08-6337-API-ZCH66

8/31/2007

0.1

Application Programming Interface for WMA10 Decoder

ABSTRACT:

Application Programming Interface for WMA10 Decoder

KEYWORDS:

Multimedia codecs, WMA10, Windows Media Audio

APPROVED:

Shang Shidong

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	31-Aug-2007	Li Jian	Initial Draft

Table of Contents

1 Introduction

1.1 Purpose

This document gives the application programming interface for the WMA10 Decoder.

1.2 Scope

This document describes only the functional interface of the WMA10 Decoder. It does not describe the internal design of the decoder. Specifically, it describes only those functions by which a software module can use the decoder.

1.3 Audience Description

The reader is expected to have basic understanding of Audio Signal processing and WMA decoding. The intended audience for this document is the development community who wish to use the WMA10 Decoder in their systems.

1.4 References

1.4.1 Standards

- "An overview of Window Media Audio Decoding", Microsoft Corporation
- ASF Specification from Microsoft Corporation, Revision 01.20.02, June 2004
- WMA Decoding Profiles Microsoft Corporation
- WMA Audio Concepts Microsoft Corporation
- WMA Decoder block diagram", Microsoft Corporation

1.4.2 General References

- Ted Painter and Andreas Spanias, "Perceptual Coding of Digital Audio", Proc. IEEE, vol-88, no.4, April 2000
- H.S.Malvar, "Lapped transforms for efficient subband/transform coding", IEEE trans. ASSP, June 1990.
- J.P.Princen, A.W.Johnson, A.B.Bradley, "Subband/transform coding using filterbank design based on time domain aliasing cancellation", in proc. IEEE Int. conference ASSP, April1987

1.4.3 Freescale Multimedia References

- WMA10 Decoder Requirements Book – wma10_dec_reqb.doc
- WMA10 Decoder Test Plan – wma10_dec_test_plan.doc
- WMA10 Decoder Release notes – wma10_dec_release_notes.doc
- WMA10 Decoder Test Results – wma10_dec_test_results.doc

- WMA10 Decoder Interface header – wma10_dec_interface.h
- WMA10 Decoder Application Code – wma10_app_test_gst_api.c.c

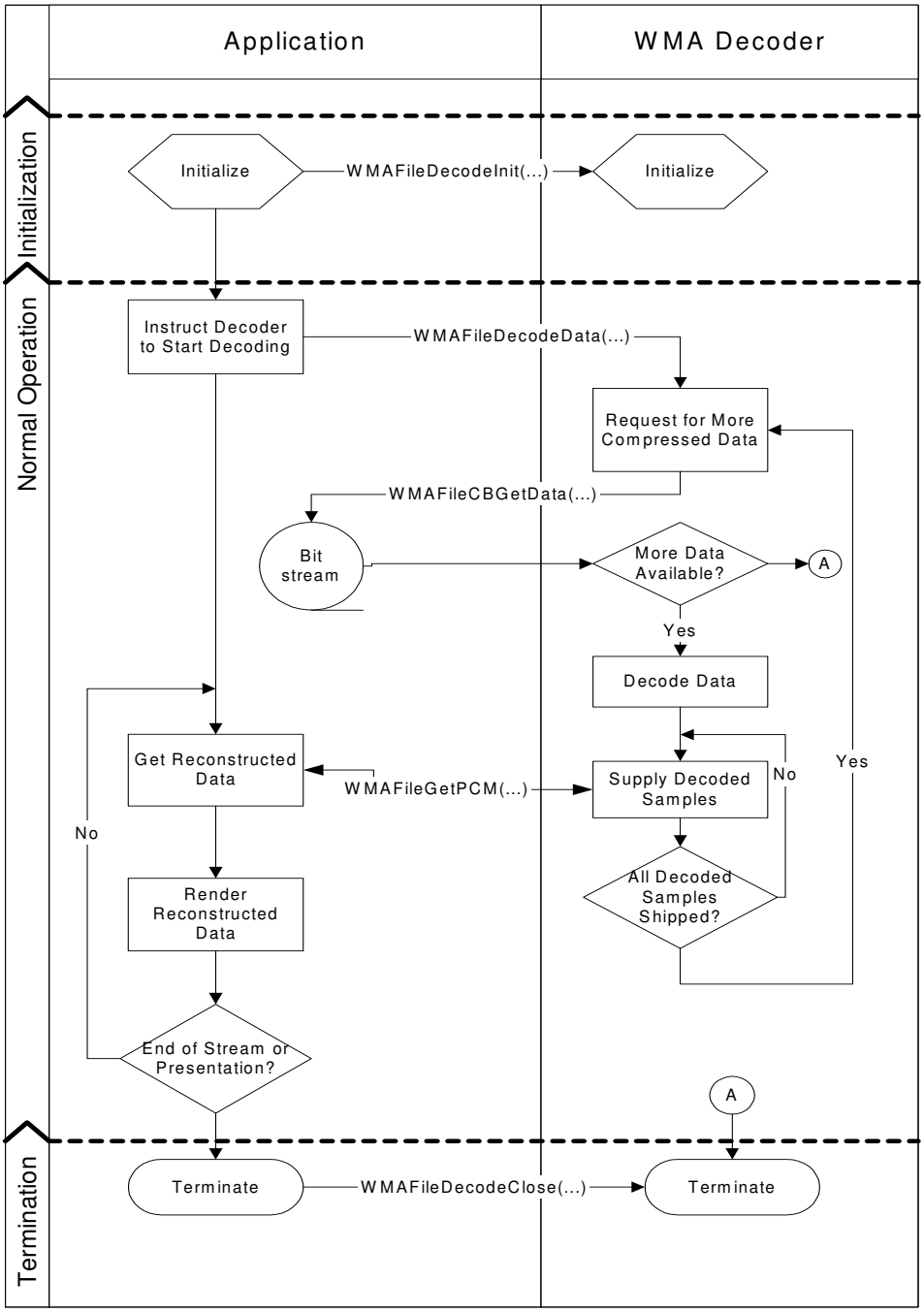
1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
API	Application Programming Interface
ARM	Advanced RISC Machine
ASF	Advanced Streaming Format
DAC	Digital to Audio Converter
LC	Low Complexity
MLT	Modulated Lapped Transform
OS	Operating System
PCM	Pulse Code Modulation
WMA	Windows Media Audio

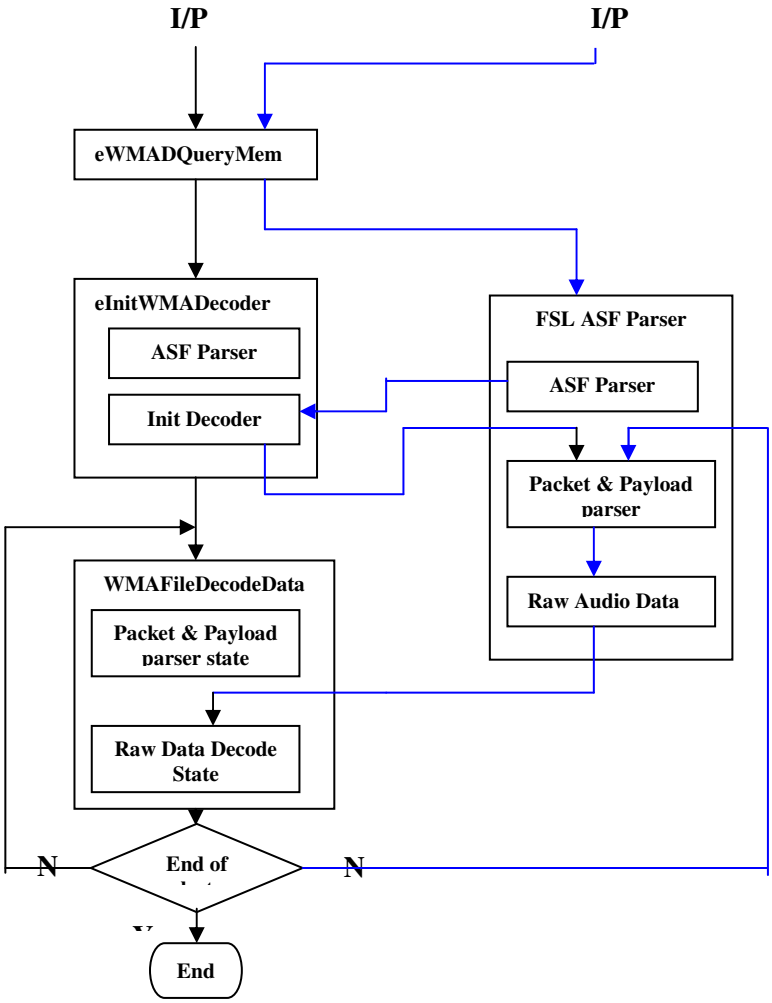
1.6 Document Location

docs/wma10_dec

2 Flow diagram of Interaction between an Application and Reference WMA10 Decoder



In the below flow chart black line indicates normal file decoding(with ASF headers) and blue line indicates flow of Raw File Decoding using ASF Parser.



3 API Description

This section describes the steps followed by the application to call the WMA10 Decoder. During each step the data structures used and the functions used will be explained. Pseudo code is given at the end of each step. The member variables inside the structure are prefixed as WMAD if that member variable needs to be initialized by the decoder.

When decoder starting, the application has to initialize the following members of WMADecoderParams with correct data.
bDropPacket, nDRCSetting, nDecoderFlags, nDstChannelMask, nInterpResampRate, nMBRTargetStream.

The output buffer size will be calculate during this step, and give out by *us32OutputBufSize* in size of bytes.

When decoder output wav file, *pWfx* shall point to *WAVEFORMATEXTENSIBLE* structure variable, otherwise, it shall be set to NULL.

The WMA10 Decoder can support ASF streams and elementary WMA raw stream as input. When raw data (elementary stream) input is supported, the application has to set *bIsRawDecoder* to true to enable raw data decoding option, otherwise, *bIsRawDecoder* shall be set to false. And also need to initialize the following members of WMADecoderParams with correct data for raw data.
us16Version,us16wFormatTag,us16Channels,us32SamplesPerSec, us32AvgBytesPerSec,us32nBlockAlign,us32ValidBitsPerSample, us32ChannelMask,us32AdvancedEncodeOpt2,us16EncodeOpt, us16AdvancedEncodeOpt.
For every payload the application has to update *bIsCompressedPayload* depending up on whether the payload is compressed or not.

When ASF input is supported, the *WMADDecoderParams* will be populated by the decoder library during the ***eInitWMADecoder*** call.

```
typedef struct {
    WMAD_Bool   bIsRawDecoder;
    WMAD_UINT16 us16Reentrant;
    WMAD_UINT8 *pus8output_file;
    WMAD_UINT8 *pus8input_file;

    WMAD_UINT16 us16Version;
    WMAD_UINT16 us16wFormatTag;
    WMAD_UINT16 us16Channels;       /* Number of channels (0 or 1) */
    WMAD_UINT32 us32SamplesPerSec;  /* Sampling frequency of the current
                                       frame in Khz */
    WMAD_UINT32 us32AvgBytesPerSec; /* Average byte rate */
    WMAD_UINT32 us32nBlockAlign;
    WMAD_UINT32 us32ValidBitsPerSample;
    WMAD_UINT32 us32ChannelMask;
    WMAD_UINT32 us32AdvancedEncodeOpt2;
    WMAD_UINT16 us16EncodeOpt;
    WMAD_UINT16 us16AdvancedEncodeOpt;
    WMAD_UINT32 us32Duration;       // bitstream duration
```



```
    WMAD_Bool   bHas_DRM;

    WMAD_Bool    bDropPacket;
    WMAD_UINT16  nDRCSetting;
    WMAD_UINT16  nDecoderFlags;
    WMAD_UINT32  nDstChannelMask;
    WMAD_UINT32  nInterpResampRate;
    WMAD_UINT16  nMBRTargetStream;

    WMAD_UINT16 us16NumSamples;
    WMAD_UINT32 us32OutputBufSize;
    WAVEFORMATEXTENSIBLE *pWfx;

} WMADDecoderParams;
```

Step 1: Allocate memory for Decoder parameter structure

The application allocates memory for the structure mentioned below. This structure contains the decoder parameters and memory information structures.

```
/* WMA10 Decoder configuration structure */
typedef struct WMADDecoderConfig
{
      WMADMemAllocInfo        sWMADMemInfo;
      void                    *psWMADecodeInfoStructPtr; // Global_struct

      WMAD_UINT32 (*app_swap_buf) (
                                  void **state,
                                  WMAD_UINT64 offset,
                                  WMAD_UINT32 *pnum_bytes,
                                  unsigned char **ppData,
                                  void *pAppContext,
                                  WMAD_UINT32 *pbIsCompressedPayload);

      void *pContext;
      WMADDecoderParams       *sDecodeParams; //for Decoder Params
} WMADDecoderConfig;
```

Description of the decoder parameter structure *WMADDecoderConfig*

sWMADMemInfo

This is memory information structure. The application needs to call the function, "eWMADQueryMem" to get the memory requirements from decoder. The decoder will fill this structure. This will be discussed in step 2.

psWMADecodeInfoStructPtr

This is a void pointer. This will be initialized by the decoder during the initialization routine. This will then be a used by the decoder and should not be changed by the application.

app_swap_buf

Function pointer to call back function. The application has to initialize this pointer.

pContext: This void pointer would be used to hold the application context which would be used in a multithreaded environment.

sDecodeParams: This is pointer to WMADDecoderParams structure. For raw data decoding, this structure needs to be updated by the application.

Step 2: Get the decoder memory requirements

The WMA10 Decoder does not do any dynamic memory allocation. The application calls the function eWMADQueryMem to get the decoder memory requirements. This function must be called before all other decoder functions are invoked.

The function prototype of eWMADQueryMem is:

C prototype:

```
tWMAFileStatus eWMADQueryMem (WMADDecoderConfig * psDecConfig);
```

Arguments:

- psDecConfig - Decoder configuration pointer.

Return value:

- cWMA_NoErr - Memory query successful.
- Other codes - Error (For other error codes refer to appendix).

This function populates the memory information structure, which is described below

Memory information structure array

```
typedef struct {
      /*  Number of valid memory requests */
      WMAD_INT32  s32WMADNumReqs;
      WMADMemAllocInfoSub sWMADMemInfo [MAX_NUM_MEM_REQS];
} WMADMemAllocInfo;
```

Description of the structure *WMADMemAllocInfo*

s32WMADNumReqs

The number of memory chunks requested by the decoder.

sWMADMemInfo

This structure contains each chunk's memory configuration parameters.

```
typedef struct {
      WMAD_INT32 s32WMADSize;       // Size in bytes
      WMAD_INT32 s32WMADType;       // Memory type Fast or Slow
      WMAD_MEM_DESC   s32WMADMemDesc;
            // to indicate if it is scratch memory
      WMAD_INT32 s32WMADPriority;
            // In case of Fast Memory type, specify the priority
      Void                    *app_base_ptr;
```

```
            // Pointer to the base memory, which will be
            // allocated and filled by application

} WMADMemAllocInfoSub
```

Description of the structure *WMADMemAllocInfoSub*

s32WMADSize

The size of each memory chunk will be in bytes.

s32WMADType

The type of the memory indicates if the requested chunk of memory needs to be allocated in external or internal memory. The type of memory can be SLOW_MEMORY, or external memory, FAST_MEMORY or internal memory. In targets where there is no internal memory, the application can allocate memory in external memory.

(Note: If the decoder request for a FAST_MEMORY for which the application allocates a SLOW_MEMORY, the decoder will still decode, but the performance (MHz) will suffer.)

s32WMADMemDesc

This indicates if the memory chunk is scratch memory

s32Priority

In case, if the decoder requests for multiple memory chunks in the Fast memory, the priority indicates the order in which the application has to prioritize placing the requested chunks in Fast memory.

app_base_ptr

This is pointer to the base memory, which will be allocated and filled by application

Step 3: Allocate Data Memory for the decoder

In this step the application allocates the memory as required by WMA10 Decoder and fills up the base memory pointer 'app_base_ptr' *of* '*WMADMemAllocInfoSub*' structure for each chunk of memory requested by the decoder.

Example pseudo code for the memory allocation and filling the base memory pointer by the application

```
WMADMemAllocInfoSub *mem;
WMADDecoderConfig   *psDecConfig;

/* Number of memory chunks requested by the decoder */
nr = psDecConfig-> sWMADMemInfo. s32NumReqs;

for(i = 0; i < nr; i++)
{
    mem = &(psDecConfig->sWMADMemInfo.sMemInfoSub[i]);

    if (mem->s32WMADType == WMAD_FAST_MEMORY)
    {
        mem->app_base_ptr = malloc(mem->s32WMADSize);
        if (mem->app_base_ptr == NULL)
            /*Error handling */
    }
    else
    {
        mem->app_base_ptr = malloc(mem->s32WMADSize);
        if (mem->app_base_ptr == NULL)
            /*Error handling */
    }
}
```

Step 4: Memory allocation for input buffer

The application has to allocate the memory needed for the input buffer. There is no restriction on the size of the input buffer to be given to the decoder. The recommended minimum size would be 2Kbytes. The decoder, whenever it needs the WMA bit-stream, shall call the function "app_swap_buf" internally from the function "e*WMADecodeFrame"*.
app_swap_buf should be implemented by the application. The application might have different techniques to implement this function. Sample code is given in section 3.1.1 . For Raw Data Decoder support the application has to also allocate sufficient memory needed for input buffer to hold one payload data.

Step 5: Initialization routine

All initializations required for the decoder are done in eInitWMA*Decoder*. This function must be called before the main decoder function is called. The WMA10 Decoder uses packet header in ASF

data to initialize its internal state variables and the decoder parameters. The input buffer pointer and the input buffer length are not used by the decoder but are kept to maintain the original interfaces.

C prototype:

```
tWMAFileStatus eInitWMADecoder (WMADDecoderConfig *psDecConfig,
                                WMADDecoderParams *sDecParams,
                                WMAD_UINT8 *pus8InputBuffer,
                                WMAD_INT32 ps32InputBufferLength);
```

Arguments:

- *psDecConfig* - Pointer to decoder configuration structure.
- *sDecParams* - Pointer to decoder output parameters structure.
- *pus8InputBuffer* - This is not used by the decoder but kept to maintain original interface.
- *ps32InputBufferLength* - This is not used by the decoder but kept to maintain original interface.

Return value:

- cWMA_NoErr - Initialization successful.
- Other codes - Initialization Error

Step 6: Memory allocation for output buffer

The application has to allocate memory for the output buffers to hold the decoded stereo PCM samples for a maximum of one frame size. The pointer to this output buffer needs to be passed to the eWMADecodeFrame function.

Step 7: Call the frame decode routine

The main WMA10 Decoder function is *eWMADecodeFrame*. This function decodes the WMA bit stream in the input buffer to generate one frame of decoder output per channel in every call. The output buffer is filled with interleaved channel PCM samples.

C prototype:

```
tWMAFileStatus eWMADecodeFrame (WMADDecoderConfig *psDecConfig,
                                WMADDecoderParams *sDecParams,
                                WMAD_INT16 *iOUTBuffer,
                                WMAD_INT32 lLength);
```

Arguments:

- *psDecConfig* - Decoder config parameter structure pointer
- *sDecParams* - Decoder parameters structure pointer
- `iOUTBuffer` - Pointer to the output buffer to hold the decoded samples
- `lLength` - Output buffer size in bytes

Return value:

- cWMA_NoErr - Indicates decoding was successful.
- **Others - Indicates error**

Example pseudo code for calling the main decode routine of the decoder

When the decoder encounters the end of bit stream, the application comes out of the loop. In case of error while decoding the current frame, the application comes out of the loop. All errors are non-recoverable.

```
do
{
      iResult = eWMADecodeFrame (psDecConfig, sDec,iOutBuffer,
                                 sDec->us32OutputBufSize);

      Framenum++;

      // Wav file dump
      wfioWrite (pwfioOut, (WMAD_UINT8*) iOutBuffer, sDec->us16NumSamples
      * wfx.Format.nChannels * wfx.Format.wBitsPerSample/8);

      // PCM file dump
      fwrite(iOutBuffer, sDec->us32ValidBitsPerSample/8,
             sDec->us16NumSamples * sDec->us16Channels,
             pfOutPCM);

      if(iResult != cWMA_NoErr)
      {
           if ( iResult == cWMA_NoMoreFrames    ||
                  iResult == cWMA_Failed              ||
                  iResult == cWMA_BrokenFrame )
           {
               iRV = 0;        // normal exit
               break;
           }
           else {
               iRV = 2;        // error decoding data
               fprintf(stderr, "DECODING ERR: decoding failed");
               break;
           }
      }
}while(1);
```

3.1.1 Call back function usage

Call back function is called by the decoder to get a new input buffer for decoding. This function is called by the WMA10 Decoder library when it runs out of current bit stream input buffer. The decoder uses this function to return the used buffer and get a new bit stream buffer. The call back function calls by the decoder will be a function pointer call. This function will be assigned to the pointer app_swap_buf before the init is called. With Raw Data Decoding feature enabled, the *Call back function* would be used to update the payload information.

The interface for this function is described below:

C prototype:

```
WMAD_UINT32 (*app_swap_buf) (
                                    void **state,
                                    WMAD_UINT64 offset,
                                    WMAD_UINT32 *pnum_bytes,
                                    unsigned char **ppData,
                                    void *pAppContext,
                                    WMAD_UINT32 *pbIsCompressedPayload);
```

Arguments:

- *Offset* - Offset address to the input file buffer.
- *pnum_bytes* - This is int pointer, holding number of bytes to be read from the input file buffer to input buffer.
- *ppData* - Pointer to the input buffer.
- *pAppContext* - void pointer to hold the application context

if the Raw Data Decoding feature is enabled the application has to update following arguments

- *pnum_bytes* - This is int pointer, the application has to update content of this pointer with raw payload size for every payload.
- *ppData* - Pointer to raw payload buffer
- *pbIsCompressedPayload* - Pointer to *bIsCompressedPayload,* the application has to set 1 or 0 depending upon the payload is compressed or not (for compressed payload set 1) for every payload

Return value:

- Data being sent back in bytes. With Raw Data Decoding feature enabled the application has to return cWMA_NoErr or cWMA_NoMoreFrames depending up on success.

Example pseudo code

```
WMAD_UINT32 WMAFileCBGetInput(void *state, WMAD_UINT64 offset,
                      WMAD_UINT32 * num_bytes,
                      unsigned char **ppData,
                      void *app_context,
                      WMAD_UINT32 * compress_payload)
{
    WMAD_UINT32 ret;

    struct_callback *pCallback=(struct_callback *) app_context;
#ifdef TEST_PERFORMANCE
    struct_input_data *input_data = pCallback->input_data;
#else
    FILE *fp = pCallback->fp;
#endif


#ifdef TEST_SPEED

    if(offset >= g_cbBuffer)
    {
        *ppData = g_pBuffer + g_cbBuffer;
```

```
        ret = 0;
    }
    else
    {
        *ppData = g_pBuffer + offset;

        if(offset + *num_bytes > g_cbBuffer)
        {
            ret = g_cbBuffer - offset;
        }
        else
        {
            ret = *num_bytes;
        }
    }

#else  /* TEST_SPEED */

    WMAD_UINT32 cbBufferlen         = pCallback->cb_buffer.cbBufferlen;
    WMAD_UINT64 qwBufferOffset      = pCallback-
>cb_buffer.qwBufferOffset;
    unsigned char *     pBuffer           = pCallback->cb_buffer.pBuffer;
    WMAD_INT64 iRelativeOffset      = (WMAD_INT64)offset -
(WMAD_INT64)qwBufferOffset;
    WMAD_UINT32 iBytesRead;

    // Initialize return values
    *ppData = NULL;
    ret = 0;

    // The ANSI-C function fread seems to do fine with files which are >
4GB,
    // but fseek fails miserably. Therefore, never call fseek unless
requested
    // read range is within first 0x7FFFFFFF.

    // Check if start of requested range is within our buffer
    if (iRelativeOffset >= 0 && iRelativeOffset < cbBufferlen)
    {
        if (iRelativeOffset + *num_bytes < cbBufferlen)
        {
            // Entire request is entirely within range of current buffer
            *ppData = pBuffer + iRelativeOffset;
            ret = *num_bytes;
            goto exit;
        }

        // Only the start of request range is in our buffer. Collapse
memory.
        memmove(pBuffer, pBuffer + iRelativeOffset,
                (size_t)(cbBufferlen - iRelativeOffset));
        cbBufferlen -= (WMAD_UINT32)iRelativeOffset;
        qwBufferOffset = offset;
    }
```

```
    else
    {
        // Start of requested range is outside of our buffer
        if (offset + *num_bytes <= 0x7FFFFFFF)
        {
#ifdef TEST_PERFORMANCE
            if (0 != my_fseek (input_data, (WMAD_UINT32)offset,
SEEK_SET))
#else
                if (0 != fseek (fp, (WMAD_UINT32)offset, SEEK_SET))
#endif
                    goto exit;
        }
        else
        {
            // If we reach this point, we can't use fseek

            if (iRelativeOffset < 0)
            {
                // Seeking backwards to > 32-bit offset is not supported
                goto exit;
            }
            // Seek forward to the desired offset.

            iRelativeOffset -= cbBufferlen;
            while (iRelativeOffset > 0)
            {
                WMAD_UINT32 iBytesToRead;


                if (iRelativeOffset > MAX_BUFSIZE)
                    iBytesToRead = MAX_BUFSIZE;
                else
                    iBytesToRead = (WMAD_UINT32)iRelativeOffset;
#ifdef TEST_PERFORMANCE
                iBytesRead = my_fread(pBuffer, 1, iBytesToRead,
input_data);
#else
                iBytesRead = fread(pBuffer, 1, iBytesToRead, fp);
#endif
                if (iBytesRead < iBytesToRead)
                {
                    // Reached EOF before reaching requested offset!
                    goto exit;
                }

                iRelativeOffset -= iBytesRead;
            }
        }

        qwBufferOffset = offset;
        cbBufferlen = 0;
    }
#ifdef TEST_PERFORMANCE
```

```
    iBytesRead = my_fread(pBuffer + cbBufferlen, 1, MAX_BUFSIZE -
cbBufferlen, input_data);
#else
    iBytesRead = fread(pBuffer + cbBufferlen, 1, MAX_BUFSIZE -
cbBufferlen, fp);
#endif
    cbBufferlen += iBytesRead;
    *ppData = pBuffer;

    if(cbBufferlen < *num_bytes)
        ret = cbBufferlen;
    else
        ret = *num_bytes;

#endif /* TEST_SPEED */

exit:
    pCallback->cb_buffer.cbBufferlen = cbBufferlen;
    pCallback->cb_buffer.qwBufferOffset= qwBufferOffset;
    return ret;
}
```

Pseudo code for Callback with Raw Data Decoding Feature

```
tWMAFileStatus WMAFileCBGetNewPayload(void *state, WMAD_UINT64
offset,
                      WMAD_UINT32 * num_bytes,
                      unsigned char **ppData,
                      void *app_context,
                      WMAD_UINT32 * compress_payload)
{
    tWMAFileStatus ret = cWMA_NoErr;
    WMAD_UINT32 iBytesRead;

    struct_callback *pCallback=(struct_callback *) app_context;
#ifdef TEST_PERFORMANCE
    struct_input_data *input_data = pCallback->input_data;
#else
    FILE *fp = pCallback->fp;
#endif
    U8 *pBuffer = pCallback->cb_buffer.pBuffer;

#ifdef TEST_PERFORMANCE
    iBytesRead = my_fread(pBuffer, 1, *num_bytes , input_data);
#else
    iBytesRead = fread(pBuffer, 1, *num_bytes , fp);
#endif
    pCallback->cb_buffer.cbBufferlen = iBytesRead;
    *ppData = pBuffer;

    if(iBytesRead != *num_bytes)
        ret = cWMA_NoMoreFrames;

    return ret;
}
```

3.2 Output version information API function

This function returns the codec library version information details. It can be called at any time and it provides the library's information: Component name, supported ARM family, Version Number, supported OS, build date and time and so on.

C prototype:

```
const WMAD_INT8 * WMA10DoderVersionInfo()
```

Arguments:

- None

Return value:

- const WMAD_INT8 *
  A pointer to the constant char string of the version information.

For Example:

Using this version information API is like below

```
#include "wma10_dec_interface.h"
… …

{
        // Output the WMA 10 Decoder Version Info
        printf("%s \n", WMA10DoderVersionInfo ());
}
```

The output of version information would be like below:
WMA10D_ARM11_02.18.00 build on Oct 15 2008 19:53:25

4 Example calling Routine

The below code snippets may not compile as is and error handling is not extensive.

4.1.1 For ASF Data

```
int main(int argc, char *argv[])
{
    WMARESULT hr = 1; // assume error
    WMADDecoderParams  sDecParams;

    if (argc < 3)
    {
        fprintf(stderr, "** Too few arguments.\n");
        exit(1);
    }

    memset((void *)(&sDecParams), 0, sizeof(sDecParams));

    sDecParams.pus8input_file = argv[1];
    sDecParams.pus8output_file = argv[2];

    sDecParams.bIsRawDecoder = WMAD_FALSE;

    //init decoder parameters
    sDecParams.bDropPacket = 0;
    sDecParams.nDRCSetting = 0;
    sDecParams.nDecoderFlags = 0;
    sDecParams.nDstChannelMask = 0;
    sDecParams.nInterpResampRate = 0;
    sDecParams.nMBRTargetStream = 1;

    CallDecode(&sDecParams);
}

void CallDecode(WMADDecoderParams *sDec)
{
    int iRV = 1;    // assume error exit return value
    short int *iOutBuffer;
    int Framenum=0;
    struct_callback *pCallbackStruct=NULL;

    WMADDecoderConfig  *psDecConfig;
    WMADMemAllocInfoSub *mem;
    WMAD_INT32 i, nr, j;
    tWMAFileStatus iResult;
    int uFileSize, words_read;
    WMAD_INT32 sample_rate;      /* sampling rate */

    WavFileIO *pwfioOut = wfioNew ();
```

```
    WAVEFORMATEXTENSIBLE wfx;

    //Initialize the structure
    memset(&wfx,0,sizeof(wfx));
    sDec->pWfx = &wfx;

    // Init Input structures
    psDecConfig = (WMADDecoderConfig
*)alloc_align(sizeof(WMADDecoderConfig));
    if (psDecConfig == NULL)
        exit(1);

    pCallbackStruct = (struct_callback
*)alloc_align(sizeof(struct_callback));
    if(pCallbackStruct == NULL)
    {
        printf("Error Allocating memory for Callback Structure \n");
        return;
    }
    memset(pCallbackStruct,0,sizeof(pCallbackStruct));

    pCallbackStruct->cb_buffer.cbBufferlen=0;
    pCallbackStruct->cb_buffer.qwBufferOffset=0;

    fp = fopen (sDec->pus8input_file, "rb");

    if (fp == NULL)
    {
        fprintf(stderr, "** Cannot open %s.\n", sDec->pus8input_file);
        exit(1);
    }

    pCallbackStruct->fp = fp;
    pCallbackStruct->cb_buffer.pBuffer = (char
*)alloc_align(sizeof(char)*WMA_MAX_DATA_REQUESTED);
    psDecConfig->app_swap_buf = WMAFileCBGetInput;
    psDecConfig->pContext = (void *)pCallbackStruct;
    psDecConfig->sDecodeParams = sDec;

    /* Query for memory */
    if ((iResult = eWMADQueryMem (psDecConfig)) != cWMA_NoErr)
        exit(1);

    /* Number of memory chunk requests by the decoder */
    nr = psDecConfig->sWMADMemInfo.s32NumReqs;
    for(i = 0; i < nr; i++)
    {
        mem = &(psDecConfig->sWMADMemInfo.sMemInfoSub[i]);

        if (mem->s32WMADType == WMAD_FAST_MEMORY)
        {
            mem->app_base_ptr = alloc_align (mem->s32WMADSize);
            if (mem->app_base_ptr == NULL)
                exit(1);
```

```
        }
        else
        {
            mem->app_base_ptr = alloc_align (mem->s32WMADSize);
            if (mem->app_base_ptr == NULL)
                exit(1);
        }
    }


    printf("\nConverting %s to %s\n",sDec->pus8input_file, sDec-
>pus8output_file);

    iResult = eInitWMADecoder (psDecConfig, sDec, NULL,0);// The last two
parameters are not used, they are kept to maintain the existing
interfaces.

    if(iResult != cWMA_NoErr)
    {
        fprintf(stderr, "** Init of WMA decoder is failed.\n");
        exit(1);
    }

    // Output buffer
    iOutBuffer = (short int *) alloc_align (sDec->us32OutputBufSize);
    if(iOutBuffer == NULL)
    {
        fprintf(stderr, "Outbuffer memory allocation failed.\n");
        exit(1);
    }


    if(wfioOpen (pwfioOut, sDec->pus8output_file, &wfx, sizeof(wfx),
wfioModeWrite) != 0)
    {
        fprintf(stderr, "Can't create file\n");
        exit(1);
    }

    /******************** Decode Call -  ***********************/
    do
    {

        iResult = eWMADecodeFrame (psDecConfig, sDec,iOutBuffer,sDec-
>us32OutputBufSize);

        Framenum++;

        if(iResult != cWMA_NoErr)
        {
            //wchen: I don't understand why failed is normal
            if ( iResult == cWMA_NoMoreFrames   ||
                    iResult == cWMA_Failed                  ||
                    iResult == cWMA_BrokenFrame )
```

```
            {
                iRV = 0;        // normal exit
                break;
            }
            else {
                iRV = 2;        // error decoding data
                fprintf(stderr, "DECODING ERR: decoding failed");
                break;
            }
        }

        wfioWrite (pwfioOut, (WMAD_UINT8*) iOutBuffer, sDec-
>us16NumSamples * wfx.Format.nChannels * wfx.Format.wBitsPerSample/8);
    } while (1);

    printf("Total Frames : %d\n",Framenum);
    printf("Done.....\n\n\n ");

    /* clean up */
    fclose(fp);
    if(pCallbackStruct->cb_buffer.pBuffer)
    {
        fflush (stdout);
        free(pCallbackStruct->cb_buffer.pBuffer);
        pCallbackStruct->cb_buffer.pBuffer = NULL;
    }

    if(pCallbackStruct)
    {
        free(pCallbackStruct);
    }

    if(iOutBuffer)
    {
        free(iOutBuffer);
    }

    for (i = 0; i < nr; i++)
    {
        if(psDecConfig->sWMADMemInfo.sMemInfoSub[i].app_base_ptr)
        {
            free (psDecConfig->sWMADMemInfo.sMemInfoSub[i].app_base_ptr);
        }
    }

    if(psDecConfig)
    {
        free (psDecConfig);
    }

    if (pwfioOut)
    {
        wfioClose(pwfioOut);
        wfioDelete (pwfioOut);
```

```
    }
}
```

4.1.2 For Raw Data

```
/*********************** Main ************************/

int main(int argc, char *argv[])
{
int               iResult = 0;
int                packet_size,count;
static packet_count = 0;
VideoStreamInfo  *v_info;
asf_parser_content_descriptor        *info_meta;
asf_parser_file_properties *f_prop;
simple_index_object *ind_test;
WMADDecoderParams  sDecParams;

int iRV = 1;    // assume error exit return value
short int *iOutBuffer;

WMADDecoderConfig  *psDecConfig;
WMADMemAllocInfoSub *mem;
WMAD_INT32 i, nr;
tWMAFileStatus iRslt;

WavFileIO *pwfioOut = wfioNew (); /* For output */
WAVEFORMATEXTENSIBLE wfx;

/* Allocate memory for parser structures - stream info, descriptor, file
properties */
a_info    = (AudioStreamInfo*)
asf_parser_allocate_mem(sizeof(AudioStreamInfo));
info_meta = (asf_parser_content_descriptor*)
asf_parser_allocate_mem(sizeof(asf_parser_content_descriptor));
op_stream = (asf_parser_stream *)
asf_parser_allocate_mem(sizeof(asf_parser_stream));
f_prop    = (asf_parser_file_properties *)
asf_parser_allocate_mem(sizeof(asf_parser_file_properties));
ind_test  = (simple_index_object *)
asf_parser_allocate_mem(sizeof(simple_index_object));


/* Initialize the output file */
iResult = Open_out_file(argv[1]);

sDecParams.pus8output_file = argv[2];

/* Initialize the stream info and get the stream properties */
mhDecoder = asf_parser_init(get_data);
asf_parse_index(mhDecoder, ind_test);
asf_parse_get_content_descriptors(mhDecoder,info_meta);
asf_parse_get_audio_stream_info(mhDecoder,a_info);
```

```
asf_parse_get_file_properties(mhDecoder,f_prop);

/* Initialize the payload information */
for(count = 0;count<ASF_MAX_PAYLOADS;count++)
{
      op_stream->raw_streams[count].payload = NULL ;
}

op_stream->num_payloads = 0;
packet_size = f_prop->packet_size;


psDecConfig = (WMADDecoderConfig*)alloc_align(sizeof(WMADDecoderConfig));

//Get Decoder Parameters
GetDecoderParams(a_info,&sDecParams);

/*************** Query for memory *************/
if ((iRslt = eWMADQueryMem (psDecConfig)) != cWMA_NoErr)
        exit(1);

/* Allocate the number of memory chunk requests by the decoder */
nr = psDecConfig->sWMADMemInfo.s32NumReqs;
for(i = 0; i < nr; i++)
{
     mem = &(psDecConfig->sWMADMemInfo.sMemInfoSub[i]);
     mem->app_base_ptr = alloc_align (mem->s32WMADSize);
}

/*************** WMADecoder Initialization *************/
psDecConfig->app_swap_buf = WMADGetNewPayload;

//Initialize the wave format structure
memset(&wfx,0,sizeof(wfx));
sDecParams.pWfx = &wfx;

iRslt = eInitWMADecoder (psDecConfig, &sDecParams, NULL,0);

// Output buffer
iOutBuffer = (short int *) alloc_align (sDec->us32OutputBufSize);

if(wfioOpen (pwfioOut, sDecParams.pus8output_file, &wfx, sizeof(wfx),
wfioModeWrite) != 0)
{
        exit(1);
}

/******************** Decode Call ***********************/

/* Keep decoding until error or No more frames */
do
{
iRslt = eWMADecodeFrame (psDecConfig, sDecParams, iOutBuffer,
                         sDecParams.us32OutputBufSize );
```

```
/* Dump the output */
wfioWrite (pwfioOut, (WMAD_UINT8*) iOutBuffer, sDecParams.us16NumSamples
* wfx.Format.nChannels * wfx.Format.wBitsPerSample/8);

if(iRslt != cWMA_NoErr)
{
      if ( iRslt == cWMA_NoMoreFrames     ||
           iRslt == cWMA_Failed                ||
           iRslt == cWMA_BrokenFrame )
          {
            iRV = 0;        // normal exit
            break;
          }
          else
          {
            iRV = 2;        // error decoding data
            break;
         }
}
} while (1);

/******************** Decode Call End ***********************/

/* clean up */

/* Free up all the used resources of the decoder and the parser */

/* close the parser*/
asf_parser_close(mhDecoder);

return iResult;
}
```