08-6466-SIS-ZCH66
MAY 25, 2009
3.5

Application Programmers Interface for JPEG Decoder

ABSTRACT:

Application Programmers Interface for JPEG Decoder

KEYWORDS:

Multimedia codecs, JPEG, image

APPROVED:

Wang Zening

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	03- Dec-2003	B.Venkatarao	Initial Draft
1.0	26-Dec-2003	B.Venkatarao	Review changes
1.1	02-Jan-2003	Harsha D G	Reformat content and added more details
1.2	21-Jan-2004	B.Venkatarao	Added appendix and changes in the interface of jpegd_get_new_data() function.
2.0	27-Jan-2004	Harsha D G	Added the scale down factor in the API
2.1	28-Jan-2004	B.Venkatarao	Error codes added in the Appendix.
2.2	28-Jan-2004	Harsha D G	Added descriptions to certain TBD items
2.3	23-Mar-2004	B.Venkatarao	Changes to use same functions for all output formats. Changed some of the Decoder structure names and definitions. Added enum for output format.
2.4	05-Jul-2004	B.Venkatarao	Updated with latest changes
2.5	16-Nov-2004	B.Venkatarao	Updated with new PCS requirements
2.6	29-Nov-2004	B.Venkatarao	Review comments incorporated
2.7	11-Jan-2005	B.Venkatarao	Prefixing data types. One section added to describe debug logs. Updated with latest changes.
2.8	08-Sep-2005	Shankar Narayan P.S.	Section added to describe implementation of call back
3.0	06-Feb-2006	Lauren Post	Using new format
3.1	28-March-2006	Sriram Shankar	Document review
3.2	1-Aug-2008	Wang Zening	Add API version information
3.3	12-Nov-2008	Eagle Zhou	Add BGR output format
3.4	02-Mar-2009	Eagle Zhou	Add API for decoding the whole frame
3.5	25-May-2009	Eagle Zhou	Add VPU decoding

Table of Contents

Introduction

1.1 Purpose

This document gives the application programmer's interface for JPEG Decoder.

1.2 Scope

This document does not give the details of implementation of the JPEG Decoder. It only explains the functions and variables exposed in the API.

1.3 Audience Description

The reader is expected to have basic understanding of Image processing and JPEG Sequential and Progressive decoding.

1.4 References

1.4.1 Standards

- DIS 10918-1 and draft DIS 10918-2
- "JPEG Still Image Data Compression Standard" by William B. Penne baker and Joan L. Mitchell published by Van No strand Reinhold, 1993, ISBN 0-442-01272-1. 638 pages, price US$59.95. This book includes the complete text of the ISO JPEG standards (DIS 10918-1 and draft DIS 10918-2).

1.4.2General References

- Wallace, Gregory K. "The JPEG Still Picture Compression Standard", Communications of the ACM, April 1991 (vol. 34 no. 4), pp. 30-44.

1.4.3Freescale Multimedia References

- JPEG Decoder Application Programming Interface – jpeg_dec_api.doc
- JPEG Decoder Requirements Book - jpeg_dec_reqb.doc
- JPEG Decoder Test Plan - jpeg_dec_test_plan.doc
- JPEG Decoder Release notes - jpeg_dec_release_notes.doc
- JPEG Decoder Test Results – jpeg_dec_test_results.doc
- JPEG Decoder Performance Results – jpeg_dec_perf_results.doc
- JPEG Decoder Interface header – jpeg_dec_interface.h
- JPEG Decoder Test Application – jpeg_dec_app.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
API	Application Programming Interface
ARM	Advanced RISC Machine
Data Unit	JPEG proposal defines a data unit as a sample in predictive codecs and a [8x8] block in case of DCT based codecs
DCT	Discrete Cosine Transform
FSL	Freescale
IDCT	Inverse Discrete Cosine Transform
IJG	Independent JPEG Group
JPEG	Joint Photographic Experts Group
MCU	Minimum Coded unit. JPEG proposal defines an MCU as the smallest group of interleaved data units
RVDS	ARM RealView Development Suite
TBD	To Be Determined
UNIX	Linux PC x/86 C-reference binaries
VPU	Video Process Unit

1.6 Document Location

docs/jpeg_dec

2 API Description

This is the most important section of this document. This section describes the steps followed by the application to call the JPEG Decoder. During each step the data structures used and the functions used will be explained.

Step 1: Allocate memory for Decoder object

The application allocates memory for the jpeg decoder object. This object contains the parameters and memory information structures which will be filled in later steps.

```
/* Decoder Object structure */
typedef struct
{
    JPEGD_Mem_Alloc_Info        mem_info;
    JPEGD_Decoder_Params        dec_param;
    JPEGD_Decoder_Info          dec_info;

    void                        *cinfo;
    JPEGD_exif_info             exif_info;
    JPEGD_jfif_info             jfif_info;

  /*Changes Made for Call back*/
  JPEGD_UINT8                       (*jpegd_get_new_data_fun)
                  (JPEGD_UINT8 **, JPEGD_UINT32 * ,JPEGD_UINT32 ,
                 JPEGD_UINT8 ,void *);
  /*Changes Made for Call back*/
} JPEGD_Decoder_Object;

typedef enum
{
    JPEGD_NO_THUMBNAIL = 0,
    JPEGD_THUMBNAIL_JPEG,
    JPEGD_THUMBNAIL_UNCOMPRESSED,
    JPEGD_THUMBNAIL_CORRUPT,
    JPEGD_THUMBNAIL_UNKNOWN,
} JPEGD_THUMBNAIL_TYPE;
```

Step 2: Register the function jpegd_get_new_data to the function pointer in the object.

The application need to register a function that could read data for the decoder. Currently, the function jpegd_get_new_data is registered using the API jpegd_register_jpegd_get_new_data. This is done in order to implement the function jpegd_get_new_data as a call back function. i.e. jpegd_get_new_data is called using the pointer *jpegd_get_new_data_fun* which is declared in the object structure.

C prototype:

```
JPEGD_RET_TYPE jpegd_register_jpegd_get_new_data(
            JPEGD_UINT8 (* func)(JPEGD_UINT8 **,
            JPEGD_UINT32 * ,JPEGD_UINT32 ,
            JPEGD_UINT8 ,void *),
            JPEGD_Decoder_Object *);
```

Arguments:

- Function Pointer
- Decoder Object pointer

Return value:

None

Step 3: Get the JPEG file information

This function gives the information about the JPEG file i.e. whether the file is a JFIF or an EXIF and whether the file contains thumbnail image or not. This function parses the bit stream and finds the required decoder information. During this process, this function calls *jpegd_get_new_data* to get input bit stream. This routine needs to be called at the beginning of every new file/stream. This function also returns one parameter “min_size_exif” if it is an EXIF file, which is the minimum size of the first bitstream buffer (the buffer which contains the bitstream from the beginning of the JPEG file) that should be passed by the application when “get_new_data” is called for first time from the functions “jpegd_query_dec_mem” and “jpegd_decoder_init”. Refer to step-6 for more details. This function returns three parameters from the function and also fills the same information in dec_obj->dec_info, for the later use in the decoding.

Important Note: The application should provide bit stream from the beginning of the JPEG stream when *jpegd_get_file_info* calls the *jpegd_get_new_data* function. It also should provide bitstream from the beginning of the JPEG stream when *jpegd_get_new_data* is called first time from the other functions *jpegd_query_dec_mem and jpegd_decoder_init.*
If the decoder is suspended from this function it can not resume again from the previous state. But the application can follow the procedure again from the beginning of step-2 and call this function again

C prototype:

```
JPEGD_RET_TYPE  jpegd_get_file_info (
            JPEGD_Decoder_Object *dec_obj,
            JPEGD_UINT8 *file_format,
            JPEGD_THUMBNAIL_TYPE *thumbnail_type,
            JPEGD_UINT32 *min_size_exif);
```

Arguments:

- Decoder Object pointer
- file_format, JFIF or EXIF
- thumbnail_type
- Minimum size of the first bitstream buffer to be passed by the application

Return value:

- JPEGD_ERR_NO_ERROR - Successful.
- JPEGD_ERR_SUSPENDED - Suspended
- Other codes - Error

file_format:

JPEGD_FILE_IS_JFIF - it is a JFIF file
JPEGD_FILE_IS_EXIF - it is an EXIF file

thumbnail_type:

JPEGD_NO_THUMBNAIL - There is no thumbnail present
JPEGD_THUMBNAIL_JPEG - JPEG compressed thumbnail is present
JPEGD_THUMBNAIL_UNCOMPRESSED - Uncompressed thumbnail is present
JPEGD_THUMBNAIL_CORRUPT, - Thumbnail is corrupted
JPEGD_THUMBNAIL_UNKNOWN - Unknown thumbnail is present

Step 4: Fill up the parameters structure

The application fills up the parameters needed to configure the jpeg decoder.

```
/* DCT/IDCT algorithm options. */
typedef enum
{
    JPEGD_IDCT_SLOW,     /* Slow but accurate integer algorithm */
    JPEGD_IDCT_FAST,     /* Faster, less accurate integer method */
    JPEGD_IDCT_FLOAT,     /* Floating-point: accurate, fast on fast HW,
                     *  not supported for TARGET
                     */
    JPEGD_IDCT_FIRST = JPEGD_IDCT_SLOW,
    JPEGD_IDCT_LAST = JPEGD_IDCT_FLOAT
} JPEGD_DCT_METHOD;

/* Output formats */
typedef enum
{
    JPEGD_OFMT_ENC_IMAGE_FMT,         /* Same as Encoded image format */
    JPEGD_OFMT_RGB_565,
    JPEGD_OFMT_RGB_888,               /* Default */
```

```
    JPEGD_OFMT_BGR_565,
    JPEGD_OFMT_BGR_888,
    JPEGD_OFMT_FIRST = JPEGD_OFMT_ENC_IMAGE_FMT,
    JPEGD_OFMT_LAST = JPEGD_OFMT_BGR_888
} JPEGD_OUTPUT_FORMAT;

/*
 * Decoder parameters. These parameters should be set by the
 *  application, before calling any decoder functions.
 */
typedef struct
{
    JPEGD_UINT16  desired_output_width;   /* If set to '0', it is equal
to original_ image_width */
    JPEGD_UINT16  desired_output_height;  /* If set to '0', it is equal
to original_image_height */
    JPEGD_DCT_METHOD dct_method;    /* default is JPEGD_IDCT_FAST, fast
IDCT */
    JPEGD_OUTPUT_FORMAT  output_format;     /* default is
JPEGD_OFMT_ENC_IMAGE_FMT */

    JPEGD_UINT8 decoding_mode;            /* JPEGD_PRIMARY,
JPEGD_THUMBNAIL */
    JPEGD_UINT8 exif_info_needed;     /* Used only if file_format is exif
*/
    JPEGD_UINT8 vpu_enable;           /*1: enable; 0:disable*/
    JPEGD_UINT32  vpu_bitstream_buf_size; /*buffer size used for raw
data*/
    JPEGD_UINT32  vpu_fill_size;    /*unit size of data which decoder
feed to vpu every time*/
    JPEGD_UINT32  vpu_wait_time;    /*decoder's waiting time before vpu
need more data, unit: millisecond*/

} JPEGD_Decoder_Params;
```

Description of structure *JPEGD_Decoder_Params*

dct_method

IDCT algorithm that the decoder uses. The decoder supports two methods, one is the slow method (JPEGD_IDCT_SLOW) which is accurate method but relatively slower than the fast method, and the other is the fast method (JPEGD_IDCT_FAST) which is faster but less accurate compared to the slow method. The floating point method (JPEGD_IDCT_FLOAT) is not supported by the decoder.

output_format

Format in which the decoder returns the output image pixels. The decoder will support 5 formats, the format same as encoded image format, RGB-565, RGB-888, BGR-565, and BGR-888

desired_output_width and desired_output height

Width and height of the output image that the decoder returns. These parameters can be configured by the user based on the display size. If the user wants the output size to be same as input image size, these parameters need to be set to zero. Currently decoder supports resizing by integer scale factors in the range 1 to 32. The decoder does the resizing of the image by maintaining same aspect ratio. If the input image size is smaller than the output image size, the decoder returns the output image with the size same as the input image size i.e., without resizing. The decoder initialization routine returns the actual output image width and height set by the decoder based on the user configuration and the input

image size. In the case of output format "*JPEGD_OFMT_ENC_IMAGE_FMT"*, resizing is not done i.e., the decoder returns the output image with the size same as the input image.

decoding_mode

Flag which indicates whether application wants to decode primary image or thumbnail image. If it is set to 'JPEGD_PRIMARY', it wants primary image otherwise it needs thumbnail image. The application can set this flag after calling the "jpegd_get_file_info" function which returns the information about the JPEG file like JFIF or EXIF, thumbnail is inserted or not. This flag should be set before calling *"jpegd_query_dec_mem"*.

exif_info_needed

*F*lag which indicates whether application wants to decode EXIF tags or not. If it is set to 1, the decoder will decode the EXIF tags otherwise it skips. The decoder currently decodes only the required tags, other tags are skipped. Refer to step-8 to see the list of tags that are currently supported.

vpu_enable

*F*lag which indicates whether enable VPU decoding. If it is set to 1, VPU decoder will be used firstly, in such case, software decoding will be used except VPU can not decode current image. If it is set to 0, software will be used.

vpu_bitstream_buf_size

It is the size of bit-stream buffer used by VPU. If it is set to 0, internal default size will be set by decoder.

vpu_fill_size

It is the unit size of reading operation for every call back function. If it is set to 0, internal default size will be set by decoder.

vpu_wait_time

It is the time for decoder to wait VPU idle. If it is set to 0, internal default time will be set by decoder.

Step 5: Query memory requirements

The JPEG decoder doesn't do any dynamic memory allocation. But the decoder memory requirements can change based on the type of the JPEG bit stream (like image size, JPEG mode baseline/progressive). The application has to allocate memory as required by the decoder. So the application first needs to query for memory by calling the function *jpegd_query_dec_mem.* This function must be called before all other decoder functions are invoked. This function parses the required decoder information from the bit stream and fills the memory information structure array. Then the application will 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 *jpegd_get_new_data* to get input bit stream. This routine needs to be called at the beginning of every new file/stream.

Important Note:

The application should provide bit stream from the beginning of the JPEG stream when *jpegd_query_dec_mem* calls the *jpegd_get_new_data* function. After this function is finished, when the decoder initialization function calls *jpegd_get_new_data*, the application should provide the bit stream again from the beginning of the JPEG stream.

If the decoder is suspended from this function it can not resume again from the previous state. But the application can follow the procedure again from the beginning of step-4 and call this function again.

C prototype:

```
JPEGD_RET_TYPE  jpegd_query_dec_mem (JPEGD_Decoder_Object *dec_obj);
```

Arguments:

- Decoder Object pointer.

Return value:

- JPEGD_ERR_NO_ERROR - Memory query successful
- JPEGD_ERR_SUSPENDED - Suspended.
- Other codes - Error

```
/* JPEGD_Mem_Alloc_Info and JPEGD_Mem_Alloc_Info_Sub are the memory
allocation
 *  structures filled by the decoder in jpegd_query_dec_mem() function.
 *  Then memory pointers (ptr) will be initialized by the application
 */
typedef struct
{
    JPEGD_INT32   align;      /* Alignment of memory in bytes */
    JPEGD_INT32   size;       /* Size in bytes */
    JPEGD_INT32   mem_type_speed; /* Memory type Fast or Slow */
    JPEGD_INT32   mem_type_usage; /* Memory type static or scratch */
    JPEGD_INT32   priority;   /* Memory priority */
    void    *ptr;       /* Pointer to the memory */
    void    *phy_ptr;       /* Pointer to the physical memory */
} JPEGD_Mem_Alloc_Info_Sub;

typedef struct
{
    JPEGD_INT32                         num_reqs;
    JPEGD_Mem_Alloc_Info_Sub    mem_info_sub[JPEGD_MAX_NUM_MEM_REQS];
} JPEGD_Mem_Alloc_Info;
```

Step 6: Data Memory allocations by application

In this step the application allocates the memory as requested by JPEG Decoder and fills up the memory pointer '*ptr' of 'JPEGD_Mem_Alloc_Info_Sub*' structures.

Step 7: Memory allocations for input buffer

The application has to allocate the memory needed for the input buffer. The decoder, whenever it needs the JPEG bit-stream, shall call the function *jpegd_get_new_data*.

jpegd_get_new_data should be implemented by the application. The application might have different techniques to implement this function. Sample code is given in released package. Refer next section for more details on 'jpegd_get_new_data'.

Step 8: Streaming Input and Suspension

The application has to allocate the memory needed for the input buffer. The decoder, whenever it needs the JPEG bit-stream, shall call the function *jpegd_get_new_data*. The application can also suspend the decoder from this function.

jpegd_get_new_data should be implemented by the application. The application might have different techniques to implement this function. Sample code is given in released package.

This function serves a dual purpose. Firstly, the decoder can decide the buffer address through filling **ppBuf* appropriately. The application can then fill data into this address. Secondly, the buffer address can also be decided by application. The application can identify which instance of the decoder has called this function through judge whether *ppBuf is equal to NULL. If the application wants to suspend the codec, then it has to save *mcu_offset* to use this when it resumes the decoding again. When the application wants to resume the decoding after suspension, then it has to calculate the pointer to the current MCU and pass the appropriate bitstream to the decoder when *get_new_data()* is called first time. The MCU pointer is equal to the current bitstream end pointer minus the saved *mcu_offset*.

When the decoder is suspended and if the application wants to resume the decoding later, the application has to maintain the whole state of the decoder which include decoder object, and the output buffers
If *begin_flag* is set to 1, application has to give the bitstream from the beginning of the JPEG file. Also if it is an EXIF file and *begin_flag* is set to 1, it has to provide the bitstream of the size greater than or equal to "min_size_exif" that is given by the function *jpegd_get_file_info.* The decoder sets *begin_flag* to 1, when this function is called first time from the functions *jpegd_query_dec_mem and jpeg_decoder_init.*

C prototype:

```
JPEGD_UINT8 jpegd_get_new_data (JPEGD_UINT8 ** ppBuf, JPEGD_UINT32  *
pLen, JPEGD_UINT32 mcu_offset, JPEGD_UINT8 begin_flag, void *obj_ptr);
```

Arguments:

- Pointer to the pointer to JPEGD_UINT8 data. The application is expected fill data into buffer pointed by this pointer. If the buffer pointed by pointer is NULL, application select one new buffer and set it to decoder.
- Pointer to *size* variable. The application is expected to fill the exact size of the buffer it passed. If buffer pointed by pointer **pBuf is NULL, application decide the length of data.
- Offset to the current MCU from the end of the current buffer.
- Flag to indicate whether decoder needs bitstream from the beginning of the file.
- Decoder object pointer.

Return value:

- JPEGD_SUCCESS - Buffer allocation successful
- JPEGD_END_OF_FILE - End of file
- JPEGD_SUSPEND - Suspend the decoder

Important Note:

- If the application passes the bitstream buffer which can hold minimum of one MCU data, the current MCU pointer will always remain in the current buffer. In case of suspension, the buffer management can be easier for the application. The recommended size will be defined in interface.h (MINIMUM_BUFFER_SIZE). This is only a recommendation from the codec.
- The application has to pass the buffers from the beginning of the JPEG file, when "begin_flag" is set to 1.
- In case of EXIF files, the decoder expects the first bitstream buffer should be of size greater than the size *min_size_exif* returned by *jpegd_get_file_info.*
- When the decoder is suspended and if the application wants to resume the decoding later, the application has to maintain the whole state of the decoder which include decoder object, and the output buffers

Step 9: Initialization routine

All initializations required for the decoder are done in *jpegd_decoder_init*. The initialization routine calls *jpegd_get_new_data* to obtain input JPEG bitstream from application. The initialization routine needs to be called at the beginning of every new file/stream.

Important Note:
When this function is being called, the application should provide the bit stream from the beginning of the JPEG stream, whenever decoder requests for new bit stream by calling *jpegd_get_new_data* routine. If the decoder is suspended from this function it can not resume again from the previous state. But the application can follow the procedure again from the beginning of step-8 and call this function again

C prototype:
```
JPEGD_RET_TYPE jpegd_decoder_init (JPEGD_Decoder_Object *dec_obj);
```

Arguments:
- Decoder Object pointer.

Return value:
- JPEGD_ERR_NO_ERROR - Initialization successful.
- JPEGD_ERR_SUSPENDED - Suspended.
- Other codes - Initialization Error

The decoder also fills a structure named *JPEGD_Decoder_Info.* The application can use the information present in this structure to allocate the memory for the output buffer. If *dec_params->exif_info_needed* is set to 1, the decoder will also fill the exif info structure *dec_obj->exif_info*.

```
typedef struct
{
    /* Relative horizontal  sampling factor of a  component */
    JPEGD_UINT8       h_samp;
    /* Relative vertical sampling factor of a component */
    JPEGD_UINT8       v_samp;

    /* Output width and height of a  component */
```

```
    JPEGD_UINT16      actual_output_width;
    JPEGD_UINT16      actual_output_height;
  /* Number of lines decoded so far of a  comp */
    JPEGD_UINT16      output_scanline;

   /* Maximum number of lines decoder can emit for a  component */
    JPEGD_INT32       max_lines;

    /* Number of lines returned for a  comp by jpegd_decode_mcu_row() */
    JPEGD_INT32       num_lines;
} JPEGD_Component_Info;

/* Following structure is the decoder info and is read-only for the
application */
typedef struct
{
    JPEGD_UINT8       mode;       /* Sequential or Progressive */
    JPEGD_UINT8       h_samp_max; /* Maximum  Horizontal sampling factor
*/
    JPEGD_UINT8       v_samp_max; /* Maximum  vertical sampling factor */
    JPEGD_UINT8       num_components; /* Number of components in JPEG
file */

    JPEGD_UINT16      original_image_width;  /* Input Image width, as
present in
                                                the JPEG bitstream */
    JPEGD_UINT16      original_image_height; /* Input Image height, as
present in
                                                the JPEG bitstream */
    /*  actual_output_width set by the decoder based on the
     *  configured parameter dec_param->desired_output_width.
     *  This can be different from (always <=) dec_param-
>desired_output_width
     */
    JPEGD_UINT16      actual_output_width;
    /*  actual_output_height set by the decoder based on the
     *  configured parameter dec_param->desired_output_height.
     *  This can be different from (always <=) dec_param-
>desired_output_height
     */
    JPEGD_UINT16      actual_output_height;

    /* For YUV outputs, following three parameters output_scanline,
max_lines,
     * and num_lines are the parameters of the maximum component present
     * in the JPEG file. */
    /* Number of lines decoded so far */
    JPEGD_UINT16      output_scanline;


    /* Maximum number of lines decoder can emit  when
jpegd_decode_mcu_row
     * is called
     */
    JPEGD_INT32       max_lines;

    /* Number of lines returned by jpegd_decode_mcu_row() */
    JPEGD_INT32       num_lines;

    //JPEGD_UINT8       thumbnail_flag;
    JPEGD_THUMBNAIL_TYPE thumbnail_type;
    JPEGD_UINT8       file_format;
```



```
    JPEGD_UINT32      min_size_exif;

    /* Information of the components present in the JPEG file  */
    JPEGD_Component_Info     comp_info[JPEGD_MAX_NUM_COMPS];
} JPEGD_Decoder_Info;

typedef struct
{
    JPEGD_UINT32 count;  /* count is size of the tag in bytes */
    void* ptr;
} JPEGD_tag;

typedef struct
{
    JPEGD_UINT32 x_resolution[2];
    JPEGD_UINT32 y_resolution[2];
    JPEGD_UINT16 resolution_unit;
    JPEGD_UINT16 ycbcr_positioning;
} JPEGD_IFD0_appinfo;

typedef struct
{
    JPEGD_UINT8 exif_version[4];
    JPEGD_UINT8 componentsconfiguration[4];
    JPEGD_UINT8 flashpix_version[4];
    JPEGD_UINT16 colorspace;
    JPEGD_UINT16 pixel_x_dimension;
    JPEGD_UINT16 pixel_y_dimension;

} JPEGD_exifIFD_appinfo;

typedef struct
{
    JPEGD_UINT32 x_resolution[2];
    JPEGD_UINT32 y_resolution[2];
    JPEGD_UINT16 resolution_unit;
    JPEGD_UINT16 compression; /* = 6 for compressed thumbnail, others
invalid */
    JPEGD_UINT32 jpeg_interchange_format; /* offset to thumbnail image */
    JPEGD_UINT32 jpeg_interchange_format_length; /* size of thumbnail
image */

} JPEGD_IFD1_appinfo;

typedef struct
{
    JPEGD_UINT8 endianness;
    /* flags to indicate the presence of IFDs */
    JPEGD_UINT8 IFD0_flag;
    JPEGD_UINT8 ExifIFD_flag;
    JPEGD_UINT8 IFD1_flag;
    JPEGD_UINT8 InteropIFD_flag;
    JPEGD_UINT8 GpsIFD_flag;

    //JPEGD_UINT8 compressed_thumbnail;
    JPEGD_THUMBNAIL_TYPE thumbnail_type;
    /* IFD structs to hold the tag info  */
    JPEGD_IFD0_appinfo ifd0_info;
    JPEGD_exifIFD_appinfo exififd_info;
    JPEGD_IFD1_appinfo ifd1_info;
    /* currently doesn't support GPS IFD and Interop IFD */
```



```
} JPEGD_exif_info;

typedef struct
{
    JPEGD_UINT8 jfif_major_version;
    JPEGD_UINT8 jfif_minor_version;
    JPEGD_UINT8 density_unit; // 0 = no unit, 1 = dots per inch, 2 = dots
per cm
    JPEGD_UINT16 Xdensity;
    JPEGD_UINT16 Ydensity;
    //JPEGD_UINT8 compressed_thumbnail;
    JPEGD_THUMBNAIL_TYPE thumbnail_type;
} JPEGD_jfif_info;
```

In case of EXIF, following tags are supported.

IFD0 tags:

x_resolution:	Number of pixels per resolution unit in the horizontal direction.
y_resolution:	Number of pixels per resolution unit in the vertical direction
resolution_unit:	Unit for measuring X-resolution and Y-resolution 2 – inches; 3- centimeters; Other- reserved
ycbcr_positioning:	The position of chrominance components in relation to the luminance component. 1 = centered 2 = co-sited

EXIF IFD tags:

exif_version:	EXIF version, 4-byte ASCII.
components_configuration:	Component configuration specific to compressed data 4,5,6,0 - RGB uncompressed data 1,2,3,0 - Other cases (Y-Cb-Cr) Other - Reserved
flashpix_version:	Flashpix version, 4-byte ASCII (Example 00.01).
colorspace:	Color space 1 - sRGB FFFF - Uncalibrated (can be treated as sRGB when it is converted to Flashpix)
pixel_x_dimension:	The valid image width, when the compressed file is recorded
pixel_y_dimension:	The valid image height, when the compressed file is recorded

IFD1 tags:

x_resolution:	Number of pixels per resolution unit in the horizontal direction, for thumbnail image.
y_resolution:	Number of pixels per resolution unit in the vertical direction, for thumbnail image
resolution_unit:	Unit for measuring X-resolution and Y-resolution of the thumbnail image 2 – inches; 3- centimeters; Other- reserved.
compression:	Compression scheme for thumbnail image 1 – Uncompressed, 6 – Compressed, Other - reserved
jpeg_interchange_format:	Offset to thumbnail image
jpeg_interchange_format_length:	Size of thumbnail image

In the case of JFIF, following parameters are returned by the decoder.

JFIF parameters:

jfif_version:	JFIF version, 4-byte info
resolution_unit:	Unit for measuring X-resolution and Y-resolution 0 – aspect ratio, 1 – inches, 2 - centimeters
x_density:	Number of horizontal pixels per resolution unit.
y_density:	Number of vertical pixels per resolution unit.
thumbnail_type:	1 - Compressed thumbnail is present 0 - Compressed thumbnail is not present

Important Note:

The above decoder information is initialized in *jpegd_decoder_init()* and will remain constant throughout the decoding except for the following parameters. The following parameters are initialized in *jpegd_decoder_init()* and will be updated in the decoder routine *jpegd_decode_mcu_row()*.

```
dec_info->output_scanline
dec_info->num_lines
dec_info->comp_info[i]->output_scanline
dec_info->comp_info[i]->num_lines
```

Also note that the Component Info structure *dec_info->comp_info* will be filled by the decoder only for YUV outputs.

Step 10: Memory allocation for output buffer

The application has to allocate memory for the output buffers to hold YUV Data or the RGB Data. The decoder does not request this memory when *jpegd_query_dec_mem* is called.

If the application wants to allocate buffers which need to hold only one MCU row of data, the size of the buffers can be calculated from the following parameters,

For RGB outputs

dec_info->actual_output_width and dec_info->max_lines

RGB565 -> pixel size = 2 bytes
RGB888 -> pixel size = 3 bytes

size = dec_info->actual_output_width * dec_info->max_lines * pixel_size;

For YUV outputs

dec_info->comp_info[i]-> actual_output_width and
dec_info->comp_info[i]-> max_lines
pixel_size -> pixel size = 1 byte (for each component)

Size of ith component = dec_info->comp_info[i]-> actual_output_width * dec_info->comp_info[i]-> max_lines;

If the application wants to allocate buffers to hold the whole image, the size of the buffers can be calculated from the following parameters.

For RGB outputs

dec_info->actual_output_width and dec_info->actual_output_height

RGB565 -> pixel size = 2 bytes
RGB888 -> pixel size = 3 bytes

For YUV outputs

dec_info->comp_info[i]-> actual_output_width and
dec_info->comp_info[i]-> actual_output_height
pixel_size -> pixel size = 1 byte (for each component)

Important Note:

To get maximum performance in speed, output buffer pointers should be aligned to 8-byte boundary. For RGB-565 outputs, make sure that output buffer pointer is aligned atleast to 2-byte boundary.

Step 11: Call the decoder routine (MCU row)

The main decoder function is *jpegd_decode_mcu_row*. This function decodes the JPEG bit stream to generate the output image in the requested format. During the process of decoding, the function *jpegd_get_new_data* gets called whenever the decoder runs out of input. The calling application needs to provide a new buffer filled with input data when *jpegd_get_new_data* is called. The decoder returns the used up buffer to the calling application. The calling application can fill up fresh data in the returned buffer and keep it ready for use in the next *jpegd_get_new_data* call.

If the bit stream has errors, the decoder handles these errors internally.
The application is responsible for passing the appropriate values in the *buffer_ptrs* array. This array shall contain the pointers to the output buffers.

This function is not supported in VPU decoding mode.

C prototype:

```
JPEGD_RET_TYPE    jpegd_decode_mcu_row  (
                        JPEGD_Decoder_Object *dec_obj,
                        JPEGD_UINT8 ** buffer_ptrs,
                        JPEGD_UINT16 *out_stride_width);
```

Arguments:

- dec_obj Decoder Object pointer
- buffer_ptrs Array of pointers to the output buffer of all components
- out_stride_width stride length of each component buffer, (row width)

Return value:

- JPEGD_ERR_NO_ERROR - indicates decoding was successful.
- JPEGD_ERR_SUSPENDED - Suspended
- **Others** - **indicates error**

When *jpegd_decode_mcu_row* is called the decoder is expected to decode 1 MCU row. The decoder shall also fill *dec_info->output_scanline*, the scanline it has started decoding. It shall also fill the number of lines it has decoded in *dec_info->num_lines*. For YUV outputs, it shall also fill the above two parameters of each component *dec_info->comp_info[i]->output_scanline and dec_info->comp_info[i]->num_lines.*

Important: In the case of RGB outputs, one buffer pointer and one output stride length is passed. In the case of YUV outputs, buffer pointer and output stride length for each component is passed. Before calling this function, user should make sure that the stride length of each component should always be greater than or equal to the actual_output_width of each component.

Step 12: Call the decoder routine (whole image)

Another main decoder function is *jpegd_decode_frame*. This function decodes the JPEG bit stream to generate the whole output image in the requested format. It has the same action with *jpegd_decode_mcu_row* except the numbers of output MCU. It is expected to decode the whole image, but not only 1 MCU row.

C prototype:

```
JPEGD_RET_TYPE    jpegd_decode_frame  (
                        JPEGD_Decoder_Object *dec_obj,
                        JPEGD_UINT8 ** buffer_ptrs,
                        JPEGD_UINT16 *out_stride_width);
```

Arguments:

- dec_obj Decoder Object pointer
- buffer_ptrs Array of pointers to the output buffer of all components
- out_stride_width stride length of each component buffer, (row width)

Return value:

- JPEGD_ERR_NO_ERROR - indicates decoding was successful.
- JPEGD_ERR_SUSPENDED - Suspended
- **Others** - **indicates error**

When *jpegd_decode_frame* is called the decoder shall also fill *output_scanline* and *num_lines* like *jpegd_decode_mcu_row*. User need to output image according to *num_lines*.

Step 13: Error codes

Any of the above functions can return an error if the decoder encountered an error during execution. The application can take appropriate action depending on the error code returned. The error codes have been classified into successful returns, application errors, and codec errors.

Note: The error codes are given in the APPENDIX.

Some error types (including all application errors) are described here.

JPEGD_ERR_NO_ERROR	No error
JPEGD_ERR_SUSPENDED	Decoder suspended by the application
JPEGD_ERR_INVALID_DCT_METHOD	Invalid dct method specified by the application
JPEGD_ERR_INVALID_OUTPUT_FORMAT	Invalid output format specified by the application
JPEGD_ERR_INVALID_OUT_BUFFER_PTR	Output buffer pointers are NULL
JPEGD_ERR_INVALID_OUT_STRIDE_WIDTH	Invalid output stride width specified by the application
JPEGD_ERR_MEM_NOT_INITIALIZED	Memory chunk pointers are NULL
JPEGD_ERR_MEM_NOT_ALIGNED	Memory chunk pointers are not aligned to the requested alignment size.
JPEGD_ERR_OUT_BUFFER_NOT_ALIGNED	Output buffer pointer not aligned to 2 byte boundary for RGB-565 or BGR-565 output format

Step 14: Free memory

The application releases the memory that it allocated to JPEG Decoder if it no longer needs the decoder instance.

Important Note For decoding thumbnail
The above procedure from **step-3 to step-10** should be repeated for the decoding of the thumbnail image.

API Version

This is the decoder function to get the API version information

Prototype:

```
const char * jpegd_CodecVersionInfo ();
```

Arguments:

- None

Return value:

const char * The pointer to the constant char string of the version information string

3 Debug logs

If debug logs are needed the codec is to be recompiled with the hash define "JPEGD_DEBUG_LEVEL" set to appropriate value in 'jpeg_dec_interface.h" file
By default no debug logs are enabled.

JPEGD_DEBUG_LEVEL = 0		No debug logs
JPEGD_DEBUG_LEVEL = 0x1	(level 0)	All errors and warnings
JPEGD_DEBUG_LEVEL = 0x2	(level 1)	Function entry and exit
JPEGD_DEBUG_LEVEL = 0x4	(level 2)	Data that occurs once per FILE
JPEGD_DEBUG_LEVEL = 0x8	(level 3)	Data that occurs once per SCAN
JPEGD_DEBUG_LEVEL = 0x10	(level 4)	Data that occurs once per MCU
JPEGD_DEBUG_LEVEL = 0x20	(level 5)	Data that occurs per block (huffman outputs)

4 Example calling Routine

#include < *jpeg_dec_interface.h*>

```
FILE *fin;

int main (void)
{
      JPEGD_UINT8 nr;
    JPEGD_RET_TYPE rc;
    JPEGD_UINT8 file_format;
    JPEGD_THUMBNAIL_TYPE thumbnail_type;
    JPEGD_UINT8 output_pixel_size;
      JPEGD_Mem_Alloc_Info_Sub *mem;
      JPEGD_Decoder_Info *dec_info;
      JPEGD_Component_Info *comp_info;
      JPEGD_UINT8 *outbuf;
      JPEGD_UINT8 *output_buf[JPEGD_MAX_NUM_COMPS];

      /* Allocate memory for the decoder object */
      dec_obj = (JPEGD_Decoder_Object *)

                        alloc_fast(sizeof(JPEGD_Decoder_Object),16);

    dec_info = &dec_obj->dec_info;
      comp_info = dec_info->comp_info;

      /* Assign id */
      /* Fill up the params structure for YUV Data */
      dec_obj->dec_param.output_format = JPEGD_OFMT_ENC_IMAGE_FMT;
    dec_obj->dec_param.dct_method = JPEGD_IDCT_FAST;
      dec_obj->dec_param.desired_output_width = 0;
      dec_obj->dec_param.desired_output_height = 0;
      dec_obj->dec_param.exif_info_needed = 0;
      dec_obj->dec_param.decoding_mode = JPEGD_PRIMARY;

      /*Changes Made for Call back*/
      ret =
jpegd_register_jpegd_get_new_data(jpegd_get_new_data,&dec_obj);

      if (ret != JPEGD_ERR_NO_ERROR)
      {
      exit (1);
      }

      /*Changes Made for Call back*/

      /* Open the input file */
      fin = fopen (file_name, “rb”);

      /* Get File information */
      rc = jpegd_get_file_info (&dec_obj, &file_format, &thumbnail_type,
```

```
                    &min_size_exif);

  if (rc == JPEGD_ERR_SUSPENDED)
              return 1;
  if (rc != JPEGD_ERR_NO_ERROR)
              return 1;

  /* Set the decoding mode */
  dec_obj->dec_param.decoding_mode = JPEGD_PRIMARY;

/* Set the EXIF flag */
if (file_format == EXIF)
{
  dec_obj->dec_param.exif_info_needed = 1;
}

/* Make sure that the size of the buffer allocated in get_new_data when it's called
     for the time, to be greater than min_size_exif if file_format == EXIF  */
/* Query for memory */
  rc = jpegd_query_dec_mem (&dec_obj);

if (rc == JPEGD_ERR_SUSPENDED)
        return 1;
 if (rc != JPEGD_ERR_NO_ERROR)
        return 1;

  nr = dec_obj.mem_info.num_req;
  mem = dec_obj.mem_info.mem_info_sub;

  for(i = 0; i < nr; i++, mem++)
  {
        if (mem->mem_type_speed == JPEGD_FAST_MEMORY)
        {
              mem->ptr = alloc_fast(mem->size, mem->align);
        }
        else if (mem->mem_type_speed == JPEGD_SLOW_MEMORY)
        {
              mem->ptr = alloc_slow(mem->size, mem->align);
        }
  }

/* Initialize the decoder. */
  rc = jpegd_decoder_init (&dec_obj);

  if (rc == JPEGD_ERR_SUSPENDED)
              return 1;
  if (rc != JPEGD_ERR_NO_ERROR)
        return 1;

  if (dec_obj.dec_param.output_format == JPEGD_OFMT_ENC_IMAGE_FMT)
  {
    for (ci = 0, comp_ptr = comp_info; ci < dec_info->num_components;
              ci++, comp_ptr++)
    {
```

```
            output_buf[ci] = (JPEGD_UINT8 *)alloc_slow
                (comp_ptr->max_lines*
                  comp_ptr->actual_output_width * SIZEOF(JPEGD_UINT8));
            out_stride_width[ci] = comp_ptr->actual_output_width;
        }
      }
      else
      {
                  if ((dec_obj.dec_param.output_format ==
JPEGD_OFMT_RGB_565)||( dec_obj.dec_param.output_format ==
JPEGD_OFMT_BGR_565))
                        output_pixel_size = 2;
                  else if ((dec_obj.dec_param.output_format ==
JPEGD_OFMT_RGB_888)||
(dec_obj.dec_param.output_format == JPEGD_OFMT_BGR_888))
                        output_pixel_size = 3;

        output_buf[0] = (JPEGD_UINT8 *)alloc_slow
                                    (output_pixel_size *
                                     dec_info->max_lines *
                                     dec_info->actual_output_width *

      SIZEOF(JPEGD_UINT8));

        out_stride_width[0] = dec_info->actual_output_width;
      }


      /* main loop */
      do
      {
           /* Decode few lines */
                  ret = jpegd_decode_mcu_row (&dec_obj, output_buf,
                              out_stride_width);

           if (ret == JPEGD_ERR_SUSPENDED)
                  {
                        continue;
                  }
                  if (ret != JPEGD_ERR_NO_ERROR)
           {
            break;
           }

      } while (dec_info->output_scanline < dec_info-
>actual_output_height);
    return 0;
}
```

The *jpegd_decoder_int and jpegd_decode_mcu_row* internally call *jpegd_get_new_data* when they run out of the input bits. The *jpegd_get_new_data* function returns the used input buffer and accepts the new input buffer.

/* This function is implemented by the application */

```
JPEGD_UINT8 jpegd_get_new_data (JPEGD_UINT8 ** buf_ptr,
                                JPEGD_UINT32 *buf_len,
                                JPEGD_UINT32 mcu_offset,
                                JPEGD_UINT8  begin_flag,
                                void *obj_ptr)
{
        JPEGD_UINT8 *  ptr;

        /* Read *new_buf_ptr and free that memory */
        ptr = *buf_ptr;
        free(ptr);

   /* Obtain the input JPEG stream */
      If it is resuming from the suspension
      {
         *buf_ptr = Obtain bitstream by going backwards by mcu_offset_saved size
                                        from the current position;
         *buf_len = Obtain the length of the current buffer;
      }
      Else
      {
         If  begin_flag = = 1
         {
                Get the bitstream from the beginning of the JPEG stream.
           }
                *buf_ptr = Obtained from some source known to application,
                        corresponding to the decoder object obj_ptr;
         *buf_len = Obtained from some source known to application,
                        corresponding to the decoder object obj_ptr;
         }
         Return JPEGD_SUCCESS to indicate that new buffer has been filled.
         OR
         Return JPEGD_END_OF_FILE  if the application  encountered End of file.
      OR
      if the application wants to suspend
      {
                mcu_offset_saved = mcu_offset;
                return JPEGD_SUSPEND;
      }
}
```

5 Appendix A: Interface

jpeg_dec_interface.h file is given as follows.

```
#ifndef JPEG_DEC_INTERFACE_H
#define JPEG_DEC_INTERFACE_H

#ifndef JPEGD_DEBUG_LEVEL
#define JPEGD_DEBUG_LEVEL 0x00
#endif

/* Defines needed for the application */
#define JPEGD_MAX_NUM_COMPS 4
#define JPEGD_MAX_NUM_MEM_REQS 5

/* Return types for jpegd_get_new_data() */
enum
{
   JPEGD_SUCCESS = 0,
   JPEGD_END_OF_FILE,
   JPEGD_SUSPEND
};

/* Memory types */
enum
{
   JPEGD_FAST_MEMORY = 0,
   JPEGD_SLOW_MEMORY
};

enum
{
   JPEGD_STATIC_MEMORY = 0,
   JPEGD_SCRATCH_MEMORY,
   JPEGD_PHY_SUCCESSIVE_MEMORY,
};

/* Data types */
typedef unsigned char       JPEGD_UINT8;
typedef char                JPEGD_INT8;
typedef unsigned short      JPEGD_UINT16;
typedef short               JPEGD_INT16;
typedef unsigned long       JPEGD_UINT32;
typedef long                JPEGD_INT32;
typedef unsigned long long  JPEGD_UINT64;
typedef long long           JPEGD_INT64;

/* DCT/IDCT algorithm options. */
```

```
typedef enum
{
   JPEGD_IDCT_SLOW,     /* Slow but accurate integer algorithm */
   JPEGD_IDCT_FAST,     /* Faster, less accurate integer method */
   JPEGD_IDCT_FLOAT,     /* Floating-point: accurate, fast on fast HW,
                  *  not supported for TARGET
                  */
   JPEGD_IDCT_FIRST = JPEGD_IDCT_SLOW,
   JPEGD_IDCT_LAST = JPEGD_IDCT_FLOAT
} JPEGD_DCT_METHOD;

/* Output formats */
typedef enum
{
   JPEGD_OFMT_ENC_IMAGE_FMT,        /* Same as Encoded image format */
   JPEGD_OFMT_RGB_565,
   JPEGD_OFMT_RGB_888,              /* Default */
   JPEGD_OFMT_BGR_565,
   JPEGD_OFMT_BGR_888,
   JPEGD_OFMT_FIRST = JPEGD_OFMT_ENC_IMAGE_FMT,
   JPEGD_OFMT_LAST = JPEGD_OFMT_BGR_888
} JPEGD_OUTPUT_FORMAT;

enum
{
   JPEGD_FILE_IS_JFIF = 0,
   JPEGD_FILE_IS_EXIF
};

// decoding modes
enum
{
   JPEGD_PRIMARY = 0,
   JPEGD_THUMBNAIL
};

enum
{
   JPEGD_EXIF_LITTLE_ENDIAN = 0,
   JPEGD_EXIF_BIG_ENDIAN,
   JPEGD_EXIF_ENDIAN_CORRUPT
};

typedef enum
{
   JPEGD_NO_THUMBNAIL = 0,
   JPEGD_THUMBNAIL_JPEG,
   JPEGD_THUMBNAIL_UNCOMPRESSED,
   JPEGD_THUMBNAIL_CORRUPT,
```

```
    JPEGD_THUMBNAIL_UNKNOWN,
} JPEGD_THUMBNAIL_TYPE;

/* Error types */
enum
{
    /* Successfull return values */
    JPEGD_ERR_NO_ERROR = 0,

    /* Warnings
     *      The application can check the warnings and can continue
     *      decoding.
     */
    JPEGD_ERR_WARNINGS_START = 11,
    JPEGD_ERR_SUSPENDED = JPEGD_ERR_WARNINGS_START,
    JPEGD_ERR_WARNINGS_END,

    /* Recoverable errors
     *      These are the application errors. The application can
     *      correct the error and call the decoder again from the beginning
     */
    JPEGD_ERR_REC_ERRORS_START = 61,
    JPEGD_ERR_INVALID_DCT_METHOD = JPEGD_ERR_REC_ERRORS_START,
    JPEGD_ERR_INVALID_OUTPUT_FORMAT,
    JPEGD_ERR_INVALID_OUT_BUFFER_PTR,
    JPEGD_ERR_INVALID_OUT_STRIDE_WIDTH,
    JPEGD_ERR_MEM_NOT_INITIALIZED,
    JPEGD_ERR_MEM_NOT_ALIGNED,
    JPEGD_ERR_OUT_BUFFER_NOT_ALIGNED,
    JPEGD_ERR_REC_ERRORS_END,

    /* Fatal errors
     *      These are the codec errors which can not be recovered
     */
    JPEGD_ERR_FATAL_ERRORS_START = 111,
    JPEGD_ERR_ARITH_NOTIMPL = JPEGD_ERR_FATAL_ERRORS_START,
    JPEGD_ERR_BAD_COMPONENT_ID,
    JPEGD_ERR_BAD_DCTSIZE,
    JPEGD_ERR_BAD_HUFF_TABLE,
    JPEGD_ERR_BAD_J_COLORSPACE,
    JPEGD_ERR_BAD_LENGTH,
    JPEGD_ERR_BAD_LIB_VERSION,
    JPEGD_ERR_BAD_MCU_SIZE,
    JPEGD_ERR_BAD_PRECISION,
    JPEGD_ERR_BAD_PROGRESSION,
    JPEGD_ERR_BAD_SAMPLING,
    JPEGD_ERR_BAD_STATE,
    JPEGD_ERR_BAD_STRUCT_SIZE,
    JPEGD_ERR_CCIR601_NOTIMPL,
```



```
    JPEGD_ERR_COMPONENT_COUNT,
    JPEGD_ERR_CONVERSION_NOTIMPL,
    JPEGD_ERR_DAC_INDEX,
    JPEGD_ERR_DAC_VALUE,
    JPEGD_ERR_DHT_INDEX,
    JPEGD_ERR_DQT_INDEX,
    JPEGD_ERR_EMPTY_IMAGE,
    JPEGD_ERR_EOI_EXPECTED,
    JPEGD_ERR_FRACT_SAMPLE_NOTIMPL,
    JPEGD_ERR_IMAGE_TOO_BIG,
    JPEGD_ERR_NOTIMPL,
    JPEGD_ERR_NOT_COMPILED,
    JPEGD_ERR_NO_HUFF_TABLE,
    JPEGD_ERR_NO_IMAGE,
    JPEGD_ERR_NO_QUANT_TABLE,
    JPEGD_ERR_NO_SOI,
    JPEGD_ERR_OUT_OF_MEMORY,
    JPEGD_ERR_SOF_DUPLICATE,
    JPEGD_ERR_SOF_NO_SOS,
    JPEGD_ERR_SOF_UNSUPPORTED,
    JPEGD_ERR_SOI_DUPLICATE,
    JPEGD_ERR_SOS_NO_SOF,
    JPEGD_ERR_TOO_LITTLE_DATA,
    JPEGD_ERR_UNKNOWN_MARKER,
    JPEGD_ERR_WIDTH_OVERFLOW,
    JPEGD_ERR_BAD_THUMBNAIL_DATA,       /* Thumbnail related errors */
    JPEGD_ERR_BAD_INPUT_PARAM_EXIF,
    JPEGD_ERR_BAD_INPUT_PARAM_MODE,
    JPEGD_ERR_UNCOMPRESSED_THUMBNAIL,
    JPEGD_ERR_THUMBNAIL_OFFSET_NOT_FOUND,
    JPEGD_ERR_BAD_THUMBNAIL_TYPE,
    JPEGD_ERR_FATAL_ERRORS_END,
    /* Vpu errors
    *      These are the vpu errors
    */
    JPEGD_ERR_VPU_START=311,
    JPEGD_ERR_VPU_SETTING_ERROR=JPEGD_ERR_VPU_START,
    JPEGD_ERR_VPU_UNSUPPORTED_FMT,
    JPEGD_ERR_VPU_INVALID_MEMORY,
    JPEGD_ERR_VPU_INIT_FAILURE,
    JPEGD_ERR_VPU_OPEN_FAILURE,
    JPEGD_ERR_VPU_FILL_BUFFER_FAILURE,
    JPEGD_ERR_VPU_GET_INFO_FAILURE,
    JPEGD_ERR_VPU_REGISTER_FRAME_FAILURE,
    JPEGD_ERR_VPU_DECODE_FAILURE,
    JPEGD_ERR_VPU_GET_OUTPUT_FAILURE,
    JPEGD_ERR_VPU_END,
} JPEGD_RET_TYPE;
```

```
/* Structure definitions */
/* JPEGD_Mem_Alloc_Info and JPEGD_Mem_Alloc_Info_Sub are the memory allocation
 *  structures filled by the decoder in jpegd_query_dec_mem() function.
 *  Then memory pointers (ptr) will be initialized by the application
 */
typedef struct
{
   JPEGD_INT32   align;      /* Alignment of memory in bytes */
   JPEGD_INT32   size;       /* Size in bytes */
   JPEGD_INT32   mem_type_speed; /* Memory type Fast or Slow */
   JPEGD_INT32   mem_type_usage; /* Memory type static or scratch */
   JPEGD_INT32   priority;   /* Memory priority */
   void    *ptr;       /* Pointer to the memory */
   void    *phy_ptr;       /* Pointer to the physical memory */
} JPEGD_Mem_Alloc_Info_Sub;

typedef struct
{
   JPEGD_INT32                  num_reqs;
   JPEGD_Mem_Alloc_Info_Sub    mem_info_sub[JPEGD_MAX_NUM_MEM_REQS];
} JPEGD_Mem_Alloc_Info;

/*
 * Decoder parameters. These parameters should be set by the
 *  application, before calling any decoder functions.
 */
typedef struct
{
   JPEGD_UINT16  desired_output_width;   /* If set to '0', it is equal to original_ image_width */
   JPEGD_UINT16  desired_output_height;  /* If set to '0', it is equal to original_image_height */
   JPEGD_DCT_METHOD dct_method;    /* default is JPEGD_IDCT_FAST, fast IDCT */
   JPEGD_OUTPUT_FORMAT  output_format;     /* default is
JPEGD_OFMT_ENC_IMAGE_FMT */

   JPEGD_UINT8 decoding_mode;                  /* JPEGD_PRIMARY, JPEGD_THUMBNAIL */
   JPEGD_UINT8 exif_info_needed;     /* Used only if file_format is exif */
   JPEGD_UINT8 vpu_enable;               /*1: enable; 0:disable*/
   JPEGD_UINT32       vpu_bitstream_buf_size; /*buffer size used for raw data*/
   JPEGD_UINT32       vpu_fill_size;    /*unit size of data which decoder feed to vpu every time*/
   JPEGD_UINT32       vpu_wait_time; /*decoder's waiting time before vpu need more data, unit:
millisecond*/
} JPEGD_Decoder_Params;

/*
 * Important Note:
 *
 * The below decoder information is initialized in jpegd_decoder_init() and
 *  will remain constant throughout the decoding except for the following
 *  parameters. The following parameters are initialized in
```

```
 *  jpegd_decoder_init() and will be updated in the decoder routine
 *  jpegd_decode_mcu_row().
 *
 *      dec_info->output_scanline
 *      dec_info->num_lines
 *      dec_info->comp_info[i]->output_scanline
 *      dec_info->comp_info[i]->num_lines
 *
 * Also note that the Component Info structure dec_info->comp_info will
 * be filled by the decoder only for YUV outputs
 */
/* Following structure is used only when
 *  output_format == JPEGD_OFMT_ENC_IMAGE_FMT
 *  This is the decoder info and is read-only for the application
 *  This structure contains information about one component of the (Y or U or
 *  V) original input color space.
 */
typedef struct
{
   /* Relative horizontal  sampling factor of a  component */
   JPEGD_UINT8       h_samp;
   /* Relative vertical sampling factor of a component */
   JPEGD_UINT8       v_samp;

   /* Output width and height of a  component */
   JPEGD_UINT16      actual_output_width;
   JPEGD_UINT16      actual_output_height;
  /* Number of lines decoded so far of a  comp */
   JPEGD_UINT16      output_scanline;

  /* Maximum number of lines decoder can emit for a  component */
   JPEGD_INT32       max_lines;

   /* Number of lines returned for a  comp by jpegd_decode_mcu_row() */
   JPEGD_INT32       num_lines;
} JPEGD_Component_Info;

/* Following structure is the decoder info and is read-only for the application */
typedef struct
{
   JPEGD_UINT8       mode;       /* Sequential or Progressive */
   JPEGD_UINT8       h_samp_max; /* Maximum  Horizontal sampling factor */
   JPEGD_UINT8       v_samp_max; /* Maximum  vertical sampling factor */
   JPEGD_UINT8       num_components; /* Number of components in JPEG file */

   JPEGD_UINT16      original_image_width;  /* Input Image width, as present in
                         the JPEG bitstream */
   JPEGD_UINT16      original_image_height; /* Input Image height, as present in
                         the JPEG bitstream */
```

```
   /*  actual_output_width set by the decoder based on the
    *  configured parameter dec_param->desired_output_width.
    *  This can be different from (always <=) dec_param->desired_output_width
    */
   JPEGD_UINT16      actual_output_width;
   /*  actual_output_height set by the decoder based on the
    *  configured parameter dec_param->desired_output_height.
    *  This can be different from (always <=) dec_param->desired_output_height
    */
   JPEGD_UINT16      actual_output_height;

   /* For YUV outputs, following three parameters output_scanline, max_lines,
    * and num_lines are the parameters of the maximum component present
    * in the JPEG file. */
   /* Number of lines decoded so far */
   JPEGD_UINT16      output_scanline;


   /* Maximum number of lines decoder can emit  when jpegd_decode_mcu_row
    * is called
    */
   JPEGD_INT32       max_lines;

   /* Number of lines returned by jpegd_decode_mcu_row() */
   JPEGD_INT32       num_lines;

   //JPEGD_UINT8       thumbnail_flag;
   JPEGD_THUMBNAIL_TYPE thumbnail_type;
   JPEGD_UINT8       file_format;
   JPEGD_UINT32      min_size_exif;

   /* Information of the components present in the JPEG file  */
   JPEGD_Component_Info     comp_info[JPEGD_MAX_NUM_COMPS];
} JPEGD_Decoder_Info;

typedef struct
{
   JPEGD_UINT32 count;  /* count is size of the tag in bytes */
   void* ptr;
} JPEGD_tag;

typedef struct
{
   JPEGD_UINT32 x_resolution[2];
   JPEGD_UINT32 y_resolution[2];
   JPEGD_UINT16 resolution_unit;
   JPEGD_UINT16 ycbcr_positioning;
} JPEGD_IFD0_appinfo;
```



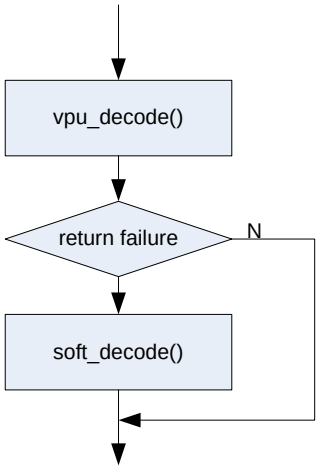
```
typedef struct
{
   JPEGD_UINT8 exif_version[4];
   JPEGD_UINT8 componentsconfiguration[4];
   JPEGD_UINT8 flashpix_version[4];
   JPEGD_UINT16 colorspace;
   JPEGD_UINT16 pixel_x_dimension;
   JPEGD_UINT16 pixel_y_dimension;

} JPEGD_exifIFD_appinfo;

typedef struct
{
   JPEGD_UINT32 x_resolution[2];
   JPEGD_UINT32 y_resolution[2];
   JPEGD_UINT16 resolution_unit;
   JPEGD_UINT16 compression; /* = 6 for compressed thumbnail, others invalid */
   JPEGD_UINT32 jpeg_interchange_format; /* offset to thumbnail image */
   JPEGD_UINT32 jpeg_interchange_format_length; /* size of thumbnail image */

} JPEGD_IFD1_appinfo;

typedef struct
{
   JPEGD_UINT8 endianness;
   /* flags to indicate the presence of IFDs */
   JPEGD_UINT8 IFD0_flag;
   JPEGD_UINT8 ExifIFD_flag;
   JPEGD_UINT8 IFD1_flag;
   JPEGD_UINT8 InteropIFD_flag;
   JPEGD_UINT8 GpsIFD_flag;

   //JPEGD_UINT8 compressed_thumbnail;
   JPEGD_THUMBNAIL_TYPE thumbnail_type;
   /* IFD structs to hold the tag info  */
   JPEGD_IFD0_appinfo ifd0_info;
   JPEGD_exifIFD_appinfo exififd_info;
   JPEGD_IFD1_appinfo ifd1_info;
   /* currently doesn't support GPS IFD and Interop IFD */

} JPEGD_exif_info;

typedef struct
{
   JPEGD_UINT8 jfif_major_version;
   JPEGD_UINT8 jfif_minor_version;
   JPEGD_UINT8 density_unit; // 0 = no unit, 1 = dots per inch, 2 = dots per cm
   JPEGD_UINT16 Xdensity;
   JPEGD_UINT16 Ydensity;
```

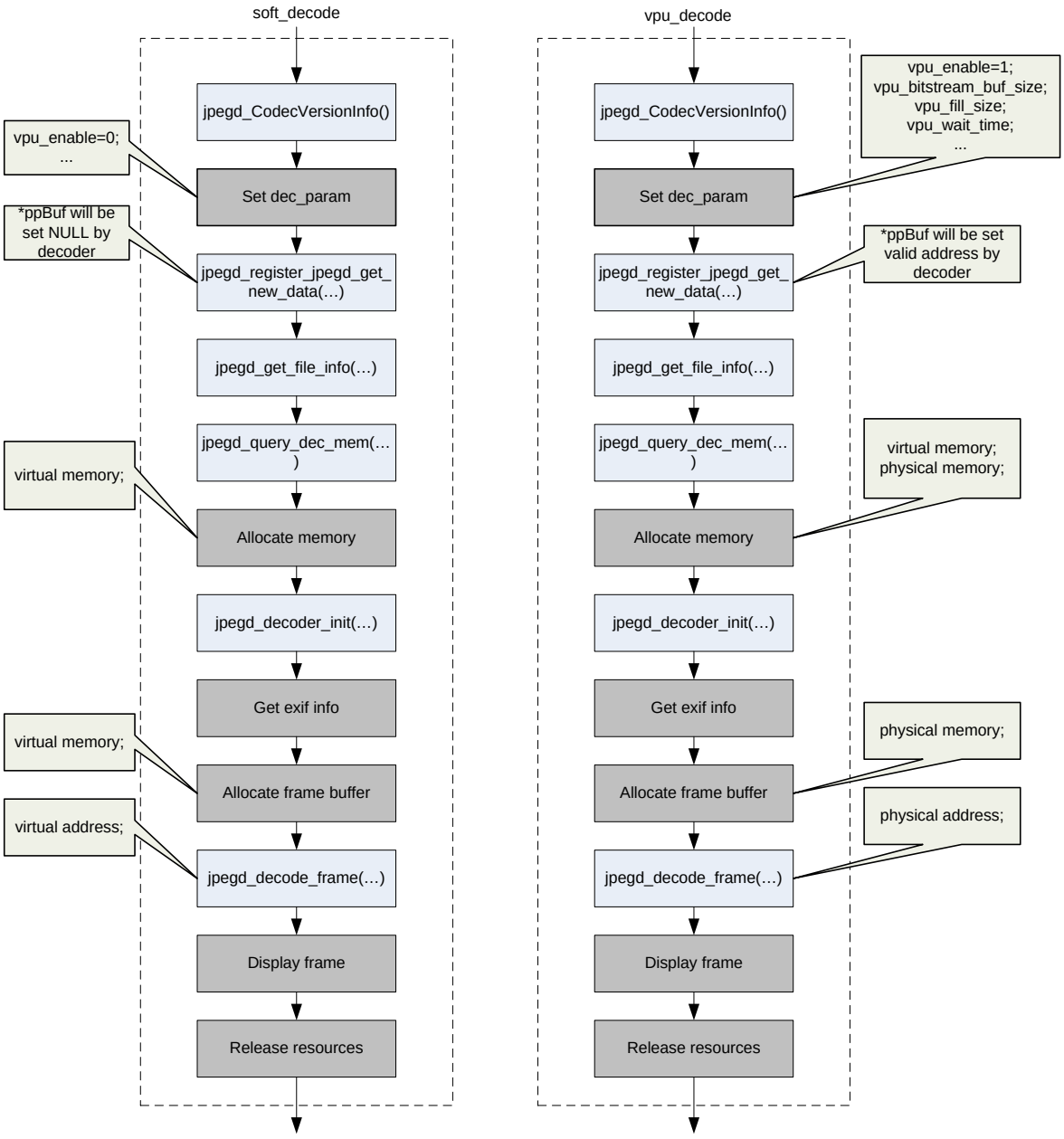
```
   //JPEGD_UINT8 compressed_thumbnail;
   JPEGD_THUMBNAIL_TYPE thumbnail_type;
} JPEGD_jfif_info;

/*
 * Decoder Object. This structure contains all the information
 *  about the decoder for one instance.
 */
typedef struct
{
   JPEGD_Mem_Alloc_Info       mem_info;
   JPEGD_Decoder_Params       dec_param;
   JPEGD_Decoder_Info         dec_info;

   void                    *cinfo;
   JPEGD_exif_info           exif_info;
   JPEGD_jfif_info           jfif_info;
} JPEGD_Decoder_Object;

JPEGD_RET_TYPE jpegd_query_dec_mem (JPEGD_Decoder_Object *dec_obj);
JPEGD_RET_TYPE jpegd_decoder_init (JPEGD_Decoder_Object *dec_obj);
JPEGD_RET_TYPE jpegd_decode_mcu_row (JPEGD_Decoder_Object *dec_obj,
                              JPEGD_UINT8 **out_buf, JPEGD_INT32 *out_stride_width);
JPEGD_UINT8 jpegd_get_new_data (JPEGD_UINT8 **ppBuf, JPEGD_UINT32 *pLen,
                    JPEGD_UINT32 mcu_offset, JPEGD_UINT8 begin_flag,
                    void *obj_ptr);

JPEGD_RET_TYPE jpegd_get_file_info (JPEGD_Decoder_Object *dec_obj,
                              JPEGD_UINT8 *file_format,
                              JPEGD_THUMBNAIL_TYPE *thumbnail_type,
                              JPEGD_UINT32 *min_size_exif);
#endif
/* End of file */
```

6 Appendix B: Work Flow

This part describes one work flow for using VPU decoding.

Application may decode image with VPU firstly. For some images, VPU decoder may not decode them successfully (only baseline is supported by VPU). In such case, software decoder should be used.




soft_decode
jpegd_CodecVersionInfo()
vpu_enable=0;
...
Set dec_param
*ppBuf will be set NULL by decoder
jpegd_register_jpegd_get_new_data(…)
jpegd_get_file_info(…)
jpegd_query_dec_mem(…)
virtual memory;
Allocate memory
jpegd_decoder_init(…)
Get exif info
virtual memory;
Allocate frame buffer
virtual address;
jpegd_decode_frame(…)
Display frame
Release resources
vpu_decode
jpegd_CodecVersionInfo()
vpu_enable=1;
vpu_bitstream_buf_size;
vpu_fill_size;
vpu_wait_time;
...
Set dec_param
*ppBuf will be set valid address by decoder
jpegd_register_jpegd_get_new_data(…)
jpegd_get_file_info(…)
jpegd_query_dec_mem(…)
virtual memory;
physical memory;
Allocate memory
jpegd_decoder_init(…)
Get exif info
physical memory;
Allocate frame buffer
physical address;
jpegd_decode_frame(…)
Display frame
Release resources