08-6467-SIS-ZCH66
DEC 18, 2008
4.2

Application Programmers Interface for JPEG Encoder

ABSTRACT:

Application Programmers Interface for JPEG Encoder

KEYWORDS:

Multimedia codecs, JPEG, image

APPROVED:

Wang Zening

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	2-Nov-2003	Harsha D G	Draft Version
1.0	18-Dec-2003	Harsha D G	Incorporated review comments from WMSG Sam/Arne
1.1	02-Jan-2004	Harsha D G	Reformat content and add more details
1.2	22-Jan-2004	Harsha D G	Reword a sentence on section 6, step 5
2.0	27-Jan-2004	Harsha D G	No changes just to have a common baseline for all documents
2.1	28-Jan-2004	Harsha D G	Added descriptions to certain TBD items
2.2	11-Feb-2004	Harsha D G	Passed a pointer to pointer to the output buffer
2.3	12-Feb-2004	Harsha D G	Minor change in comment for the ‘output’ parameter.
2.4	23-Mar-2004	Harsha D G	Removed ‘interleaved’ parameter from params structure. The ‘yuv_format’ in params structure made as enum and this will also tell whether the input is interleaved or non interleaved. Added some more description to the contents of params structure. Added void pointer to the encoder object in header file.
2.5	02-May-2004	Harsha D G	Width and height of each component provided in API. Modified Error codes. Replaced header file with latest
2.6	03-May-2004	Harsha D G	Prefixed JPEGENC_ERR for error types
2.7	05-Jul-2004	Harsha D G	Added description on how application should set height and width in step 2 of section 2
3.0	19-Nov-2004	Harsha D G	API changes to support new PCS requirements i.e. suspension,thumbnails,exif, streaming output etc
3.1	22-Nov-2004	Harsha D G	Review/Rework Done
3.2	03-Dec-2004	Harsha D G	The return type of call back function changed. 1 – success, 0 -failure
3.3	06-Jan-2004	Harsha D G	Updated for ARM11 release
4.0	06-Feb-2006	Lauren Post	Using new format
4.1	31-March-2006	Sriram Shankar	Document review
4.2	18-Dec-2008	Eagle Zhou	Add API version, input cropping , raw data output and context pointer

Table of Contents

Introduction

1.1 Purpose

This document gives details of the application programmer's interface of JPEG Encoder.

1.2 Scope

This document does not give the details of implementation of the JPEG Encoder. 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 Baseline encoding.

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).

1.4.2 General References

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

1.4.3 Freescale Multimedia References

- JPEG Encoder Application Programming Interface – jpeg_enc_api.doc
- JPEG Encoder Requirements Book - jpeg_enc_reqb.doc
- JPEG Encoder Test Plan - jpeg_enc_test_plan.doc
- JPEG Encoder Release notes - jpeg_enc_release_notes.doc
- JPEG Encoder Test Results – jpeg_enc_test_results.doc
- JPEG Encoder Performance Results – jpeg_enc_perf_results.doc
- JPEG Encoder Interface header – jpeg_enc_interface.h
- JPEG Encoder Test Application – jpeg_enc_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
DPI	Dots Per Inch
FSL	Freescale
IDCT	Inverse Discrete Cosine Transform - Transform used to convert samples from Frequency Domain to Spatial Domain
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

1.6 Document Location

docs/jpeg_enc

2 API Description

This section describes the steps followed by the application to call the JPEG Encoder. During each step the data structures used and the functions used will be explained.

Step 1: Allocate memory for Encoder object

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

```
typedef struct
{
      jpeg_enc_parameters parameters;
      jpeg_enc_memory_infos mem_infos;
      /* The application should not change the function pointer
           * once the init routine is called */
      JPEG_ENC_UINT8 (*jpeg_enc_push_output) (JPEG_ENC_UINT8 **
      out_buf_ptrptr, JPEG_ENC_UINT32 *out_buf_len_ptr, JPEG_ENC_UINT8
      flush,
      void * context, JPEG_ENC_MODE enc_mode);
      /* The application should not bother about the object entries below
       * and should not modify them */
      void *cinfo;
      void * context;
}jpeg_enc_object;
```

Description of structure *jpeg_enc_object*

jpeg_enc_parameters

Explained in next section

jpeg_enc_push_output

Call back function implemented by the application and called by the codec.

The application has to assign this function pointer to an appropriate value.

- `out_buf_ptr : Pointer to Pointer for output buffer`
- `out_buf_len_ptr : Pointer to the length of the output buffer`
- `flush : A flag to indicate to the application that codec no longer needs more buffers and that it may have partially filled up the previous output buffer. The extent to which the output buffer is filled up is given in the` *`out_buf_len`* `field.`
- *`context`* `: Context pointer, user can use it store their private variables`
- *`enc_mode`* `: The mode to which the encoder was configured (JPEG_ENC_MAIN_ONLY, JPEG_ENC_MAIN, JPEG_ENC_THUMB)`

Return Type:

If the call back function returns 1, it is considered successful. If the call back function returns '0' it is considered as a failure. In such a scenario, the codec shall return back to the calling application with an error message. `The encoder can not resume if such an error occurs.`

1. The codec allocates two memory locations a) U8 * out_buf_ptr b) U32 out_buf_len
2. Before calling the call back function for the first time, 'out_buf_ptr' is made NULL and 'out_buf_len' is made to 'zero' by the codec. These 2 variables are passed by value to the call back function
3. Inside the call back function, these 2 variables are assigned to proper values by the application
4. In all subsequent calls, these variables are not changed by the codec. This means, that codec has used up the entire buffer that was passed to it
5. Before calling the call back function for the last time, the 'flush' flag is set to 1 by the codec, to indicate that it no longer needs a new buffer. Also, 'out_buf_len' will be modified to reflect the exact amount of data filled up the codec.

Step 2: Fill up the parameters structure

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

```
typedef enum
{
      JPEG_ENC_MAIN_ONLY = 0,
      JPEG_ENC_MAIN,
      JPEG_ENC_THUMB
} JPEG_ENC_MODE;

typedef enum
{
      JPEG_ENC_YUV_444_NONINTERLEAVED,
      JPEG_ENC_YUV_422_NONINTERLEAVED,
      JPEG_ENC_YUV_420_NONINTERLEAVED,
      JPEG_ENC_YU_YV_422_INTERLEAVED,
      JPEG_ENC_YV_YU_422_INTERLEAVED,
      JPEG_ENC_UY_VY_422_INTERLEAVED,
      JPEG_ENC_VY_UY_422_INTERLEAVED
} JPEG_ENC_YUV_FORMAT;

typedef struct
{
    JPEG_ENC_UINT32 x_resolution[2];
    JPEG_ENC_UINT32 y_resolution[2];
    JPEG_ENC_UINT16 resolution_unit;
    JPEG_ENC_UINT16 ycbcr_positioning;
} jpeg_enc_IFD0_appinfo;

typedef struct
{
    /* currently empty. further tags/variables can be added if req */
    void * ptr;
} jpeg_enc_exifIFD_appinfo;

typedef struct
{
    JPEG_ENC_UINT32 x_resolution[2];
    JPEG_ENC_UINT32 y_resolution[2];
    JPEG_ENC_UINT16 resolution_unit;
```

```
} jpeg_enc_IFD1_appinfo;

/* No error checking is done on the jpeg_enc_exif_parameters
 * and the application is expected to have passed valid
 * values */
typedef struct
{
    jpeg_enc_IFD0_appinfo IFD0_info;
    jpeg_enc_exifIFD_appinfo exififd_info;
    jpeg_enc_IFD1_appinfo IFD1_info;
} jpeg_enc_exif_parameters;

typedef struct
{
    /* JFIF code for pixel size units */
    JPEG_ENC_UINT8 density_unit;
    /* Horizontal pixel density */
    JPEG_ENC_UINT16 X_density;
    /* Vertical pixel density */
    JPEG_ENC_UINT16 Y_density;
} jpeg_enc_jfif_parameters;

typedef struct
{
    JPEG_ENC_YUV_FORMAT yuv_format;
    JPEG_ENC_UINT8 quality;
    /* 1   JFIF (Send restart markers every MCU row),
     *     EXIF (Send restart markers for every 4 MCUs )
     * 0 - No restart markers */
    JPEG_ENC_UINT8 restart_markers;
    JPEG_ENC_UINT8 compression_method;  /* Baseline or Progressive */
    JPEG_ENC_MODE mode;
    /* Primary image widht/height should be set in
     * all the modes */
    JPEG_ENC_UINT16 primary_image_height;
    JPEG_ENC_UINT16 primary_image_width;
    JPEG_ENC_UINT16 y_height;
    JPEG_ENC_UINT16 u_height;
    JPEG_ENC_UINT16 v_height;
    JPEG_ENC_UINT16 y_width;
    JPEG_ENC_UINT16 u_width;
    JPEG_ENC_UINT16 v_width;
    /* cropping */
    JPEG_ENC_UINT16 y_left;
    JPEG_ENC_UINT16 y_top;
    JPEG_ENC_UINT16 y_total_width;
    JPEG_ENC_UINT16 y_total_height;
    JPEG_ENC_UINT16 u_left;
    JPEG_ENC_UINT16 u_top;
    JPEG_ENC_UINT16 u_total_width;
    JPEG_ENC_UINT16 u_total_height;
    JPEG_ENC_UINT16 v_left;
    JPEG_ENC_UINT16 v_top;
    JPEG_ENC_UINT16 v_total_width;
    JPEG_ENC_UINT16 v_total_height;
    /* raw data output ?*/
    JPEG_ENC_UINT8 raw_dat_flag;
    /* 1 - EXIF.... 0 - JFIF */
    JPEG_ENC_UINT8 exif_flag;
    jpeg_enc_exif_parameters exif_params;
    jpeg_enc_jfif_parameters jfif_params;
}jpeg_enc_parameters;
```


Description of structure *jpeg_enc_parameters*

compression_method

JPEG_ENC_SEQUENTIAL: Sequential Method

JPEG_ENC_PROGRESSIVE: Progressive Method

exif_flag

0 – JFIF file format, 1 – EXIF file format (If '0' is selected, *jpeg_enc_jfif_parameters* shall be filled. If '1' is selected, *jpeg_enc_exif_parameters* shall be filled)

mode

JPEG_ENC_MAIN_ONLY or JPEG_ENC_THUMB or JPEG_ENC_MAIN

quality

[0,100] Controls the quality and bpp of the Image. The quality factor scales the quantization values in the quantization table.

restart_markers

0 – No restart markers,

1 – Put restart markers every MCU Row for JFIF. In case of EXIF, put restart markers for every 4 MCUs

Description of structure *exif_parameters*

resolution_unit

Unit for measuring X Resolution and Y Resolution. The same unit is used for both Xresolution and Yresolution. If the image resolution is unknown 2(inches) is designated.

x_resolution

The number of pixels per Resolution unit in the ImageWidth direction. When the image resolution is unknown, 72[dpi] is designated. x_resolution[0] is the numerator and x_resolution[1] is the denominator.

y_resolution

The number of pixels per Resolution unit in the ImageLength direction. The same value as *x_resolution* is designated. y_resolution[0] is the numerator and y_resolution[1] is the denominator.

ycbcr_positioning

The position of chrominance components in relation to the luminance component. 1 = centered 2 = co-sited.

The codec does not perform any checks on the exif parameters that are passed. These values are directly written into the bit-stream. The following are some recommended values.

```
/* IFD0 params */
params->IFD0_info.x_resolution[0] = 72;
params->IFD0_info.x_resolution[1] = 1;
params->IFD0_info.y_resolution[0] = 72;
params->IFD0_info.y_resolution[1] = 1;
params->IFD0_info.resolution_unit = 2;
params->IFD0_info.ycbcr_positioning = 1;

/* IFD1 params */
params->IFD1_info.x_resolution[0] = 72;
params->IFD1_info.x_resolution[1] = 1;
params->IFD1_info.y_resolution[0] = 72;
```

```
params->IFD1_info.y_resolution[1] = 1;
params->IFD1_info.resolution_unit = 2;
```

Description of structure *jfif_parameters*

density_unit

Units for the X and Y pixel densities. Zero means than no densities are used. The next two fields specify the aspect ratio. One means that the next fields specify pixels per inch. Two means that they specify pixels per cm.

X_density

Horizontal pixel density

Y_density

Vertical pixel density

The codec does not perform any checks on the jfif parameters that are passed. These values are directly written into the bit-stream. The following are some recommended values.

```
/* Pixel size is unknown by default */
params->jfif_params.density_unit = 0;
/* Pixel aspect ratio is square by default */
params->jfif_params.X_density = 1;
params->jfif_params.Y_density = 1;
```

yuv_format

The encoder supports both non interleaved inputs as well as certain kinds of interleaved inputs. It supports 4 different types of interleaved inputs.

JPEG_ENC_YUV_444_NONINTERLEAVED - Non interleaved 444
JPEG_ENC_YUV_422_NONINTERLEAVED - Non interleaved 422
JPEG_ENC_ YUV_420_NONINTERLEAVED - Non interleaved 420
JPEG_ENC_YU_YV_422_INTERLEAVED - Interleaved 422 with pixel ordering as YU_YV
JPEG_ENC_ YV_YU_422_INTERLEAVED - Interleaved 422 with pixel ordering as YV_YU
JPEG_ENC_UY_VY_422_INTERLEAVED - Interleaved 422 with pixel ordering as UY_VY
JPEG_ENC_VY_UY_422_INTERLEAVED - Interleaved 422 with pixel ordering as VY_UY

y_width

Valid width of the Luminance Image buffer in number of pixels

y_height

Valid Height of the Luminance Image buffer in number of pixels

u_width

Valid width of the Cb Image buffer in number of pixels

u_height

Valid height of the Cb Image buffer in number of pixels

v_width

Valid width of the Cr Image buffer in number of pixels

v_height

Valid height of the Cr Image buffer in number of pixels

Width and height are given separately for each component, to cater to the case of odd image width and height. Say, if the image width is 213 and the sampling format 422, the width of U image buffer can be 106 or 107. The application has to provide 106 or 107 depending on the input format.

Detailed description:

1. If Y width (*y_width*) is even, the application has to pass (*u_width*) and (*v_width*) as follows :
   a) 444 Non Interleaved format : *u_width = v_width = y_width*
   b) 422 Non Interleaved format : *u_width = v_width = y_width/2*
   c) 420 Non Interleaved format : *u_width = v_width = y_width/2*
   d) 422 Interleaved format : *u_width = v_width = y_width*/2

2. If Y height (*y_height*) is even, the application has to pass (*u_height*) and (*v_height*) as follows :
   a) 444 Non Interleaved format : *u_height = v_height = y_height*
   b) 422 Non Interleaved format : *u_height = v_height = y_height*
   c) 420 Non Interleaved format : *u_height = v_height = y_height/2*
   d) 422 Interleaved format : *u_height = v_height = y_height*

3. If Y width (*y_width*) is odd, the application has to pass (*u_width*) and (*v_width*) as follows :
   a) 444 Non Interleaved format : *u_width = v_width = y_width*
   b) 422 Non Interleaved format : *u_width = v_width = y_width/2 + 1*
   c) 420 Non Interleaved format : *u_width = v_width = y_width/2 + 1*
   d) 422 Interleaved format :
      d.1) JPEG_ENC_YU_YV_422_INTERLEAVED :
      u_width = y_width/2 + 1; v_width = y_width/2
      d.2) JPEG_ENC_YV_YU_422_INTERLEAVED
      v_width = y_width/2 + 1; u_width = y_width/2
      d.3) JPEG_ENC_UY_VY_422_INTERLEAVED
      u_width = y_width/2 + 1; v_width = y_width/2
      d.4) JPEG_ENC_VY_UY_422_INTERLEAVED
      v_width = y_width/2 + 1; u_width = y_width/2

4. If Y height (*y_height*) is odd, the application has to pass (*u_height*) and (*v_height*) as follows :
   a) 444 Non Interleaved format : *u_height = v_height = y_height*
   b) 422 Non Interleaved format : *u_height = v_height = y_height*
   c) 420 Non Interleaved format : *u_height = v_height = y_height/2 + 1*
   d) 422 Interleaved format : *u_height = v_height = y_height*

y_left

Base address of valid Luminance Image buffer in number of pixels at horizontal direction

y_top

Base address of valid Luminance Image buffer in number of pixels at vertical direction

y_total_width

Total width of the Luminance Image buffer in number of pixels

y_total_height

Total height of the Luminance Image buffer in number of pixels

u_left

Base address of valid Cb Image buffer in number of pixels at horizontal direction

u_top

Base address of valid Cb Image buffer in number of pixels at vertical direction

u_total_width

Total width of the Cb Image buffer in number of pixels

u_total_height

Total height of the Cb Image buffer in number of pixels

v_left

Base address of valid Cr Image buffer in number of pixels at horizontal direction

v_top

Base address of valid Cr Image buffer in number of pixels at vertical direction

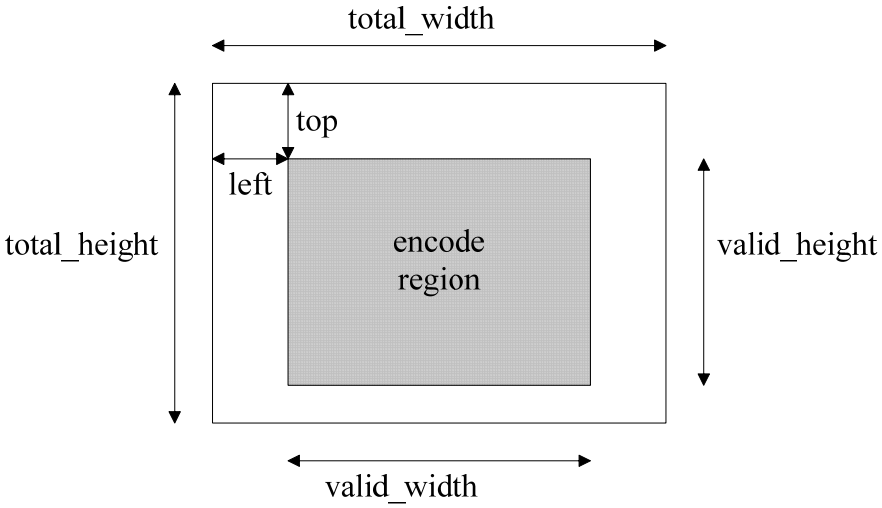
v_total_width

Total width of the Cr Image buffer in number of pixels

v_total_height

Total height of the Cr Image buffer in number of pixels

Below figure show the relation of above coordinate, we can implement cropping feature based on them.



raw_dat_flag

0 - jpeg file format (begin with SOI),
1 - raw data format (begin with DQT)

Step 3: Query memory requirements

The application asks the jpeg encoder about its data memory requirements.

C prototype:

```
JPEG_ENC_RET_TYPE  jpeg_enc_query_mem_req(jpeg_enc_object * enc_obj);
```

Arguments:

- Encoder Object pointer.

Return value:

- *JPEG_ENC_RET_TYPE*

JPEG Encoder would fill the *'size', 'alignment' and the* memory_type of the memory info structure present in the encoder object. The JPEG Encoder would also fill a data field (*no_entries*) that

contains the number of memory info structures it has filled. The '*memtype_fast_slow'* is a suggestion from the encoder and it is not a hard constraint. (i.e.) If the encoder requested for fast memory and if for some reason the application allocated slow memory the encoder would still run, but, may run slow. The '*memtype_static_scratch*' is a hard constraint. If the codec has requested for static memory, the application must give static memory.

Step 4: Data Memory allocation by application

In this step the application allocates the memory as requested by JPEG Encoder and fills up the '*memptr' of 'jpeg_enc_memory_info'* structures. The application must provide buffers of the 'memory type' and 'alignment' as requested by the codec.

When the codec is configured in progressive mode, the codec requests for very big buffers (to hold a frame worth of DCT coefficients) that must be double word (8byte) aligned. If the application does not provide buffers with the required alignment, the codec returns an error "JPEG_ENC_ERR_COEF_BUFFERS_UNALIGNED"

'priority' is a number starting from '0'. A buffer with priority '0' is more important than a buffer of priority '1' and a buffer of priority '1' is more important than a buffer of priority '2' and so on. Priority is given independently for FAST and SLOW memory requests.

Step 5: Memory allocation for input and output buffers

Input buffer : The application shall allocate memory to hold 1 MCU row of input and pass to the encode functions. In case of non interleaved input, the 'y_buff', 'u_buff', 'v_buff' have to be assigned properly and in case of interleaved 'i_buff' should be assigned properly. If the application uses 'jpeg_enc_encodeframe' API, then it must allocate memory to hold the entire frame.

Output Buffer: The application shall pass the output buffer pointer and the length of the buffer when the call back function '*jpeg_enc_push_output'* is called.

Step 6: Initialization routine

The application calls the encoder initialization routine

C prototype:

```
JPEG_ENC_UINT8          jpeg_enc_init(jpeg_enc_object * enc_obj);
```

Arguments:

- Encoder Object pointer.

Return value:

- *JPEG_ENC_RET_TYPE*

When this function is called, the encoder initializes the members of the object. The members can be data variables or pointers to data variables.

Step 7: Call the encode MCU Row routine

The application calls the MCU row encode routine.

C prototype:

```
JPEG_ENC_UINT8 jpeg_enc_encodemcurow (
                   jpeg_enc_object * enc_obj,
                   JPEG_ENC_UINT8 * i_buff,
                   JPEG_ENC_UINT8 * y_buff,
                   JPEG_ENC_UINT8 * u_buff,
                   JPEG_ENC_UINT8 * v_buff);
```

Arguments:

- enc_obj - Encoder Object pointer.
- i_buff - Pointer to the interleaved Y,U,V MCU Row buffer
- y_buff - pointer to Y MCU row buffer
- u_buff – pointer to cb MCU row buffer
- v_buff – pointer to cr MCU row buffer

Return value:

- *JPEG_ENC_RET_TYPE*

(The return value can be a successful return or an application error or a codec error. Refer the enum JPEG_ENC_RET_TYPE in appendix to see the list of return values)

If the buffer being encoded were in interleaved format, then the application would pass a pointer to an interleaved buffer in '*i_buff*' and pass '*NULL*' for '*y_buff*', '*u_buff*', and '*v_buff*'

<For interleaved input>

```
jpeg_enc_encodemcurow(enc_obj, i_buff, NULL, NULL, NULL);
```

```
<For non-interleaved input>
jpeg_enc_encodemcurow(enc_obj, NULL, y_buff, u_buff, v_buff);
```

The application has to keep calling this function in a loop till it receives a *JPEG_ENC_ERR_ENCODINGCOMPLETE* message.

Baseline:
If the encoder was configured for baseline, this function shall also emit bits into the output buffer.

Progressive:
If the encoder is configured for progressive, this function may not emit bits into the output buffer.

Note : Depending on the sampling format chosen, the 'y_buff', 'u_buff' and 'v_buff' should contain the following number of pixels.

Sampling Format : 420 :
y_buff -> Y_Image width*16
u_buff -> U_Image width*8

v_buff -> V_Image width*8

Sampling Format : 422 :
y_buff -> Y_Image width*8
u_buff -> U_Image width*8
v_buff -> V_Image width*8

Sampling Format : 444 :
y_buff -> Y_Image width*8
u_buff -> U_Image width*8
v_buff -> V_Image width*8

Step 8: Call Encode Pass MCU Row routine

This function is to be called by the application only when the encoder is configured for progressive.

C prototype:

```
JPEG_ENC_RET_TYPE jpeg_enc_encodepassmcurow (jpeg_enc_object *
enc_obj);
```

Arguments:

- enc_obj - Encoder Object pointer.

Return value:

- *JPEG_ENC_RET_TYPE*

The application does not need to understand the exact functionality of this function. The application has to keep calling this function in a loop till it receives a *JPEG_ENC_ERR_ENCODINGCOMPLETE* message. This function may not write bits into the output buffer each time it is called.

Using encode Frame API (Alternative for Step 7 and Step 8)
If the application has an entire frame available then it can call the 'jpeg_enc_encodeframe' API. In such a case, the application need NOT take step 7 and step 8

C prototype:

```
JPEG_ENC_RET_TYPE jpeg_enc_encodeframe (
                        jpeg_enc_object * enc_obj,
                        JPEG_ENC_UINT8 * i_buff,
                        JPEG_ENC_UINT8 * y_buff,
```

```
                        JPEG_ENC_UINT8 * u_buff,
                        JPEG_ENC_UINT8 * v_buff);
```

Arguments:

- enc_obj - Encoder Object pointer.
- i_buff - Pointer to the interleaved Y,U,V frame buffer
- y_buff - pointer to Y frame buffer
- u_buff – pointer to cb frame buffer
- v_buff – pointer to cr frame buffer

Return value:

- *JPEG_ENC_RET_TYPE*

(The return value can be a successful return or an application error or a codec error. Refer the enum JPEG_ENC_RET_TYPE in appendix to see the list of return values)

Step 9: Find Length and Offset

If the encoder was configured to run in thumb mode, this API shall be used.

When the function '*jpeg_enc_encodemcurow' or 'jpeg_enc_encodeframe'* is called in thumb mode, the codec will write '0' in the length field of the APP1/APP0Ext marker that is sent out. The codec will also write zero in the length field of a tag related to thumbnail (For EXIF). The application shall call the following API, to know the actual lengths and the positions of the length fields from the start of the file.

The application must call this function only after calling the encoder in thumb mode. The same object pointer used to call the encoder in THUMB mode, must be passed to this function.

C prototype:

```
JPEG_ENC_RET_TYPE jpeg_enc_find_length_position(
                        jpeg_enc_object * obj_ptr,
                        JPEG_ENC_UINT32 offset[],
                        JPEG_ENC_UINT8 value[],
                        JPEG_ENC_UINT8 *num_entries);
```

Arguments:

- enc_obj - Encoder Object pointer.
- Offset[] is an array of size JPEG_ENC_NUM_OF_OFFSETS to be passed by app
- Value[] is an array of size JPEG_ENC_NUM_OF_OFFSETS to be passed by app
- num_entries – Number of entries in the tables that codec has written

Return value:

- *JPEG_ENC_RET_TYPE*

The application must allocate memory for the tables and Offset[] and Value[] and call this function. Depending on the file format (JFIF/EXIF), the codec shall fill up the tables. The application must go back to the buffer/file system where it had stored the JPEG bit-stream and overwrite byte location offset[i] with value[i]

Step 10: Error codes

Any of the above functions can return an error if the encoder 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.
Here are some error codes with some explanations.

JPEG_ENC_ERR_NO_ERROR:
No Error
JPEG_ENC_ERR_ENCODINGCOMPLETE:
Frame encode completed. Returned by either jpeg_enc_encodemcurow or jpeg_enc_encodepassmcurow.
JPEG_ENC_ERR_OUTPUT_PTR_PASSED_TO_ENC_NULL:
The call back function to get output buffer returned a NULL ptr.
JPEG_ENC_ERR_OUTPUT_LEN_PASSED_TO_ENC_ZERO:
The call back function to get output buffer returned a buffer of zero length.
JPEG_ENC_ERR_OUTPUT_CALLBACK_FAIL :
The call back function to get output buffer returned a 'zero'.
JPEG_ENC_ERR_INVALID_YUV_FORMAT:
Invalid YUV format supplied.
JPEG_ENC_ERR_INVALID_QUALITY:
Quality parameter is not with in the range [0,100]
JPEG_ENC_ERR_INVALID_RESTART_MARKERS:
Parameter is not with in the range [0,1]
JPEG_ENC_ERR_INVALID_MODE:
Out of Range
JPEG_ENC_ERR_INVALID_WIDTH:
Out of Range [1,65500]
JPEG_ENC_ERR_INVALID_HEIGHT:
Out of Range [1,65500]
JPEG_ENC_ERR_INSUFFICIENT_MEMORY_REQUESTED_BY_ENC:
Insufficient memory requested by codec during query_mem (Codec error)
JPEG_ENC_ERR_MEMORY_PTRS_PASSED_TO_ENC_NULL:
The memory pointers passed are NULL
JPEG_ENC_ERR_IMAGE_PTRS_PASSED_TO_ENC_NULL:
The image buffer pointers passed are NULL

Step 11: Free memory

The application releases the memory that it allocated to JPEG Encoder if it no longer needs the encoder instance.
i.e. the application should free the memory it allocated on the basis of *'jpeg_enc_memory_info'* structure and also free the encoder object.

Important Notes: Calling Encoder for different Modes

If a JPEG image with out thumbnail is needed, then the encoder shall be configured in the mode 'MAIN_ONLY'.

If a JPEG Image with thumbnail is needed, then the encoder shall be configured for
a) 'THUMB'. All the steps 1 to 10 shall be done
b) 'MAIN'. All the steps 1 to 10 shall be done

The application has to then concatenate the buffers that are output in these modes. The application can also do first (b) and then (a). During (a), the SOI marker and the entire APP0/APP1 marker (APP0 or APP1 depending on 'exif_flag') is sent out. During (b), the rest of the JPEG image is sent out.

Step 12: API Version

This is the decoder function to get API version.

C prototype:

```
const char * jpege_CodecVersionInfo (void)
```

Arguments:

- None

Return value:

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

3 Example calling Routine

```
#include “jpeg_enc_interface.h”

/* Image buffers for QCIF YUV */
JPEG_ENC_UINT8 ybuff[176*144];
JPEG_ENC_UINT8 ubuff[88*72];
JPEG_ENC_UINT8 vbuff[88*72];

JPEG_ENC_UINT8 * ybuff_mcurow;
JPEG_ENC_UINT8 * ubuff_mcurow;
JPEG_ENC_UINT8 * vbuff_mcurow;

main()
{

        JPEG_ENC_UINT8 number_mem_info;
        JPEG_ENC_UINT16 i,j;
        JPEG_ENC_RET_TYPE error_code;
        JPEG_ENC_UINT8 * ibuff; /* Interleaved buffer ptr*/
        jpeg_enc_object * obj_ptr;
        jpeg_enc_memory_info * mem_info;
        JPEG_ENC_UINT32 size_outputbuffer;
        jpeg_enc_memory_info  mem_info;
        JPEG_ENC_UINT8 * outputbuffer_ptr;
        JPEG_ENC_UINT8 * outputbuffer_refptr;
        JPEG_ENC_UINT8 *temp_ptr;

        /* Allocate memory for JPEG encoder Object */
        obj_ptr = (jpeg_enc_object *) malloc(sizeof(jpeg_enc_object));

        /* Fill jpeg_enc_parameters’ structure */
        obj_ptr ->parameters.y_width = 176;
        obj_ptr->parameters.y_height = 144;
        obj_ptr ->parameters.u_width = 176/2;
        obj_ptr->parameters.u_height = 144/2;
        obj_ptr ->parameters.v_width = 176/2;
        obj_ptr->parameters.v_height = 144/2;

        obj_ptr ->parameters.y_left=0;
        obj_ptr ->parameters.y_top=0;
        obj_ptr ->parameters.y_total_width= obj_ptr ->parameters.y_width;
        obj_ptr ->parameters.y_total_height= obj_ptr ->parameters.y_height;
        obj_ptr ->parameters.u_left=0;
        obj_ptr ->parameters.u_top=0;
```

```
obj_ptr ->parameters.u_total_width= obj_ptr ->parameters.u_width;
obj_ptr ->parameters.u_total_height= obj_ptr ->parameters.u_height;
obj_ptr ->parameters.v_left=0;
obj_ptr ->parameters.v_top=0;
obj_ptr ->parameters.v_total_width= obj_ptr ->parameters.v_width;
obj_ptr ->parameters.v_total_height= obj_ptr ->parameters.v_height;
obj_ptr ->parameters. raw_dat_flag=0;

obj_ptr->parameters.quality = 50;                    /* Range [0..100]*/
obj_ptr->parameters.yuv_format = JPEG_ENC_YUV_420_NONINTERLEAVED; /* 420
format */
obj_ptr->parameters.restart_markers = 1;             /*  1 – Put Restart markers */
obj_ptr->parameters.mode = JPEG_ENC_MAIN_ONLY;
obj_ptr->parameters. compression_method = JPEG_ENC_SEQUENTIAL;
obj_ptr->parameters. exif_flag = 0;  /*Generate JFIF file */.
obj_ptr->parameters.jfif_params.density_unit = 0; /* No densities are used */
obj_ptr->parameters.jfif_params.X_density = 1;
obj_ptr->parameters.jfif_params.Y_density = 1;
/* Initialize the function pointer. This pointer should not be
changed once init is called */
obj_ptr->jpeg_enc_push_output = push_output;


/* query JPEG Encoder its memory requirement */
error_code  = jpeg_enc_query_mem_req(obj_ptr);
if(error_code != JPEG_ENC_ERR_NO_ERROR)
{
        /* Application takes corrective action based on “error_code”
        if it’s an application error*/
}

number_mem_info   =  obj_ptr-> jpeg_enc_memory_infos.no_entries;

/* Allocate memory */
for (i = 0; i < number_mem_info  ; i++)
{
        /* This example code ignores the ‘alignment’ and memory type, but some other
        applications might want to allocate memory based on this */
        mem_info = &(obj_ptr-> mem_infos.mem_info[i]);
        mem_info->memptr = (void *)  malloc(mem_info->siize);
}

/* Call Encoder Init */
error_code =  jpeg_enc_init(obj_ptr);

/* Check if Encoder returned an error */
if(error_code != JPEG_ENC_ERR_NO_ERROR)
{
        /* Application takes corrective action based on “error_code”
```

```
        if it's an application error*/
}


/* Encode Frame */
ibuff = NULL;

/* Fill up the contents of  y_buff, u_buff and v_buff with valid image data*/
error_code = JPEG_ENC_ERR_NO_ERROR;
ybuff_mcurow = ybuff;
ubuff_mcurow = ubuff;
vbuff_mcurow = vbuff;

while(error_code == JPEG_ENC_ERR_NO_ERROR)
{
        error_code = jpeg_enc_encodemcurow(obj_ptr, i_buff,
        y_buff_mcurow,u_buff_mcurow,v_buff_mcurow );
        ybuff_mcurow += 176*16;
        ubuff_mcurow += 88*8;
        vbuff_mcurow += 88*8;

}

if(error_code != JPEG_ENC_ERR_ENCODINGCOMPLETE)
{
        /* Application takes corrective action based on "error_code"
        if it's an application error*/
}

if(params->compression_method == JPEG_ENC_PROGRESSIVE)
{
      while(1)
      {
            return_val = jpeg_enc_encodepassmcurow(obj_ptr);

           if(return_val != JPEG_ENC_ERR_NO_ERROR)
           {
              break;
           }
      }

      if(return_val == JPEG_ENC_ERR_ENCODINGCOMPLETE)
      {
           printf("Encoding of Progressive Image completed\n");
      }
      else
      {
          printf("JPEG encoder returned an error when
```

```
                    jpeg_enc_encodepassmcurow was called \n");
                    printf("Return Val %d\n",return_val);
                   /* Application takes corrective action based on the error code*/
            }
        }

        /* Free data memory memory */
        for (i = 0; i < number_mem_info  ; i++)
        {
                mem_info = &(obj_ptr-> mem_infos.mem_info[i]);
                free(mem_info->memptr);
        }

                free(obj_ptr);

} /* End of main */

 /*
 This function is an example implementation of call back
 function 'jpeg_enc_push_output'. This function reads the
 data from the buffer and writes the content into an
 output file. Currently this function ignores the ‘enc_mode’
 and  ‘obj_ptr’. Final implementations may need to use the same
  */

#define APP_OUT_BUFFER_SIZE 1000
U8 APP_OUT_BUFFER[APP_OUT_BUFFER_SIZE];

JPEG_ENC_UINT8 push_output(JPEG_ENC_UINT8 ** out_buf_ptrptr, JPEG_ENC_UINT32
*out_buf_len_ptr, JPEG_ENC_UINT8 flush, void * context, JPEG_ENC_MODE enc_mode)
{
   JPEG_ENC_UINT32 i,j;
   JPEG_ENC_UINT8 temp;
   if(*out_buf_ptrptr == NULL)
   {
      /* This function is called for the 1'st time from the
       * codec */
      *out_buf_ptrptr = APP_OUT_BUFFER;
      *out_buf_len_ptr = APP_OUT_BUFFER_SIZE;
   }
   else if(flush == 1)
   {
      /* Flush the buffer*/
      /* This example code flushes the buffer into a file */
      for(i = 0; i < *out_buf_len_ptr;i++)
      {
         fputc(*(*out_buf_ptrptr + i),fp_out);
      }
```

```
        fclose(fp_out);
    }
    else
    {
        /* This example code flushes the buffer into a file */
        for(i = 0; i < APP_OUT_BUFFER_SIZE ;i++)
        {
            fputc(*(*out_buf_ptrptr + i),fp_out);
        }
        /* Now provide a new buffer */
        *out_buf_ptrptr = APP_OUT_BUFFER;
        *out_buf_len_ptr = APP_OUT_BUFFER_SIZE;
    }
    return(1); /* Success */
}
```

Appendix A Debug Logs

If debug logs are needed the codec is to be recompiled with the hash define “DBG_LVL” set to appropriate value in ‘jpeg_enc_interface.h” file
By default no debug logs are enabled.

#define DBG_LVL 0x0

```
/* Debug Level : "0" - No Debug Logs
 * Bit position 0 - Debug Level 0 -  Function entry/exit
 * Bit position 1 - Debug Level 1 -  Image Level
 * Bit position 2 - Debug Level 2 -  MCU level
 * Bit position 3 - Debug Level 3 -  DCT Huffman Level
 * Bit position 4 - Debug Level 4 -  Error/Warning Level
 */
```

Appendix B

File name : jpeg_enc_interface.h

```
#ifndef JPEG_ENC_INTERFACE_H
#define JPEG_ENC_INTERFACE_H

typedef unsigned char JPEG_ENC_UINT8;
typedef unsigned short JPEG_ENC_UINT16;
typedef unsigned long JPEG_ENC_UINT32;
typedef char JPEG_ENC_INT8;
typedef short JPEG_ENC_INT16;
typedef long JPEG_ENC_INT32;

#define JPEG_ENC_FAST_MEMORY 0
#define JPEG_ENC_SLOW_MEMORY 1
#define JPEG_ENC_STATIC_MEMORY 2
#define JPEG_ENC_SCRATCH_MEMORY 3

/* Debug Level : "0" - No Debug Logs
 * Bit position 0 - Debug Level 0 -  Function entry/exit
 * Bit position 1 - Debug Level 1 -  Image Level
 * Bit position 2 - Debug Level 2 -  MCU level
 * Bit position 3 - Debug Level 3 -  DCT Huffman Level
 * Bit position 4 - Debug Level 4 -  Error/Warning Level
 */
#define DBG_LVL 0x0

typedef enum
{
      JPEG_ENC_YUV_444_NONINTERLEAVED,
      JPEG_ENC_YUV_422_NONINTERLEAVED,
      JPEG_ENC_YUV_420_NONINTERLEAVED,
      JPEG_ENC_YU_YV_422_INTERLEAVED,
      JPEG_ENC_YV_YU_422_INTERLEAVED,
      JPEG_ENC_UY_VY_422_INTERLEAVED,
      JPEG_ENC_VY_UY_422_INTERLEAVED
} JPEG_ENC_YUV_FORMAT;

#define JPEG_ENC_ALIGN_8BIT 0     /* Align start of buffer to 8 bit
boundary */
#define JPEG_ENC_ALIGN_16BIT 1    /* Align start of buffer to a 16 bit
boundary */
#define JPEG_ENC_ALIGN_32BIT 2    /* Align start of buffer to a 32 bit
boundary */
#define JPEG_ENC_ALIGN_64_BIT 3    /* Align start of buffer to a 64 bit
boundary */
                          /* Double word alignment */

#define JPEG_ENC_YUV_444 1
```

```
#define JPEG_ENC_YUV_422 2
#define JPEG_ENC_YUV_420 3

/* These defines are used for setting the compression_method
 * in the params structure */
#define JPEG_ENC_SEQUENTIAL 0
#define JPEG_ENC_PROGRESSIVE 1

#define JPEG_ENC_NUM_OF_OFFSETS 6

/* This dictates the maximum size of the array of jpeg_enc_memory_info.
 * The codec is not expected to request more than
JPEG_ENC_MAX_MEMORY_INFO_ENTRIES
 * buffers */
#define JPEG_ENC_MAX_MEMORY_INFO_ENTRIES 5

typedef enum
{
    /* Successful return values */
    JPEG_ENC_ERR_NO_ERROR = 0,
    JPEG_ENC_ERR_ENCODINGCOMPLETE,

    /*Application Errors */
    JPEG_ENC_ERR_RECL_START = 101,
    JPEG_ENC_ERR_INVALID_YUV_FORMAT,
    JPEG_ENC_ERR_INVALID_QUALITY,
    JPEG_ENC_ERR_INVALID_RESTART_MARKERS,
    JPEG_ENC_ERR_INVALID_MODE,
    JPEG_ENC_ERR_INVALID_COMPMETHOD,
    JPEG_ENC_ERR_INVALID_WIDTH,
    JPEG_ENC_ERR_INVALID_HEIGHT,
    JPEG_ENC_ERR_MEMORY_PTRS_PASSED_TO_ENC_NULL,
    JPEG_ENC_ERR_IMAGE_PTRS_PASSED_TO_ENC_NULL,
    JPEG_ENC_ERR_OUTPUT_PTR_PASSED_TO_ENC_NULL,
    JPEG_ENC_ERR_OUTPUT_LEN_PASSED_TO_ENC_ZERO,
    JPEG_ENC_ERR_OUTPUT_CALLBACK_FAIL,
    JPEG_ENC_ERR_THUMB_SIZE_TOO_BIG,
    JPEG_ENC_ERR_PROGRESSIVE_NOT_COMPILED,
    JPEG_ENC_ERR_EXIF_NOT_COMPILED,
    JPEG_ENC_ERR_COEF_BUFFERS_UNALIGNED,
    JPEG_ENC_ERR_RECL_END,

    /*Codec Errors*/
    JPEG_ENC_ERR_FATAL_START = 151,
    JPEG_ENC_ERR_INSUFFICIENT_MEMORY_REQUESTED_BY_ENC,
    JPEG_ENC_ERR_INVALID_MCUROW,
    JPEG_ENC_ERR_ARITH_NOTIMPL,
    JPEG_ENC_ERR_BAD_BUFFER_MODE,
    JPEG_ENC_ERR_BAD_DCT_COEF,
    JPEG_ENC_ERR_BAD_HUFF_TABLE,
    JPEG_ENC_ERR_BAD_IN_COLORSPACE,
    JPEG_ENC_ERR_BAD_J_COLORSPACE,
    JPEG_ENC_ERR_BAD_LENGTH,
    JPEG_ENC_ERR_BAD_LIB_VERSION,
```



```
    JPEG_ENC_ERR_BAD_MCU_SIZE,
    JPEG_ENC_ERR_BAD_PRECISION,
    JPEG_ENC_ERR_BAD_PROG_SCRIPT,
    JPEG_ENC_ERR_BAD_SAMPLING,
    JPEG_ENC_ERR_BAD_SCAN_SCRIPT,
    JPEG_ENC_ERR_BAD_STATE,
    JPEG_ENC_ERR_BAD_STRUCT_SIZE,
    JPEG_ENC_ERR_CANT_SUSPEND,
    JPEG_ENC_ERR_COMPONENT_COUNT,
    JPEG_ENC_ERR_DQT_INDEX,
    JPEG_ENC_ERR_EMPTY_IMAGE,
    JPEG_ENC_ERR_HUFF_CLEN_OVERFLOW,
    JPEG_ENC_ERR_HUFF_MISSING_CODE,
    JPEG_ENC_ERR_IMAGE_TOO_BIG,
    JPEG_ENC_ERR_MISSING_DATA,
    JPEG_ENC_ERR_NOT_COMPILED,
    JPEG_ENC_ERR_NO_HUFF_TABLE,
    JPEG_ENC_ERR_NO_QUANT_TABLE,
    JPEG_ENC_ERR_OUT_OF_MEMORY,
    JPEG_ENC_ERR_TOO_LITTLE_DATA,
    JPEG_ENC_ERR_WIDTH_OVERFLOW,
    JPEG_ENC_ERR_FATAL_END,
}JPEG_ENC_RET_TYPE;

/* JPEG_ENC_ERR_COEF_BUFFERS_UNALIGNED - This is the error returned
 * by the codec if the COEF buffers passed by the application are not
 * aligned. The codec will not return such an error if the other buffers
 * are not aligned. Except the COEF buffers, the codec internally aligns
 * all other memory chunks passed to it. COEF buffers are requested
 * by codec only in JPEG_ENC_PROGRESSIVE compression mode */


typedef struct
{
    JPEG_ENC_UINT8 alignment;                /* JPEG_ENC_ALIGN_8BIT OR
16 or 32  or 64*/
    JPEG_ENC_UINT32 size;                    /* Size in number of bytes
*/
    /* JPEG_ENC_FAST_MEMORY (OR) JPEG_ENC_SLOW_MEMORY  - Only a
Recommendation from codec */
    JPEG_ENC_UINT16 memtype_fast_slow;
    /* STATIC memory or SCRATCH Memory. It is mandatory that application
     * allocates static memory as requested. However, for the ones that
     * are requested as 'scratch', the application can choose to allocate
     * either static or scratch */
    JPEG_ENC_UINT16 memtype_static_scratch;
    /* Priority of memory chunk */
    JPEG_ENC_UINT8 priority;
    /* ptr to the memory allocated by application */
    void * memptr;
} jpeg_enc_memory_info;


/* Current JPEG implementation does not request multiple memory chunks.
```

```
 * One single memory chunk for FAST and one single chunk for SLOW is
being
 * requested. So, now, priority is not of importance.
 * However, during system integration, if the application has only
limited
 * FAST memory available, then we will divide the codec FAST memory
requests
 * into multiple smaller memory chunks and give appropriate priority. */

/* In the current JPEG Encoder implementation, all the memory requested
is
 * of type STATIC. */

typedef struct
{
    JPEG_ENC_UINT8 no_entries;
    jpeg_enc_memory_info mem_info[JPEG_ENC_MAX_MEMORY_INFO_ENTRIES];
}jpeg_enc_memory_infos;

typedef enum
{
    JPEG_ENC_MAIN_ONLY = 0,
    JPEG_ENC_MAIN,
    JPEG_ENC_THUMB
}JPEG_ENC_MODE;

typedef struct
{
    JPEG_ENC_UINT32 count;  /* count is size of the jpeg_enc_tag in bytes
*/
    void* ptr;
} jpeg_enc_tag;

typedef struct
{
    JPEG_ENC_UINT32 x_resolution[2];
    JPEG_ENC_UINT32 y_resolution[2];
    JPEG_ENC_UINT16 resolution_unit;
    JPEG_ENC_UINT16 ycbcr_positioning;
} jpeg_enc_IFD0_appinfo;

typedef struct
{
    /* currently empty. further tags/variables can be added if req */
    void * ptr;
} jpeg_enc_exifIFD_appinfo;

typedef struct
{
    JPEG_ENC_UINT32 x_resolution[2];
    JPEG_ENC_UINT32 y_resolution[2];
    JPEG_ENC_UINT16 resolution_unit;
} jpeg_enc_IFD1_appinfo;
```

```
/* No error checking is done on the jpeg_enc_exif_parameters
 * and the application is expected to have passed valid
 * values */
typedef struct
{
    jpeg_enc_IFD0_appinfo IFD0_info;
    jpeg_enc_exifIFD_appinfo exififd_info;
    jpeg_enc_IFD1_appinfo IFD1_info;
} jpeg_enc_exif_parameters;

/* No error checking is done on the jpeg_enc_jfif_parameters
 * and the application is expected to have passed valid
 * values */

/*
 * These three values are not used by the JPEG code, merely copied
 * into the JFIF APP0 marker.  density_unit can be 0 for unknown,
 * 1 for dots/inch, or 2 for dots/cm.  Note that the pixel aspect
 * ratio is defined by X_density/Y_density even when density_unit=0
 */
typedef struct
{
    /* JFIF code for pixel size units */
    JPEG_ENC_UINT8 density_unit;
    /* Horizontal pixel density */
    JPEG_ENC_UINT16 X_density;
    /* Vertical pixel density */
    JPEG_ENC_UINT16 Y_density;
} jpeg_enc_jfif_parameters;


/* Refer API doc for description of these parameters */
typedef struct
{
    JPEG_ENC_YUV_FORMAT yuv_format;
    JPEG_ENC_UINT8 quality;
    /* 1   JFIF (Send restart markers every MCU row),
     *     EXIF (Send restart markers for every 4 MCUs )
     * 0 - No restart markers */
    JPEG_ENC_UINT8 restart_markers;
    JPEG_ENC_UINT8 compression_method;  /* Baseline or Progressive */
    JPEG_ENC_MODE mode;
    /* Primary image widht/height should be set in
     * all the modes */
    JPEG_ENC_UINT16 primary_image_height;
    JPEG_ENC_UINT16 primary_image_width;
    JPEG_ENC_UINT16 y_height;
    JPEG_ENC_UINT16 u_height;
    JPEG_ENC_UINT16 v_height;
    JPEG_ENC_UINT16 y_width;
    JPEG_ENC_UINT16 u_width;
    JPEG_ENC_UINT16 v_width;
    /* cropping */
    JPEG_ENC_UINT16 y_left;
```



```
    JPEG_ENC_UINT16 y_top;
    JPEG_ENC_UINT16 y_total_width;
    JPEG_ENC_UINT16 y_total_height;
    JPEG_ENC_UINT16 u_left;
    JPEG_ENC_UINT16 u_top;
    JPEG_ENC_UINT16 u_total_width;
    JPEG_ENC_UINT16 u_total_height;
    JPEG_ENC_UINT16 v_left;
    JPEG_ENC_UINT16 v_top;
    JPEG_ENC_UINT16 v_total_width;
    JPEG_ENC_UINT16 v_total_height;
    /* raw data output ?*/
    JPEG_ENC_UINT8 raw_dat_flag;
    /* 1 - EXIF.... 0 - JFIF */
    JPEG_ENC_UINT8 exif_flag;
    jpeg_enc_exif_parameters exif_params;
    jpeg_enc_jfif_parameters jfif_params;
} jpeg_enc_parameters;

typedef struct
{
    jpeg_enc_parameters parameters;
    jpeg_enc_memory_infos mem_infos;
    /* The application should not change the function pointer
     * once the init routine is called */
    JPEG_ENC_UINT8 (*jpeg_enc_push_output) (JPEG_ENC_UINT8 **
out_buf_ptrptr,JPEG_ENC_UINT32 *out_buf_len_ptr,
    JPEG_ENC_UINT8 flush, void * context, JPEG_ENC_MODE enc_mode);
    /* The application should not bother about the object entries below
     * and should not modify them */
    void *cinfo;
    void * context ;
} jpeg_enc_object;

 JPEG_ENC_RET_TYPE jpeg_enc_query_mem_req(jpeg_enc_object * enc_obj);
 JPEG_ENC_RET_TYPE jpeg_enc_init(jpeg_enc_object * enc_obj);
 JPEG_ENC_RET_TYPE jpeg_enc_flush_outputbuffer(jpeg_enc_object *
obj_ptr);
 /* Application has to write the value at the offset.
  * See the application.c and API doc for more details.
  * The size of the arrays is JPEG_ENC_NUM_OF_OFFSETS 6 */
 JPEG_ENC_RET_TYPE jpeg_enc_find_length_position(jpeg_enc_object *
obj_ptr,
                                  JPEG_ENC_UINT32 offset[],
JPEG_ENC_UINT8 value[],
                                  JPEG_ENC_UINT8 *num_entries);
 /* Row level APIs */
 JPEG_ENC_RET_TYPE jpeg_enc_encodemcurow (jpeg_enc_object * enc_obj,
JPEG_ENC_UINT8 * i_buff,
                          JPEG_ENC_UINT8 * y_buff, JPEG_ENC_UINT8 *
u_buff, JPEG_ENC_UINT8 * v_buff);
 JPEG_ENC_RET_TYPE jpeg_enc_encodepassmcurow (jpeg_enc_object * enc_obj);
 /* Frame Level API. This can be used for both Baseline and Progressive
*/
```

```
 JPEG_ENC_RET_TYPE jpeg_enc_encodeframe (jpeg_enc_object * enc_obj,
JPEG_ENC_UINT8 * i_buff,
                            JPEG_ENC_UINT8 * y_buff, JPEG_ENC_UINT8 *
u_buff, JPEG_ENC_UINT8 * v_buff);
#endif /* JPEG_ENC_INTERFACE_H */
```