



08-6369-SIS-ZCH66
10-AUG-2009
3.2

Application Programmers Interface for WMV9 Decoder

ABSTRACT:

Application Programmers Interface WMV9 Decoder

KEYWORDS:

Multimedia codecs, WMV9, Windows Media Video

APPROVED:

Wang Zening

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	04-Nov-2004	Debashis Sarkar	Initial Draft
0.2	13-Dec-2004	Debashis Sarkar	Updated after internal discussions
0.3	16-Dec-2004	Debashis Sarkar	Updated after review comments
0.4	29-Jan-2005	Debashis Sarkar	Updated after PCS review comments
0.5	06-May-2005	Anurag Goel	Updated performance numbers
1.0	26-May-2005	Prachi Bhatkar	Updated performance numbers
2.0	06-Feb-2006	Lauren Post	Using new format
2.1	31-Mar-2006	Prachi	Updated
2.2	11-Jan-2007	Abhishek	Updated the APIs to provide Padded or Non-Padded data to the application
2.3	22-May-2007	Anirudh R	Added new API and updated return parameter for Key frame seek.
3.0	07-Nov-2007	Ding Qiang	Add Direct Rendering module.
3.1	31-Mar-2008	Ellick Chen	Updated Direct Rendering API
3.2	10-Aug-2009	Eagle Zhou	Add feature to drop B frames

Table of Contents

Introduction	5
1.1 Purpose	5
1.2 Scope	5
1.3 Audience Description	5
1.4 References	5
1.4.1 Standards	5
1.4.2 References	5
1.4.3 Freescale Multimedia References	5
1.5 Definitions, Acronyms, and Abbreviations	6
1.6 Document Location	6
2 API Description	7
2.1 Data structures	7
Basic data types	7
sWmv9DecObjectType	7
Buffer Manager	8
sWmv9DecMemAllocInfoType	8
sWmv9DecMemBlockType	9
sWmv9DecParamsType	9
sWmv9DecYCbCrBuferType	10
2.2 Enumerations and macros	12
2.2.1 Library API return codes	12
2.2.2 Memory alignment	12
2.2.3 VOP types	13
2.2.4 Compression types	13
2.2.5 Frame manager types	13
2.2.6 Output Format	14
2.2.7 Memory types	14
2.2.8 Buffer Releaser	15
2.2.9 Buffer getter	15
2.2.10 Buffer Rejecter	15
2.3 Application programmer interface functions	16
2.3.1 Query memory	16
2.3.2 Initialization	16
2.3.3 Enable Skip to Next I Mode	17
2.3.4 Enable Skip B frames	17
2.3.5 Decode a frame	17
2.3.6 Get output frame	18
2.3.7 Free decoder	18
2.3.8 Input buffer callback interface	19
2.3.9 Debug logging	19
2.3.10 Set Buffer Manager	21
2.3.11 Set Additional Callback Function	21
3 Example library usage	22

Introduction

This document describes the application programmer's interface of WMV9.0 video decoder library and its usage. The interface assumes that the input is not a bare WMV9 video bit stream, and relevant information from ASF format is available with the application.

1.1 Purpose

This document gives the application programmer's interface to WMV9 decoder library. End of the document contains an example application written using these APIs.

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 decoder library.

1.3 Audience Description

The reader is expected to have basic understanding of video processing and windows media video coding standard.

1.4 References

1.4.1 Standards

- WMV Version 9.0, Windows Media Video V9 Decoding Specification, revision 87
- WMT Version 9.0, Functional Specification, Recommended Media Decoding – rev 8.1.

1.4.2 References

- Arm codec coding guidelines
- Advanced System format (ASF) Specification, Revision 01.20.02, Microsoft Corporation, June 2004

1.4.3 Freescale Multimedia References

- WMV9 Decoder Requirements Book - wmv9_dec_reqb.doc
- WMV9 Decoder Test Plan - wmv9_dec_test_plan.doc
- WMV9 Decoder Release notes - wmv9_dec_release_notes.doc
- WMV9 Decoder Test Results – wmv9_dec_test_results.doc
- WMV9 Decoder Interface Header – wmv9mp_dec_api.h
- WMV9 Decoder Application Code – wmv9_testapp.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
AVC	Advanced Video Coding
API	Application Programming Interface
ARM	Advanced RISC Machine
ASF	Advanced System Format,
FSL	Freescale
ISO	International Standards Organization
ITU	International Telecommunication Union
MPEG	Moving Pictures Expert Group
NAL	Network Abstraction Layer
RVDS	RealView Development Suite
SP	Simple Profile
RVDS	RealView Development Suite
TBD	To Be Determined
UNIX	Linux PC x/86 C-reference binaries
WMV	Windows Media Video

1.6 Document Location

docs/wmv9mp_dec

2 API Description

This section describes the data structures, enumerations and macros used in the WMV9 decoder library interface.

2.1 Data structures

This section describes the data structures used in the decoder interface. The basic data type definitions are given in section .

Basic data types

These are the typedefs for the basic data types.

```
typedef void          WMV9D_Void ;
typedef int           WMV9D_S32 ;
typedef unsigned int  WMV9D_U32 ;
typedef short int     WMV9D_S16 ;
typedef short unsigned int WMV9D_U16 ;
typedef char          WMV9D_S8 ;
typedef unsigned char WMV9D_U8 ;
```

sWmv9DecObjectType

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
{
    sWmv9DecMemAllocInfoType  sMemInfo;
    sWmv9DecParamsType        sDecParam;
                                WMV9D_Void *pvWmv9Obj;
                                WMV9D_Void *pvBitBuffer;
                                WMV9D_Void *pvAppContext;
    FpWmv9DecReadCbkJType     pfCbkJBuffRead;
    WMV9D_FrameManager        frameManager;
} sWmv9DecObjectType;
```

Description of structure *sWmv9DecObjectType*

sMemInfo

This is memory information structure. This is further described later.

sDecParam

The output of the decoder is encapsulated in this structure. This is further described later.

pvWmv9Obj

This is an internal video object context for the decoder and application should not change this.

pvBitBuffer

This buffer is used for holding the bit stream data inside the library.

pvAppContext

This space is provided for the application to keep its context and the decoder does not

modify it. It is passed in the call back function to get the bit stream data.

pfCbkBuffRead

Function to be used for reading the bit stream data. For more information see section 2.3.8.

frameManager

Frame buffer manager, it will be set by the framework.

Buffer Manager

For clarifying the concept and simplifying the API, we group the 2 function pointer we described above into a structure named DR_BufferManager:

```
typedef struct _WMV9D_FrameManager
{
    bufferGetter      BfGetter;
    bufferRejecter    BfRejector;
} WMV9D_FrameManager;
```

sWmv9DecMemAllocInfoType

This structure holds the memory block allocation information for decoder library. The decoder memory requirements are given to the application when *eWMV9DQuerymem* is called. The decoder specifies number of memory blocks needed by filling *s32NumReqs* in this structure. For each memory block, all parameters are set in *asMemBlks* array based on decoder requirement. Application shall allocate the memory required by looking at this structure before initializing the decoder.

```
typedef struct
{
    WMV9D_S32          s32NumReqs;
    sWmv9DecMemBlockType asMemBlks [WMV9D_MAX_NUM_MEM_REQS];
    WMV9D_S32          s32MinFrameBufferNum;
} sWmv9DecMemAllocInfoType;
```

Description of the structure *sWmv9DecMemAllocInfoType*

s32NumReqs

Number of memory blocks required. Decoder will set this to required value when *eWmv9DQuerymem* function is called.

asMemBlks

Array of memory block structure. For each request defined in *s32NumReqs* application should allocate the memory. To see how this information can be used, please see (Example library usage).

s32MinFrameBufferNum

The minimum frame buffer number, it is set when querying memory process.

WMV9D_MAX_NUM_MEM_REQS is the maximum number of memory block requests the decoder can make. WMV9D_MAX_NUM_MEM_REQS has been set to 256.

sWmv9DecMemBlockType

This describes the memory block requirement details such as size, type etc. The application shall allocate memory depending on the requirement and set the pointer in the space provided. The library uses the memory given by the application.

```
typedef struct
{
    WMV9D_S32      s32Size;
    WMV9D_S32      s32MemType;
    WMV9D_S32      s32Priority;
    eWmv9DecMemAlignType  eMemAlign;
    WMV9D_S32      s32OldSize;
    WMV9D_S32      s32MaxSize;
    WMV9D_Void      *pvUnalignedBuffer;
    WMV9D_Void      *pvBuffer;
} sWmv9DecMemBlockType;
```

Description of the structure *sWmv9DecMemBlockType*

s32Size

The actual size of the memory required (populated by the decoder).

s32MemType

The type of memory needed. It can be one of slow/fast along with scratch/static memory. Please see the section Memory types to get the possible type of values (populated by the decoder).

s32Priority

It suggests the importance of this memory block. Higher the impact of the speed of this memory on the decoder performance, lower will be the value (>0). Priorities are neither unique nor continuous (populated by the decoder).

eMemAlign

Required alignment of the memory block. The values are defined in the Memory alignment section.(populated by the decoder).

s32OldSize

Size of the block in the last allocation. It is useful only if the block is resized.

s32MaxSize

This gives the maximum size that can be requested for this type. This is fixed on compile time, based on the maximum size supported. It is only useful if the application does not want to reallocate memory.

pvUnalignedBuffer

This storage is provided to keep track of the unaligned buffer address (by the application). This is not used by the decoder.

pvBuffer

This will be updated by the application based on the memory requirement. The decoder assumes that it contains a valid value of a buffer for the required size, type and alignment.

sWmv9DecParamsType

This defines the data structure for the decoder parameters. This structure is used to exchange multiple parameters between application and decoder.

```
typedef struct
{
```

```

    sWmv9DecYCbCrBufferType  sOutputBuffer;
    eWmv9DecCompFmtType      eCompressionFormat;
                               WMV9D_S32    s32FrameRate;
                               WMV9D_S32    s32BitRate;
    WMV9D_U16                 u16FrameWidth;
    WMV9D_U16                 u16FrameHeight;
                               WMV9D_U32    u32PrevFrameNum;
                               WMV9D_U32    u32CurrFrameNum;
    eWmv9DecVOPType          eVopType;
} sWmv9DecParamsType;

```

Description of the structure *sWmv9DecParamsType*

sOutputBuffer

Decoded output is stored in this structure.

eCompressionFormat

Compression format used to encode the sequence. This is set by the application.

s32FrameRate

Frame rate of the bit stream to be decoded (if not known, set it to 0)

s32BitRate

Bit rate of the bit stream to be decoded (if not known, set to 0).

u16FrameWidth

Width of the frame.

u16FrameHeight

Height of the frame.

u32PrevFrameNum

Assumed frame number of the previous frame. This is used to calculate the current frame number. For proper usage of this field, please see the Example library usage. This is set to u32CurFrameNum after decoding a frame, so that the application need not set it, unless there is a skip.

u32CurrFrameNum

Frame number of the current decoded frame, calculated from the u32PrevFrameNum, and number of frames skipped in the encoder. For proper usage of this field, please see the Example library usage.

eVopType

Type of the VOP of the last decoded frame.

sWmv9DecYCbCrBuferType

This structure encapsulates the YCbCr buffer, which stores the decoded frame in YUV 4:2:0 format, non-interleaved. The buffer addresses and row sizes shall be set by the decoder in *eWMV9DDecode* function. The values can be changed after each call of *eWMV9DDecode* function. As the pointers point inside the decoder data structure, application shall not modify the pointers or the content.

```

typedef struct
{
    const          WMV9D_U8    *pu8YBuf;
    const          WMV9D_U8    *pu8CbBuf;
    const          WMV9D_U8    *pu8CrBuf;
                   WMV9D_S32    s32YRowSize;
                   WMV9D_S32    s32CbRowSize;
                   WMV9D_S32    s32CrRowSize;
}

```

```

    tVideoFormat_WMC    tOutputFormat;
} sWmv9DecYCbCrBufferType;

```

Description of the structure *sWmv9DecYCbCrBufferType*

pu8YBuf

Buffer pointer to decoded frames Y data.

pu8CbBuf

Buffer pointer to decoded frames Cb data.

pu8CrBuf

Buffer pointer to decoded frames Cr data.

s32YRowSize

Offset in bytes between start pixels of two consecutive rows of Y.

s32CbRowSize

Offset in bytes between start pixels of two consecutive rows of Cb.

s32CrRowSize

Offset in bytes between start pixels of two consecutive rows of Cr.

tOutputFormat

Specifies the format of output data (YUV_420)

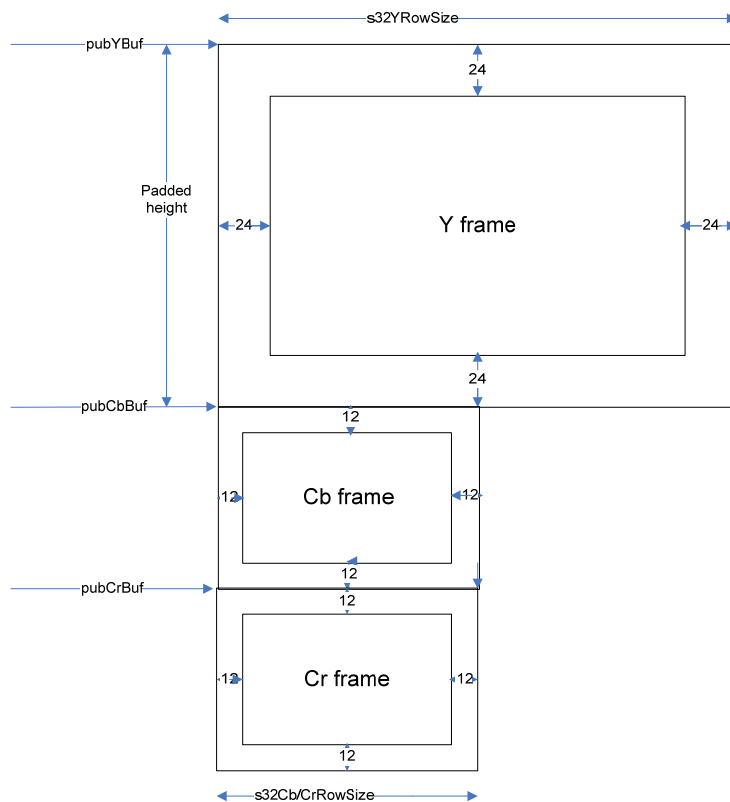


Figure 1: Storage format and pointers for one padded output frame

2.2 Enumerations and macros

2.2.1 Library API return codes

This enum holds the return types of the APIs.

```
typedef enum
{
    /* Successfull return values */
    E_WMV9D_SUCCESS = 0,                /* Successful completion*/

    /* Successful return with a warning, no action needs to be taken */
    E_WMV9D_ERROR_CONCEALED = 11,        /* Error in the bit stream,
    but concealed */

    E_WMV9D_ENDOF_BITSTREAM,             /* End of Bit Stream*/
    E_WMV9D_SKIPPED_FRAME,               /* The frame has been skipped */

    /* Successful return with a warning, correct the situation and
    continue */
    E_WMV9D_NOT_ENOUGH_BITS=31,          /* Not enough bits are
    provided */
    E_WMV9D_BAD_MEMORY,                 /* Out of memory*/
    E_WMV9D_WRONG_ALIGNMENT,            /* Incorrect memory alignment */

    E_WMV9D_SIZE_CHANGED,               /* Image size changed*/
    E_WMV9D_NO_OUTPUT,                 /* No output frame is
    available */
    E_WMV9D_BROKEN_FRAME,               /* Should have more data in
    this frame*/

    /* irrecoverable error type, may need re-initialization to continue
    */
    E_WMV9D_CORRUPTED_BITS=51,          /* Error bit stream*/
    E_WMV9D_FAILED,                    /* Failure*/
    E_WMV9D_UNSUPPORTED,               /* Unsupported format*/
    E_WMV9D_NO_KEYFRAME_DECODED,       /* first frame is not an I
    frame */
    E_WMV9D_SIZE_NOT_FOUND,            /* Frame size not found in
    bit stream*/
    E_WMV9D_NOT_INITIALIZED,           /* Decoder is not initialized */

    E_WMV9D_INVALID_ARGUMENTS,         /* Argument to the API is
    invalid */
    E_WMV9D_SET_CB_OK,                 /*!< Decoder Register call
    back OK */
    E_WMV9D_SET_CB_FAIL               /*!< Decoder Register call
    back FAIL */
} eWmv9DecRetType;
```

2.2.2 Memory alignment

This enumeration defines the memory alignment type.

```
typedef enum
{
    E_WMV9D_ALIGN_NONE = 0,            /* buffer can start at any place */
    E_WMV9D_ALIGN_HALF_WORD,          /* start address's last bit has
    to be 0 */
}
```

```

    E_WMV9D_ALIGN_WORD,          /* start address's last 2 bits
    has to be 0                    */
    E_WMV9D_ALIGN_DWORD,         /* address's last 3 bits has to
    be 0                           */
    E_WMV9D_ALIGN_QWORD,         /* address's last 4 bits has to
    be 0                           */
    E_WMV9D_ALIGN_OCTAWORD       /* address's last 5 bits has to
    be 0                           */
} eWmv9DecMemAlignType;

```

2.2.3 VOP types

This enumeration describes the encoding type of the VOP last decoded.

```

typedef enum
{
    E_WMV9D_INTRA_VOP,           /* Intra VOP or I-VOP*/
    E_WMV9D_INTER_VOP,          /* Inter VOP or P-VOP*/
    E_WMV9D_BIDIR_VOP,          /* Bidirectional VOP or B-VOP*/
    E_WMV9D_UNKNOWN_VOP,       /* Unknown, should not happen*/
} eWmv9DecVOPType;

```

2.2.4 Compression types

The following enumeration is used to specify the compression type that is supported by the decoder.

```

typedef enum
{
    E_WMV9D_COMP_FMT_WMV9,
    E_WMV9D_COMP_FMT_WMV8,
    E_WMV9D_COMP_FMT_WMV7,
    E_WMV9D_COMP_FMT_UNSUPPORTED
} eWmv9DecCompFmtType;

```

To facilitate the conversion of ASF file compression type to the required enumeration, library also provides a utility function to convert a string to the compression type.

eWmv9DecCompFmtType eWMV9DCompFormat (const WMV9D_U8 format);*

It returns the compression format enumeration from the string used in the ASF file format.

2.2.5 Frame manager types

The following enumeration is used to specify the manager type that is supported by the decoder.

```

typedef enum
{
    E_RELEASE_FRAME = 0,
    E_REJECT_FRAME,
    E_GET_FRAME
} eWmv9CallbackType;

```

add this to indicate additional callback function type .

2.2.6 Output Format

```
typedef enum
{
    YUV_420=0,
    YUV_420_PADDED,
    UYVY_WMV
} tVideoFormat_WMC;
```

Specifies the format of the output data.

2.2.7 Memory types

The following defines specifies the memory types of the memory blocks requested by the decoder. As various properties are combined in single variable, please use the macros provided to extract the required value.

Speed of memory block

WMV9D_SLOW_MEMORY, WMV9D_FAST_MEMORY

Usage of memory block over API calls

WMV9D_STATIC_MEMORY, WMV9D_SCRATCH_MEMORY

Whether there is any size dependency, and if it is, whether we need to copy old data on resizing.

WMV9D_SIZE_DEPENDENT, WMV9D_SIZE_CHANGED, WMV9D_COPY_AT_RESIZE

To specify whether it contains output data

WMV9D_OUTPUT_MEMORY

The memory type will be a combination of these properties. Do not check the value directly, use the macros defined below instead.

WMV9D_IS_FAST_MEMORY(memType) Returns non zero, if memory type is fast.

WMV9D_IS_SLOW_MEMORY(memType) Returns non-zero, if memory type is slow.

WMV9D_IS_STATIC_MEMORY(memType) Returns non-zero, if memory type is static.

WMV9D_IS_SCRATCH_MEMORY(memType) Returns non-zero, if memory type is scratch.

WMV9D_IS_SIZE_DEPENDENT_MEMORY(memType)

Returns non-zero, if buffer is frame size dependent.

WMV9D_IS_SIZE_CHANGED(memType)

Returns non-zero, if buffer size needs to be changed, since the last allocation happened.

WMV9D_COPY_AT_RESIZE(memType)

Returns non-zero, if memory content has to be copied when the buffer is resized.

WMV9D_IS_OUTPUT_MEMORY(memType)

Returns non-zero, if memory content is output data.

2.2.8 Buffer Releaser

Since the Direct Rendering is adopted, the decoder will tell the APP that some frame buffers will not be used as reference frame. A function which is implemented by the application (frame work) will be used to release a frame buffer.

This function's prototype is listed in the wmv9mp_dec_api.h:

Prototype:

```
typedef void (*bufferReleaser)(void*, void*);
```

Arguments:

- void

Return value:

- A frame buffer

2.2.9 Buffer getter

Since the Direct Rendering is adopted, the decoder will ask to get a new buffer for decoding. A function which is implemented by the application (frame work) will be used to perform getting frame buffer.

This function's prototype is listed in the wmv9mp_dec_api.h:

Prototype:

```
typedef void* (*bufferGetter)( void);
```

Arguments:

- void

Return value:

- A frame buffer

2.2.10 Buffer Rejecter

Since the Direct Rendering is adopted, the decoder will ask to get a new buffer for decoding.

It's possible that the gotten frame buffer may be refused by the decoder. Decoder need to inform the application (framework) that this frame is rejected.

A function which is implemented by the application (frame work) will be used to perform reject ion of a frame buffer.

This function's prototype is listed in the wmv9mp_dec_api.h:

Prototype:

```
typedef void (*bufferRejecter)(void*, void*);
```

Arguments:

- A rejected frame buffer

Return value:

- None

2.3 Application programmer interface functions

2.3.1 Query memory

This is the first function to be called by the application. This function returns the memory requirement for the decoder. The decoder will populate *psWmv9Obj->sMemInfo* structure based on width and height of the frame. The application shall use this information to allocate the requested memory blocks and set the pointers of *asMemBlks* in *psWmv9Obj->sMemInfo* structure to the memory block of required size, type & alignment.

C prototype:

```
eWmv9DecRetType eWMV9DQuerymem ( sWmv9DecObjectType *psWmv9Obj,
                                   WMV9D_S32 s32Height,
                                   WMV9D_S32 s32Width);
```

Arguments:

- *psWmv9Obj* - Decoder object pointer
- *s32Height* - Height of the frame size of the sequence to be decoded
- *s32Width* - Width of the frame size of the sequence to be decoded

Return value:

eWmv9DecRetType Tells whether the decoder was successful to set the parameters needed for memory allocation.

Return values are -

- | | |
|------------------------|------------------------|
| <i>E_WMV9D_SUCCESS</i> | - Function successful. |
| Other values | - Error |

2.3.2 Initialization

All initializations required for the decoder are done in *eWMV9DInit*. This function must be called before main decoder functions are invoked. This function also parses the sequence header, and uses the call back function to get the required data.

C prototype:

```
eWmv9DecRetType eWMV9DInit (sWmv9DecObjectType *psWmv9Obj);
```

Arguments:

- *psWmv9Obj* - Decoder object pointer.

Return value:

eWmv9DecRetType Tells whether decoder has been successfully initialized or not.

Return values are -

- | | |
|------------------------|------------------------|
| <i>E_WMV9D_SUCCESS</i> | - Function successful. |
| Other values | - Error |

2.3.3 Enable Skip to Next I Mode

This API enables the ‘skip to next I mode’ in the decoder. Refer to Decode a frame

C prototype:

```
eWmv9DecRetType eWMV9DEnableSkipMode (sWmv9DecObjectType *psWmv9Obj);
```

Arguments:

- psWmv9Obj - Decoder object pointer.

Return value:

eWmv9DecRetType Tells whether frames were decoded successfully or not.

Return values are:

- | | |
|----------------|------------------------|
| E_WM9D_SUCCESS | - Function successful. |
| Other values | - Error |

2.3.4 Enable Skip B frames

This API enables or disables the ‘skip B frames’ in the decoder.

C prototype:

```
eWmv9DecRetType eWMV9DEnableSkipBMode (sWmv9DecObjectType *psWmv9Obj, int flag);
```

Arguments:

- psWmv9Obj - Decoder object pointer.
- Flag - 0: disable skipping B frames, otherwise, enable skipping B frames

Return value:

eWmv9DecRetType Tells whether setting is successful or not.

Return values are:

- | | |
|----------------|------------------------|
| E_WM9D_SUCCESS | - Function successful. |
| Other values | - Error |

2.3.5 Decode a frame

This is the frame decoding function. It decodes the WMV bit stream to generate one frame of decoder output in every call. When running in ‘skip to next I mode’ the decoder skips all the non-key frames until it finds a key frame, after which it decodes the key frame and disables the ‘skip to next I mode’.

C prototype:

```
eWmv9DecRetType eWMV9DDecode (sWmv9DecObjectType *psWmv9Obj, 0
                               WMV9D_U32 u32FrameDataSize);
```

Arguments:

- psWmv9Obj - Decoder object pointer.
- u32FrameDataSize - Bit stream data size in bytes for this frame (or sequence header)

Return value:

eWmv9DecRetType Tells whether frames were decoded successfully or not.

Return values are:

- | | |
|-----------------------|------------------------------|
| E_WMV9D_SUCCESS | - Function successful. |
| E_WMV9D_SKIPPED_FRAME | - The frame has been skipped |
| Other values | - Error |

2.3.6 Get output frame

This API gives the recently decoded frame output. In case of YUV_420 output format,

- It copies decoded video frame (after removing padded data) into a buffer allocated by the application.
- The pointers to the buffer allocated by the application must be populated on the structure members of *sWmv9DecYCbCrBufferType* (pu8YBuf, pu8CbBuf and pu8CrBuf).

In case of YUV_420_PADDED output format, this API sets the pointers pu8YBuf, pu8CbBuf and pu8CrBuf to point to the padded video frame buffers. This will be helpful if the application can handle removal of padded data using hardware.

C Prototype:

```
eWmv9DecRetType eWmv9DecGetOutputFrame (sWmv9DecObjectType *psWmv9Obj);
```

Arguments:

- psWmv9Obj - Decoder object pointer.

Return value:

eWmv9DecRetType Tells whether the output data is available.

Return values are -

- | | |
|-----------------|------------------------|
| E_WMV9D_SUCCESS | - Function successful. |
| Other values | - Error |

2.3.7 Free decoder

This is the decoder function to release any resources used by the decoder. It doesn't free memory that has been allocated by the application on behalf of the decoder.

Prototype:

```
eWmv9DecRetType eWMV9DFree (sWmv9DecObjectType *psWmv9Obj);
```

Arguments:

- psWmv9Obj - Decoder object pointer.

Return value:

eWmv9DecRetType - Tells whether memory was freed successfully or not.

Return values are -

- E_WMV9D_SUCCESS - Function successful.
- Other values - Error

2.3.8 Input buffer callback interface

Input buffer callback function should be a synchronous function used by the decoder to read the portion of bit stream from the application. This function is called by the decoder in *eWMV9DInit* and *eWMV9DDecode* functions, when it needs the bit stream data. The decoder assumes that the application knows the end of the bit stream data for the current frame, and data returned is for current frame only. While initializing, only sequence header data should be returned.

This function is not a part of the library and has to be implemented by the application, user of the decoder library. Application developer has complete control on implementing this function based on his/her system requirements. The name of the function given is only for illustration purpose, and the application developer can name the function as he/she likes. The decoder cannot continue till this function returns with the required data. For example usage, see section Example library usage.

C prototype:

```
int cbkWMV9BuffRead (int s32BufLen, unsigned char *pu8Buf, int
                    *bEndOfFrame, void * pvAppContext);
```

Arguments:

- *s32BufLen* Length of the buffer provided.
- *pu8Buf* Pointer to the buffer.
- *bEndOfFrame* Set to 1, if no more data for the current frame (or sequence header) is available.
- *pvAppContext* Caller context, as set while initializing the decoder. It can be used to distinguish between calls from different decoding threads in multi-threaded environment.

Return value:

Returns the size of the buffer copied else return -1.

Note:

The function *cbkWMV9BuffRead* is in application space and can have any name. The function pointer in the *sWmv9DecObject* structure has to be initialized to this function.

sWmv9DecObject.pfCbkbuffRead = *cbkWMV9BuffRead*

2.3.9 Debug logging

Debug logs for the decoder can be enabled by setting the debug levels appropriately. They are described in *debug.h* file. The logs can be extracted with various debug level enabled depending on the need. Here are the list of levels and its meaning. These settings are compilation time options.

Table 1 Different debug levels

DEBUG_1	Data that occurs once per bitstream
DEBUG_2	Data that occurs once per VOP
DEBUG_3	Data that occurs once per MB
DEBUG_4	Data that occurs once per block

The debug log interface functions are defined in the `log_api.h` file, which is supplied with the library. The application should implement the specific logging functionality required, according to the interface.

2.3.9.1 Initializing debug log module

Prototype

```
int DebugLogInit (void/* any parameter defined by the application*/);
```

2.3.9.2 Logging a data value

This function is used by the decoder to log a data value. Return value is ignored.

C prototype

```
int DebugLogData (short int msgid, void *ptr, int size);
```

Arguments

- *msgid* Message id for the particular codec/data type.
- *ptr* Memory where the data is present.
- *size* Data size in bytes.

2.3.9.3 Logging a text message

This function is used to log a text message, similar to `printf` format. Return value is ignored.

C prototype

```
int DebugLogText (short int msgid, char *fmt, ...);
```

Arguments

- *msgid* Message id for the particular codec/message type.
- *fmt* Format of the message, similar to `printf` function.
- ... Any other arguments (possibly none) that the format specifies.

2.3.9.4 Closing debug logs

This function closes the debug logs. The function shall be called by the application after closing the decoder.

C prototype

```
int DebugLogClose();
```

2.3.10 Set Buffer Manager

Set the Frame manager to decoder. It will be called after querying memory and before decode init.

C prototype

```
extern void WMV9DSetBufferManager (sWmv9DecObjectType *psWmv9Obj,
WMV9D_FrameManager* manager);
```

Arguments

- *psWmv9Obj* WMV9 Decoder object
- *manager* Frame buffer manager

return value

null

2.3.11 Set Additional Callback Function

Set the Frame manager to decoder via callback function. It will be called after decode initialization.

At current version, only bufferReleaser need to be used via this callback function.

C prototype

```
extern eWmv9DecRetType eWMV9DSetAdditionalCallbackFunction
(sWmv9DecObjectType *psWmv9Obj, eWmv9CallbackType funcType, void* cbFunc);
```

Arguments

- *psWmv9Obj* WMV9 Decoder object
- *funcType* Frame buffer manager type
- *cbFunc* Frame buffer manager function

Return value:

eWmv9DecRetType -Tells whether the callback function was registered successfully or not.

Return values are -

- | | |
|----------------------------|------------------------|
| <i>E_WMV9D_SET_CB_OK</i> | - Function successful. |
| <i>E_WMV9D_SET_CB_FAIL</i> | - Error |

3 Example library usage

This example shows how to use the WMV9 decoder library. Here are the steps involved in a typical usage.

1. Get the memory requirement
2. Allocate the memory as required by the decoder.
3. Initialize the decoder
4. Allocate memory as per the requirements.
5. While there are more frames, decode and display the decoded frames.
6. Close the decoder.
7. Free any memory allocated for the decoder and application.

The encoded bit stream is fed through *cbkWMV9DBufRead* function. Please note that the error handling is not shown properly and the code may not compile as it is. Also, at few places only one field of a structure is set, while the application needs to set all the fields.

```

/***** portion of the application code *****/

sWmv9DecObject      sWmv9DecObj;   /* instance of a decoder */

/* Call QueryMem to get the size and the type of the memory needed by
the decoder */
error = eWMV9DQuerymem (&sWmv9DecObj, frameHeight, frameWidth);

/* Give memory to the decoder for the size, type and alignment
returned */
AllocateMemory (&sWmv9DecObj);

/* Set the call back function and app context, as Init uses these */
sWmv9DecObj.pfCbkmv9BufRead = cbkWMV9DBufRead;
sWmv9DecObj.pvAppContext = (void*)&sWmv9DecObj; /* or anything you
need */

/* Initialize the decoder. */
error = eWMV9DInit (&sWmv9DecObj);

/* Decode the bit stream and produce the outputs */
while (1) {

    if(SkiptoNextIVOP)
    {
        error= eWMV9DEnableSkipMode (psWmv9Obj);
        if (E_WMV9D_SUCCESS == error)
            SkiptoNextIVOP = 0;

    }

    /* Decode one frame.*/

```

```

error = eWMV9DDecode (psWmv9DecObj, s32NumBytes);
/* if success get the output*/
if (E_WMV9D_SUCCESS == error){
    error = eWMV9DecGetOutputFrame(&psWmv9DecObj);
}
if (error != E_WMV9D_SUCCESS) {
    /* do the appropriate depending upon the return value */
}
else{

    /* the current frame number is available at currFrameNum */

    /* The output frame is available on the YCbCr buffer. Application
    can access the buffer for post-processing (filtering / colour
    conversion / rotation / resizing). Make sure that the buffer is
    not corrupted as this will probably be used by the next decode
    call.
    */
    const unsigned char* pu8RowPtr =
        sWmv9DecObj.sDecParam.sOutputBuffer.pu8YBuf;
    s32RowSize = sWmv9DecObj.sDecParam.sOutputBuffer.s32YRowSize;

    for (row = 0; row < height; row++)
    {
        /* one row data from pu8RowPtr to pu8RowPtr + width */
        pu8RowPtr += s32RowSize;
    }
}

/* free the decoder */
error = eWMV9DFree(&sWmv9DecObj);

FreeMemory (&sWmv9DecObj);

/***** End of the decoding a video bit stream *****/

/* function called by the decoder for reading the bitstream */
int cbkWMV9DBufRead (int s32BufLen, unsigned char *pu8Buf, int
                    *bEndOfFrame, void *pvAppContext)
{
    // Copy bit stream data to the buffer from the bit stream buffer to
    decoder space
    // Set the bEndOfFrame to be 1, if there is no more data for this
    frame or sequence header.
    // return the amount of data copied.
}

/* couple of helper function used in memory allocation */
void* AllocateFastMem (int s32NumBytes);
void* AllocateSlowMem (int s32NumBytes);
void* MemAlign(void* pvUnalignedBuf, eWmv9DecMemAlignType ememAlign);
void  MemCopy(const void* pvSrc, void* pvDst, s32NumBytes);

```

```

void    MemFree(void* pvBuffer);

/* Give memory to the decoder for the size, type and alignment returned
*/
void AllocateMemory (sWmv9DecObject  *psObj)
{
    sMemAllocInfo *psMemInfo = &psObj->sMemInfo;

    for (i = 0; i < psMemInfo->s32NumReqs ; i++) {

        sWmv9DecMemBlock* curBlk = psMemInfo->asMemBlks + i;

        memType = curBlk->type;

        if (WMV9D_IS_SIZE_CHANGED(memType)) {

            /* See if we need to allocate new memory buffer */
            if (curBlk->size > 0)
            {
                /* allocate the memory fast */
                if (WMV9D_IS_FAST_MEMORY(memType))
                    tempBuf = AllocateFastMem (curBlk->size);
                else
                    tempBuf = AllocateSlowMem (curBlk->size);

                /* Now align the memory for the requested alignment */
                alignBuf = MemAlign (tempBuf, curBlk->eMemAlign);
            } else
                alignBuf = tempBuf = NULL;

            /* if required copy the data from the old place */
            if (curBlk->pvBuffer != NULL)
            {
                if (WMV9D_COPY_AT_RESIZE(memType))
                    MemCopy (curBlk->pvBuffer, alignBuf, smaller of two
sizes);

                /* free the old buffer */
                MemFree (currBlk->pvUnalignedBuffer);
            }

            /* Set the new buffer */
            currBlk->pvUnalignedBuffer = tempBuf;
            curBlk->pvBuffer = alignBuf;

        } /* if current blocks size has been changed */
    } /* for each valid entry in the array */
}

/* function to free the memory allocated for the decoder */
void FreeMemeory (sWmv9DecObject  *psObj)
{
    sMemAllocInfo *psMemInfo = &psObj->sMemInfo;

    for (i = 0; i < psMemInfo->s32NumReqs ; i++) {

```

```
    sWmv9DecMemBlock* curBlk = psMemInfo->asMemBlks + i;  
    if (curBlk->size > 0)  
        free (curBlk->pvUnalignedBuffer);  
    }  
}
```