08-6468-SIS-ZCH66
MAR 4, 2009
2.4

Application Programmers Interface for PNG Decoder

ABSTRACT:

Application Programmers Interface for PNG Decoder

KEYWORDS:

Multimedia codecs, Image, PNG

APPROVED:

Wang Zening

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	30-Aug-2004	Sameer Rapate Shailesh R	Created
1.0	07-Sep--2004	Sameer Rapate Shailesh R	Updated with review comments
1.1	05-Nov--2004	Sameer Rapate Shailesh R	Updated to support transparency and other additional features
1.2	07-Dec--2004	Sameer Rapate Shailesh R	Updated for release 1.0
1.3	04-Apr-2005	Sameer Rapate Shailesh R	Added description for RGB formats
1.4	20-Jan--2006	Atul Duggal	Edited variable names for output formats and scaling modes.
1.5	25-Jan-2006	Kunal Goel	Updated with review comments
2.0	06-Feb-2006	Lauren Post	Draft version using new format
2.1	29-Mar-2006	Sanjeev Kumar	Document review
2.2	05-Dec-2006	Durgaprasad Bilagi	Modified for new output format, grayscale
2.3	11-Dec-2008	Eagle Zhou	Add BGR format; Add API to query decoder version; Add recommending output format to application when return PNG_DEC_INVALID_OUTFORMAT
2.4	04-Mar-2009	Eagle Zhou	Add frame decode API

Table of Contents

Introduction

1.1 Purpose

This document gives the application programmer's interface for the PNG Decoder. The purpose of this document is to specify the functional interface of the PNG decoder.

1.2 Scope

This document describes only the functional interface of the PNG decoder. It does not describe the internal design of the decoder. Specifically, it describes only those functions needed by calling application to use the decoder. This PNG Decoder uses *libpng 1.2.8* (www.libpng.org/pub/png/pngcode.html) and *zlib 1.2.1* (www.zlib.org).

1.3 Audience Description

The reader is expected to have basic understanding of PNG decoding. The intended audience for this document is the development community who wish to use the PNG decoder in their systems.

1.4 References

1.4.1 Standards

- PNG Specifications 1.0 (RFC 2083) (http://www.libpng.org/pub/png/spec)

1.4.2 References

- Compressed Image File formats by John Miano, ACM Press/Addison Wesley Longman.
- Libpng 1.2.7 (http://www.libpng.org/pub/png/libpng.html)
- Zlib 1.2.1(www.zlib.org)
- ZLIB data format v3.3 (RFC 1950)
- 'Deflate' compressed data format – Spec v1.3 (RFC 1951)

1.4.3 Freescale Multimedia References

- PNG Decoder Application Programming Interface – png_dec_api.doc
- PNG Decoder Requirements Book - png_dec_reqb.doc
- PNG Decoder Test Plan - png_dec_test_plan.doc
- PNG Decoder Release notes - png_dec_release_notes.doc
- PNG Decoder Test Results – png_dec_test_results.doc
- PNG Decoder Performance Results – png_dec_perf_results.doc
- PNG Decoder Interface Header – png_dec_interface.h
- PNG Decoder Application Code – png_test_wrapper.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
API	Application Programming Interface
ARM	Advanced RISC Machine
FSL	Freescale
OS	Operating System
PNG	Portable Network Graphics
PNM	**P**ortable a**N**y**M**ap file. It refers collectively to PBM, PGM, and PPM formats (**P**ortable **B**i-level-image **M**ap, **P**ortable **G**rayscale Map and **P**ortable **P**ixel **M**ap respectively)
RGB	Raw pixel data organized in the order of Red, green and blue components. **RGB888** denotes 8 bits per pixel each for R, G, and B components
TBD	To Be Determined
UNIX	Linux PC x/86 C-reference binaries

1.6 Document Location

docs/png_dec

2 API Description

This PNG decoder uses *libpng* 1.2.8 (www.libpng.org/pub/png/pngcode.html) and zlib 1.2.1 (www.zlib.org)

The external software interface to this PNG Decoder consists of the following functions:

`PNG_dec_init`: Initialization API

`PNG_decode_row`:: API to decode the PNG file row by row

`PNG_cleanup`: This cleanup API is responsible for 'destroying' all the PNG structures allocated during initialization

Functions used for allocating memory, freeing memory and for reading the data from input stream need to be implemented by the calling application. The PNG decoder API uses function pointers (refer to section 4.6 for details) to invoke these functions.

`PNG_app_malloc: Function pointer to the function that` allocates memory (the function needs to be implemented by calling application)

`PNG_app_free: Function pointer to the function that` frees up the allocated memory (the function needs to be implemented by calling application)

`PNG_app_read_data: Function pointer to the function that reads data from the input stream` (the function needs to be implemented by calling application)

An example of the calling sequence is shown below. (The functions implemented by calling application have not been included, since they are invoked internally by the depicted API functions)

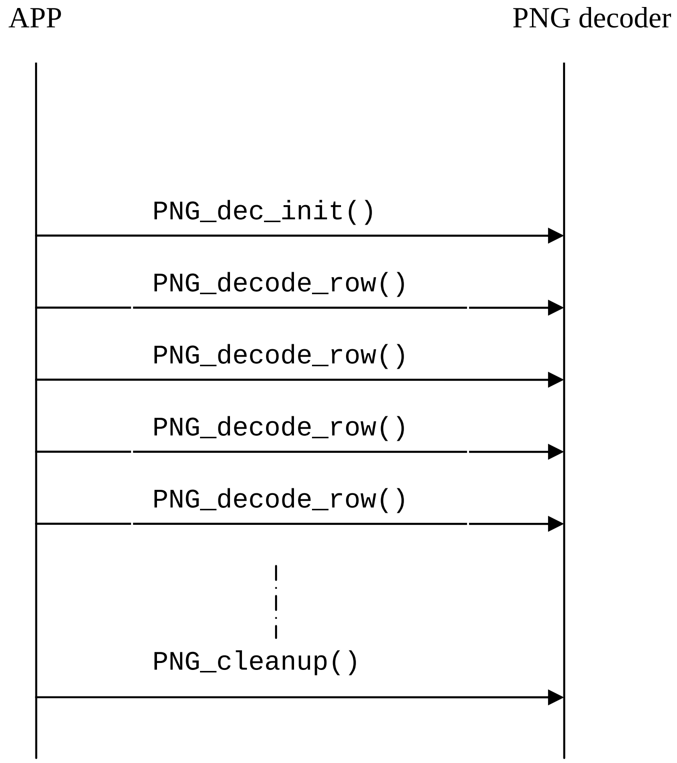

Fig 2. Ladder diagram of calling sequence

3 PNG Decoder – Data Structures

3.1 BASIC DATA TYPES [1]

```
typedef                 int                     PNG_INT32;
typedef                 unsigned int            PNG_UINT32;
typedef                 char                    PNG_INT8;
typedef                 unsigned char           PNG_UINT8;
typedef                 short                   PNG_INT16;
typedef                 unsigned short          PNG_UINT16;
```

3.2 PNG_DECODER_OBJECT

In order to call any PNG decode function, the application that calls the PNG decoder needs to create a new instance of the decoder object. The calling application maintains a list of pointers to all currently active instances of the object, and manages them. The caller should also ensure that there is sufficient memory available to run the instance that is being created. All data structures used by the PNG functions need to be allocated by the caller on a per instance basis, and hence are part of PNG Decoder Object instance structure. Input data that is required for this particular instance of the decoder should be filled into the instance structure by the calling function. After completion of the intended functions, the caller needs to delete the instance and free all memory associated with it.

```
typedef struct {
    PNG_Decoder_Info_Init dec_info_init;
    PNG_Decoder_Params dec_param;
    void *png_ptr;
    void *info_ptr;
    void *end_info_ptr;
    PNG_INT32 *pixels;
    PNG_UINT8 *row_buf;
    PNG_UINT8 *out_interlaced_buf;
    void *pAppContext;
    PNGD_RET_TYPE (*PNG_app_read_data)(void *, PNG_UINT8*,
                             PNG_UINT32, PNG_UINT32);

    void * (*PNG_app_malloc)(void *, PNG_UINT32);
    void (*PNG_app_free)(void *);
    PNG_UINT32    rows_decoded;
} PNG_Decoder_Object;
```

[1] The above typedefs are based on the current development platforms

Element	Description
`PNG_Decoder_Params dec_param`	Caller needs to fill this structure before calling the decoder functions
`PNG_Decoder_Info_Init dec_info_init`	PNG_dec_init fills this structure up, which can be used by the caller
`png_ptr`	Pointer to PNG internal structure. This is a codec specific structure not needed by the caller
`Info_ptr`	Pointer to PNG info structure used by the codec for storing png information. The caller does not need this.
`row_buf`	Pointer to Buffer for non interlaced data
`Pixels`	Pointer to Row buffer for pixels colr conversion
`out_interlaced_buf`	Pointer to Buffer for interlaced data
`pAppContext`	Pointer to Application data
`PNG_app_malloc`	Pointer to malloc function implemented by calling application
`PNG_app_PNG_read_data`	Pointer to app_PNG_read_data to be implemented by calling application
`PNG_app_free`	Pointer to free function to be implemented by calling application
`Rows_decoded`	Number of rows decoded

3.3 PNG_DECODER_PARAMS

PNG_Decoder_Params needs to be filled by the application calling the PNG decoder before it calls the Decoder functions. The calling application needs to indicate the desired output format. In case the calling application needs the PNG decoder to also rescale the decoded output, it needs to set the `png_scaling_mode` structure member to 1. In such a case, the calling application also provides information on the width and height of output to be displayed[2]. It should be noted that it is the responsibility of the calling application to ensure that all the structure members of PNG_Decoder_Params are initialized to the correct values.

[2] Note that only scaling down is supported – if the output dimensions configured are greater than the PNG image size as it occurs in the header, the PNG image is left unscaled.

```
typedef struct
{
    png_output_format   outformat;
    png_scaling_mode    scale_mode;
    PNG_UINT16          output_width;
    PNG_UINT16          output_height;
} PNG_Decoder_Params;
```

Element	Description
png_output_format	Enum for output formats supported
png_scaling_mode	Enum for scaling mode
output_width	Width of output to be displayed (to be specified by calling application if software scaling is enabled)
output_height	Height of output to be displayed (to be specified by calling application if software scaling is enabled)

```
typedef enum
{
            E_PNG_OUTPUTFORMAT_RGB888,
            E_PNG_OUTPUTFORMAT_RGB565,
            E_PNG_OUTPUTFORMAT_RGB555,
            E_PNG_OUTPUTFORMAT_RGB666,
            E_PNG_OUTPUTFORMAT_BGR888,
            E_PNG_OUTPUTFORMAT_BGR565,
            E_PNG_OUTPUTFORMAT_BGR555,
            E_PNG_OUTPUTFORMAT_BGR666,
            E_PNG_OUTPUTFORMAT_ARGB,
            E_PNG_OUTPUTFORMAT_BGRA,
            E_PNG_OUTPUTFORMAT_AG,
            E_PNG_OUTPUTFORMAT_G
            E_PNG_LAST_OUTPUT_FORMAT,
}png_output_format;
```

E_PNG_OUTPUTFORMAT_AG: This signifies the output format to be Grayscale with alpha channel.
E_PNG_OUTPUTFORMAT_G: This signifies the output format to be only Grayscale.

For more details on these formats refer to Appendix [A]

This enum for the output format indexes into an array of function pointers – the functions are responsible for rendering the output in the required format.

```
typedef enum
{
      E_PNG_NO_SCALE,                      /* No software scaling */
      E_PNG_INT_SCALE_PRESERVE_AR,       /* Software scaling using
                                 integer scaling factor preserving
                                 pixel aspect ratio */
      E_PNG_LAST_SCALE_MODE
} png_scaling_mode;
```

3.4 PNG_DECODER_INFO_INIT

PNG_Decoder_Info_Init is filled by the decoder whenever the application invoking the PNG decoder calls the PNG decoder initialization function PNG_decoder_init.

The information that is available after the initialization includes the width, height, output width, output height, number of bytes in a row, number of channels, number of passes, number of entries in the palette, bit depth of each channel, compression method and level, filter method, interlace type, pixel depth, palette data for indexed images, histogram information, RGB color space information, file gamma, background information, transparency information, significant bits in the original PNG stream, chromaticity information and physical dimensions information.

```
typedef struct
{
    PNG_UINT32                  width;
    PNG_UINT32                  height;
    PNG_UINT32                  output_width;
    PNG_UINT32                  output_height;
    PNG_UINT32                  rowbytes;
    PNG_UINT8                   channels_orig;
    PNG_UINT8                   channels_after_transform;
    PNG_UINT8                   number_passes;
    PNG_UINT16                  num_palette;
    PNG_UINT16                  num_trans;
    PNG_UINT32                  bit_depth;
    PNG_UINT32                  color_type;
    PNG_UINT32                  interlace_type;
    PNG_UINT8                   pixel_depth;
    PNG_UINT32                  scaling_factor;
    PNG_UINT8                   pass;
    PNG_INT32                   compression_type;
    PNG_INT32                   filter_method;
    PNG_UINT8                   compression_level;
    PNG_UINT8                   srgb_info;
    PNG_UINT32                  image_gamma;
    Background_Info             bkgd_info;
    Trans_Info_Rgb_And_Gray     trans_rgb_gray;
    Trans_Info_Indexed          trans_indexed;
    Significant_Bits_Info       sig_bits;
    Chromaticity_Info           chrm_info;
    Phy_Dimension_Info          phy_dim_info;
} PNG_Decoder_Info_Init;
```

Element	Description
`Width`	Input Width as specified in the PNG image header

Height	Input Height as specified in the PNG image header
output_width	Width of the output image to be rendered[3]
output_height	Height of the output image to be rendered
Rowbytes	Number of bytes in a decoded row
channels_orig	Number of data channels in the input stream pixel. Valid range 0 to 4
channels_after_transform	Number of data channels in the decoded pixel.(after input transform are applied) Value=4 (if alpha is present) Value=3 (if alpha is absent)
number_passes	Number of passes in the interlaced image. If Value =1, it's a non–interlaced image If Value =7, it's a interlaced image (uses Adam7 interlacing)
num_palette	Number of color entries in palette. This is applicable only if image is indexed-color.
num_trans	number of transparent palette color (tRNS)
bit_depth	Number of bits per channel. Valid values For indexed color images --1, 2, 4, or 8. For gray-scale images -- 1, 2, 4, 8, or 16. For true-color, true-color with alpha data, and gray-scale (with alpha) – 8 or 16
color_type	Type of image 0 Grayscale: gray 2 True-color: red, green, blue. 3 Indexed-color: palette index. 4 Grayscale with alpha: grey, alpha. 6 True-color with alpha: red, green, blue, and alpha.
interlace_type	Interlacing Flag. If value=0, image is non-interlaced If value=1, image is interlaced.
pixel_depth	Number of bits per pixel. Valid values For indexed–color images-- 1,2,4 or 8. For true-color -- 24 or 48. For gray-scale -- 1, 2, 4, 8 or 16.
scaling_factor	Its value depends on scale_factor
Pass	current interlace pass (0 - 6)

[3] The rendered output size may not exactly match the display size configured in PNG_Decoder_Params since the decoded output is scaled down by an integral multiple with aspect ratio preserved, to yield the rendered output.

`compression_type`	Method of Compression used. Only compression method 0 (deflate/inflate compression with a sliding window of at most 32768 bytes) is supported as per the specifications.
`filter_method`	Method of filtering used. Only filter method 0 (adaptive filtering with five basic filter types) is supported as the per the specifications.
`compression_level`	Level of compression (Value ranges from 0 to 9)
`srgb_info`	Rendering intent 0-Perceptual 1-Relative colorimetric 2-Saturation 3-Absolute colorimetric
`image_gamma`	Image gamma information. The value is encoded as a four-byte PNG unsigned integer, representing gamma times 100000
`Background_Info`	Structure indicating the background color information (as provided in PNG Background chunk)
`Trans_Info_Rgb_And_Gray`	Structure indicating transparency information for true-color (color type 2) and grayscale (color type 0) images (as provided in PNG Transparency chunk)
`Trans_Info_Indexed`	Structure indicating transparency info for indexed color images (color type 3), as provided in PNG Transparency chunk
`Significant_Bits_Info`	Structure indicating significant bit info
`Chromaticity_Info`	Structure indicating chromaticity info
`Phy_Dimension_Info`	Structure indicating physical dimension info

```
typedef struct
{
  /*Following RGB values can be used as a default background color
   Applicable for True-color Images (Color type 2 and 6)*/

   PNG_UINT16 red;
   PNG_UINT16 green;
   PNG_UINT16 blue;

  /*Following grayscale value can be used as a default background color
   Applicable for Grayscale Images (Color type 0)*/

   PNG_UINT16 gray;

 /*Following index value can be used as a default background color.
   Applicable for Indexed-Color Images (Color type 3)*/

  PNG_UINT8 index;

} Background_Info;

typedef struct
{
  /*Pixels of the specified RGB sample values are treated as
transparent. Applicable for True-color Images without alpha (Color type
2)*/

   PNG_UINT16 red;
   PNG_UINT16 green;
   PNG_UINT16 blue;

  /* Pixels of the specified grey sample values are treated as
    transparent. Applicable for True-color Images without alpha (Color
    type 0)*/

   PNG_UINT16 gray;

} Trans_Info_Rgb_And_Gray;

typedef struct
{
   /*Array indicating transparency information for indexed (color    type
   3)  images  (as  provided  in  PNG  Transparency  chunk).  There  are
   "num_trans"  transparency  values  stored  in  the  same  order  as  the
   palette colors, starting from index 0. Values for the data are in the
   range  [0,  255],  ranging  from  fully  transparent  to  fully  opaque,
   respectively*/


  /*Number of transparent palette colors*/

  PNG_UINT16 num_trans;
} Trans_Info_Indexed;

typedef struct

{
/*  Following  values  provided  significant  red,  green  and  blue  bits  for
true-color and indexed images (color types 2 and 3) files */

   PNG_UINT8 red;
   PNG_UINT8 green;
   PNG_UINT8 blue;

/*  Following  value  provides  significant  gray  bits  for  grayscale  images
(color type 0) files */

   PNG_UINT8 gray;

/*  Following  value  provides  significant  alpha  bits  for  grayscale  and
true-color images with alpha channel (color types 4 and 6) files */

   PNG_UINT8 alpha; /* for alpha channel files */
} Significant_Bits_Info;

typedef struct
{
  /*Each  value  is  encoded  as  a  four-byte  PNG  unsigned  integer,
  representing the x or y value times 100000. Refer spec for details*/

   PNG_UINT32 white_x; /*White point x*/
   PNG_UINT32 white_y; /*White point y*/
   PNG_UINT32 red_x; /*Red x*/
   PNG_UINT32 red_y; /*Red y*/
   PNG_UINT32 green_x; /*Green x*/
   PNG_UINT32 green_y; /*Green y*/
   PNG_UINT32 blue_x; /*Blue x*/
   PNG_UINT32 blue_y; /*Blue y*/
} chromaticity_info;

typedef struct
{
   PNG_UINT32 x_pixels_per_unit;  /* horizontal pixel density */
   PNG_UINT32 y_pixels_per_unit;  /* vertical pixel density */
   PNG_UINT8 phys_unit_type;     /* resolution type  */
} Phy_dimension_info;
```

4 PNG Decoder Interface

4.1 Initialization

All initializations required for the decoder are done in PNG_dec_init(). This function must be called after the allocation for PNG decoder object has been done. The routine internally uses PNG Lib API for initialization purpose. It calls the function pointed by function pointer PNG_app_malloc() for allocation of the memory needed by decoder. Function pointed by function pointer PNG_app_read_data() is called for reading the input bits required for initialization. Header information is available after call to the initialization routine is made. Members of PNG_Decoder_Info_Init structure are initialized in this function. This routine needs to be called at the beginning of every new file/stream. When PNG_DEC_INVALID_OUTFORMAT is returned, the output format will be modified with one recommended format by decoder, caller should re-call PNG_dec_init() with the new output format.

C prototype:

```
PNGD_RET_TYPE PNG_dec_init (PNG_Decoder_Object *png_dec_object)
```

Arguments:

png_dec_obj	- Decoder Object pointer

Return value:

PNGD_OK	-	indicates initialization was successful.
Other Codes	-	indicates error

4.2 Decoding and Post Processing (row)

The main decoding function is PNG_decode_row().This function decodes the PNG bit stream row by row to generate the image pixels in requested format. The decoder should be initialized before this function is called. During the process of decoding, the function pointed by function pointer PNG_app_read_data() gets called whenever the decoder needs data from the input stream. The output buffer is filled with RGB pixels of the required output format and intended size for display. The decoded output is available after each row since the decoding and post processing are carried out row by row. If errors are encountered in the bit stream, the decoder handles these errors internally.

C prototype:

```
PNGD_RET_TYPE PNG_decode_row (PNG_Decoder_Object *png_dec_object,
PNG_UINT8 *outbuf);
```

Arguments:

png_dec_obj	- Decoder Object pointer
outbuf	- Output Buffer

Return value:

PNGD_OK	-	indicates decoding was successful.
Other Codes	-	indicates error

4.3 Decoding and Post Processing (whole image)

The main decoding function is PNG_decode_frame().This function decodes the PNG bit stream to generate the whole output image in requested format. The decoder has the same action with PNG_decode_row except the numbers of output rows. It is expected to decode the whole image, but not only 1 row.

C prototype:

```
PNGD_RET_TYPE PNG_decode_frame (PNG_Decoder_Object *png_dec_object,
PNG_UINT8 *outbuf);
```

Arguments:

png_dec_obj	- Decoder Object pointer
outbuf	- Output Buffer

Return value:

PNGD_OK	-	indicates decoding was successful.
Other Codes	-	indicates error

4.4 Cleanup

The cleanup API, PNG_cleanup(), is responsible for 'destroying' all the PNG structures allocated during initialization. Freeing up of these structures is done by the function pointed by function pointer PNG_app_free (the actual free function needs to be implemented by the calling application).

C prototype:

```
PNGD_RET_TYPE PNG_cleanup (PNG_Decoder_Object *png_dec_object)
```

Arguments:

png_dec_obj	- Decoder Object pointer

Return value:

PNGD_OK	-	indicates cleanup was successful.
Other Codes	-	indicates error

4.5 API Version

This is the decoder function to get the API version information.

C prototype:

```
const char * PNGD_CodecVersionInfo(void)
```

Arguments:

None

Return value:

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

4.6 Functions Calling Applications Must Implement

The PNG decoder requires certain functions, to handle memory allocation (and freeing) and to read data from input stream, which need to be implemented by the calling application. The PNG decoder API uses function pointers to invoke these functions.

4.6.1 Allocation of Memory

The function (implemented by calling application) that allocates memory is accessed through the following function pointer. The application has to update the passed argument ptr with the starting location of the allocated memory.

C prototype:

```
void *(*PNG_app_malloc)(void *ptr, PNG_UINT32 size)
```

Arguments:

ptr – Pointer to the allocated memory (to be updated by calling application)
size – Number of bytes to be allocated

Return value:

Pointer to the allocated memory

4.6.2 Freeing of Memory

The function (implemented by calling application) that frees up memory is accessed through the following function pointer. The application has to free the memory pointed to by the passed argument ptr.

C prototype:

```
void (*PNG_app_free_fun)(void *ptr)
```

Arguments:

Ptr – Pointer to the memory that needs to be freed

Return value:

None.

4.6.3 Reading Data from an Input Stream

The function (implemented by calling application), which allows the PNG decoder library to read the input stream, is accessed through the following function pointer. The decoder library needs to pass a pointer to the input stream (Note: In case of the first call to this function the PNG decoder calls this function with a NULL). The application returns the data of specified length, pointed to by `input_data` pointer. As an example, a calling application that uses a file system may choose to implement this using the fread() function.

C prototype:

```
PNGD_RET_TYPE (*PNG_app_read_data)(void *input_ptr, PNG_UINT8
*input_data, PNG_UINT32 length_requested, PNG_UINT32 length_returned);
```

Arguments:

input_ptr –	Pointer to input stream
input_data –	Pointer to input data
`length_requested` –	Number of bytes requested by the library
`length_returned -`	Number of bytes returned by the application

Note: If length_returned is not equal to length_requested, the library will perform appropriate error handling

Return value:
Error Code

4.7 Suspension

There are two ways the application can suspend the PNG decoder. The first method is with the use of PNG_decode_row() after which control is returned to the calling application. The second method is by the use PNG_app_read_data().

Suspension using the second method takes place as follows:

- A flag TEST_SUSPENSION is defined in the application code
- A static variable is declared in PNG_app_read_data() function and is incremented each time the function is called.
- If the calling application chooses to suspend the decoding process, PNG_app_read_data() returns the code PNGD_ERR_SUSPEND.
- The library comes out of the decoding function with return code as PNGD_ERR_SUSPEND.
- The application sets the state of the decoder as suspended.
- When the data is ready, the application sets the input pointer to the start of the image and the decoding proceeds by calling PNG_dec_init() and PNG_decode_row() sequentially irrespective of the routine it was suspended from PNG_dec_init() or PNG_decode_row()

5 Overview of API Usage

- Calling application allocates memory for PNG decoder object.
- Calling application configures the decoding parameters (i.e. desired output format, rescale enabling, the width and height of output to be displayed)
- Calling application uses PNG_dec_init() and performs validity check for PNG data stream and populates the PNG decoder init info structure (with width, height, output width, output height, number of bytes in a row, number of channels, pass, number of entries in the palette, bit depth of each channel, compression type, filter type, interlace type and pixel depth.)
- Calling application allocates memory for output buffer
- For non-interlaced images, calling application calls PNG_decode_row() for each row to obtain the decoded data for each row in the output buffer .
- For interlaced images, calling application[4] calls PNG_decode_row () for each row within each pass (for instance, in a nested loop). After iteration of each pass is done, decoded output of the image for that pass is available in the output buffer.
- Calling application frees up the output buffer
- Calling application calls PNG_cleanup() which internally 'destroys' PNG structures
- Calling application frees up the PNG decoder object, the pointer to which has already been set to NULL by PNG_cleanup().

Note: The calling application needs to implement functions that handle memory allocation (and de-allocation) and allow the PNG decoder library to read data from input stream.

[4] The release will contain a sample test application with information on how the application needs to call the PNG_decoder_row()

Appendix A RGB output formats supported

5.1 RGB888 FORMAT

5.1.1Unwrapped format

In the RGB888 image data format, each pixel requires 3 bytes. The image data is organized as follows.

Unwrapped RGB888 Image data format

DATA (MSB -> LSB)

$R_7\ R_6\ R_5\ R_4\ R_3\ R_2\ R_1\ R_0\ G_7G_6\ G_5\ G_4\ G_3\ G_2\ G_1\ G_0\ B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0$

The library provides data in the aforementioned unwrapped format.

5.1.2Wrapped format

In order to facilitate easy viewing of the raw RGB888 data, the sample test wrapper prepends headers to make it compatible with the Portable Bit-Map formats, i.e. PPM (Portable PixelMap) in case of colour data.

Wrapped RGB888 Image Fields

HEADER	**DATA (MSB -> LSB)** $R_7\ R_6\ R_5\ R_4\ R_3\ R_2\ R_1\ R_0\ G_7G_6\ G_5\ G_4\ G_3\ G_2\ G_1\ G_0\ B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0$

Please refer to http://netpbm.sourceforge.net/doc/ppm.html for details on PPM header and http://netpbm.sourceforge.net/doc/pgm.html for details on PGM header format.

5.2 RGB565 FORMAT

5.2.1Unwrapped format

In the RGB565 image data format, each pixel requires 2 bytes. Consider the RGB888 data depicted in the previous section. The derived RGB 565 data would be as follows.

Unwrapped RGB565 Image data format

DATA (MSB -> LSB)
$\mathbf{R_7\ R_6\ R_5\ R_4\ R_3\ G_7G_6\ G_5\ G_4\ G_3\ G_2\ B_7\ B_6\ B_5\ B_4\ B_3}$

The library provides data in the aforementioned unwrapped format. Note that this data can be organized in the little endian or big endian format, depending on the endianness of the target of execution.

5.2.2Wrapped format

In order to be consistent with the wrapped format for RGB888, the sample test wrapper prepends headers to make it compatible with the Portable Bit-Map formats, i.e. PPM (Portable PixelMap) in case of colour data.

Wrapped RGB565 Image Fields

HEADER	**DATA (MSB -> LSB)** $\mathbf{R_7\ R_6\ R_5\ R_4\ R_3\ G_7G_6\ G_5\ G_4\ G_3\ G_2\ B_7\ B_6\ B_5\ B_4\ B_3}$

Please refer to http://netpbm.sourceforge.net/doc/ppm.html for details on PPM header and http://netpbm.sourceforge.net/doc/pgm.html for details on PGM header format.

5.3 RGB555 FORMAT

5.3.1Unwrapped format

In the RGB555 image data format, each pixel requires 2 bytes. Consider the RGB888 data depicted in the previous section. The derived RGB 555 data would be as follows

Unwrapped RGB555 Image data format

DATA (MSB -> LSB)
0 R_7 R_6 R_5 R_4 R_3 G_7G_6 G_5 G_4 G_3 B_7 B_6 B_5 B_4 B_3

Among the 16 bits, the most significant bit is set to zero.

The library provides data in the aforementioned unwrapped format. Note that this data can be organized in the little endian or big endian format, depending on the endianness of the target of execution.

5.3.2Wrapped format

In order to be consistent with the wrapped format for RGB888, the sample test wrapper prepends headers to make it compatible with the Portable Bit-Map formats, PPM (Portable PixelMap) in case of colour data.

Wrapped RGB555 Image Fields

HEADER	**DATA (MSB -> LSB)** **0 R_7 R_6 R_5 R_4 R_3 G_7G_6 G_5 G_4 G_3 B_7 B_6 B_5 B_4 B_3**

Please refer to http://netpbm.sourceforge.net/doc/ppm.html for details on PPM header and http://netpbm.sourceforge.net/doc/pgm.html for details on PGM header format.

5.4 RGB666 FORMAT

5.4.1Unwrapped format

In the RGB666 image data format, each pixel requires 3 bytes. Consider the RGB888 data depicted in the previous section. The derived RGB 666 data would be as follows

Unwrapped RGB666 Image data format

DATA (MSB -> LSB)
R_7 R_6 R_5 R_4 R_3 R_2 0 0 G_7 G_6 G_5 G_4 G_3 G_2 0 0 B_7 B_6 B_5 B_4 B_3 B_2 0 0

Within each byte, the two least significant bits are set to zero. This choice of padding zeros towards the LSB lends itself to easy viewing of the rendered RGB666 data.

The library provides data in the aforementioned unwrapped format.

5.4.2Wrapped format

In order to facilitate easy viewing of the raw RGB666 data, the sample test wrapper prepends headers to make it compatible with the Portable Bit-Map formats, i.e. PPM (Portable PixelMap) in case of colour data.

Wrapped RGB555 Image Fields

HEADER	**DATA (MSB -> LSB)** R_7 R_6 R_5 R_4 R_3 R_2 0 0 G_7 G_6 G_5 G_4 G_3 G_2 0 0 B_7 B_6 B_5 B_4 B_3 B_2 0 0

Please refer to http://netpbm.sourceforge.net/doc/ppm.html for details on PPM header and http://netpbm.sourceforge.net/doc/pgm.html for details on PGM header format.

5.5 Grayscale FORMAT

5.5.1 Unwrapped format

In the grayscale 8-bit image data format, each pixel requires 1 byte. The image data is organized as follows.

Unwrapped 8-bit Grayscale Image data format

DATA (MSB -> LSB)
G_7G_6 G_5 G_4 G_3 G_2 G_1 G_0

The library provides data in the aforementioned unwrapped format.

5.5.2Wrapped format

In order to facilitate easy viewing of the raw grayscale 8-bit data, the sample test wrapper prepends headers to make it compatible with the Portable Bit-Map formats, i.e. PGM (Portable Grayscale Map) in case of grayscale image.

Wrapped 8-bit Grayscale Image Fields

HEADER	**DATA (MSB -> LSB)** **G_7G_6 G_5 G_4 G_3 G_2 G_1 G_0**

Please refer to http://netpbm.sourceforge.net/doc/ppm.html for details on PPM header and http://netpbm.sourceforge.net/doc/pgm.html for details on PGM header format.

5.6 BGR FORMAT

In BGR format, R component and B component are exchanged in store order according to corresponding RGB format addressed above.

Appendix B Suspension and Resumption Mechanism

To test the suspension mechanism, compile time flag TEST_SUSPENSION is used. This flag is defined in the file /ARM11/src/image/png_dec/test/c_source/png_test_wrapper.c
For simulating the suspension and resumption mechanism this flag needs to be enabled. To simulate this suspension mechanism following concept is implemented in the application code.

1. A flag TEST_SUSPENSION is defined in the application code
2. A static variable is declared in PNG_app_read_data () function and is incremented each time the function is called.
3. After a few calls to the function, PNG_app_read_data () returns the code PNGD_ERR_SUSPEND.
4. The library comes out of the decoding function with return code as PNGD_ERR_SUSPEND.
5. The application sets the state of the decoder as suspended.
6. When the data is ready, the application sets the input pointer to the start of the image and the decoding proceeds by calling PNG_dec_init and PNG_decode_row sequentially irrespective of the routine it was suspended from PNG_dec_init or PNG_decode_row

Appendix C Debug and Log Support

The current release uses the debug and log support provided by the PNG library code. Some modifications have been made to the “png.h” file of the PNGlib code for unification with logging mechanism. The modifications are given below

```
/********************Original Code******************************/
#ifdef PNG_DEBUG
#if (PNG_DEBUG > 0)
#if !defined(PNG_DEBUG_FILE) && defined(_MSC_VER)
#include <crtdbg.h>
#if (PNG_DEBUG > 1)
#define png_debug(l,m)  _RPT0(_CRT_WARN,m)
#define png_debug1(l,m,p1)  _RPT1(_CRT_WARN,m,p1)
#define png_debug2(l,m,p1,p2) _RPT2(_CRT_WARN,m,p1,p2)
#endif
#else // PNG_DEBUG_FILE || !_MSC_VER
#ifndef PNG_DEBUG_FILE
#define PNG_DEBUG_FILE stderr
#endif // PNG_DEBUG_FILE
#if (PNG_DEBUG > 1)
#define png_debug(l,m) \
{ \
     int num_tabs=l; \
     fprintf(PNG_DEBUG_FILE,"%s"m,(num_tabs==1 ? "\t" : \
       (num_tabs==2 ? "\t\t":(num_tabs>2 ? "\t\t\t":"")))); \
}
#define png_debug1(l,m,p1) \
{ \
     int num_tabs=l; \
     fprintf(PNG_DEBUG_FILE,"%s"m,(num_tabs==1 ? "\t" : \
       (num_tabs==2 ? "\t\t":(num_tabs>2 ? "\t\t\t":""))),p1); \
}
#define png_debug2(l,m,p1,p2) \
{ \
     int num_tabs=l; \
     fprintf(PNG_DEBUG_FILE,"%s"m,(num_tabs==1 ? "\t" : \
       (num_tabs==2 ? "\t\t":(num_tabs>2 ? "\t\t\t":""))),p1,p2); \
}
#endif // (PNG_DEBUG > 1)
#endif // _MSC_VER
#endif // (PNG_DEBUG > 0)
#endif // PNG_DEBUG
*****************End of original code *************************/

/************Modified Code*********************************/
#include "log_api.h"

#ifndef PNG_DEBUG_FILE
#define PNG_DEBUG_FILE stderr
#endif

#ifndef PNG_DEBUG
#  define PNG_DEBUG 0
#endif
```

```
#if (PNG_DEBUG > 1)
#define png_debug(l,m) \
{ \
       DebugLogText(l,m);\
}
#define png_debug1(l,m,p1) \
{ \
       DebugLogText(l,m,p1);\
}
#define png_debug2(l,m,p1,p2) \
{ \
       DebugLogText(l,m,p1,p2);\
}
#endif
```

The following description of the debug logging mechanism used for this release has been excerpted from http://www.libpng.org/pub/png/libpng-manual.txt
with the necessary modifications.

Requesting debug printout

The macro definition PNG_DEBUG can be used to request debugging printout. Set it to an integer value in the range 0 to 3. Higher numbers result in increasing amounts of debugging information. The information is printed to the "stderr" file, unless another file name is specified in the PNG_DEBUG_FILE macro definition.

This release logs the messages and data to the debug.bin file.

When PNG_DEBUG > 0, the following functions (macros) become available:

```
  png_debug(level, message)
  png_debug1(level, message, p1)
  png_debug2(level, message, p1, p2)
```

in which "level" is compared to PNG_DEBUG to decide whether to print the message, "message" is the formatted string to be printed, and p1 and p2 are parameters that are to be embedded in the string according to printf-style formatting directives. For example,

```
  png_debug1(2, "foo=%d\n", foo);
```

is expanded to

```
if(PNG_DEBUG > 2)
    DebugLogText(PNG_DEBUG_FILE, "foo=%d\n", foo);

  /*Original PNG code has the following code statement*/
if(PNG_DEBUG > 2)
     fprintf(PNG_DEBUG_FILE, "foo=%d\n", foo);
```

When PNG_DEBUG is defined but is zero, the macros aren't defined, but you can still use PNG_DEBUG to control your own debugging:

```
   #ifdef PNG_DEBUG
      DebugLogText(PNG_DEBUG_FILE,… //Modified
  #endif
/*Original PNG code has the following code statement*/
#ifdef PNG_DEBUG
                  fprintf(stderr, ...
#endif
```

When PNG_DEBUG = 1, the macros are defined, but only png_debug statements having level = 0 will be printed. There aren't any such statements in this version of libpng, but if you insert some they will be printed.