Image Library Reference

©2005–2014, QNX Software Systems Limited, a subsidiary of BlackBerry Limited.
All rights reserved.

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

QNX, QNX CAR, Momentics, Neutrino, and Aviage are trademarks of BlackBerry Limited, which are registered and/or used in certain jurisdictions, and used under license by QNX Software Systems Limited. All other trademarks belong to their respective owners.

Electronic edition published: Wednesday, October 8, 2014

Table of Contents

About This Guide

The *Image Library Reference* is intended for developers who want to write applications that render images using the `libimg` library.

This table may help you find what you need in this guide:

To find out about:	Go to:
Using the library	*Working with Images* (p. 11)
The functions for rendering images	*Image API* (p. 17)

© 2014, QNX Software Systems Limited

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	`if( stream == NULL)`
Command options	`-lR`
Commands	`make`
Constants	`NULL`
Data types	`unsigned short`
Environment variables	***PATH***
File and pathnames	`/dev/null`
Function names	*exit()*
Keyboard chords	**Ctrl–Alt–Delete**
Keyboard input	`Username`
Keyboard keys	**Enter**
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
User-interface components	**Navigator**
Window title	**Options**

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective** **Show View**.

We use notes, cautions, and warnings to highlight important messages:

Notes point out something important or useful.

Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

© 2014, QNX Software Systems Limited

Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we typically use a forward slash (/) as a delimiter in pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

© 2014, QNX Software Systems Limited

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (*www.qnx.com*). You'll find a wide range of support options, including community forums.

© 2014, QNX Software Systems Limited

Chapter 1
About This Guide

The *Image Library Reference* is intended for developers who want to write applications that render images using the `libimg` library.

This table may help you find what you need in this guide:

To find out about:	Go to:
Using the library	*Working with Images* (p. 11)
The functions for rendering images	*Image API* (p. 17)

© 2014, QNX Software Systems Limited

Chapter 2
Working with Images

This chapter shows you how to load and render images using the image library, `libimg`.

The image library is a static library that provides a common interface for image codecs. This means that while the library is linked into your executable, you need to put any required image codecs, plus the image configuration file, onto targets running your application. You'll need to include (at least):

- the configuration file at `/etc/system/config/img.conf` (see its description in the documentation for *img_lib_attach()* (p. 84))
- the image codecs (`img_codec_*.so`) in ***$PROCESSOR***`/lib/dll`

The codecs used by the image library are:

`img_codec_bmp.so`

Windows Bitmap format codec. Provides full Microsoft BMP support for all known variants (except Header Version 5.x) as well as the older OS/2 variant. Does not support v1.x DDB format.

`img_codec_gif.so`

Graphics Interchange Format codec. This codec supports GIF 87a and GIF 89a variants and the graphics control extension which provides most of the significant features for this format (transparency, interlacing, multiframe etc). It ignores other extensions which allow embedded text, comments, application data etc.

`img_codec_jpg.so`

Joint Photographic Experts Group file format codec. Supports 24-bit RGB, YUV, and grayscale.

`img_codec_pcx.so`

Personal Computer Exchange Format decoder.

`img_codec_png.so`

Portable Networks Graphics codec. Provides full PNG support with alpha, transparency, and interlacing support. This codec ignores the following chunks:

- background color
- chromaticity

© 2014, QNX Software Systems Limited

- gamma
- histogram
- physical pixel dimension
- significant bits
- text data
- image last-modified time

`img_codec_sgi.so`

SGI format codec. It supports black-and-white, grayscale, and color images (*sgi, *.rgb, *.rgba, *.bw).

This codec supports only decoding.

`img_codec_tga.so`

Truevision Graphics Adapter format codec. For decoding, this codec handles run length encoding (RLE) compression and supports these formats:

- true color 16-, 24-, and 32-bit
- cmap 15-, 16-, 24-, and 32-bit
- black and white 8-bit

For encoding, this codec supports true color (8888) 32-bit RLE.

`img_codec_tiff.so`

Tagged Image File Format codec. Supports full Baseline TIFF decoding from the Adobe TIFF Revision 6.0 specification (for example, bilevel, grayscale, RGB, multiple subfiles, PackBits and Huffman compression). Some TIFF extensions are supported, such as CCITT bilevel encodings (enables fax image decoding), LZW compression, and associated alpha.

The TIFF encoder is limited to encoding grayscale and RGB images only.

As TIFF images are not sequential streams of data, the TIFF decoder must be able to seek in the image stream to find image data. As a result, it may not be compatible with certain unidirectional or unbuffered IO streams. For the best results, decode a TIFF image from a file or a full memory buffer.

`img_codec_wbmp.so`

Wireless Application Protocol Bitmap file format codec. Supports decoding and encoding of monochrome images.

© 2014, QNX Software Systems Limited

To display an image, your application needs to:

- attach to the image library
- load the image file
- clean up allocated resources and detach from the image library

Let's look at each of these steps in a little more detail.

© 2014, QNX Software Systems Limited

Attaching to the image library

When you call *img_lib_attach()* (p. 84), the image library initializes and loads the codecs it finds listed in the `img.conf` configuration file. You can customize this file to load just the codecs your application requires, and change the location where the image library looks for it by setting the ***LIBIMG_CFGFILE*** environment variable. If this environment variable isn't set, the library checks the default location `/etc/system/config/img.conf`. See the documentation for *img_lib_attach()* for more information about the format of this file.

To use *img_lib_attach()*:

```
img_lib_t ilib = NULL;
int rc;
...
if ((rc = img_lib_attach(&ilib)) != IMG_ERR_OK) {
                fprintf(stderr, "img_lib_attach() failed: %d\n", rc);
                return -1;
        }
```


© 2014, QNX Software Systems Limited

Load an image

To load an image, follow these steps:

1. Enumeration of codecs

   First, you need a list of codecs that are installed, which you can retrieve by calling *img_codec_list()* (p. 22).

   

   If you have additional information about the data (for example, a mime-type or extension), you could use a variant such as *img_codec_list_byext()* (p. 24) or *img_codec_list_bymime()* (p. 26). This will give you a list of codecs including only those that match the specified criteria. Keep in mind though, that extensions or mime types do not necessarily guarantee the data is of a specific format (that is, they can lie). So it's always good to be prepared to try all the codecs if one that handles the format that data claims to be in fails.

2. Establishing the underlying input source

   The image data has to come from a source, such as a file, TCP/IP socket, or memory buffer. This step involves establishing the origin of the data. This step may involve no work at all (that is, if it's a file already stored in memory), or it may involve opening a file or performing some other task.

3. Associating the image library conventional IO interface with the input source

   The image library decoders need a conventional way to access the data, which is where the IO streams come in. Use *io_open()* (p. 111) to associate an `io_stream_t` with the data source from the previous step.

4. Data recognition

   This step involves allowing the codecs you've enumerated to peek at the data to determine which one is capable of handling the data. You can do this with *img_decode_validate()* (p. 57), which runs through the list of codecs and indicates which (if any) approved of the data. You can then use that codec to decode the data.

5. Initializing the decoder

   This step notifies the decoder of an imminent decode operation, and allows it to set up any resources it may require. Use *img_decode_begin()* (p. 34) to perform this step.

6. Decoding frames

   Decode frames using *img_decode_frame()* (p. 47) until you're finished or there are no more (when *img_decode_frame()* returns `IMG_ERR_NODATA`).

© 2014, QNX Software Systems Limited

7. Finalizing the decoding

   Call *img_decode_finish()* (p. 45) to allow the decoder to clean up after itself.

Although this process may seem complicated, there are two higher-level API calls that simplify the process:

***img_load_file()* (p. 89)**

This function takes care of all of the steps outlined above. However, this function loads only the first frame, and works only with a file source.

***img_load()* (p. 87)**

This function takes care of all the steps, except for establishing the input source and associating it with an `io_stream_t`. This provides the convenience of *img_load_file()* (p. 89) but lifts the file only restriction. Like *img_load_file()*, it's limited to loading only the first frame.

Here's an example of using *img_load_file()*:

```
int rc;
img_t img;

...

/* initialize an img_t by setting its flags to 0 */
img.flags = 0;

/* if we want, we can preselect a format (ie force the image to be
   loaded in the format we specify) by enabling the following two
   lines */

img.format = IMG_FMT_PKLE_ARGB1555;
img.flags |= IMG_FORMAT;

/* likewise, we can 'clip' the loaded image by enabling the following */

img.w = 100;
img.flags |= IMG_W;

img.h = 100;
img.flags |= IMG_H;

if ((rc = img_load_file(ilib, argv[optind], NULL, &img)) != IMG_ERR_OK) {
        fprintf(stderr, "img_load_file(%s) failed: %d\n", argv[optind], rc);
        return -1;
}

fprintf(stdout, "img is %dx%dx%d\n", img.w, img.h, IMG_FMT_BPP(img.format));


/* for our purposes we're done with the img lib */
img_lib_detach(ilib);
```


© 2014, QNX Software Systems Limited

Chapter 3
Image API

The image API includes the functions and data types described here.

To use the functionalities that this image API offers, along with the data types and functions in here, you need to open an I/O stream, which is managed by the *io_open()* (p. 111) and *io_close()* (p. 110) functions. The *io_open()* (p. 111) is used to associate an io_stream_t value with image data (a file, TCIP/IP socket, or memory buffer). The *io_close()* (p. 110) function releases the input stream and frees resources. Since the *io_open()* (p. 111) and *io_close()* (p. 110) are not part of the `<img.h>` header, you need to include the `<io/io.h>` header file, which includes the `<img.h>` header file within it. Therefore, to work with the image API, you can simply include the `<io/io.h>` header file.

© 2014, QNX Software Systems Limited

img_cfg_read()

Read a configuration file and load codecs

Synopsis:

```
#include <img/img.h>

int img_cfg_read (img_lib_t   ilib,
                  const char *path )
```

Arguments:

ilib

The library handle filled in by *img_lib_attach()*.

path

The path to a configuration file to read.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function reads the configuration file specified by *path*, and loads the codecs listed in it.

See *img_lib_attach()* (p. 84) for a description of the configuration file format.

Returns:

`IMG_ERR_OK`

Success

`IMG_ERR_MEM`

Memory-allocation failure.

`IMG_ERR_CFG`

Couldn't open the specified file, or the file has an incorrect format.

© 2014, QNX Software Systems Limited

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_codec_get_criteria()

Get the extension and mime information for a given codec

Synopsis:

```
#include <img/img.h>

void img_codec_get_criteria( img_codec_t codec,
                             const char **ext,
                             const char **mime );
```

Arguments:

codec

The codec for which to return the criteria.

ext

A pointer to the codec's extension type.

mime

A pointer to the codec's mime type.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function gets the extension type and mime type for a given codec.

Examples:

```
#include <stdio.h>
#include <stdlib.h>
#include <img/img.h>

int main (int argc, char *argv[])
{
  img_lib_t  ilib = NULL;

  if(img_lib_attach(&ilib) == IMG_ERR_OK)
  {
    int count = img_codec_list(ilib, NULL, NULL, NULL, 0);
    if( count > 0)
    {
       img_codec_t     *codecs;
       if((codecs = (img_codec_t *)calloc(count, sizeof(img_codec_t))) != NULL)
```


© 2014, QNX Software Systems Limited

```
        {
          if((count = img_codec_list(ilib, codecs, count, NULL, 0)) > 0)
          {
             int i;

             for( i = 0; i < count; i++)
             {
                char const * mime;
                char const * ext;

                img_codec_get_criteria(codecs[i], &ext, &mime);
                printf("codecs[%d]: ext = %s: mime = %s\n", i, ext, mime);
             }
           }
           free(codecs);
        }
      }
    }

    return (0);
}
```

Running this example produces the following output:

```
codecs[0]: ext = pcx: mime = application/pcx
codecs[1]: ext = tga: mime = application/tga
codecs[2]: ext = sgi: mime = image/sgi
codecs[3]: ext = png: mime = image/png
codecs[4]: ext = jpg: mime = image/jpeg
codecs[5]: ext = gif: mime = image/gif
codecs[6]: ext = bmp: mime = image/bmp
```

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_codec_list()

Enumerate codecs

Synopsis:

```
#include <img/img.h>

size_t img_codec_list( img_lib_t    ilib,
                       img_codec_t* buf,
                       size_t       nbuf,
                       img_codec_t* exclude,
                       size_t       nexclude );
```

Arguments:

ilib

The handle for the image library, returned by *img_lib_attach()* (p. 84).

buf

The address of an array that the function populates with handles for available codecs.

nbuf

The number of items in the *buf* array.

exclude

The address of an array of codec handles that you'd like the function to exclude from the list.

nexclude

The number of items in the *exclude* array.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function lists all codecs installed, except for those specified by *exclude*.

The function copies up to *nbuf* handles into the array specified by *buf*. No copying is done if *nbuf* is 0. This function returns the total number of matching codecs, which

© 2014, QNX Software Systems Limited

may be larger than *nbuf* if your buffer was not big enough to store all the matched codecs.

Returns:

The the total number of matching codecs.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_codec_list_byext()

Enumerate codecs by file extension

Synopsis:

```
#include <img/img.h>

size_t img_codec_list_byext( img_lib_t      ilib,
                             const char*    string,
                             img_codec_t*   buf,
                             size_t         nbuf);
```

Arguments:

ilib

The handle for the image library, returned by *img_lib_attach()* (p. 84).

string

A string containing the file extension to identify. For example, `.jpg` or `my_file.jpg` for codecs that handle JPEGs.

The extension must start with a period.

buf

The address of an array that the function populates with handles for the available codecs.

nbuf

The number of items in the *buf* array.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function enumerates codecs that handle files with the specified extension.

© 2014, QNX Software Systems Limited

While there are no standards defining what extensions are and how they should be named, there seems to be a *de facto* standard governing their use. The term "extension" originates as an intrinsic filename property although most contemporary implementations no longer treat them as separate components of the filename. Thus the term has evolved to loosely describe the portion of characters in a filename that follow the last occurrence of the `.` character. This principle can be seen in use throughout the world wide web, and through many OSs that use extensions as a content-recognition mechanism. Although extensions do not in themselves impose any guarantee on the nature of data a file contains, they are generally appropriately assigned from a well-known set, and as such, they represent reasonable criteria in deciding which codec should handle the data, at least in an initial pass.

The function copies up to *nbuf* handles into the array specified by *buf*. No copying is done if *nbuf* is 0. This function returns the total number of matching codecs, which may be larger than *nbuf* if your buffer was not big enough to store all the matched codecs.

Returns:

The total number of matching codecs.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_codec_list_bymime()

Enumerate codecs by MIME type

Synopsis:

```
#include <img/img.h>

size_t img_codec_list_bymime( img_lib_t     ilib,
                              const char*   mime,
                              img_codec_t*  buf,
                              size_t        nbuf );
```

Arguments:

ilib

The handle for the image library, returned by *img_lib_attach()* (p. 84).

mime

A string describing the desired MIME type (in accordance with *RFC 2046*).

buf

The address of an array that the function populates with handles for available codecs.

nbuf

Number of items in the *buf* array.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function enumerate codecs that handle a specified MIME type.

The function copies up to *nbuf* handles into the array specified by *buf*. No copying is done if *nbuf* is 0. This function returns the total number of matching codecs, which may be larger than *nbuf* if your buffer was not big enough to store all the matched codecs.

© 2014, QNX Software Systems Limited

Returns:

The the total number of matching codecs.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_convert_data()

Convert data from one image format to another

Synopsis:

```
#include <img/img.h>

int img_convert_data( img_format_t          sformat,
                      const uint8_t*        src,
                      img_format_t          dformat,
                      uint8_t*              dst,
                      size_t                n );
```

Arguments:

sformat

The format of the data you are converting from; see `img_format_t` (p. 82).

src

A pointer to a buffer containing the source data.

dformat

The format you would like to convert the data to.

dst

A pointer to a buffer to store the converted data. This may point to a different buffer, or it can point to the same buffer as *src*, as long as you've ensured that the source buffer is large enough to store the converted data (the *IMG_FMT_BPL()* (p. 80) macro can help you with this).

n

The number of samples to convert.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

© 2014, QNX Software Systems Limited

Description:

This function converts data from one image format to another. The conversion may be done from one buffer to another, or in place.

The neither the destination nor the source formats can be a palette-based format (for example `IMG_FMT_PAL8` or `IMG_FMT_PAL4`). Both must be "direct" formats. See *img_expand_getfunc()* (p. 78) to convert a palette-based image to a direct format.

If you're repeatedly converting data, it's better to call *img_convert_getfunc()* (p. 30) to get the conversion function, and then call the conversion function as required.

Returns:

`IMG_ERR_OK`

Success.

`IMG_ERR_NOSUPPORT`

One of the formats specified is invalid.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_convert_getfunc()

Get a function to convert one image format to another

Synopsis:

```
#include <img/img.h>

img_convert_f *img_convert_getfunc( img_format_t src,
                                    img_format_t dst )
```

Arguments:

src

The `img_format_t` (p. 82) image format to convert from

dst

The `img_format_t` image format to convert to.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function returns a pointer to a data conversion function (or `NULL` if the requested conversion is not supported) which you can call to convert a run of pixels from format *src* to format *dst*. The source and destination formats must be "direct" — palette-based formats are not supported. A conversion function takes the form:

```
void convert_f( const uint8_t *src,
                uint8_t *dst,
                unsigned n )
```

A conversion function is called to convert *n* pixels from the *src* buffer, writing the results in the *dst* buffer. The conversions can be done in place (that is, *src* can be the same as *dst*).

Use this function instead of *img_convert_data()* (p. 28) if you need to repeatedly convert data from one format to another. Calling *img_convert_data()* each time will add overhead because it has to get the conversion function each time its called. Using this function, you can just call the correct conversion function yourself directly.

Returns:

© 2014, QNX Software Systems Limited

IMG_ERR_OK

Success.

IMG_ERR_NOSUPPORT

One of the formats specified is invalid.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_crop()

Crop an image

Synopsis:

```
#include <img/img.h>

int img_crop( const img_t    *src,
              img_t          *dst,
              const unsigned  x,
              const unsigned  y );
```

Arguments:

src

The address of the source `img_t` structure to crop.

dst

The address of the destination `img_t` structure, which requires the width (*IMG_W*) and height (*IMG_H*) to be specified at a minimum.

x

The horizontal coordinate in the source image to begin the crop operation.

y

The vertical coordinate in the source image to begin the crop operation.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function will crop a source image at a given set of source coordinates *x*, *y* to the dimensions specified by the destination image.

Returns:

`IMG_ERR_OK`

© 2014, QNX Software Systems Limited

Success

IMG_ERR_PARM

- source image width, height, and/or format not set.
- destination image width, and/or height not set.
- destination image larger than source image.
- crop co-ordinates plus size of destination image exceed source image bounds.

IMG_ERR_NOSUPPORT

Destination image format not supported

IMG_ERR_MEM

Memory allocation failure

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_decode_begin()

Prepare to decode one or more frames from a stream

Synopsis:

```
#include <img/img.h>

int img_decode_begin( img_codec_t  codec,
                      io_stream_t *input,
                      uintptr_t    *decode_data );
```

Arguments:

codec

The codec to use. To figure out a codec to use, see img_codec_list, list_byext, list_bymime, and img_decode_validate.

input

The input source.

decode_data

An address of a `uintptr_t` which the decoder uses to store data it needs across the decode process. You should not pass `NULL`, but instead pass a valid address of a `uintptr_t` initialized to `0`.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function prepares to decode a frame (or series of frames) from a stream.

Returns:

`IMG_ERR_OK`

Success.

`IMG_ERR_NOTIMPL`

The *codec* doesn't provide an implementation for this function.

© 2014, QNX Software Systems Limited

Other

Any other code that a decoder's *begin()* function may pass back to flag an error (see `img_errno.h` for a list of defined errors).

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_decode_callouts_t

Decoder callout table

Synopsis:

```
#include <img/img.h>

typedef struct {
        img_decode_choose_format_f    *choose_format_f;
        img_decode_setup_f            *setup_f;
        img_decode_abort_f            *abort_f;
        img_decode_scanline_f         *scanline_f;
        img_decode_set_palette_f      *set_palette_f;
        img_decode_set_transparency_f *set_transparency_f;
        img_decode_frame_f            *frame_f;
        img_decode_set_value_f        *set_value_f;
        uintptr_t                      data;
} img_decode_callouts_t;
```

Description:

The `img_decode_callouts_t` structure defines a decoder callout table. It provides the decoder with a list of callouts for it to invoke at various stages of the decoding:

- `img_decode_choose_format_f` **choose_format_f* (p. 36)
- `img_decode_setup_f` **setup_f* (p. 37)
- `img_decode_abort_f` **abort_f* (p. 38)
- `img_decode_scanline_f` ** scanline_f* (p. 39)
- `img_decode_set_palette_f` **set_palette_f* (p. 41)
- `img_decode_set_transparency_f` **set_transparency_f* (p. 40)
- `img_decode_frame_f` **frame_f* (p. 42)
- `img_decode_set_value_f` **set_value_f* (p. 42)
- `uintptr_t` *data* (p. 43)

`img_decode_choose_format_f` **choose_format_f*

A pointer to a function that chooses a format for the image from the list provided by the decoder. You will get this callout first (unless the format was preselected, in which case it will not be called at all).

The function takes this form:

```
typdef unsigned (img_decode_choose_format_f) (
        uintptr_t data,
        img_t *img,
        const img_format_t *formats,
        unsigned nformats );
```

The arguments for this function are:

© 2014, QNX Software Systems Limited

data

Application data — the value of the *data* member of the `img_decode_callouts_t`.

img

A pointer to a partially filled `img_t` structure providing vital information about the frame.

formats

An array of possible `img_format_t` (p. 82) formats to choose from.

nformats

The number of elements in the *formats* array.

If you do not supply a *choose_format()* callout, the library will supply a default that chooses the first format in the list.

The function should return an index within the *formats* array, or an out-of-bounds value (for example, *nformats*) to indicate that no format was desired; the decoder will error out with `IMG_ERR_NOSUPPORT`.

Alternatively, it can return an out-of-bounds value (for example, *nformats*) to indicate that none of the formats were acceptable. In this case it can pick a different format by setting *img->format* and asserting the `IMG_FORMAT` bit of *img->flags*. In this case, the end result will be the same as if this format was initially selected before load time.

`img_decode_setup_f*` *setup_f*

A pointer to a function that does any setup required to begin frame decoding.

The function takes this form:

```
typedef int (img_decode_setup_f) ( uintptr_t data,
                                   img_t *img,
                                   unsigned flags );
```

The arguments for this function are:

data

Application data — the value of the *data* member of the `img_decode_callouts_t`.

img

A pointer to a partially filled `img_t` structure providing vital information about the frame being decoded.

flags

© 2014, QNX Software Systems Limited

Flags to provide hints about the data and how it will arrive, which may influence assumptions and actions your function takes. Flags can have the following values:

- `IMG_SETUP_PAL_SHARED` — the palette (if any) can be shared between this and other frames in the decoding. This is meaningful for multiple frame formats, but can otherwise be ignored.
- `IMG_SETUP_TOP_DOWN` — scanlines are completed in order, moving from the top down
- `IMG_SETUP_BOTTOM_UP` — scanlines are completed in order, moving from the bottom up
- `IMG_SETUP_MULTIPASS` — scanline processing is fragmented across multiple passes (for example, planar and some interlacing schemes).

Your setup function can, but does not have to, set up the *access* field for the `img_t`. This field tells the decoder how to record the image data being decoded. You set this up by setting up either *img->access.direct* or *img->access.indirect* and asserting the appropriate flags, `IMG_DIRECT` or `IMG_INDIRECT`.

Alternatively, the code that called *img_decode_frame()* (or *img_load_file()* etc) could set this up before calling *img_deocode_frame()*, *img_load_file()*, and so on, or you can rely on the library's default behavior, which is to allocate memory from system RAM for the image and/or palette (the library will do this only if the `IMG_DIRECT`, `IMG_INDIRECT` and `IMG_PALETTE` flags aren't already set).

If you rely on this default behavior, you can later free the memory by simply freeing `img_t`::*access.direct.data* or `img_t`::*palette* (but not both). You only need to do this if the image decode succeeds, and only once you are done with the image. You do not need to worry about freeing if the decoding fails, as it's taken care of automatically.

It should return `IMG_ERR_OK` if everything is ok, otherwise return some other error code. Anything other than `IMG_ERR_OK` causes the decoding to stop and the error code is propagated back to the application.

`img_decode_abort_f*` *abort_f*

A pointer to a function that called if the decoding fails (after setup_f has been called). The library will automatically release any memory it allocated as part of the setup (if you have relied on the default behavior described above in `img_decode_setup_f` (p. 37)).

The function takes this form:

```
typedef void (img_decode_abort_f) (
        uintptr_t data,
        img_t *img );
```

The arguments for this function are:

© 2014, QNX Software Systems Limited

data

Application data — the value of the *data* member of the `img_decode_callouts_t`.

info

A pointer to an `img_t` structure that needs to be released.

`img_decode_scanline_f*` *scanline_f*

A pointer to a function that's invoked to notify the application when a scanline has been decoded.

The function takes this form:

```
typedef int (img_decode_scanline_f) (
        uintptr_t data,
        img_t *img,
        unsigned row,
        unsigned npass_line,
        unsigned npass_total );
```

The arguments for this function are:

data

Application data — the value of the *data* member of the `img_decode_callouts_t`.

img

A pointer to an `img_t` describing the frame.

row

The index of the scanline that has been decoded. Scanlines are numbered starting at `0` (topmost scanline) to (*h* - 1), where *h* represents the height of the image.

npass_line

The number of additional passes required (after this pass) to complete the scanline. Planar formats and some interlaced formats partition scanlines across multiple passes, delivering only portions of the data at a time (this is the case when the `IMG_SETUP_MULTIPASS` flag bit is set in the *setup_f()* callout). You know the scanline is complete when this value is `0`.

npass_total

© 2014, QNX Software Systems Limited

The number of additional scanline passes remaining (after this pass) to complete the entire frame. You will know the frame is complete when this value is passed in as `0`.

This total includes partial passes, where the `IMG_SETUP_MULTIPASS` flag was set in the *setup_f()* callout. If this flag was not set, then a "scanline pass" is equivalent to "scanline completion", that is, *npass_total* reflects the number of lines left to be decoded. In either case, this value gives you a gauge of the total work left versus what has already been completed.

This function should return `IMG_ERR_OK` to continue decoding or some other value to abort the decoding. The code you return is propagated back to the application. Normally `IMG_ERR_INTR` is a good value to use in this case, unless there's another value you wish to use.

`img_decode_set_transparency_f*` *set_transparency_f*

A pointer to a function that notifies the application that the image has a transparency color, which means that the designer of the image intented for a particular color in the image to be treated as though it were transparent. Pixels of that color in the source image should not be rendered to the destination.

This function is called only for image formats that support transparency in the form of a color mask (as opposed to transparency achieved through alpha blending), which currently only applies to the GIF format.

The function takes this form:

```
typedef void (img_decode_set_transparency_f) (
        uintptr_t data,
        img_t *img,
        img_color_t color );
```

The arguments for this function are:

data

Application data — the value of the *data* member of the `img_decode_callouts_t`.

img

A pointer to an `img_t` describing the frame.

color

The color to treat as transparent. The value will be:

- an index number (0-255) for palette-based or `IMG_FMT_G8`

© 2014, QNX Software Systems Limited

- a 16-bit value for 16bpp formats (encoding is the same as the image format)
- a 32-bit value for 24 or 32 bpp formats (encoding is IMG_FMT_PKHE_ARGB8888)

The *transparency* field of the img_t will have already been set up by the time you receive this callout. The only reason for this callout is to do additional processing as a result of the presence of the transparency.

img_decode_set_palette_f* ***set_palette_f***

A pointer to a function that notifies the application of an image's palette. This function is called after setup before any decoding of the body is done.

If you do not supply a callout, or the supplied callout returns nonzero, the library copies the palette data into the buffer pointed to by *img->palette* (if the IMG_PALETTE flag bit is set), converting the data to IMG_FMT_PKHE_ARGB8888 as necessary (the proper representation of an img_color_t). If you supply a callout that returns 0, no copying takes place.

The function takes this form:

```
typedef int (img_decode_set_palette_f) (
            uintptr_t data,
            img_t *img,
            const uint8_t *palette,
            img_format_t format );
```

The arguments for this function are:

data

Application data — the value of the *data* member of the img_decode_callouts_t.

img

A pointer to an img_t describing the frame.

palette

A pointer to a palette that is set for the image. The format of the palette data is described by *format*.

format

The format of the palette. Return 0 if you want the palette data copied to *img->palette*, or nonzero to indicate that you don't want this copy/conversion to take place.

© 2014, QNX Software Systems Limited

It should return `IMG_ERR_OK` if everything is ok, otherwise return some other error code. Anything other than `IMG_ERR_OK` causes the decoding to stop and the error code is propagated back to the application.

`img_decode_frame_f*` *frame_f*

A pointer to a function that is called once a frame is successfully decoded.

The function takes this form:

```
typedef void ( img_decode_frame_f ) (
        uintptr_t data,
        img_t *img );
```

`img_decode_set_value_f*` *set_value_f*

A pointer to a function that is called to tell the application about additional properties encountered in the file.

The function takes this form:

```
typedef int (img_decode_set_value_f) (
        uintptr_t data,
        img_t *img,
        unsigned type,
        uintptr_t value );
```

Some image file formats specify additional information that is not represented in an `img_t` (p. 106), for example DPI, x/y position, comments, and so on. The purpose of this callout is to provide a way for the application to receive this information.

Currently, this callout is called only when processing progressive images (such as progressive JPEG image files).

The arguments for this function are:

data

Application data — the value of the *data* member of the `img_decode_callouts_t`.

img

A pointer to an `img_t` describing the frame.

type

One of the types from the `img_value_type` enum.

value

© 2014, QNX Software Systems Limited

See the `img_value_type` enum.

The `img_value_type` enumeration provides value types that are used by the library and takes the following form:

```
#include <img/img.h>
enum img_value_type {
        IMG_VALUE_TYPE_INVALID = 0,
        IMG_VALUE_TYPE_PROGRESSIVE,
        IMG_VALUE_TYPE_ANIM_PLAY_COUNT,
        IMG_VALUE_TYPE_ANIM_FRAME_DELAY,
        IMG_VALUE_TYPE_FRAME_COUNT
};
```

The enumeration has these possible types of values:

IMG_VALUE_TYPE_INVALID

Not a valid type.

IMG_VALUE_TYPE_PROGRESSIVE

Indicates whether the image is progressive or non-progressive. A value of `0` indicates a non-progressive image while a value of `1` indicates a progressive one.

IMG_VALUE_TYPE_ANIM_PLAY_COUNT

The number of times an animation should be played. A value of `0` means playing forever.

IMG_VALUE_TYPE_ANIM_FRAME_DELAY

The minimum time for which the current frame must be displayed, in milliseconds.

IMG_VALUE_TYPE_FRAME_COUNT

The number of frames in the image. This value is never `0`.

Codecs are not required to implement the *`set_value_f()`* (p. 42) callout. Either the information is unavailable, or reasonable defaults should be assumed (see *img_value_type* (p. 43)).

It should return `IMG_ERR_OK` if everything is OK, otherwise it should return some other error code. Anything other than `IMG_ERR_OK` causes the decoding to stop and the error code to be propagated back to the application.

`uintptr_t` *data*

User-defined data passed as an additional argument to callouts.

© 2014, QNX Software Systems Limited

Classification:

Image library

© 2014, QNX Software Systems Limited

img_decode_finish()

Release decode resources

Synopsis:

```
#include <img/img.h>

int img_decode_finish( img_codec_t  codec,
                       io_stream_t *input,
                       uintptr_t   *decode_data );
```

Arguments:

codec

The handle of the codec that was used to decode.

input

A pointer to an input stream for the image data.

decode_data

The address of the `uintptr_t` that was used for *img_decode_begin()* (p. 34) and *img_decode_frame()* (p. 47).

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function finalizes the decode process and releases resources allocated during a decoding session. You should call this function after you have finished decoding a series of frames, to release any resources that the decoder may have allocated in association with those frames.

You do not need to decode all the frames in a stream, but you should always follow up with *img_decode_finish()* (p. 45) when you have decoded the frames you are interested in to avoid potential memory leaks.

Returns:

`IMG_ERR_OK`

© 2014, QNX Software Systems Limited

Success.

IMG_ERR_NOTIMPL

The *codec* doesn't provide an implementation for this function.

Other

Any other code that a decoder's *begin()* function may pass back to flag an error (see `img_errno.h` for a list of defined errors).

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_decode_frame()

Decode a frame

Synopsis:

```
#include <img/img.h>

int img_decode_frame( img_codec_t                  codec,
                      io_stream_t                 *input,
                      const img_decode_callouts_t *callouts,
                      img_t                       *img,
                      uintptr_t                   *decode_data
 );
```

Arguments:

codec

The handle of the codec to use to decode the frame.

input

The input source.

callouts

A pointer to an `img_decode_callouts_t` (p. 36) structure that provides system callouts for the decoder. If you pass `NULL` for this value, a set of default callouts is used. See the description of `img_decode_callouts_t` (p. 36) for more details.

img

The address of an `img_t` (p. 106) structure to fill with information regarding the decoded frame.

decode_data

The address of the `uintptr_t` that was used for *img_decode_begin()* (p. 34).

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

© 2014, QNX Software Systems Limited

Description:

This function decodes a frame. You need to call *img_decode_begin()* (p. 34) first to prepare for the decode, and *img_decode_finish()* (p. 45) to release any resources allocated for the decode.

Returns:

IMG_ERR_OK

Success. The complete frame was decoded.

IMG_ERR_MEM

Memory-allocation failure.

IMG_ERR_NOSUPPORT

Output data format not supported; the codec and application could not agree on an output format.

IMG_ERR_NODATA

No frame data was present. This return code indicates the end of a multi-frame data source.

IMG_ERR_CORRUPT

Invalid data was encountered in the stream, preventing the decode from proceeding. Some of the frame may be intact.

IMG_ERR_TRUNC

Premature end of data encountered. Some of the frame may be intact.

IMG_ERR_INTR

Decoding was interrupted by the application.

IMG_ERR_DLL

Error accessing the codec DLL; check *errno* and/or try running your application with `DL_DEBUG=1`.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No

© 2014, QNX Software Systems Limited

Safety:	
Thread	No

© 2014, QNX Software Systems Limited

img_decode_frame_resize()

Decode and resize a frame from a stream

Synopsis:

```
#include <img/img.h>

int img_decode_frame_resize(img_codec_t                codec,
                            io_stream_t                *input,
                            const img_decode_callouts_t
*callouts,
                            img_t                      *img,
                            uintptr_t
*decode_data );
```

Arguments:

codec

The handle of the codec to use to decode the frame.

input

The input source.

callouts

A pointer to an `img_decode_callouts_t` (p. 36) structure that provides system callouts for the decoder. If you pass `NULL` for this value, a set of default callouts is used. See the description of `img_decode_callouts_t` (p. 36) for more details.

img

The address of an `img_t` (p. 106) structure to fill with information regarding the decoded frame.

If you set the image width and height (`img.w` and `img.h`) before calling this function, then the image is sized to fit the specified dimensions rather than clipped, as it is with `img_load_file()`. The resizing is performed on-the-fly during the decoding, without incurring the memory penalty of loading the entire original image and then subsequently resizing it.

If you specify only one of the dimensions, then the other dimension is calculated based on the aspect ratio of the original image.

© 2014, QNX Software Systems Limited

You also set the corresponding dimension flag in the `img` structure. For example, if you specify the width, then you set the `IMG_W` bit in flags.

decode_data

The address of the `uintptr_t` that was used for *img_decode_begin()* (p. 34).

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function decodes a frame and optionally resizes it if the image width and height are specified. You need to call *img_decode_begin()* (p. 34) first to prepare for the decode, and *img_decode_finish()* (p. 45) to release any resources allocated for the decode.

Returns:

`IMG_ERR_OK`

Success. The complete frame was decoded.

`IMG_ERR_MEM`

Memory-allocation failure.

`IMG_ERR_NOSUPPORT`

Output data format not supported; the codec and application could not agree on an output format.

`IMG_ERR_NODATA`

No frame data was present. This return code indicates the end of a multi-frame data source.

`IMG_ERR_CORRUPT`

Invalid data was encountered in the stream, preventing the decode from proceeding. Some of the frame may be intact.

`IMG_ERR_TRUNC`

Premature end of data encountered. Some of the frame may be intact.

`IMG_ERR_INTR`

© 2014, QNX Software Systems Limited

Decoding was interrupted by the application.

IMG_ERR_DLL

Error accessing the codec DLL; check *errno* and/or try running your application with `DL_DEBUG=1`.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_decode_get_frame_count()

Get the total number of frames in the image stream

Synopsis:

```
#include <img/img.h>

int img_decode_get_frame_count( img_codec_t  codec,
                                io_stream_t *input,
                                uintptr_t   *decode_data,
                                unsigned    *count );
```

Arguments:

codec

The handle of the codec to use to decode the frame.

input

The input source.

decode_data

The address of the `uintptr_t` that was used for *img_decode_begin()* (p. 34).

count

A pointer to an unsigned integer where the number of frames can be stored.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

Gets the total number of image frames in the image stream.

This function may only be called after calling *img_decode_begin()* (p. 34), and before calling *img_decode_finish()* (p. 45). If the function is successful, the count parameter will contain a valid number of frames.

Calling this function may require reading the entire image stream to determine the correct count of frames that can be decoded. For example, some image file formats do not store the total number of frames in the header (for example,

© 2014, QNX Software Systems Limited

GIF), thus the whole stream must be read to determine EOF and get an exact count. If you do not want this behavior, provide a `set_value_f` callout instead, and wait for the `IMG_VALUE_TYPE_FRAME_COUNT` value to be returned once it is known.

Returns:

`IMG_ERR_OK`

Success. The complete frame was decoded.

`IMG_ERR_PARM`

One of the parameters supplied was invalid.

`IMG_ERR_NOTIMPL`

The codec does not provide an implementation for this function.

`IMG_ERR_NODATA`

No frame data was present. This return code indicates the end of a multi-frame data source.

`IMG_ERR_CORRUPT`

Invalid data was encountered in the stream, preventing the decode from proceeding. Some of the frame may be intact.

`IMG_ERR_TRUNC`

Premature end of data encountered. Some of the frame may be intact.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_decode_set_frame_index()

Set the current frame index in the image decoder

Synopsis:

```
#include <img/img.h>

int img_decode_set_frame_index(img_codec_t  codec,
                               io_stream_t *input,
                               uintptr_t   *decode_data,
                               unsigned     index );
```

Arguments:

codec

The handle of the codec to use to decode the frame.

input

The input source.

decode_data

The address of the `uintptr_t` that was used for *img_decode_begin()* (p. 34).

index

The frame index that should be set. The first frame has index 0, and the last frame has index `img_decode_get_frame_count() - 1`.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

Sets the current frame index for the current decode. Use this function to perform non-sequential decoding of frames from an image stream.

This function may only be called after calling *img_decode_begin()* (p. 34), and before calling *img_decode_finish()* (p. 45). If the function is successful, the frame index will be updated and will be used on the next call to *img_decode_frame()* (p. 47) or *img_decode_frame_resize()* (p. 50).

© 2014, QNX Software Systems Limited

Returns:

IMG_ERR_OK

Success. The complete frame was decoded.

IMG_ERR_PARM

One of the parameters supplied was invalid.

IMG_ERR_NOTIMPL

The codec does not provide an implementation for this function.

IMG_ERR_NODATA

No frame data was present. This return code indicates the end of a multi-frame data source.

IMG_ERR_CORRUPT

Invalid data was encountered in the stream, preventing the decode from proceeding. Some of the frame may be intact.

IMG_ERR_TRUNC

Premature end of data encountered. Some of the frame may be intact.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_decode_validate()

Find a codec for decoding

Synopsis:

```
#include <img/img.h>

int img_decode_validate( const img_codec_t *codecs,
                         size_t             ncodecs,
                         io_stream_t       *input,
                         unsigned          *codec );
```

Arguments:

codecs

A pointer to an array of `img_codec_t` handles providing a list of codecs to try. The function will try each codec in order until it finds one that validates the data in the stream.

ncodecs

The number of items in the *codecs* array.

input

The input source.

codec

The address of an unsigned value where the function stores the index of the codec that validated the datastream. This memory is left untouched if no such codec is found.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function finds a suitable codec for decoding.

Returns:

Status of the operation:

© 2014, QNX Software Systems Limited

IMG_ERR_OK

Success; an appropriate codec was found. Check *codec* for the index of the codec in the *codecs* array which validated the datastream.

IMG_ERR_DLL

An error occurred processing the DLL that handles the file type. Check to make sure that the DLL is not missing or corrupt.

IMG_ERR_FORMAT

No installed codec recognized the input data as a format it supports. This could mean the data is of a format that's not supported, or the datastream is corrupt.

IMG_ERR_NOTIMPL

The codec doesn't provide an implementation for this function.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_dtransform()

Convert an image from one format to another

Synopsis:

```
#include <img/img.h>

int img_dtransform( const img_t *src,
                    img_t *dst );
```

Arguments:

src

The image you want to convert from.

dst

The image you want to convert to.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function converts an `img_t` from one format to another. Best to use this when you only need to do the transform once on a single img_t. Returns IMG_ERR_OK if the transform succeeed, otherwise returns one of the documented error codes of *img_dtransform_create()* (p. 63).

This function cannot convert from a direct format source to a palette-based format destination.

Returns:

`IMG_ERR_OK`

Success.

`IMG_ERR_NOSUPPORT`

One of the formats specified is invalid.

© 2014, QNX Software Systems Limited

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_dtransform_apply()

Apply a data transform

Synopsis:

```
#include <img/img.h>

void img_dtransform_apply( img_dtransform_t xform,
                           const uint8_t *src,
                           uint8_t *dst,
                           unsigned n );
```

Arguments:

xform

An opaque `img_dtransform_t` filled in by *img_dtransform_create()* (p. 63).

src

The source pixel data buffer.

dst

The destination pixel data buffer.

n

The number of pixels to transform.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function applies a previously created transform *xform* to transform *n* pixels of pixel data contained in *src* and store the results in *dst*. The transform can be done in place (that is, *src* can be the same as *dst*).

Classification:

Image library

© 2014, QNX Software Systems Limited

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_dtransform_create()

Prepare to transform an image

Synopsis:

```
#include <img/img.h>

int img_dtransform_create( const img_t *src,
                           const img_t *dst,
                           img_dtransform_t *xform );
```

Arguments:

src

The image you want to convert from

dst

The image you want to convert to

xform

The address to an opaque `img_dtransform_t` where the function stores the transform it creates.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function pepares a data transformation from one format to another, from the format in the *src* image to the format in the *dst* image. Once this function is called, you call *img_dtransform_apply()* (p. 61) to apply the transformation, then *img_dtransform_free()* (p. 65) to free the *xform* opaque structure.

- Data transforms are capable of handling palette-based formats, abstracting the details of conversions and/or expansion. It's generally easiest to use this construct when converting data from one arbitrary format to another.
- Conversion to a palette-based format is not supported.

© 2014, QNX Software Systems Limited

Returns:

IMG_ERR_OK

Success. The *xform* is valid and must be freed when the transform is finished. For any other return code (error), the *xform* isn't valid, and it must not be freed.

IMG_ERR_PARM

Required bits in the flags member of *src* aren't set (at a minimum `IMG_H` and `IMG_W` need to be set).

IMG_ERR_MEM

Insufficient memory for transform

IMG_ERR_NOSUPPORT

No support for the requested transform.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_dtransform_free()

Free a transform structure

Synopsis:

```
#include <img/img.h>

void img_dtransform_free (img_dtransform_t xform );
```

Arguments:

xform

The transform structure created by *img_dtransform_create()* (p. 63).

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function releases a transform previously created by *img_dtransform_create()* (p. 63) and thereby render it invalid.

Returns:

`IMG_ERR_OK`

Success.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_encode_begin()

Prepare to encode one or more frames to a stream

Synopsis:

```
#include <img/img.h>

int img_encode_begin( img_codec_t codec,
                      io_stream_t *output,
                      uintptr_t *encode_data );
```

Arguments:

codec

The codec to use. To figure out a codec to use, see *img_codec_list()*, *img_codec_list_byext()*, and *img_codec_list_bymime()*.

output

The output destination.

encode_data

An address of a `uintptr_t` which the encoder uses to store data it needs across the encode process. Pass a valid address of a `uintptr_t` initialized to `0`, not `NULL`.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function prepares to encode a frame (or series of frames) to a stream.

Returns:

`IMG_ERR_OK`

Success.

`IMG_ERR_NOTIMPL`

The codec doesn't provide an implementation for this function.

© 2014, QNX Software Systems Limited

Other

Any other code that a encoder's *encode_begin()* function may pass back to flag an error (see `img_errno.h` for a list of defined errors).

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_encode_callouts_t

Encoder callout table

Synopsis:

```
#include <img/img.h>

typedef struct {
    img_encode_choose_format_f      *choose_format_f;
    img_encode_setup_f              *setup_f;
    img_encode_abort_f              *abort_f;
    img_encode_scanline_f           *scanline_f;
    img_encode_set_palette_f        *get_palette_f;
    img_encode_set_transparency_f   *get_transparency_f;
    img_encode_frame_f              *frame_f;
    uintptrt_t                       data;

} img_encode_callouts_t;
```

Description:

The img_encode_callouts_t (p. 68) structure defines an encoder callout table. It provides the encoder with a list of callouts for it to invoke at various stages of the encoding:

- img_encode_choose_format_f *choose_format_f (p. 68)
- img_encode_setup_f* setup_f (p. 69)
- img_encode_abort_f* abort_f (p. 70)
- img_encode_scanline_f* scanline_f (p. 70)
- img_encode_get_transparency_f* get_transparency_f (p. 71)
- img_encode_get_palette_f* get_palette_f (p. 72)
- img_encode_frame_f* frame_f (p. 73)
- uintptrt_t data (p. 73)

img_encode_choose_format_f *choose_format_f

A pointer to a function that chooses an alternate format for the image from the list provided by the decoder. This is the first callout called during an encode, but it will only be called if the format of the supplied image cannot be represented by the encoder. In this case, the application may prepare to provide the data in one of the formats requested, or it may abort the encode altogether.

The function takes this form:

```
typdef unsigned (img_encode_choose_format_f) (
        uintptrt_t data,
        const img_t *img,
```


© 2014, QNX Software Systems Limited

```
        const img_format_t *formats,
        unsigned nformats );
```

The arguments for this function are:

data

Application data -- the value of the *data* member of the `img_encode_callouts_t`.

img

A pointer to the `img_t` structure describing the frame being encoded.

formats

An array of possible `img_format_t` formats to choose from.

nformats

The number of elements in the *formats* array.

If you do not supply a *choose_format()* callout, the library will choose the format that best matches the provided image, and it will automatically convert the data to that format as needed.

The function should return an index within the formats array, or an out-of-bounds value (for example, *nformats*) to indicate that no format was desired; the encoder will error out with `IMG_ERR_NOSUPPORT`.

img_encode_setup_f* ***setup_f***

A pointer to a function that does any setup required to begin frame encoding.

The function takes this form:

```
typedef int (img_encode_setup_f) (
        uintptrt_t data,
        img_t *img,
        unsigned flags );
```

The arguments for this function are:

data

Application data — the value of the *data* member of the `img_encode_callouts_t`.

img

A pointer to the `img_t` structure describing the frame being encoded.

flags

© 2014, QNX Software Systems Limited

Flags to provide hints about how the data will be encoded, which may influence assumptions and actions your function takes. It can have the following values:

- `IMG_SETUP_TOP_DOWN` — scanlines are encoded in order, moving from the top down
- `IMG_SETUP_BOTTOM_UP` — scanlines are encoded in order, moving from the bottom up
- `IMG_SETUP_MULTIPASS` — scanline are fragmented across multiple passes (for example, planar and some interlacing schemes).

This function should return `IMG_ERR_OK` if everything is ok, otherwise return some other error code. Anything other than `IMG_ERR_OK` causes the encode to stop and the error code is propagated back to the application.

img_encode_abort_f* ***abort_f***

A pointer to a function that called if the encode fails (after *setup_f()* has been called).

The function takes this form:

```
typedef void (img_encode_abort_f) (
        uintptrt_t data,
        img_t *img );
```

The arguments for this function are:

data

Application data — the value of the *data* member of the `img_encode_callouts_t`.

img

A pointer to the `img_t` structure describing the frame being encoded.

img_encode_scanline_f* ***scanline_f***

A pointer to a function that's invoked to notify the application when a scanline has been encoded.

The function takes this form:

```
typedef int (img_encode_scanline_f) (
        uintptrt_t data,
        img_t *img,
        unsigned row,
        unsigned npass_line,
        unsigned npass_total );
```

The arguments for this function are:

© 2014, QNX Software Systems Limited

data

Application data — the value of the *data* member of the `img_encode_callouts_t`.

img

A pointer to an `img_t` describing the frame being encoded.

row

The index of the scanline that has been encoded. Scanlines are numbered starting at 0 (the topmost scanline) to image height - 1.

npass_line

The number of additional passes required (after this pass) to complete the scanline. Planar formats and some interlaced formats partition scanlines across multiple passes, encoding only portions of the data at a time (this is the case when the `IMG_SETUP_MULTIPASS` flag bit is set in the *setup_f()* callout). You know the scanline is complete when this value is `0`.

npass_total

The number of additional scanline passes remaining (after this pass) to complete the entire frame. You will know the frame is complete when this value is passed in as *0*.

This total includes partial passes, where the `IMG_SETUP_MULTIPASS` flag was set in the *setup_f()* callout. If this flag was not set, then a "scanline pass" is equivalent to "scanline completion", that is, *npass_total* reflects the number of lines left to be encoded. In either case, this value gives you a gauge of the total work left versus what has already been completed.

This function should return `IMG_ERR_OK` to continue encoding or some other value to abort the encode. The code you return is propagated back to the application. Normally `IMG_ERR_INTR` is a good value to use in this case, unless there's another value you wish to use.

img_encode_get_transparency_f* ***get_transparency_f***

A pointer to a function that satisfies the request of the image transparency color. You only need to provide this function if, for some reason, the transparency color is not accurately represented in the transparency field of the `img_t`.

The function takes this form:

```
typedef int (img_encode_get_transparency_f) (
        uintptrt_t data,
        img_t *img,
        img_color_t *color );
```

© 2014, QNX Software Systems Limited

The arguments for this function are:

data

Application data — the value of the data member of the `img_encode_callouts_t`.

img

A pointer to an `img_t` describing the frame.

color

A pointer to an `img_color_t` to fill with the transparent color. The format of this data should match the format of the frame data itself.

If you do not provide this callout, the library will automatically use and convert the image transparency field as needed.

This function should return `IMG_ERR_OK` to signify the validity of the requested field. If some other value is returned, the encoder will ignore the transparency and it will not be represented in the resulting encoded data. The encode will proceed regardless.

img_encode_get_palette_f* ***get_palette_f***

A pointer to a function that satisfies the request of an image's palette. You only need to provide this function if, for some reason, the palette is not accurately represented in the `img_t`.

The function takes this form:

```
typedef int (img_encode_get_palette_f) (
        uintptrt_t data,
        img_t *img,
        uint8_t *palette,
        img_format_t format );
```

The arguments for this function are:

data

Application data — the value of the *data* member of the `img_encode_callouts_t`.

img

A pointer to an `img_t` describing the frame being encoded.

palette

A pointer to a buffer to fill with palette data.

© 2014, QNX Software Systems Limited

format

The format that the encoder is expecting the palette data to be in.

This function should return `IMG_ERR_OK` if everything is ok, otherwise return some other error code. Anything other than `IMG_ERR_OK` causes the encode to stop and the error code is propagated back to the application.

img_encode_frame_f* ***frame_f***

A pointer to a function that is called once a frame is successfully encoded.

The function takes this form:

```
typedef void ( img_encode_frame_f ) (
        uintptrt_t data,
        img_t *img );
```

The arguments for this function are:

data

Application data — the value of the *data* member of the `img_encode_callouts_t`.

img

A pointer to an `img_t` describing the frame being encoded.

uintptrt_t ***data***

User-defined data passed as an additional argument to callouts.

Classification:

Image library

© 2014, QNX Software Systems Limited

img_encode_finish()

Release encode resources

Synopsis:

```
#include <img/img.h>

int img_encode_finish( img_codec_t  codec,
                       io_stream_t *output,
                       uintptr_t   *encode_data );
```

Arguments:

codec

The handle of the codec that was used to encode.

output

A pointer to an output stream for the image data.

encode_data

The address of the `uintptr_t` that was used for *img_encode_begin()* (p. 66) and *img_encode_frame()* (p. 76).

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function finalizes the encode process and releases resources allocated during an encoding session. You should call this function after you've finished encoding a series of frames, to release any resources that the encoder may have allocated in association with those frames.

Returns:

`IMG_ERR_OK`

Success.

`IMG_ERR_NOTIMPL`

© 2014, QNX Software Systems Limited

The codec doesn't provide an implementation for this function.

Other

Any other code that a decoder's *encode_finish()* function may pass back to flag an error (see `img_errno.h` for a list of defined errors).

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_encode_frame()

Encode a frame

Synopsis:

```
#include <img/img.h>

int img_encode_frame( img_codec_t                  codec,
                      io_stream_t                 *output,
                      const img_encode_callouts_t *callouts,
                      img_t                       *img,
                      uintptr_t                   *encode_data
 );
```

Arguments:

codec

The handle of the codec to use to encode the frame.

output

The output destination.

callouts

A pointer to an `img_encode_callouts_t` structure that provides system callouts for the encoder. If you pass `NULL` for this value, the library uses a set of default callouts. See the description of `img_encode_callouts_t` (p. 68) for more details.

img

The address of an `img_t` structure describing the frame to be encoded.

encode_data

The address of the `uintptr_t` that was used for *img_encode_begin()* (p. 66).

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

© 2014, QNX Software Systems Limited

Description:

This function encodes a frame. You need to call *img_encode_begin()* (p. 66) first to prepare for the encode, and *img_encode_finish()* (p. 74) to release any resources allocated for the encode.

Returns:

IMG_ERR_OK

Success. The complete frame was encoded.

IMG_ERR_MEM

Memory-allocation failure.

IMG_ERR_NOSUPPORT

Input data format not supported; the codec and application could not agree on an output format.

IMG_ERR_TRUNC

Error writing data; file was truncated.

IMG_ERR_INTR

Encoding was interrupted by the application.

IMG_ERR_DLL

Error accessing the codec DLL; check *errno* and/or try running your application with `DL_DEBUG=1`.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_expand_getfunc()

Get a function to convert a palette format to a direct format

Synopsis:

```
#include <img/img.h>

img_expand_f *img_expand_getfunc( img_format_t src,
                                  img_format_t lut )
```

Arguments:

src

The palette-based `img_format_t` (p. 82) image format to convert from

lut

The `img_format_t` (p. 82) direct image format to convert to

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function returns a pointer to a data conversion function (or `NULL` if the requested conversion is not supported) which you can call to "expand" (that is, convert) a run of pixels from a palette-based format *src* to a "direct" format in a lookup table *lut*. A conversion function takes the form:

```
void img_expand_f( const uint8 *src,
                   uint8 *dst,
                   unsigned n,
                   const uint8 *lut);
```

Here's a sample procedure for converting from PAL8 to ARGB1555:

1. Convert your lookup table to the destination format:

   ```
   img_convert_data(IMG_FMT_PKHE_ARGB8888, palette, IMG_FMT_PKLE_ARGB1555, palette, npalette);
   ```

2. Get the handle to the expand function that will expand the data:

   ```
   img_expand_getfunc(IMG_FMT_PAL8, IMG_FMT_PKLE_ARGB1555);
   ```

3. Call the expand function for each run of indexed data you want to expand:

   ```
   expand_f(sptr, dptr, npixels, palette);
   ```


© 2014, QNX Software Systems Limited

You can use *img_dtransform_create()* (p. 63) instead to abstract the details of expansion and conversion.

Returns:

IMG_ERR_OK

Success.

IMG_ERR_NOSUPPORT

One of the formats specified is invalid.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

IMG_FMT_BPL()

Calculate the minimum number of bytes required to represent a pixel run

Synopsis:

```
#include <img/img.h>

#define IMG_FMT_BPL(_fmt, _w) ...
```

Arguments:

_fmt

An `img_format_t` (p. 82) specifying the data format of the frame.

_w

The width of the frame, in pixels.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This macro calculates the minimum number of bytes required to represent a run of pixels.

Returns:

The minimum number of bytes required per scanline of the frame.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

IMG_FMT_BPP()

Determine the number of bits per pixel for the specified format

Synopsis:

```
#include <img/img.h>

#define IMG_FMT_BPP(_fmt) ...
```

Arguments:

_fmt

An `img_format_t` (p. 82) specifying the data format of the frame.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This macro determines the number of bits per pixel for the specified data format.

Returns:

The number of bits required to represent a single pixel.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_format_t

Image formats

Synopsis:

```
#include <img/img.h>

typedef enum {
   ...
} img_format_t;
```

Description:

The `img_format_t` is an enumeration of these possible image formats:

IMG_FMT_INVALID

Invalid image format

IMG_FMT_MONO

Monochromatic bitmap with 1 bit/pixel, packing 8 pixels per byte.

IMG_FMT_G8

8-bits/pixel graymap.

IMG_FMT_PAL1

1-bit/pixel index into a palette of 2 entries, packing 8 pixels per byte.

IMG_FMT_PAL4

4-bits/pixel index into a palette of up to 16 entries, packing 2 pixels per byte.

IMG_FMT_PAL8

8-bits/pixel index into a palette of up to 256 entries.

IMG_FMT_PKLE_RGB565

16-bits/pixel RGB packed into 16-bit little-endian integer type with bits 0-4 for B, 5-10 for G, and 11-15 for R.

IMG_FMT_PKBE_RGB565

A big-endian version of `IMG_FMT_PKLE_RGB565`

IMG_FMT_PKLE_ARGB1555

© 2014, QNX Software Systems Limited

16-bits/pixel ARGB packed into 16-bit little-endian integer type with bits 0-4 for B, 5-9 for G, 10-14 for R and most significant bit for A.

IMG_FMT_PKBE_ARGB1555

A big-endian version of IMG_FMT_PKLE_ARGB1555

IMG_FMT_BGR888

24-bits/pixel BGR with 8 bits per channel as an ordered byte sequence.

IMG_FMT_RGB888

24-bits/pixel RGB with 8 bits per channel as an ordered byte sequence.

IMG_FMT_RGBA8888

32-bits/pixel RGBA with 8 bits per channel as an ordered byte sequence.

IMG_FMT_PKLE_ARGB8888

32-bits/pixel ARGB packed into 32-bit little-endian integer type with byte 0 (least-significant byte) for B, byte 1 for G, byte 2 for R and byte 3 for A.

IMG_FMT_PKBE_ARGB8888

A big endian version of IMG_FMT_PKLE_ARGB8888

IMG_FMT_PKLE_XRGB8888

24-bits/pixel BGR with 8 bits per channel as an ordered byte sequence, followed by a single byte of padding.

IMG_FMT_PKBE_XRGB8888

A big endian version of IMG_FMT_PKLE_XRGB8888

In addition to PKLE and PKBE variants listed above, there are PKHE and PKOE variants that make it easier to identify host-endian (HE) formats and other-endian (OE). So for example, if your code is executing on an x86 platform, IMG_FMT_PKHE_ARGB1555 equals IMG_FMT_PKLE_ARGB1555.

Classification:

Image library

© 2014, QNX Software Systems Limited

img_lib_attach()

Initialize the image library

Synopsis:

```
#include <img/img.h>

int img_lib_attach( img_lib_t*  ilib );
```

Arguments:

ilib

The address where the function stores a handle to the library.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function initializes the image library, looks for a configuration file, and loads the codecs listed in that file.

If the function can't find a configuration file, it still fills in a valid library handle, but no codecs are loaded, and images can't be decoded. In this situation, the function returns `IMG_ERR_CFG`. You can load codecs after library initialization by calling *img_cfg_read()* (p. 18).

The `img.conf` Configuration File

The image library uses a configuration file to determine which codecs to load, This function first checks the environment variable ***`LIBIMG_CFGFILE`***, which is the full path for the configuration file. If it isn't set, then checks the default location `/etc/system/config/img.conf`.

Codecs in the configuration file are specified as sections. Each section of the configuration file is demarcated by the codec name in square brackets, followed by an unordered list of properties specifying additional information about that codec. There should be at least a mimetype line and list of extensions to associate with the codec. For example:

```
[img_codec_jpg.so]
mime=image/jpeg:image/jpg
ext=jpg:jpeg
```


© 2014, QNX Software Systems Limited

This example illustrates how to specify multiple entries in the same line, but has been simplified from the original. The default configuration file contains more mimetype entries.

Returns:

IMG_ERR_OK

Success.

IMG_ERR_MEM

Memory-allocation failure.

IMG_ERR_CFG

Bad or missing configuration file. The handle returned is still valid, however no codecs have been preloaded.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_lib_detach()

Detach from the library

Synopsis:

```
#include <img/img.h>
void img_lib_detach( img_lib_t  ilib );
```

Arguments:

ilib

The library handle filled in by *img_lib_attach()*. The handle will no longer be valid.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function detaches from the image library and frees all associated resources.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_load()

Decode a frame from a stream

Synopsis:

```
#include <img/img.h>

int img_load( img_lib_t                    ilib,
              io_stream_t                  *input,
              const img_decode_callouts_t *callouts,
              img_t                        *img );
```

Arguments:

ilib

A handle for the image library, returned by *img_lib_attach()* (p. 84).

input

The input stream.

callouts

A pointer to an `img_decode_callouts_t` (p. 36) structure that provides system callouts for the decoder. If you specify `NULL` for this value, a set of default callouts is supplied.

img

The address of an `img_t` (p. 106) structure the function fills in with information about the decoded frame.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function decodes a frame from a streaming source. This function decodes only the first frame encountered.

Returns:

`IMG_ERR_OK`

© 2014, QNX Software Systems Limited

Success

IMG_ERR_MEM

Memory-allocation failure.

IMG_ERR_FORMAT

No appopriate codec could be found. The codec you require could be missing or corrupt, or the file could be corrupt.

IMG_ERR_NOSUPPORT

Output data not supported; the codec and application could not agree on an output format.

IMG_ERR_NODATA

No frame data was present.

IMG_ERR_CORRUPT

Invalid data encountered in the file, preventing the decode from proceeding. Some of the frame may be intact.

IMG_ERR_TRUNC

Premature end of file encountered. Some of the frame may be intact.

IMG_ERR_INTR

Decode was interrupted by application.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_load_file()

Decode a frame from a file on the filesystem

Synopsis:

```
#include <img/img.h>

int img_load_file( img_lib_t                    ilib,
                   const char*                  path,
                   const img_decode_callouts_t* callouts,
                   img_t*                       img );
```

Arguments:

ilib

A handle for the image library, returned by *img_lib_attach()* (p. 84).

path

The full path to the file from which the data can be read.

callouts

A pointer to an `img_decode_callouts_t` (p. 36) structure that provides system callouts for the decoder. If you specify `NULL` for this value, a set of default callouts is supplied.

img

The address of an `img_t` (p. 106) structure the function fills in with information about the decoded frame.

You can override elements, such as the format, before the call to *img_load_file()*:

```
img.format = IMG_FMT_G8;
img.flags |= IMG_FORMAT;

img_load_file(...);
```

In the above example, because the format is set before the load occurs, the `libimg` will force the loaded image into `IMG_FMT_G8` format, regardless of the actual source image format.

Library:

`libimg`

© 2014, QNX Software Systems Limited

Use the `-l img` option to `qcc` to link against this library.

Description:

This function decodes a frame from a file on the filesystem. This function decodes only the first frame encountered.

If you want to resize a file when it's loaded, use *img_load_resize_file()* (p. 95).

Returns:

IMG_ERR_OK

Success

IMG_ERR_CORRUPT

Invalid data encountered in the file, preventing the decode from proceeding. Some of the frame may be intact.

IMG_ERR_DLL

An error occurred processing the DLL that handles the file type. Check to make sure that the DLL is not missing or corrupt.

IMG_ERR_FILE

Error accessing path (*errno* is set).

IMG_ERR_FORMAT

No appopriate codec could be found. The codec you require could be missing or corrupt, or the file could be corrupt.

IMG_ERR_INTR

Decode was interrupted by application.

IMG_ERR_MEM

Memory-allocation failure.

IMG_ERR_NODATA

No frame data was present. This error is highly unlikely, as files generally contain at least one frame.

IMG_ERR_NOSUPPORT

Output data not supported; the codec and application could not agree on an output format.

IMG_ERR_TRUNC

© 2014, QNX Software Systems Limited

Premature end of file encountered. Some of the frame may be intact.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_load_resize()

Decode and resize a frame from a stream

Synopsis:

```
#include <img/img.h>

int img_load_resize( img_lib_t                    ilib,
                     io_stream_t                  *input,
                     const img_decode_callouts_t  *callouts,
                     img_t                        *img );
```

Arguments:

ilib

A handle for the image library, returned by *img_lib_attach()* (p. 84).

input

The input stream.

callouts

A pointer to an `img_decode_callouts_t` (p. 36) structure that provides system callouts for the decoder. If you specify `NULL` for this value, a set of default callouts is supplied.

img

The address of an `img_t` (p. 106) structure the function fills in with information about the decoded frame.

If you set the image width and height (*img.w* and *img.h*) before calling this function, then the image is sized to fit the specified dimensions rather than clipped, as it is with *img_load_file()* (p. 89). The resizing is performed on-the-fly during the decoding, without incurring the memory penalty of loading the entire original image and then subsequently resizing it.

If you specify only one of the dimensions, then the other is calculated based on the aspect ratio of the original image.

You need to also set the corresponding dimension flag in the *img* structure. For example, if you specify the width, you need to set the `IMG_W` bit in *flags*.

© 2014, QNX Software Systems Limited

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function decodes a frame from a stream, and optionally resizes it if the image width and height are specified. This function decodes only the first frame encountered.

At least these callouts are supported:

- *set_value_f*
- *choose_format_f*

Returns:

IMG_ERR_OK

Success

IMG_ERR_CORRUPT

Invalid data encountered in the file, preventing the decoding from proceeding. Some of the frame may be intact.

IMG_ERR_DLL

An error occurred processing the DLL that handles the file type. Check to make sure that the DLL is not missing or corrupt.

IMG_ERR_FILE

Error accessing path (*errno* is set).

IMG_ERR_FORMAT

No appropriate codec could be found. The codec you require could be missing or corrupt, or the file could be corrupt.

IMG_ERR_INTR

Decoding was interrupted by application.

IMG_ERR_MEM

Memory-allocation failure.

IMG_ERR_NODATA

No frame data was present. This error is highly unlikely, as files generally contain at least one frame.

IMG_ERR_NOSUPPORT

© 2014, QNX Software Systems Limited

Output data not supported; the codec and application could not agree on an output format.

IMG_ERR_TRUNC

Premature end of file encountered. Some of the frame may be intact.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_load_resize_file()

Decode and resize a frame from a file on the filesystem

Synopsis:

```
#include <img/img.h>

int img_load_resize_file( img_lib_t                    ilib,
                          const char                   *path,
                        const img_decode_callouts_t *callouts,

                          img_t                        *img );
```

Arguments:

ilib

A handle for the image library, returned by *img_lib_attach()* (p. 84).

path

The full path to the file from which the data can be read.

callouts

A pointer to an `img_decode_callouts_t` (p. 36) structure that provides system callouts for the decoder. If you specify `NULL` for this value, a set of default callouts is supplied.

img

The address of an `img_t` (p. 106) structure the function fills in with information about the decoded frame.

If you set the image width and height (*img.w* and *img.h*) before calling this function, then the image is sized to fit the specified dimensions rather than clipped, as it is with *img_load_file()* (p. 89). The resizing is performed on-the-fly during the decoding, without incurring the memory penalty of loading the entire original image and then subsequently resizing it.

If you specify only one of the dimensions, then the other is calculated based on the aspect ratio of the original image.

You need to also set the corresponding dimension flag in the *img* structure. For example, if you specify the width, you need to set the `IMG_W` bit in *flags*.

© 2014, QNX Software Systems Limited

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function decodes a frame from a file on the filesystem, and optionally resizes it if the image width and height are specified. This function decodes only the first frame encountered.

If you don't specify the height and width of your image before calling this function, the Image library sets the height and width to the original size of the image. An application can choose to resize the image when its callout, *choose_format_f callback*, is called.

At least these callouts are supported:

- *set_value_f*
- *choose_format_f*

Returns:

`IMG_ERR_OK`

Success

`IMG_ERR_CORRUPT`

Invalid data encountered in the file, preventing the decoding from proceeding. Some of the frame may be intact.

`IMG_ERR_DLL`

An error occurred processing the DLL that handles the file type. Check to make sure that the DLL is not missing or corrupt.

`IMG_ERR_FILE`

Error accessing path (*errno* is set).

`IMG_ERR_FORMAT`

No appropriate codec could be found. The codec you require could be missing or corrupt, or the file could be corrupt.

`IMG_ERR_INTR`

Decoding was interrupted by application.

`IMG_ERR_MEM`

Memory-allocation failure.

© 2014, QNX Software Systems Limited

IMG_ERR_NODATA

No frame data was present. This error is highly unlikely, as files generally contain at least one frame.

IMG_ERR_NOSUPPORT

Output data not supported; the codec and application could not agree on an output format.

IMG_ERR_TRUNC

Premature end of file encountered. Some of the frame may be intact.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_resize_fs()

Resize an image

Synopsis:

```
#include <img/img.h>

int img_resize_fs( const img_t *src,
                   img_t *dst );
```

Arguments:

src

The image to resize.

dst

The address of an *img_t* describing the destination. If you do not specify one of width or height in the *dst* (that is, the field isn't marked as valid in *flags*) then this function will calculate the missing dimension based on the aspect ratio of the *src* image.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function resizes the image *src* to fit into the image described by *dst*. The algorithm is a simple "fast smooth" algorithm (that is, the algorithm yields results much more visually pleasing and smooth than simple pixel replication, but is faster than applying a filter function).

The formats of *src* and *dst* do not have to be the same; if they are different the data will be converted. However, a palette-based *dst* format is not supported.

Resize can be done in place, but the *src* and *dst* data pointers must be the same. You will get unpredictable results by partially overlapping *src* and *dst* data buffers.

Returns:

`IMG_ERR_OK`

Success

© 2014, QNX Software Systems Limited

IMG_ERR_PARM

Some fields of *src* are missing (that is, not marked as valid in *flags*)

IMG_ERR_NOSUPPORT

Unsupported format/conversion

IMG_ERR_MEM

Insufficient memory (the function requires a small amount of working memory)

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_rotate_ortho()

Rotate an image by 90-degree increments

Synopsis:

```
#include <img/img.h>

int img_rotate_ortho( const img_t *src,
                      img_t *dst,
                      img_fixed_t angle );
```

Arguments:

src

The image to rotate.

dst

The address of an `img_t` describing the destination. If you don't specify width or height (or both) in the *dst* then this function will calculate the missing dimension(s) based on the *src* image, taking into account the rotation. If you do specify either width or height (or both), the image is clipped as necessary; unused data remains untouched.

angle

A 16.16 fixed point representation of the angle (in radians). The following constants are provided for convenience:

- `IMG_ANGLE_90CW` — 90 degrees clockwise (to the right)
- `IMG_ANGLE_180` — 180 degrees
- `IMG_ANGLE_90CCW` — 90 degrees counter-clockwise (to the left)

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function rotates the *src* image by 90-degree increments. The rotation is not a true “rotation” in that the image is not rotated about a fixed point. Rather, the image itself is rotated and the new origin of the image becomes the upper-left corner of the rotated image.

© 2014, QNX Software Systems Limited

The formats of *src* and *dst* don't have to be the same; if they are different, the data is converted. A palette-based *dst* format is only supported if the *src* data also is palette-based.

Rotation cannot be done in place.

Returns:

IMG_ERR_OK

Success.

IMG_ERR_PARM

Some fields of src are missing (that is, not marked as valid in *flags*).

IMG_ERR_NOSUPPORT

Unsupported format conversion or angle.

IMG_ERR_MEM

Insufficient memory (the function requires a small amount of working memory).

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_write()

Encode a frame to an output stream on the filesystem

Synopsis:

```
#include <img/img.h>

int img_write( img_lib_t                    ilib,
               io_stream_t*                 output,
               const img_encode_callouts_t* callouts,
               img_t*                       img
               img_codec_t*                 codec );
```

Arguments:

ilib

A handle for the image library, returned by *img_lib_attach()*.

output

An `io_stream_t` to use for the output.

callouts

A pointer to an `img_encode_callouts_t` (p. 68) structure that provides system callouts for the encoder. If you specify `NULL` for this value, the library supplies a set of default callouts.

img

The address of an `img_t` structure describing the frame to be encoded.

codec

The codec to use for the encoding. You must use one of *img_codec_list_byext()* or *img_codec_list_bymime()* to get the codec.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function encodes a frame to an `io_stream_t` output stream on the filesystem.

© 2014, QNX Software Systems Limited

Returns:

IMG_ERR_OK

Success

IMG_ERR_FILE

Error accessing path (*errno* is set).

IMG_ERR_MEM

Memory-allocation failure.

IMG_ERR_FORMAT

No appropriate codec could be found that handles the extension included in the provided filename. The codec you require could be missing or corrupt.

IMG_ERR_NOSUPPORT

Input data format not supported; the codec and application could not agree on an output format.

IMG_ERR_TRUNC

Error writing data; file was truncated.

IMG_ERR_INTR

Encode was interrupted by application.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_write_file()

Encode a frame to a file on the filesystem

Synopsis:

```
#include <img/img.h>

int img_write_file( img_lib_t                    ilib,
                    const char*                  path,
                    const img_encode_callouts_t* callouts,
                    img_t*                       img );
```

Arguments:

ilib

A handle for the image library, returned by *img_lib_attach()*.

path

The full path to the file to create.

callouts

A pointer to an `img_encode_callouts_t` (p. 68) structure that provides system callouts for the encoder. If you specify `NULL` for this value, the library supplies a set of default callouts.

img

The address of an `img_t` structure describing the frame to be encoded.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function encodes a frame to a file on the filesystem. This function is only capable of encoding a single frame. A codec is chosen based on the extension included in the provided filename.

The file will be automatically unlinked if the encode fails for any reason.

Returns:

© 2014, QNX Software Systems Limited

IMG_ERR_OK

Success

IMG_ERR_FILE

Error accessing path (*errno* is set).

IMG_ERR_MEM

Memory-allocation failure.

IMG_ERR_FORMAT

No appropriate codec could be found that handles the extension included in the provided filename. The codec you require could be missing or corrupt.

IMG_ERR_NOSUPPORT

Input data format not supported; the codec and application could not agree on an output format.

IMG_ERR_TRUNC

Error writing data; file was truncated.

IMG_ERR_INTR

Encode was interrupted by application.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

img_t

Information describing an image

Synopsis:

```
#include <img/img.h>

typedef struct {
    union {
        struct {
            uint8           *data;
            unsigned        stride;
        } direct;

        struct {
            img_access_f    *access_f;
            Uintptrt        data;
        }               indirect;
    } access;
    unsigned            w, h;
    img_format_t        format;
    unsigned            npalette;
    img_color_t         *palette;
    unsigned            flags;
    union {
            uint8           index;
            uint16          rgb16;
            img_color_t     rgb32;
    }                   transparency;
    unsigned            quality;
} img_t;
```

Description:

The `img_t` structure describes a decoded frame. The members include:

access

A union of two structures, *direct* and *indirect*, depending on whether you want the image data to be accessed directly or indirectly. The `IMG_DIRECT` or `IMG_INDIRECT` flag should be set to indicate which mode of access is in place.

Using the direct access model, anyone operating on the image data can access it directly via a pointer. The beginning of the image data is pointed to by *direct.data*, and it is assumed that the data pointed to is a contiguous buffer of *h* scanlines of *direct.stride* bytes each.

The *stride* can be much larger (if needed) than the actual number of bytes required to represent a single scanline in the specified

© 2014, QNX Software Systems Limited

format; anyone operating on the image should never overwrite or otherwise give any regard to the "in between" padding bytes.

Using the indirect access model, anyone operating on the image data does it through a function; the function pointer is given by *indirect.access_f*, and *indirect.data* provides a facility to give your access function some context.

An access function is a function you provide to read or write a run of pixels to or from your image. An access function must be coded either as a reader or writer, there is no way to tell from the parameters the direction of data flow.

```
void access_f(uintptr_t data, unsigned x,
       unsigned y, unsigned n, uint8_t *pixels)
```

- *data* — the data field (from img_t::*access.indirect.data*)
- *x*, *y* — the x and y position within the image of the pixel run being accessed
- *n* — the number of pixels in the run
- *pixels* — pointer to pixel data. If your function is a reader, it should copy the prescribed run of image data to this buffer; if it's a write it should copy the pixels in this buffer to your image

---

The format of the data in *pixels* will be the same as the format of the image; that is, no data transformation is required at this level. The *x*, *y*, and *n* arguments are guaranteed not to exceed the boundary of your image so you don't have to check for that.

---

w*, *h

The width and height of the image frame, in pixels. These members are only valid if the `IMG_W` and `IMG_H` flag bits are set.

format

The `img_format_t` (p. 82) format of the image's pixel data. This field is valid if the `IMG_FORMAT` flag bit is set

npalette

The number of colors in the image *palette* color table. This field should be used only if the format is palette-based (that is, the `IMG_FMT_PALETTE` bit is set in *format*).

palette

© 2014, QNX Software Systems Limited

The palette color table. This field is valid if the IMG_PALETTE flag bit is set.

flags

Flags indicating which of the fields in the structure are valid. Can be one or more of:

IMG_TRANSPARENCY

The *transparency* field is valid and the specified color within the image should be treated as transparent.

IMG_FORMAT

The *format* field is valid.

IMG_W

The *w* field is valid.

IMG_H

The *h* field is valid.

IMG_DIRECT

The *direct* field is valid.

IMG_INDIRECT

The *indirect* field is valid.

IMG_PALETTE

The *palette* field is valid.

IMG_QUALITY

The *quality* field is valid.

IMG_PAL8_ALPHA

The PAL8 image palette contains alpha data. Therefore, color values can be treated as 8888 instead of 888.

IMG_TRANSPARENCY_TO_ALPHA

Convert transparency into alpha values if supported by destination format. This capability allows transparency to be utilized even if chroma is not supported.

IMG_SRC_FMT_TRANSPARENCY

© 2014, QNX Software Systems Limited

Indicates whether or not the source image contains any type of transparency. This flag is set by the Image library only. Applications can't set this flag, but they can use it to determine the existence of transparency for optimizations.

transparency

The transparency color. This is valid only if the IMG_TRANSPARENCY flag bit is set. The union field that should be used depends on the format of the image:

- *index* — for palette-based or grayscale images (the IMG_FMT_PALETTE bit is set in *format* or the format is IMG_FMT_G8)
- *rgb16* — for 16bpp images. Encoded the same as the image data.
- *rgb32* — for 24 or 32 bpp RGB images. Encoding is always IMG_FMT_PKHE_ARGB8888.

quality

If the IMG_QUALITY flag is set, the codecs may process the new img_t member unsigned quality. For example, the *img_codec_jpg.so* will use this value to determine the output quality for encoding. For example, when *img_write()* is invoked.

Classification:

Image library

© 2014, QNX Software Systems Limited

io_close()

Release an input stream

Synopsis:

```
#include <img/img.h>
void io_close( io_stream_t *stream );
```

Arguments:

stream

A pointer to the stream object returned by *io_open()* (p. 111).

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function releases the resources associated with an input stream.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

io_open()

Initialize an IO stream

Synopsis:

```
#include <img/img.h>

io_stream_t *io_open( io_open_f *open_f,
                      io_mode_t  mode, ...)
```

Arguments:

open_f

A pointer to a function to establish a stream. There are two functions supplied by the library: *IO_FD()* (for fd-based reading/writing) and *IO_MEM()* (for memory buffer based reading/writing). See below.

mode

The open mode, either `IO_READ` or `IO_WRITE`.

...

Additional parameters depending on the *open_f* specified, described below.

Library:

`libimg`

Use the `-l img` option to `qcc` to link against this library.

Description:

This function initializes a stream. The stream can be a fd-based, or a memory buffer, depending on the *open_f* specified:

IO_FD()

Buffered streaming for unix-type fd's. An additional parmameter is required: an `int` specifying the (previously opened) fd that is ready for reading or writing.

IO_MEM()

Streaming support for a memory buffer. Additional parameters are required (in order):

© 2014, QNX Software Systems Limited

1. an `unsigned` to specify size of the memory buffer. This must be non-zero.
2. a `void` pointer to specify the address of the buffer.

When your application is finished with a stream, it should call *io_close()* (p. 110) to release it.

Returns:

A pointer to the stream object, or `NULL` if an error occurred (*errno* is set).

Errors:

ENOMEM

Insufficient memory to allocate structures.

EINVAL

Invalid *open_f* or *mode*.

ENOTSUP

Mode not supported for stream.

Classification:

Image library

Safety:	
Interrupt handler	No
Signal handler	No
Thread	No

© 2014, QNX Software Systems Limited

Index

© 2014, QNX Software Systems Limited

© 2014, QNX Software Systems Limited