

# VisualOn AAC Encoder Reference Manual

*VisualOn, Inc.*

170 Knowles Drive, Suite 211

Los Gatos, CA 95032, USA

<http://www.visualon.com>

## Revision History

Date	Version	Changes	Author
May 21 2009	1.0.0	Initial Version	Jacky Lin

---

## Table of Contents

1	Overview.....	4
2	Files in SDK.....	6
2.1	Header Files .....	6
2.2	Sample Code Files.....	6
2.3	Encoder Library Files.....	6
3	Input & Output.....	6
3.1	Input .....	6
3.2	Output.....	6
3.2.1	Output Data.....	6
3.2.2	Return Code .....	6
4	Encoder Data Definition .....	7
4.1	Common Audio Encoder Data Structure.....	7
4.1.1	STRUCTURE VO_CODECBUFFER.....	7
4.1.2	STRUCTURE VO_AUDIO_OUTPUTINFO.....	8
4.1.3	STRUCTURE VO_CODEC_INIT_USERDATA .....	8
4.1.4	ENUM VO_AUDIO_CODINGTYPE.....	9
4.2	AAC Encoder Data Structure .....	9
4.3	Parameter IDs.....	10
5	Supported OSs and CPUs .....	10
6	How to Use the API .....	11
6.1	Only One API.....	11
6.2	Six Functions in VO_AUDIO_CODECAPI .....	11
7	Sample Code Details.....	13
7.1	Memory .....	13
7.2	Input Mode.....	14
7.3	Encoding Process .....	14
8	Support.....	14

# 1 Overview

The document details the Application Programming Interfaces (APIs) of Advance Audio Codec (AAC) encoder. It allows you to compress PCM data to standard AAC compliant bit streams. The AAC encoder is based on AAC LC profile. The supported output formats are ADTS and raw AAC data with mono and stereo channels. The encoder is optimized for various ARM instruction sets, including v5, and v7 with NEON, instructions.

Figure 1 show a sample sequence of function execution flow in the AAC audio encoder.

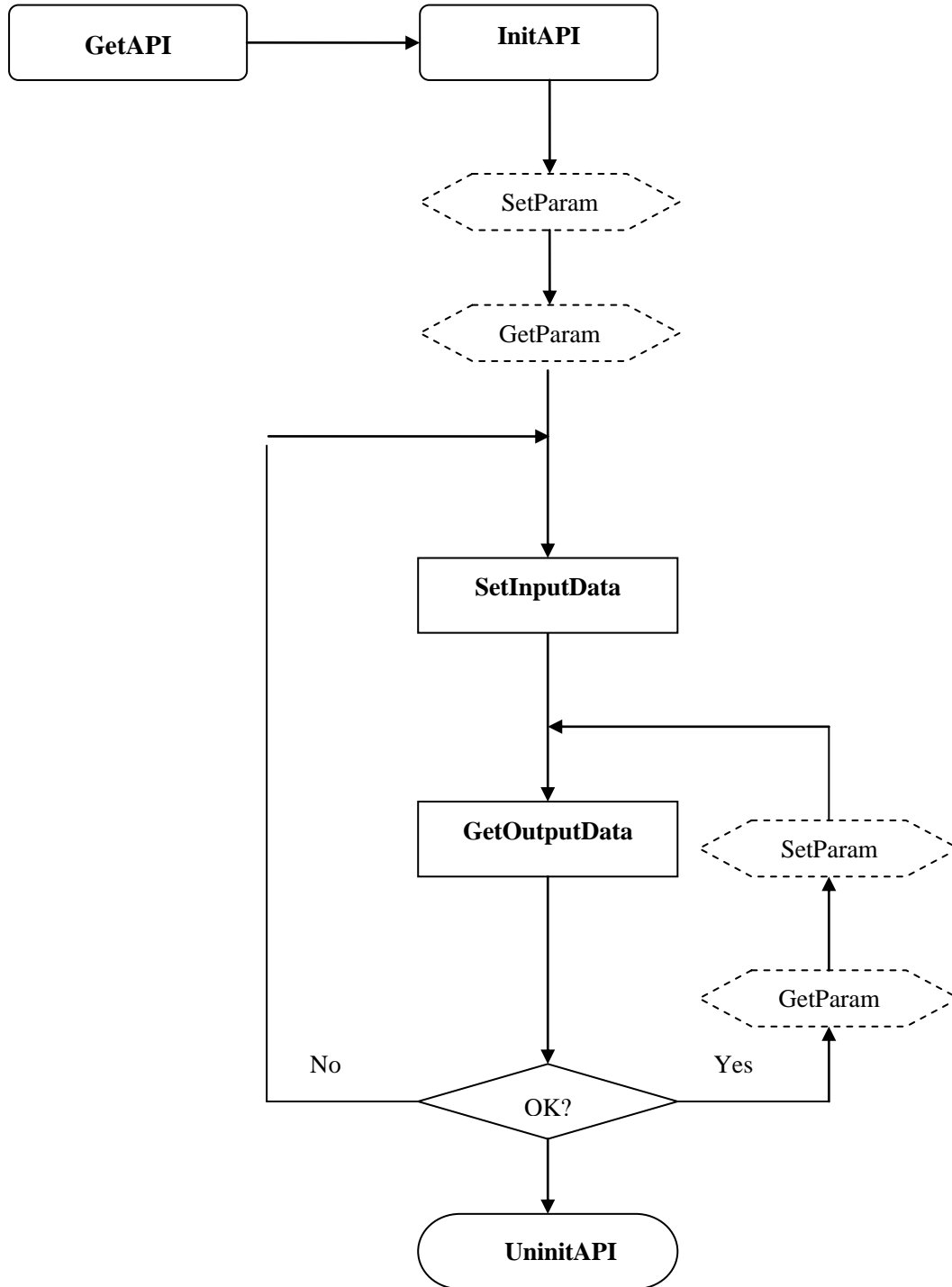


Figure 1. AAC Audio Encoder Diagram.

## 2 Files in SDK

### 2.1 Header Files

- 1) Common header files also used by other VisualOn codecs: voIndex.h, voType.h, voAudio.h, voMem.h, cmnMemory.h
- 2) Special header file used by the AAC encoder: voAAC.h

### 2.2 Sample Code Files

Sample application to run the encoder: AAC\_E\_SAMPLE.c

Memory implementation sample code to enhance portability: cmnMemory.c

### 2.3 Encoder Library Files

Lib files for core encoder: voAACEnc.\*

It may include other files for debugging purpose.

## 3 Input & Output

### 3.1 Input

The encoder supports mono or stereo line PCM input data. In case of stereo, the right and left channels of PCM data shall be interleaved, and the bits per sample shall be 16.

### 3.2 Output

#### 3.2.1 Output Data

The output data include ADTS or raw AAC bit stream, current encoded AAC frame length, channel information, sample rate and the used data length in the current PCM buffer.

#### 3.2.2 Return Code

All return codes are outlined below.

Table 4: Encoder Return Code

Return Code ID	Description
VO_ERR_NONE	Process data successful
VO_ERR_FAILED	Process data failed
VO_ERR_OUTOF_MEMORY	Out of memory
VO_ERR_NOT_IMPLEMENT	Features not implemented
VO_ERR_INVALID_ARG	Error in the input parameter
VO_ERR_INPUT_BUFFER_SMALL	Input buffer data is too small to encode a frame; please increase the input data size
VO_ERR_OUTPUT_BUFFER_SMALL	Output buffer size is too small; please realloc a bigger buffer
VO_ERR_WRONG_STATUS	Wrong encoder run-time status for the function call
VO_ERR_WRONG_PARAM_ID	Unsupported Parameter ID
VO_ERR_LICENSE_ERROR	License error; check with VisualOn for a new license.
VO_ERR_AUDIO_UNCHANNEL	Unsupported number of channels
VO_ERR_AUDIO_UNSSAMPLERATE	Unsupported sample rate
VO_ERR_AUDIO_UNFEATURE	Unsupported feature

## 4 Encoder Data Definition

### 4.1 Common Audio Encoder Data Structure

#### 4.1.1 STRUCTURE VO\_CODECBUFFER

VO\_CODECBUFFER is used for settings of input or output data buffer. It is defined as below.

```
typedef struct {
    VO_PBYTE Buffer;           /*!< Buffer pointer */
    VO_U32   Length;         /*!< Buffer size in byte */
    VO_S64   Time;          /*!< The time of the buffer */
} VO_CODECBUFFER;
```

### 4.1.2 STRUCTURE VO\_AUDIO\_OUTPUTINFO

VO\_AUDIO\_OUTPUTINFO is used for get audio information, including VO\_AUDIO\_FORMAT, bytes of data used in the input buffer, and a reserved value. It is defined as below.

```
typedef struct
{
    VO_AUDIO_FORMAT    Format;           /*!< Audio format information */
    VO_U32              InputUsed;      /*!< Bytes of data used in input buffer */
    VO_U32              Resever;        /*!< Resevered */
} VO_AUDIO_OUTPUTINFO;
```

VO\_AUDIO\_FORMAT, defined as below, specifies audio format information, including sample rate, number of channels and bits per sample.

```
typedef struct
{
    VO_S32 SampleRate; /*!< Sample rate */
    VO_S32 Channels;   /*!< Channel count */
    VO_S32 SampleBits; /*!< Bits per sample */
} VO_AUDIO_FORMAT;
```

### 4.1.3 STRUCTURE VO\_CODEC\_INIT\_USERDATA

VO\_CODEC\_INIT\_USERDATA allows users to defined standard C run-time library, such as memory operation functions, to increase the portability of the codec libraries. It is defined as below.

```
typedef struct{
    VO_INIT_MEM_FLAG    memflag; /*!<memory flag */
    VO_PTR              memData; /*!<a pointer to VO_MEM_OPERATOR
or a pre-allocated buffer */
    VO_U32              reserved1; /*!<reserved */
    VO_U32              reserved2; /*!<reserved */
} VO_CODEC_INIT_USERDATA;
```

VO\_INIT\_MEM\_FLAG, defined as below, specifies two types of memory operation. Currently the audio encoder only supports VO\_IMF\_USERMEMOPERATOR type. For the definition of VO\_MEM\_OPERATOR, please refer to voMem.h header file in our SDK.

```
typedef enum{
```





```

VO_IMF_USERMEMOPERATOR    =0, /*!< memData is a
                               VO_MEM_OPERATOR pointer */
VO_IMF_PREALLOCATEDBUFFER =1, /*!< memData is a pre-allocated buffer*/
VO_IMF_MAX = VO_MAX_ENUM_VALUE
}VO_INIT_MEM_FLAG;

```

#### 4.1.4 ENUM VO\_AUDIO\_CODINGTYPE

VO\_AUDIO\_CODINGTYPE is used to specify the audio coding type. All VisualOn supported audio types are listed in VO\_AUDIO\_CODINGTYPE as below.

```

typedef enum VO_AUDIO_CODINGTYPE {
    VO_AUDIO_CodingUnused = 0, /*!< Place holder value when coding is N/A */
    VO_AUDIO_CodingPCM,      /*!< Any variant of PCM coding */
    VO_AUDIO_CodingADPCM,    /*!< Any variant of ADPCM encoded data */
    VO_AUDIO_CodingAMRNB,    /*!< Any variant of AMR encoded data */
    VO_AUDIO_CodingAMRWB,    /*!< Any variant of AMR encoded data */
    VO_AUDIO_CodingAMRWBP,   /*!< Any variant of AMR encoded data */
    VO_AUDIO_CodingQCELP13,  /*!< Any variant of QCELP 13kbps encoded data */
    VO_AUDIO_CodingEVRC,     /*!< Any variant of EVRC encoded data */
    VO_AUDIO_CodingAAC,      /*!< Any variant of AAC encoded data, 0xA106 -
ISO/MPEG-4 AAC, 0xFF - AAC */
    VO_AUDIO_CodingAC3,      /*!< Any variant of AC3 encoded data */
    VO_AUDIO_CodingFLAC,     /*!< Any variant of FLAC encoded data */
    VO_AUDIO_CodingMP1,      /*!< Any variant of MP1 encoded data */
    VO_AUDIO_CodingMP3,      /*!< Any variant of MP3 encoded data */
    VO_AUDIO_CodingOGG,      /*!< Any variant of OGG encoded data */
    VO_AUDIO_CodingWMA,      /*!< Any variant of WMA encoded data */
    VO_AUDIO_CodingRA,       /*!< Any variant of Real Audio encoded data */
    VO_AUDIO_CodingMIDI,     /*!< Any variant of MIDI encoded data */
    VO_AUDIO_CodingDRA,      /*!< Any variant of DRA encoded data */
    VO_AUDIO_Coding_MAX     = VO_MAX_ENUM_VALUE
} VO_AUDIO_CODINGTYPE;

```

## 4.2 AAC Encoder Data Structure

The AACENC\_PARAM structure allows user to set properties of encoding process, including sample rate, bit rate, number of channels and if ADTS used.

```

typedef struct {
    int    sampleRate;        /*! audio file sample rate */
    int    bitRate;          /*! encoder bit rate in bits/sec */
    short  nChannels;        /*! number of channels on input (1,2) */

```

```

short    adtsUsed;                /*! add ADTS header in bit stream, if nonzero. Else do
nothing */
} AACENC_PARAM;
    
```

The default values are sample rate 44100, bit rate 128000, number of channels 2, and ADTS used.

## 4.3 Parameter IDs

The section lists the valid parameter IDs used in two API functions, SetParam and GetParam.

### VO\_PID\_COMMON\_HEADATA

Set or get the header data of audio tracks. The parameter is in VO\_CODECBUFFER structure.

### VO\_PID\_COMMON\_FLUSH

Reset encoder status when seeking or restart. The parameter is an interger. If nozero, reset it. Otherwise, do nothing.

### VO\_PID\_AUDIO\_FORMAT

Set or get the audio format. The parameter is in the struture of VO\_AUDIO\_FORMAT.

### VO\_PID\_AUDIO\_SAMPLERATE

Set or get the audio sample rate. The parameter is an interger indicating the sample rate.

### VO\_PID\_AUDIO\_CHANNELS

Set or get the number of audio channels. The parameter is an interger indicating the channel number.

### VO\_PID\_AAC\_PROFILE

Set or get AAC profile. The parameter is an integer as defined in VOAACOBJECTTYPE. Currently the AAC encoder supports AAC LC profile only.

### VO\_PID\_AAC\_ENCPARAM

Set or get the encoder parameters. The parameter is in the struture of AACENC\_PARAM.

## 5 Supported OSs and CPUs

- 1) OS: Android, iPhone, Windows Mobile, Linux, RIM, Nucleus, Windows XP, ...
- 2) x86, ARMv4, ARMv5, ARMv6, ARMv7(NEON)

## 6 How to Use the API

### 6.1 Only One API

VO\_S32 VO\_API voGetAACEncAPI (VO\_AUDIO\_CODECAPI \*pDecHandle);

To simplify the interface, we only provide one API, voGetAACEncAPI, for the SDK. voGetAACEncAPI is to get the API handle of the encoder. Please refer to voAudio.h header file in SDK for the definition of the API handle, VO\_AUDIO\_CODECAPI \* pDecHandle. Currently there are six available functions in pDecHandle as described below. Additional required or customized functions can be easily extended using SetParam function with different Parameter IDs.

### 6.2 Six Functions in VO\_AUDIO\_CODECAPI

- 1) VO\_U32 Init (VO\_HANDLE \* phDec,
  - VO\_AUDIO\_CODINGTYPE vType,
  - VO\_CODEC\_INIT\_USERDATA \* pUserData);

Initialize the audio encoder and return a encoder handle.

phCodec [OUT]: Audio encoder handle.

vType[IN]: The codec type if the SDK support multiple codecs.

pUserData[IN]: The initialized parameter, including memory operator or allocated memory.

Return VO\_ERR\_NONE if succeeded. Otherwise, return an error code.

**Note:**

- a) This is the first call for every encoder instance.
- b) By configuring VO\_CODEC\_INIT\_USERDATA, input memory used by the encoder can be allocated by users to optimize the performance.

- 2) VO\_U32 SetInputData (VO\_HANDLE hDec,
  - VO\_CODECBUFFER \* pInput);

Input PCM data.

hCodec [IN] The audio encoder handle created in Init function.

pInput [IN] The input buffer param.

Return VO\_ERR\_NONE if succeeded. Otherwise, return an error code.

**Note:**

Encoder supports input data in one frame or any length of PCM data at each call. For the best performance, users should input one frame at a call.

- 3) VO\_U32 GetOutputData (VO\_HANDLE hDec,  
VO\_CODECBUFFER \*pOutBuffer,  
VO\_AUDIO\_OUTPUTINFO \* pOutInfo);

Get the compressed audio data and output audio information. Please refer to voAudio.h header file in the SDK for the definition of VO\_AUDIO\_OUTPUTINFO.

hCodec [IN] The audio encoder handle created in Init function.

pOutBuffer [OUT] Output buffer with one compressed audio frame

pOutInfo [OUT] Audio output audio data information and input data length used.

Return VO\_ERR\_NONE if succeeded. Otherwise, return an error code, or return VO\_ERR\_INPUT\_BUFFER\_SMALL if input data are all finished, or the leftover of input data are not enough to encode one frame.

**Note:**

This function outputs one frame of compressed audio data at each call. Users can use a loop to call GetOutputData to encode all the frames of the input data in SetInputData until return VO\_ERR\_INPUT\_BUFFER\_SMALL if there are possibilities the input data are more than one PCM frame.

- 4) VO\_U32 SetParam (VO\_HANDLE hDec,  
VO\_S32 uParamID,  
VO\_PTR pData);

Set the data of the specified parameter ID.

hCodec [IN] The audio encoder handle created in Init function.

uParamID [IN] A parameter ID.

pData [IN] The value of the parameter for the ID. It can be a pointer or a value.

Return VO\_ERR\_NONE if succeeded. Otherwise, return an error code.

**Note:**

There are pre-defined parameter IDs as described in “Parameter IDs” section.

Additionally, the function provides the flexibility to easily add customized functions for special customer requirements by adding new parameter IDs.

- 5) VO\_U32 GetParam (VO\_HANDLE hDec,  
                    VO\_S32 uParamID,  
                    VO\_PTR pData);

Get the data of the specified parameter ID.

hCodec [IN] The audio encoder handle created in Init function.

uParamID [IN] The param ID.

pData [OUT] The value of the parameter for the ID. It can be a pointer or a value.

Return VO\_ERR\_NONE if succeeded. Otherwise, return an error code.

**Note:**

GetParam function is the counterpart of SetParam function. They share the same definition of parameter ID.

- 6) VO\_U32 Uninit (VO\_HANDLE hDec);

Un-initialize the encoder after all encoder operations are done. It will free up all internal memory used inside encoder.

hCodec [IN] The audio encoder handle created in Init function.

Return VO\_ERR\_NONE if succeeded. Otherwise, return an error code.

## 7 Sample Code Details

### 7.1 Memory

- 1) Input memory:

Memory used by PCM audio data is allocated by application. It gives users the flexibility to share memory with other module without unnecessary memory copying.

- 2) Encoder Internal memory:

There are two methods to provide the internal memory used by the encoder.

- a) Default method.

Encoder call standard C run-time function malloc to allocate memory.

- b) Users provide memory operation functions

Users can set VO\_MEM\_OPERATOR, as defined in voMEM.h, to the encoder during initialization.

## 7.2 Input Mode

We support two input modes, Frame and Stream. The default is Frame mode.

- a) Frame mode

When calling SetInputData, the input data length shall be equal to or more than one complete frame.

- b) Stream mode

When calling SetInputData, the input data length can be less than one complete frame.

## 7.3 Encoding Process

Please refer to the comments in the sample code.

# 8 Support

If you have any problems or questions about this SDK, please feel free to contact [info@visualon.com](mailto:info@visualon.com).