROR	MPS library initialization error.
AACENC_ENCODE_ERROR	The encoding process was interrupted by an unexpected error.
AACENC_ENCODE_EOF	End of file reached.

AACENC_BufferIdentifier

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_ANCILLRY_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 superframing.

AACENC_METADATA_DRC_PROFILE

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.

Enumerator

AACENC_METADATA_DRC_MUSICSTANDARD	Music standard.
AACENC_METADATA_DRC_MUSICLIGHT	Music light.
AACENC_METADATA_DRC_SPEECH	Speech.
AACENC_METADATA_DRC_NOT_PRESENT	Disable writing gain factor (used for comp_profile only).

AACENC_CTRLFLAGS

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.

AACENC_PARAM

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	<p>Audio object type. See <code>::AUDIO_OBJECT_TYPE</code> in <code>FDK_audio.h</code>.</p> <ul style="list-style-type: none"> • 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 <code>::AUDIO_OBJECT_TYPE</code> for ELD in combination with SBR defined, enable SBR explicitly by AACENC_SBR_MODE parameter. The ELD v2 212 configuration can be configured by AACENC_CHANNELMODE parameter. • 129: MPEG-2 AAC Low Complexity. • 132: MPEG-2 AAC Low Complexity with Spectral Band Replication (HE-AAC). <p>Please note that the virtual MPEG-2 AOT's basically disables non-existing Perceptual Noise Substitution tool in AAC encoder and controls the <code>MPEG_ID</code> flag in <code>adts</code> header. The virtual MPEG-2 AOT doesn't prohibit specific transport formats.</p>
AACENC_BITRATE	<p>Total encoder bitrate. This parameter is mandatory and interacts with AACENC_BITRATEMODE.</p> <ul style="list-style-type: none"> • CBR: Bitrate in bits/second. • VBR: Variable bitrate. Bitrate argument will be ignored. See Supported Bitrates for details.

Enumerator

AACENC_BITRATEMODE	<p>Bitrate mode. Configuration can be different kind of bitrate configurations:</p> <ul style="list-style-type: none"> • 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. • 1: Variable bitrate mode, very low bitrate. • 2: Variable bitrate mode, low bitrate. • 3: Variable bitrate mode, medium bitrate. • 4: Variable bitrate mode, high bitrate. • 5: Variable bitrate mode, very high bitrate.
AACENC_SAMPLERATE	<p>Audio input data sampling rate. Encoder supports following sampling rates: 8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100, 48000, 64000, 88200, 96000</p>
AACENC_SBR_MODE	<p>Configure SBR independently of the chosen Audio Object Type ::AUDIO_OBJECT_TYPE. This parameter is for ELD audio object type only.</p> <ul style="list-style-type: none"> • -1: Use ELD SBR auto configurator (default). • 0: Disable Spectral Band Replication. • 1: Enable Spectral Band Replication.
AACENC_GRANULE_LENGTH	<p>Core encoder (AAC) audio frame length in samples:</p> <ul style="list-style-type: none"> • 1024: Default configuration. • 512: Default length in LD/ELD configuration. • 480: Length in LD/ELD configuration. • 256: Length for ELD reduced delay mode (x2). • 240: Length for ELD reduced delay mode (x2). • 128: Length for ELD reduced delay mode (x4). • 120: Length for ELD reduced delay mode (x4).
AACENC_CHANNELMODE	<p>Set explicit channel mode. Channel mode must match with number of input channels.</p> <ul style="list-style-type: none"> • 1-7, 11,12,14 and 33,34: MPEG channel modes supported, see ::CHANNEL_MODE in FDK_audio.h.

Enumerator

AACENC_CHANNELORDER	<p>Input audio data channel ordering scheme:</p> <ul style="list-style-type: none"> • 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_SBR_RATIO	<p>Controls activation of downsampled SBR. With downsampled SBR, the delay will be shorter. On the other hand, for achieving the same quality level, downsampled SBR needs more bits than dual-rate SBR. With downsampled SBR, the AAC encoder will work at the same sampling rate as the SBR encoder (single rate). Downsampled SBR is supported for AAC-ELD and HE-AACv1.</p> <ul style="list-style-type: none"> • 1: Downsampled SBR (default for ELD). • 2: Dual-rate SBR (default for HE-AAC).
AACENC_AFTERBURNER	<p>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.</p> <ul style="list-style-type: none"> • 0: Disable afterburner (default). • 1: Enable afterburner.
AACENC_BANDWIDTH	<p>Core encoder audio bandwidth:</p> <ul style="list-style-type: none"> • 0: Determine audio bandwidth internally (default, see chapter Bandwidth). • 1 to fs/2: Audio bandwidth in Hertz. Limited to 20kHz max. Not usable if SBR is active. This setting is for experts only, better do not touch this value to avoid degraded audio quality.

Enumerator

AACENC_PEAK_BITRATE	<p>Peak bitrate configuration parameter to adjust maximum bits per audio frame. Bitrate is in bits/second. The peak bitrate will internally be limited to the chosen bitrate AACENC_BITRATE as lower limit and the $\text{number_of_effective_channels} \times 6144$ bit as upper limit. Setting the peak bitrate equal to AACENC_BITRATE does not necessarily mean that the audio frames will be of constant size. Since the peak bitrate is in bits/second, the frame sizes can vary by one byte in one or the other direction over various frames. However, it is not recommended to reduce the peak bitrate to AACENC_BITRATE - it would disable the bitreservoir, which would affect the audio quality by a large amount.</p>
AACENC_TRANSMUX	<p>Transport type to be used. See <code>::TRANSPORT_TYPE</code> in <code>FDK_audio.h</code>. Following types can be configured in encoder library:</p> <ul style="list-style-type: none"> • 0: raw access units • 1: ADIF bitstream format • 2: ADTS bitstream format • 6: Audio Mux Elements (LATM) with <code>muxConfigPresent = 1</code> • 7: Audio Mux Elements (LATM) with <code>muxConfigPresent = 0</code>, out of band <code>StreamMuxConfig</code> • 10: Audio Sync Stream (LOAS)
AACENC_HEADER_PERIOD	<p>Frame count period for sending in-band configuration buffers within LATM/LOAS transport layer. Additionally this parameter configures the PCE repetition period in <code>raw_data_block()</code>. See <code>encPCE</code>.</p> <ul style="list-style-type: none"> • 0xFF: auto-mode default 10 for <code>TT_MP4_ADTS</code>, <code>TT_MP4_LOAS</code> and <code>TT_MP4_LATM_MCP1</code>, otherwise 0. • n: Frame count period.

Enumerator

AACENC_SIGNALING_MODE	<p>Signaling mode of the extension AOT:</p> <ul style="list-style-type: none"> • 0: Implicit backward compatible signaling (default for non-MPEG-4 based AOT's and for the transport formats ADIF and ADTS) <ul style="list-style-type: none"> – A stream that uses implicit signaling can be decoded by every AAC decoder, even AAC-LC-only decoders – An AAC-LC-only decoder will only decode the low-frequency part of the stream, resulting in a band-limited output – This method works with all transport formats – This method does not work with downsampled SBR • 1: Explicit backward compatible signaling <ul style="list-style-type: none"> – A stream that uses explicit backward compatible signaling can be decoded by every AAC decoder, even AAC-LC-only decoders – An AAC-LC-only decoder will only decode the low-frequency part of the stream, resulting in a band-limited output – A decoder not capable of decoding PS will only decode the AAC-LC+SBR part. If the stream contained PS, the result will be a a decoded mono downmix – This method does not work with ADIF or ADTS. For LOAS/LATM, it only works with AudioMuxVersion==1 – This method does work with downsampled SBR • 2: Explicit hierarchical signaling (default for MPEG-4 based AOT's and for all transport formats excluding ADIF and ADTS) <ul style="list-style-type: none"> – A stream that uses explicit hierarchical signaling can be decoded only by HE-AAC decoders – An AAC-LC-only decoder will not decode a stream that uses explicit hierarchical signaling – A decoder not capable of decoding PS will not decode the stream at all if it contained PS – This method does not work with ADIF or ADTS. It works with LOAS/LATM and the MPEG-4 File format – This method does work with downsampled SBR <p>For making sure that the listener always</p>
	<p>experiences the best audio quality, explicit hierarchical signaling should be used. This makes sure that only a full HE-AAC-capable decoder will decode those streams. The audio is played at full bandwidth. For best backwards compatibility, it is recommended to encode with implicit SBR signaling. A decoder capable of AAC-LC only will</p>

Enumerator

AACENC_TPSUBFRAMES	<p>Number of sub frames in a transport frame for LOAS/LATM or ADTS (default 1).</p> <ul style="list-style-type: none"> • ADTS: Maximum number of sub frames restricted to 4. • LOAS/LATM: Maximum number of sub frames restricted to 2.
AACENC_AUDIOMUXVER	<p>AudioMuxVersion to be used for LATM. (AudioMuxVersionA, currently not implemented):</p> <ul style="list-style-type: none"> • 0: Default, no transmission of tara Buffer fullness, no ASC length and including actual latm Buffer fullness. • 1: Transmission of tara Buffer fullness, ASC length and actual latm Buffer fullness. • 2: Transmission of tara Buffer fullness, ASC length and maximum level of latm Buffer fullness.
AACENC_PROTECTION	<p>Configure protection in transport layer:</p> <ul style="list-style-type: none"> • 0: No protection. (default) • 1: CRC active for ADTS transport format.
AACENC_ANCILLARY_BITRATE	<p>Constant ancillary data bitrate in bits/second.</p> <ul style="list-style-type: none"> • 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	<p>Configure Meta Data. See AACENC_MetaData for further details:</p> <ul style="list-style-type: none"> • 0: Do not embed any metadata. • 1: Embed dynamic_range_info metadata. • 2: Embed dynamic_range_info and ancillary_data metadata. • 3: Embed ancillary_data metadata.
AACENC_CONTROL_STATE	<p>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.</p>
AACENC_NONE	

7.1.4 Function Documentation

aacEncOpen()

```
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

<i>phAacEncoder</i>	A pointer to an encoder handle. Initialized on return.
<i>encModules</i>	Specify encoder modules to be supported in this encoder instance: <ul style="list-style-type: none"> • 0x0: Allocate memory for all available encoder modules. • else: Select memory allocation regarding encoder modules. Following flags are possible and can be combined. <ul style="list-style-type: none"> – 0x01: AAC module. – 0x02: SBR module. – 0x04: PS module. – 0x08: MPS module. – 0x10: Metadata module. – example: (0x01 0x02 0x04 0x08 0x10) allocates all modules and is equivalent to default configuration denoted by 0x0.
<i>maxChannels</i>	Number of channels to be allocated. This parameter can be used in different ways: <ul style="list-style-type: none"> • 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.

aacEncClose()

```
AACENC_ERROR aacEncClose (
    HANDLE_AACENCODER * phAacEncoder )
```

Close the encoder instance.

Deallocate encoder instance and free whole memory.

Parameters

<i>phAacEncoder</i>	Pointer to the encoder handle to be deallocated.
---------------------	--

Returns

- AACENC_OK, on success.
- AACENC_INVALID_HANDLE, on failure.

aacEncEncode()

```
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

<i>hAacEncoder</i>	A valid AAC encoder handle.
--------------------	-----------------------------

Parameters

<i>inBufDesc</i>	Input buffer descriptor, see AACENC_BufDesc : <ul style="list-style-type: none"> • At least one input buffer with audio data is expected. • Optionally a second input buffer with ancillary data can be fed.
<i>outBufDesc</i>	Output buffer descriptor, see AACENC_BufDesc : <ul style="list-style-type: none"> • Provide one output buffer for the encoded bitstream.
<i>inargs</i>	Input arguments, see AACENC_InArgs .
<i>outargs</i>	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, AACENC_INIT_MPS_ERROR, on failure in encoder initialization.
- AACENC_UNSUPPORTED_PARAMETER, on incorrect input or output buffer descriptor initialization.
- AACENC_ENCODE_EOF, when flushing fully concluded.

aacEncInfo()

```
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 StreamMux-Config 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

<i>hAacEncoder</i>	A valid AAC encoder handle.
<i>pInfo</i>	Pointer to AACENC_InfoStruct . Filled on return.

Returns

- AACENC_OK, on succes.
- AACENC_INIT_ERROR, on failure.

aacEncoder_SetParam()

```
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

<i>hAacEncoder</i>	A valid AAC encoder handle.
<i>param</i>	Parameter to be set. See AACENC_PARAM .
<i>value</i>	Parameter value. See parameter description in AACENC_PARAM .

Returns

- AACENC_OK, on success.
- AACENC_INVALID_HANDLE, AACENC_UNSUPPORTED_PARAMETER, AACENC_INVALID_CONFIG, on failure.

aacEncoder_GetParam()

```
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

<i>hAacEncoder</i>	A valid AAC encoder handle.
<i>param</i>	Parameter to be returned. See AACENC_PARAM .

Returns

Internal configuration value of specified parameter [AACENC_PARAM](#).

aacEncGetLibInfo()

```
AACENC_ERROR aacEncGetLibInfo (
    LIB_INFO * info )
```

Get information about encoder library build.

Fill a given LIB_INFO structure with library version information.

Parameters

<i>info</i>	Pointer to an allocated LIB_INFO struct.
-------------	--

Returns

- AACENC_OK, on success.
- AACENC_INVALID_HANDLE, AACENC_INIT_ERROR, on failure.

Index

AACENC_BufDesc, 25
 bufElSizes, 26
 bufSizes, 25
 bufferIdentifiers, 25
 bufs, 25
 numBufs, 25
AACENC_BufferIdentifier
 aacenc.lib.h, 36
AACENC_CTRLFLAGS
 aacenc.lib.h, 37
AACENC_ERROR
 aacenc.lib.h, 35
AACENC_InArgs, 26
 numAncBytes, 26
 numInSamples, 26
AACENC_InfoStruct, 26
 confBuf, 28
 confSize, 28
 frameLength, 27
 inBufFillLevel, 27
 inputChannels, 27
 maxAncBytes, 27
 maxOutBufBytes, 27
 nDelay, 27
 nDelayCore, 28
AACENC_METADATA_DRC_PROFILE
 aacenc.lib.h, 36
AACENC_MetaData, 28
 centerMixLevel, 30
 comp_TargetRefLevel, 29
 comp_profile, 29
 dmxGain2, 31
 dmxGain5, 31
 dmxGainEnable, 31
 dolbySurroundMode, 30
 drc_TargetRefLevel, 29
 drc_profile, 29
 drcPresentationMode, 30
 ETSI_DmxLvl_present, 29
 extAncDataEnable, 30
 extDownmixLevel_A, 30
 extDownmixLevel_B, 30
 extDownmixLevelEnable, 30
 ExtMetaData, 31
 lfeDmxEnable, 31
 lfeDmxLevel, 31
 PCE_mixdown_idx_present, 29
 prog_ref_level, 29
 prog_ref_level_present, 29
 surroundMixLevel, 30
AACENC_OutArgs, 31
 bitResState, 32
 numAncBytes, 32
 numInSamples, 32
 numOutBytes, 32
AACENC_PARAM
 aacenc.lib.h, 37
aacEncClose
 aacenc.lib.h, 44
aacEncEncode
 aacenc.lib.h, 45
aacEncGetLibInfo
 aacenc.lib.h, 47
aacEncInfo
 aacenc.lib.h, 46
aacEncOpen
 aacenc.lib.h, 44
aacEncoder_GetParam
 aacenc.lib.h, 47
aacEncoder_SetParam
 aacenc.lib.h, 46
aacenc.lib.h, 33
 AACENC_BufferIdentifier, 36
 AACENC_CTRLFLAGS, 37
 AACENC_ERROR, 35
 AACENC_METADATA_DRC_PROFILE, 36
 AACENC_PARAM, 37
 aacEncClose, 44
 aacEncEncode, 45
 aacEncGetLibInfo, 47
 aacEncInfo, 46
 aacEncOpen, 44
 aacEncoder_GetParam, 47
 aacEncoder_SetParam, 46
 HANDLE_AACENCODER, 35
AACENC_AFTERBURNER
 aacenc.lib.h, 40
AACENC_ANCILLARY_BITRATE

- aacenc.lib.h, 43
- AACENC_AOT
 - aacenc.lib.h, 38
- AACENC_AUDIOMUXVER
 - aacenc.lib.h, 43
- AACENC_BANDWIDTH
 - aacenc.lib.h, 40
- AACENC_BITRATE
 - aacenc.lib.h, 38
- AACENC_BITRATEMODE
 - aacenc.lib.h, 39
- AACENC_CHANNELMODE
 - aacenc.lib.h, 39
- AACENC_CHANNELORDER
 - aacenc.lib.h, 40
- AACENC_CONTROL_STATE
 - aacenc.lib.h, 43
- AACENC_ENCODE_EOF
 - aacenc.lib.h, 36
- AACENC_ENCODE_ERROR
 - aacenc.lib.h, 36
- AACENC_GRANULE_LENGTH
 - aacenc.lib.h, 39
- AACENC_HEADER_PERIOD
 - aacenc.lib.h, 41
- AACENC_INIT_AAC_ERROR
 - aacenc.lib.h, 36
- AACENC_INIT_ALL
 - aacenc.lib.h, 37
- AACENC_INIT_CONFIG
 - aacenc.lib.h, 37
- AACENC_INIT_ERROR
 - aacenc.lib.h, 36
- AACENC_INIT_META_ERROR
 - aacenc.lib.h, 36
- AACENC_INIT_MPS_ERROR
 - aacenc.lib.h, 36
- AACENC_INIT_NONE
 - aacenc.lib.h, 37
- AACENC_INIT_SBR_ERROR
 - aacenc.lib.h, 36
- AACENC_INIT_STATES
 - aacenc.lib.h, 37
- AACENC_INIT_TP_ERROR
 - aacenc.lib.h, 36
- AACENC_INIT_TRANSPORT
 - aacenc.lib.h, 37
- AACENC_INVALID_CONFIG
 - aacenc.lib.h, 36
- AACENC_INVALID_HANDLE
 - aacenc.lib.h, 36
- aacenc.lib.h
 - AACENC_AFTERBURNER, 40
 - AACENC Ancillary_Bitrate, 43
 - AACENC_AOT, 38
 - AACENC_AUDIOMUXVER, 43
 - AACENC_BANDWIDTH, 40
 - AACENC_BITRATE, 38
 - AACENC_BITRATEMODE, 39
 - AACENC_CHANNELMODE, 39
 - AACENC_CHANNELORDER, 40
 - AACENC_CONTROL_STATE, 43
 - AACENC_ENCODE_EOF, 36
 - AACENC_ENCODE_ERROR, 36
 - AACENC_GRANULE_LENGTH, 39
 - AACENC_HEADER_PERIOD, 41
 - AACENC_INIT_AAC_ERROR, 36
 - AACENC_INIT_ALL, 37
 - AACENC_INIT_CONFIG, 37
 - AACENC_INIT_ERROR, 36
 - AACENC_INIT_META_ERROR, 36
 - AACENC_INIT_MPS_ERROR, 36
 - AACENC_INIT_NONE, 37
 - AACENC_INIT_SBR_ERROR, 36
 - AACENC_INIT_STATES, 37
 - AACENC_INIT_TP_ERROR, 36
 - AACENC_INIT_TRANSPORT, 37
 - AACENC_INVALID_CONFIG, 36
 - AACENC_INVALID_HANDLE, 36
 - AACENC_MEMORY_ERROR, 36
 - AACENC_METADATA_DRC_FILMLIGHT, 36
 - AACENC_METADATA_DRC_FILMSTANDARD, 36
 - AACENC_METADATA_DRC_MUSICLIGHT, 37
 - AACENC_METADATA_DRC_MUSICSTANDARD, 37
 - AACENC_METADATA_DRC_NONE, 36
 - AACENC_METADATA_DRC_NOT_PRESENT, 37
 - AACENC_METADATA_DRC_SPEECH, 37
 - AACENC_METADATA_MODE, 43
 - AACENC_NONE, 43
 - AACENC_OK, 36
 - AACENC_PEAK_BITRATE, 41
 - AACENC_PROTECTION, 43
 - AACENC_RESET_INBUFFER, 37
 - AACENC_SAMPLERATE, 39
 - AACENC_SBR_MODE, 39
 - AACENC_SBR_RATIO, 40
 - AACENC_SIGNALING_MODE, 42
 - AACENC_TPSUBFRAMES, 43
 - AACENC_TRANSMUX, 41
 - AACENC_UNSUPPORTED_PARAMETER, 36
 - IN_Ancillary_Data, 36
 - IN_AUDIO_DATA, 36
 - IN_METADATA_SETUP, 36

- OUT_AU_SIZES, 36
- OUT_BITSTREAM_DATA, 36
- AACENC_MEMORY_ERROR
 - aacenc.lib.h, 36
- AACENC_METADATA_DRC_FILMLIGHT
 - aacenc.lib.h, 36
- AACENC_METADATA_DRC_FILMSTANDARD
 - aacenc.lib.h, 36
- AACENC_METADATA_DRC_MUSICLIGHT
 - aacenc.lib.h, 37
- AACENC_METADATA_DRC_MUSICSTANDARD
 - aacenc.lib.h, 37
- AACENC_METADATA_DRC_NONE
 - aacenc.lib.h, 36
- AACENC_METADATA_DRC_NOT_PRESENT
 - aacenc.lib.h, 37
- AACENC_METADATA_DRC_SPEECH
 - aacenc.lib.h, 37
- AACENC_METADATA_MODE
 - aacenc.lib.h, 43
- AACENC_NONE
 - aacenc.lib.h, 43
- AACENC_OK
 - aacenc.lib.h, 36
- AACENC_PEAK_BITRATE
 - aacenc.lib.h, 41
- AACENC_PROTECTION
 - aacenc.lib.h, 43
- AACENC_RESET_INBUFFER
 - aacenc.lib.h, 37
- AACENC_SAMPLERATE
 - aacenc.lib.h, 39
- AACENC_SBR_MODE
 - aacenc.lib.h, 39
- AACENC_SBR_RATIO
 - aacenc.lib.h, 40
- AACENC_SIGNALING_MODE
 - aacenc.lib.h, 42
- AACENC_TPSUBFRAMES
 - aacenc.lib.h, 43
- AACENC_TRANSMUX
 - aacenc.lib.h, 41
- AACENC_UNSUPPORTED_PARAMETER
 - aacenc.lib.h, 36
- bitResState
 - AACENC_OutArgs, 32
- bufElSizes
 - AACENC_BufDesc, 26
- bufSizes
 - AACENC_BufDesc, 25
- bufferIdentifiers
 - AACENC_BufDesc, 25
- bufs
- AACENC_BufDesc, 25
- centerMixLevel
 - AACENC_MetaData, 30
- comp_TargetRefLevel
 - AACENC_MetaData, 29
- comp_profile
 - AACENC_MetaData, 29
- confBuf
 - AACENC_InfoStruct, 28
- confSize
 - AACENC_InfoStruct, 28
- dmxGain2
 - AACENC_MetaData, 31
- dmxGain5
 - AACENC_MetaData, 31
- dmxGainEnable
 - AACENC_MetaData, 31
- dolbySurroundMode
 - AACENC_MetaData, 30
- drc_TargetRefLevel
 - AACENC_MetaData, 29
- drc_profile
 - AACENC_MetaData, 29
- drcPresentationMode
 - AACENC_MetaData, 30
- ETSI_DmxLvl_present
 - AACENC_MetaData, 29
- extAncDataEnable
 - AACENC_MetaData, 30
- extDownmixLevel_A
 - AACENC_MetaData, 30
- extDownmixLevel_B
 - AACENC_MetaData, 30
- extDownmixLevelEnable
 - AACENC_MetaData, 30
- ExtMetaData
 - AACENC_MetaData, 31
- frameLength
 - AACENC_InfoStruct, 27
- HANDLE_AACENCODER
 - aacenc.lib.h, 35
- inBufFillLevel
 - AACENC_InfoStruct, 27
- IN_ANCILLRY_DATA
 - aacenc.lib.h, 36
- IN_AUDIO_DATA
 - aacenc.lib.h, 36
- IN_METADATA_SETUP
 - aacenc.lib.h, 36

inputChannels
 AACENC_InfoStruct, [27](#)

lfeDmxEnable
 AACENC_MetaData, [31](#)

lfeDmxLevel
 AACENC_MetaData, [31](#)

maxAncBytes
 AACENC_InfoStruct, [27](#)

maxOutBufBytes
 AACENC_InfoStruct, [27](#)

nDelay
 AACENC_InfoStruct, [27](#)

nDelayCore
 AACENC_InfoStruct, [28](#)

numAncBytes
 AACENC_InArgs, [26](#)
 AACENC_OutArgs, [32](#)

numBufs
 AACENC_BufDesc, [25](#)

numInSamples
 AACENC_InArgs, [26](#)
 AACENC_OutArgs, [32](#)

numOutBytes
 AACENC_OutArgs, [32](#)

OUT_AU_SIZES
 aacenc.lib.h, [36](#)

OUT_BITSTREAM_DATA
 aacenc.lib.h, [36](#)

PCE_mixdown_idx_present
 AACENC_MetaData, [29](#)

prog_ref_level
 AACENC_MetaData, [29](#)

prog_ref_level_present
 AACENC_MetaData, [29](#)

surroundMixLevel
 AACENC_MetaData, [30](#)