06-3648-API-TX30
MARCH 9, 2006
1.0

Application Programmers Interface for Ogg Vorbis Decoder

ABSTRACT:

Application Programmers Interface for Ogg Vorbis Decoder

KEYWORDS:

Multimedia codecs, Ogg Vorbis

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	19-Dec-2005	Anand Narayanan	Initial Draft
0.2	02-Jan-2006	Anand Narayanan	Modified according to the latest API header
0.3	18-Jan-2006	Anand Narayanan	Included the Relocation API header
1.0	06-Feb-2006	Lauren Post	Using new format
2.0	28-Dec-2008	Lyon Wang	Support for Vorbis Raw data input decode

Table of Contents

Introduction

1.1 Purpose

This document gives the details of the application programmer's interface of the Ogg Vorbis Audio Decoder. The purpose of the document is to define the interfaces, data structures and the sequencing of the API calls to facilitate decoding of Ogg Vorbis bit streams.

1.2 Scope

This document does not detail the implementation of this decoder. It only explains the functions and data structures exposed for the application developer to use the Ogg Vorbis decoder (hereafter referred as decoder) library.

1.3 Audience Description

The reader is expected to have basic understanding of Audio Compression and Vorbis decoding.

1.4 References

1.4.1 Standards

- Vorbis I Specification

1.4.2 Freescale Multimedia References

- Ogg Vorbis Decoder Application Programming Interface – oggvorbis_dec_api.doc
- Ogg Vorbis Decoder Requirements Book - ogg vorbis_dec_reqb.doc
- Ogg Vorbis Decoder Test Plan – oggvorbis_dec_test_plan.doc
- Ogg Vorbis Decoder Release notes – oggvorbis_dec_release_notes.doc
- Ogg Vorbis Decoder Test Results – oggvorbis_dec_test_results.doc
- Ogg Vorbis Decoder Performance Results – oggvorbis_dec_perf_results.doc
- Ogg Vorbis Decoder Interface Header – oggvorbis_dec_api.h, oggvorbis_dec_os_types.h
- Ogg Vorbis Decoder Application Code – ivorbisfile_example.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
API	Application Programming Interface
ARM	Advanced RISC Machine
FSL	Freescale
MDCT	Modified Discrete Cosine Transform
OS	Operating System
PCM	Pulse Code Modulation
RVDS	ARM RealView Development Suite
TBD	To Be Determined
UNIX	Linux PC x/86 C-reference binaries

1.6 Document Location

docs/oggvorbis_dec

2 API Description

This section describes the functions and data structures supported by the decoder followed by an example usage of the Ogg Vorbis decoder. The salient features of the Decoder are mentioned below.

- Decoder shall be available as a static and shared library
- Decoder provides APIs along with a set of Data structures to facilitate the process of configuring and decoding
- Library exposes C-style API interfaces

The library is provided along with header files that are required for the API definition. There shall be two header files namely, oggvorbis_dec_api.h which defines the APIs of the Decoder and oggvorbis_dec_os_types.h which defines platform independent basic data types to facilitate portability.

2.1 Data Structures

This section describes the data structures used in the decoder interface.

sOggVorbisDecObj

This is the main data structure which should be passed to all the decoder functions. The definition of the structure is given below.

```
typedef struct {
      void *datasource;
      int current_section;
      char *initial_buffer;
      int buffer_length;

      void *pvOggDecObj;
      unsigned int OggDecObjSize;
      void *pvAppContext;
       size_t (*read_func)  (void *ptr, size_t size, size_t nmemb, void
      *datasource);
      int    (*seek_func)  (void *datasource, ogg_int64_t offset, int
      whence);
      int    (*close_func) (void *datasource);
      long   (*tell_func)  (void *datasource);

      char *pcmout;
      int output_length;

      unsigned char *decoderbuf;
      unsigned int buf_size;

      int TotalCodeBookEntries;
      int NoOfChannels;
      int SampleRate;
      int max_bitrate;
      int ave_bitrate;
```

```
        int min_bitrate;
        int mPacketCount;
        int byteConsumed;
        int outNumSamples;
} sOggVorbisDecObj;
```

Description of structure `sOggVorbisDecObj`

datasource

Pointer to file or other ogg source. When using stdio based file/stream access, this field contains a FILE pointer. When using custom IO via callbacks, decoder treats this void pointer as a black box only to be passed to the callback routines provided by the application. Ex: This can be used to store the application context.

current_section

The decoder sets this to the logical bit stream number. Application should not modify its value.

initial_buffer

Typically set to NULL. This parameter is useful if some data has already been read from the file and the stream is not seekable. It is used in conjunction with *buffer_length*. In this case, *initial_buffer* should be a pointer to a buffer containing the data read.
(Note: when raw data input decoding, this parameter is the input buffer of the raw data.)

buffer_length

Typically set to 0. This parameter is useful if some data has already been read from the file and the stream is not seekable. In this case, buffer_length should contain the length (in bytes) of the buffer. Used together with initial_buffer.
((Note: when raw data input decoding, this parameter indicates the length(in bytes) of the buffer.)

pvOggDecObj

This is an internal object context for the decoder and application should not change this. The memory should be allocated by the application using *OggDecObjSize* returned by the decoder.

OggDecObjSize

This is the total size of *OggVorbis_File* structure(internal to the decoder). Application should not modify its value.

pvAppContext

This space is provided for the application to keep its context and the decoder does not use it. *read_func, seek_func, close_func* and *tell_func* are the function pointer. The details are given in the section below.

pcmout:

Output buffer pointer. The memory should be allocated in the application and has to be aligned.

output_length

Output buffer length. The length of the output buffer should be passed to the decoder.

Decoderbuf:

This is the pointer to decoder internal buffer. The memory should be allocated by the application for the size returned by the QueryMem. This memory should be zero initialized.

TotalCodeBookEntries

This is the total number of code book entries in the stream returned by the decoder after QueryMem. Application should not modify its value.

NoOfChannels

This is the number of channels in the stream returned by the decoder after

QueryMem. Application should not modify its value.

`SampleRate`
This is the sample rate in the stream returned by the decoder after QueryMem. Application should not modify its value.

`max_bitrate`
This is the maxium bitrate in the stream returned by the decoder after QueryMem. Application should not modify its value.

`ave_bitrate`
This is the average bitrate in the stream returned by the decoder after QueryMem. Application should not modify its value.

`min_bitrate`
This is the minimum bitrate in the stream returned by the decoder after QueryMem. Application should not modify its value.

`mPacketCount`
This is a Counter for packet count in the decoder, as first 3 packets are the necessarily information for whole stream decoding. Application will initialize it to 0 in initialization.

`byteConsumed`
This is the byte consumed value returned after decodinig one raw data packet. It indicates that how much data has been consumed in this packet decode.

`outNumSamples`
This is the number of sample in each channel returned after decoding one raw data packet.

2.1.1 CallBack functions

read_func
This function pointer will be used by the decoder to read data from the application. The application has to implement this function. This function is similar to the fread function call.

seek_func
This function pointer will be used to move the read position of the input. The application has to implement this function. This function is similar to the fseek function call.

close_func
This function pointer will be used during the clean up of the decoder to indicate the end of read operation. The application has to implement this function. This function is similar to the fclose function call.

tell_func
This function pointer will be used to know the read position in the input buffer. The application has to implement this function. This function is similar to the ftell function call.

Note: when raw data input decoding , call back functions will be no use, the application using the push mode.

2.2 Return Codes

2.2.1 Library API Return codes

The return values of the APIs are listed below.

1. When decoding the data with OGG container format:

Values	**Notation (used in this document)**
0	OGGV_SUCCESS
-1	OGGV_FALSE
-2	OGGV_EOF
-3	OGGV_HOLE
-128	OGGV_EREAD
-129	OGGV_EFAULT
-130	OGGV_EIMPL
-131	OGGV_EINVAL
-132	OGGV_ENOTVORBIS
-133	OGGV_EBADHEADER
-134	OGGV_EVERSION
-135	OGGV_ENOTAUDIO
-136	OGGV_EBADPACKET
-137	OGGV_EBADLINK
-138	OGGV_ENOSEEK

The description of the error codes is given below.

OGGV_SUCCESS
: True or success

OGGV_FALSE
: Not true, or no data available

OGGV_EOF
: Indicates stream is at end of file immediately after a seek

OGGV_HOLE
: Decoder encountered missing or corrupt data in the bitstream. Recovery is normally automatic and this return code is for informational purposes only.

OGGV_EREAD
: Read error while fetching compressed data for decode

OGGV_EFAULT
: Internal inconsistency in decode state. Continuing is likely not possible.

OGGV_EIMPL
: Feature not implemented

OGGV_EINVAL
: Either an invalid argument or incompletely initialized argument passed to API call

OGGV_ENOTVORBIS
: The given file/data was not recognized as Ogg Vorbis data.

OGGV_EBADHEADER
: The file/data is apparently an Ogg Vorbis stream, but contains a corrupted or undecipherable header.

OGGV_EVERSION
: The bitstream format revision of the given stream is not supported.

OGGV_EBADLINK
: The given link exists in the Vorbis data stream, but is not decipherable due to garbage or corruption.

OGGV_ENOSEEK
: The given stream is not seekable

2. When decoding the raw data input without OGG container format:

```
typedef enum {
   VORBIS_DATA_OK = 0,
   VORBIS_HEADER_OK,
   VORBIS_COMMENT_OK,
   VORBIS_CODEBOOK_OK,
   VORBIS_ERROR_MIN = 64,
   VORBIS_HEADER_BAD,
   VORBIS_COMMENT_BAD,
   VORBIS_CODEBOOK_BAD,
   VORBIS_SYNTH_FAILED,
   VORBIS_BLOCKIN_FAILED,
   VORBIS_OUTBUF_OVFLOW
}eVorbisResult ;
```

2.3 Application Programmer Interface Functions

2.3.1 Query Memory

The decoder does not perform any dynamic memory allocation. However, the decoder memory requirements may depend on the input bitstream. The application has to allocate memory as required by the decoder. So the application first needs to query for memory by calling the function *OggVorbisDecQueryMem*. This function must be called before any other decoder functions are invoked. Also, the structure members, *datasource*, *decoderbuf, buffer_length* and the *call back function pointers* needs to be updated by the application before calling this API.

This function parses the required decoder information from the bitstream and fills the memory information structure array. The application will then allocate memory and gives the memory pointers to the decoder by calling the initialization function, which is given in the next section. During the memory query, this function calls *read_func* to get input bitstream required for the memory query. This routine needs to be called at the beginning of every new file/stream.

C prototype:

```
int OggVorbisQueryMem(sOggVorbisDecObj *psOVDecObj)
```

Arguments:

Input Parameters

- ***psOVDecObj*** Decoder object pointer.

Output Parameters

Returntype ***int***

Return value**:**

Memory in bytes required by the decoder for a particular stream.
The application should provide zero initialized memory to the decoder aligned to 4 byte boundary

2.3.2 Initialization

All initializations required for the decoder are done in *OggVorbisDecoderInit*. This function must be called before the main decoder functions are invoked.

C prototype:

```
int OggVorbisDecoderInit(sOggVorbisDecObj *psOVDecObj)
```

Arguments:

Input Parameters

- ***psOVDecObj*** Decoder object pointer.

Output Parameters

Returntype int

***Return value*:**

Indicates whether the decoder was initialized properly.

2.3.3 Decode

The main decoder function is *OggVorbisDecode*. This function decodes the Ogg bit stream in the input buffer and returns up to the specified number of bytes of decoded PCM audio.

C prototype:

```
int OggVorbisDecode (sOggVorbisDecObj * psOVDecObj)
```

Arguments:

Input Parameters

- ***psOVDecObj*** Decoder object pointer.

Output Parameters

1. with OGG container format input:

Returntype ***int***

***Return value*:**

long Number of bytes of PCM actually returned or the error

Return values are -

Number of bytes of PCM actually returned	- Function successful.
0	- EOF
<0	- Error

2. raw data without OGG container format input:

Returntype ***eVorbisResult***

***Return value*:**

Return status of current packet decoding

Return values are -

VORBIS_DATA_OK = 0	-- function successed
Other data	-- decoding abnormal
psOVDecObj-> byteConsumed	-- return decoder data consumed
psOVDecObj-> outNumSamples	-- return output sample numbers in each channel

2.3.4 Relocate

This function is used to relocate the internal buffer of the decoder. This function must be called by the application with the new internal buffer pointer that the decoder should relocate to. The application should copy the old internal buffer content to the newly allocated buffer and call *OggVorbisDecoderReLocate* function before calling *OggVorbisDecode*.

C prototype:

```
int OggVorbisDecoderReLocate(sOggVorbisDecObj *psOVDecObj);
```

Arguments:

Input Parameters

- **sOggVorbisDecObj** Decoder object pointer.

Output Parameters

Returntype int

***Return value*:**

0 – indicates success

2.3.5 Cleanup Decoder

This is the decoder function to release any resources used by the decoder.

C prototype:

```
int OggVorbisCleanup(sOggVorbisDecObj *psOVDecObj)
```

Arguments:

Input Parameters

- ***psOVDecObj*** Decoder object pointer.

Output Parameters

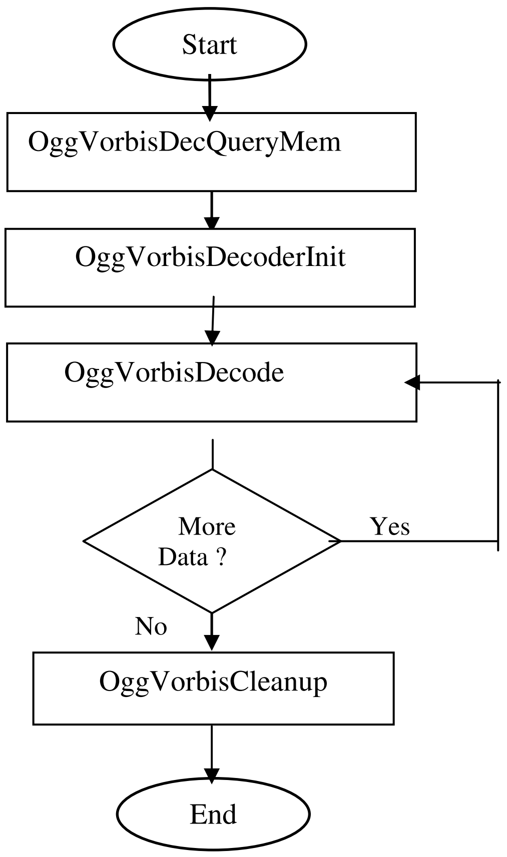
Returntype int

***Return value*:**

Indicates whether the clean up of the decoder was properly done.

3 Decoder Application Flow

1. The Flow Chart for using the Ogg Vorbis Decoder is as shown below (with OGG container format input)

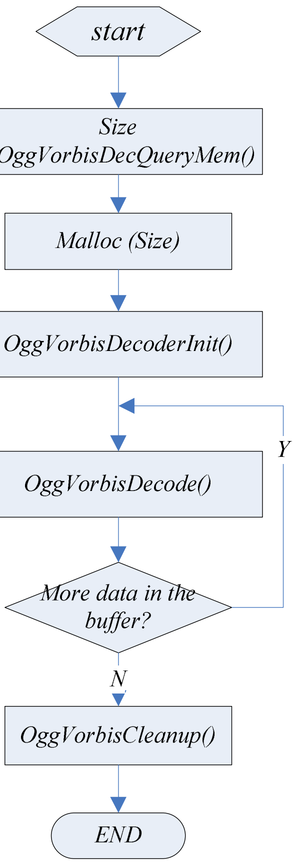


This example shows how to use the Ogg Vorbis Decoder. Before calling the decoder, we call the decoder initialization function OggVorbisDecoderInit. This function initializes the decoder. The main decode function is called in a loop. Please note that the error handling is not shown properly and the code may not compile as it is.

```
int main(int argc,char *argv[])
{
      sOggVorbisDecObj sOVDecObj;
      struct_input_file *input_file;
      long size=0;
      char *pcmout=NULL;
      FILE *fpin=NULL,*fpout=NULL,*fperr=NULL;
      int eof=0;

      fpin = fopen(argv[1],"rb");
      if(fpin == NULL)
      {
            printf("Error opening i/p file %s\n",argv[1]);
            return ;
      }

      input_file = (struct_input_file
*)malloc(sizeof(struct_input_file));
      if(input_file == NULL)
      {
            printf("Error Allocating memory \n");
            return;
      }

      input_file->fp = fpin;
      input_file->file_size = 0;
      input_file->bytes_read = 0;

      while(fgetc(fpin)!=EOF)                   //i/p file size
            input_file->file_size++;

      input_file->buff = (char *)malloc(sizeof(char)*input_file-
>file_size);
      input_file->buff_head = input_file->buff;
      fseek(fpin,SEEK_SET,0);
      fread(input_file->buff,sizeof(char),input_file->file_size,fpin);

      fpout = fopen(argv[2],"wb");
      if(fpout == NULL)
      {
            printf("Error opening o/p file %s\n",argv[2]);
            return ;
      }
      fperr = fopen(argv[3],"wb");
      if(fperr == NULL)
      {
            printf("Error opening err file %s\n",argv[3]);
            return ;
      }
      /*Allocating aligned memory for output buffer*/
      pcmout = (char *)malloc(sizeof(char)*CHUNK_SIZE + 4);
      pcmout = (char *)(((unsigned int)pcmout + 3) & ~0x3);

    /* Initialization of the decoder object */
      sOVDecObj.datasource = (void *)input_file->buff;
      sOVDecObj.read_func = my_fread;
      sOVDecObj.seek_func = my_fseek;
      sOVDecObj.close_func = my_fclose;
      sOVDecObj.tell_func = my_ftell;
      sOVDecObj.pcmout = pcmout;
    sOVDecObj.output_length = CHUNK_SIZE;

      size = OggVorbisQueryMem(&sOVDecObj);

    sOVDecObj.pvOggDecObj = (void *)malloc(sOVDecObj.OggDecObjSize);
    memset(sOVDecObj.pvOggDecObj,0,sOVDecObj.OggDecObjSize);
      sOVDecObj.buf_size = size;

      /*decoder internal buffer allocation*/
      sOVDecObj.decoderbuf = (unsigned char*)malloc(size);
      if(!sOVDecObj.decoderbuf)
      {
            printf("Internal Decoder buffer allocation failed!!");
            return ;
```

```
	}
	memset(sOVDecObj.decoderbuf,0,size);

	sOVDecObj.datasource = (void *)input_file;

	if(OggVorbisDecoderInit(&sOVDecObj) != OV_SUCCESS) {
		fprintf(fperr,"Input does not appear to be an Ogg
bitstream.\n");
		exit(1);
	}

	while(!eof){
		long ret = OggVorbisDecode(&sOVDecObj);		//Decode Call
		if (ret == 0) {
			/* EOF */
			eof=1;
		} else if (ret < 0) {
			/* Application specific Error Handling */
		} else {
			fwrite(pcmout,1,ret,fpout);
		}
	}
	/* cleanup */
	OggVorbisCleanup(&sOVDecObj);

	fprintf(fperr,"Done.\n");
	fclose(fpin);
	fclose(fpout);
	fclose(fperr);
	free(sOVDecObj.decoderbuf);
	free(sOVDecObj.pvOggDecObj);
	free(pcmout);
	return 0;
}
```

2. The Flow Chart for using the Ogg Vorbis Decoder is as shown below (raw data without OGG container format input)



```
Example code :
int main(int argc,char *argv[]) {
    long ret =0;
    sOggVorbisDecObj sOVDecObj;
    unsigned char *old_buffer;
    struct_input_file *input_file;
    long size=0;
    char *pcmout=NULL;
    FILE *fpin=NULL,*fpout=NULL;

    fpin = fopen(argv[1],"rb");
    fpout = fopen(argv[2],"wb");

    input_file->fp = fpin;
    input_file->file_size = 0;
    input_file->bytes_read = 0;


    fseek(fpin, 0, SEEK_END);
    input_file->file_size = ftell(fpin);
    fseek(fpin, 0, SEEK_SET);

    input_file->buff = (char *)malloc(sizeof(char)*input_file-
>file_size);
    input_file->buff_head = input_file->buff;
    fread(input_file->buff,sizeof(char),input_file->file_size,fpin);


    /*Allocating aligned memory for output buffer*/
    pcmout = (char *)malloc(sizeof(char)*CHUNK_SIZE + 4);
    pcmout = (char *)(((unsigned int)pcmout + 3) & ~0x3);

    /* Initialization of the decoder object */
    sOVDecObj.datasource = (void *)input_file->buff;

    sOVDecObj.pcmout = pcmout;
    sOVDecObj.output_length = CHUNK_SIZE;

    size = OggVorbisQueryMem(&sOVDecObj);

    sOVDecObj.pvOggDecObj = (void *)malloc(sOVDecObj.OggDecObjSize);

    memset(sOVDecObj.pvOggDecObj,0,sOVDecObj.OggDecObjSize);
    sOVDecObj.buf_size = size;

    sOVDecObj.decoderbuf = (unsigned char*)malloc(size);
    sOVDecObj.datasource = (void *)input_file;

    OggVorbisDecoderInit(&sOVDecObj);

    {
        int inputSize, byte, output_sample;
        int byte_consumed = 0;
        inputSize = my_fread(input_buffer, 1, INPUT_BUFFER_SIZE,
sOVDecObj.datasource);
        sOVDecObj.initial_buffer = input_buffer;
        while (inputSize>0) {
            sOVDecObj.buffer_length = inputSize;
            ret = OggVorbisDecode(&sOVDecObj);
            if(ret) {printf("decode error!\n"); break;}
            output_sample = sOVDecObj.outNumSamples;
            if (output_sample)
                fwrite(pcmout, output_sample,
sizeof(short)*(sOVDecObj.NoOfChannels), fpout);
            byte_consumed = sOVDecObj.byteConsumed;
            memmove(input_buffer, input_buffer+byte_consumed,  inputSize-
byte_consumed );
            byte = my_fread(input_buffer+(inputSize-byte_consumed), 1,
byte_consumed, sOVDecObj.datasource);
            inputSize = byte + (inputSize-byte_consumed);
        }

    }
    /* cleanup */
    OggVorbisCleanup(&sOVDecObj);
    printf("Done.\n");
    fclose(fpin);
    fclose(fpout);
    free(sOVDecObj.decoderbuf);
    free(sOVDecObj.pvOggDecObj);
    free(pcmout);
    return ;

}
```