Reference Manual

Generated by Doxygen 1.8.13

Contents

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

2 File Index

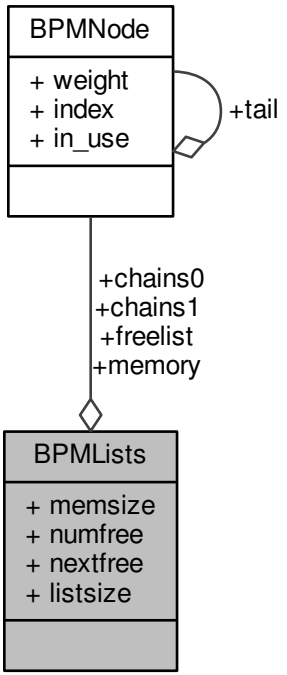
2.1 File List

Here is a list of all files with brief descriptions:

3 Class Documentation

3.1 BPMLists Struct Reference

Collaboration diagram for BPMLists:



Public Attributes

- unsigned memsize
- BPMNode * memory
- unsigned numfree
- unsigned nextfree
- BPMNode ** freelist
- unsigned listsize
- BPMNode ** chains0
- BPMNode ** chains1

3.1.1 Detailed Description

Definition at line 683 of file lodepng.cpp.

3.1.2 Member Data Documentation

3.1.2.1 chains0

```
BPMNode** BPMLists::chains0
```

Definition at line 693 of file lodepng.cpp.

3.1.2.2 chains1

```
BPMNode** BPMLists::chains1
```

Definition at line 694 of file lodepng.cpp.

3.1.2.3 freelist

```
BPMNode** BPMLists::freelist
```

Definition at line 690 of file lodepng.cpp.

3.1.2.4 listsize

```
unsigned BPMLists::listsize
```

Definition at line 692 of file lodepng.cpp.

3.1.2.5 memory

```
BPMNode* BPMLists::memory
```

Definition at line 687 of file lodepng.cpp.

3.1.2.6 memsize

```
unsigned BPMLists::memsize
```

Definition at line 686 of file lodepng.cpp.

3.1.2.7 nextfree

```
unsigned BPMLists::nextfree
```

Definition at line 689 of file lodepng.cpp.

3.1.2.8 numfree

```
unsigned BPMLists::numfree
```

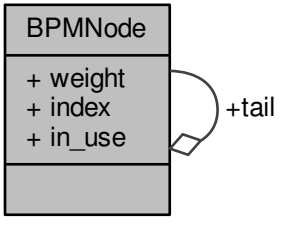
Definition at line 688 of file lodepng.cpp.

The documentation for this struct was generated from the following file:

- lodepng.cpp

3.2 BPMNode Struct Reference

Collaboration diagram for BPMNode:



Public Attributes

- int weight
- unsigned index
- struct BPMNode * tail
- int in_use

3.2.1 Detailed Description

Definition at line 674 of file lodepng.cpp.

3.2.2 Member Data Documentation

3.2.2.1 in_use

```
int BPMNode::in_use
```

Definition at line 679 of file lodepng.cpp.

3.2.2.2 index

```
unsigned BPMNode::index
```

Definition at line 677 of file lodepng.cpp.

3.2.2.3 tail

```
struct BPMNode* BPMNode::tail
```

Definition at line 678 of file lodepng.cpp.

3.2.2.4 weight

```
int BPMNode::weight
```

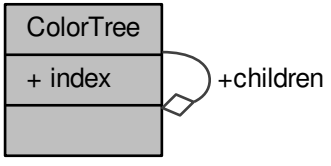
Definition at line 676 of file lodepng.cpp.

The documentation for this struct was generated from the following file:

- lodepng.cpp

3.3 ColorTree Struct Reference

Collaboration diagram for ColorTree:



Public Attributes

- ColorTree * children [16]
- int index

3.3.1 Detailed Description

Definition at line 3000 of file lodepng.cpp.

3.3.2 Member Data Documentation

3.3.2.1 children

```
ColorTree* ColorTree::children[16]
```

Definition at line 3002 of file lodepng.cpp.

3.3.2.2 index

```
int ColorTree::index
```

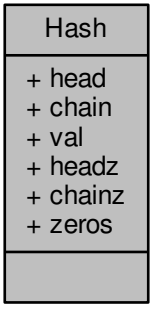
Definition at line 3003 of file lodepng.cpp.

The documentation for this struct was generated from the following file:

- lodepng.cpp

3.4 Hash Struct Reference

Collaboration diagram for Hash:



Public Attributes

- int * head
- unsigned short * chain
- int * val
- int * headz
- unsigned short * chainz
- unsigned short * zeros

3.4.1 Detailed Description

Definition at line 1367 of file lodepng.cpp.

3.4.2 Member Data Documentation

3.4.2.1 chain

```
unsigned short* Hash::chain
```

Definition at line 1371 of file lodepng.cpp.

3.4.2.2 chainz

```
unsigned short* Hash::chainz
```

Definition at line 1377 of file lodepng.cpp.

3.4.2.3 head

```
int* Hash::head
```

Definition at line 1369 of file lodepng.cpp.

3.4.2.4 headz

```
int* Hash::headz
```

Definition at line 1376 of file lodepng.cpp.

3.4.2.5 val

```
int* Hash::val
```

Definition at line 1372 of file lodepng.cpp.

3.4.2.6 zeros

```
unsigned short* Hash::zeros
```

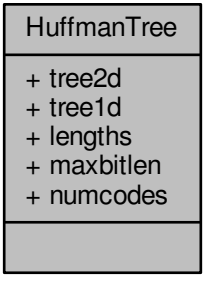
Definition at line 1378 of file lodepng.cpp.

The documentation for this struct was generated from the following file:

- lodepng.cpp

3.5 HuffmanTree Struct Reference

Collaboration diagram for HuffmanTree:



Public Attributes

- unsigned * tree2d
- unsigned * tree1d
- unsigned * lengths
- unsigned maxbitlen
- unsigned numcodes

3.5.1 Detailed Description

Definition at line 509 of file lodepng.cpp.

3.5.2 Member Data Documentation

3.5.2.1 lengths

```
unsigned* HuffmanTree::lengths
```

Definition at line 513 of file lodepng.cpp.

3.5.2.2 maxbitlen

```
unsigned HuffmanTree::maxbitlen
```

Definition at line 514 of file lodepng.cpp.

3.5.2.3 numcodes

```
unsigned HuffmanTree::numcodes
```

Definition at line 515 of file lodepng.cpp.

3.5.2.4 tree1d

```
unsigned* HuffmanTree::tree1d
```

Definition at line 512 of file lodepng.cpp.

3.5.2.5 tree2d

```
unsigned* HuffmanTree::tree2d
```

Definition at line 511 of file lodepng.cpp.

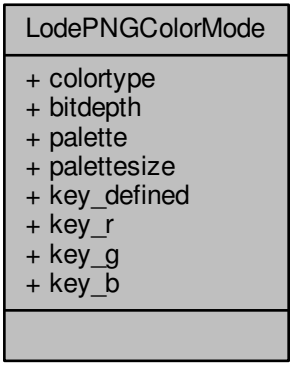
The documentation for this struct was generated from the following file:

- lodepng.cpp

3.6 LodePNGColorMode Struct Reference

`#include <lodepng.h>`

Collaboration diagram for LodePNGColorMode:



Public Attributes

- LodePNGColorType colortype
- unsigned bitdepth
- unsigned char * palette
- size_t palettesize
- unsigned key_defined
- unsigned key_r
- unsigned key_g
- unsigned key_b

3.6.1 Detailed Description

Definition at line 318 of file lodepng.h.

3.6.2 Member Data Documentation

3.6.2.1 bitdepth

```
unsigned LodePNGColorMode::bitdepth
```

Definition at line 322 of file lodepng.h.

3.6.2.2 colortype

```
LodePNGColorType LodePNGColorMode::colortype
```

Definition at line 321 of file lodepng.h.

3.6.2.3 key_b

```
unsigned LodePNGColorMode::key_b
```

Definition at line 354 of file lodepng.h.

3.6.2.4 key_defined

```
unsigned LodePNGColorMode::key_defined
```

Definition at line 351 of file lodepng.h.

3.6.2.5 key_g

```
unsigned LodePNGColorMode::key_g
```

Definition at line 353 of file lodepng.h.

3.6.2.6 key_r

```
unsigned LodePNGColorMode::key_r
```

Definition at line 352 of file lodepng.h.

3.6.2.7 palette

```
unsigned char* LodePNGColorMode::palette
```

Definition at line 337 of file lodepng.h.

3.6.2.8 palettesize

```
size_t LodePNGColorMode::palettesize
```

Definition at line 338 of file lodepng.h.

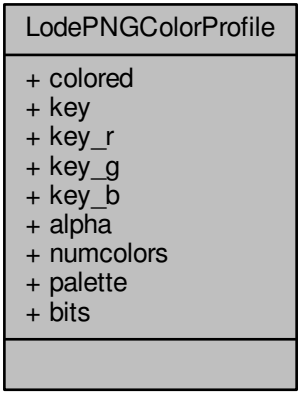
The documentation for this struct was generated from the following file:

- lodepng.h

3.7 LodePNGColorProfile Struct Reference

```
#include <lodepng.h>
```

Collaboration diagram for LodePNGColorProfile:



Public Attributes

- unsigned colored
- unsigned key
- unsigned short key_r
- unsigned short key_g
- unsigned short key_b
- unsigned alpha
- unsigned numcolors
- unsigned char palette [1024]
- unsigned bits

3.7.1 Detailed Description

Definition at line 559 of file lodepng.h.

3.7.2 Member Data Documentation

3.7.2.1 alpha

```
unsigned LodePNGColorProfile::alpha
```

Definition at line 566 of file lodepng.h.

3.7.2.2 bits

```
unsigned LodePNGColorProfile::bits
```

Definition at line 569 of file lodepng.h.

3.7.2.3 colored

```
unsigned LodePNGColorProfile::colored
```

Definition at line 561 of file lodepng.h.

3.7.2.4 key

```
unsigned LodePNGColorProfile::key
```

Definition at line 562 of file lodepng.h.

3.7.2.5 key_b

```
unsigned short LodePNGColorProfile::key_b
```

Definition at line 565 of file lodepng.h.

3.7.2.6 key_g

```
unsigned short LodePNGColorProfile::key_g
```

Definition at line 564 of file lodepng.h.

3.7.2.7 key_r

```
unsigned short LodePNGColorProfile::key_r
```

Definition at line 563 of file lodepng.h.

3.7.2.8 numcolors

```
unsigned LodePNGColorProfile::numcolors
```

Definition at line 567 of file lodepng.h.

3.7.2.9 palette

```
unsigned char LodePNGColorProfile::palette[1024]
```

Definition at line 568 of file lodepng.h.

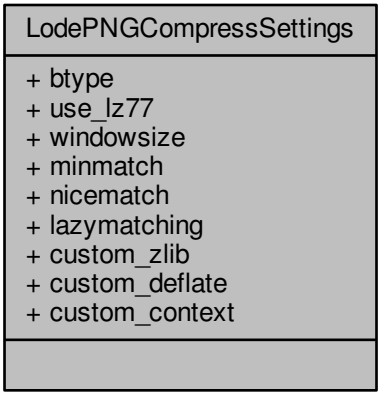
The documentation for this struct was generated from the following file:

- lodepng.h

3.8 LodePNGCompressSettings Struct Reference

`#include <lodepng.h>`

Collaboration diagram for LodePNGCompressSettings:



Public Attributes

- unsigned btype
- unsigned use_lz77
- unsigned windowsize
- unsigned minmatch
- unsigned nicematch

- unsigned lazymatching
- unsigned(∗ custom_zlib)(unsigned char ∗∗, size_t ∗, const unsigned char ∗, size_t, const LodePNGCompressSettings ∗)
- unsigned(∗ custom_deflate)(unsigned char ∗∗, size_t ∗, const unsigned char ∗, size_t, const LodePNGCompressSettings ∗)
- const void ∗ custom_context

3.8.1 Detailed Description

Definition at line 284 of file lodepng.h.

3.8.2 Member Data Documentation

3.8.2.1 btype

```
unsigned LodePNGCompressSettings::btype
```

Definition at line 287 of file lodepng.h.

3.8.2.2 custom_context

```
const void* LodePNGCompressSettings::custom_context
```

Definition at line 305 of file lodepng.h.

3.8.2.3 custom_deflate

```
unsigned(* LodePNGCompressSettings::custom_deflate) (unsigned char **, size_t *, const unsigned char *, size_t, const
LodePNGCompressSettings *)
```

Definition at line 301 of file lodepng.h.

3.8.2.4 custom_zlib

```
unsigned(* LodePNGCompressSettings::custom_zlib) (unsigned char **, size_t *, const unsigned char *, size_t, const
LodePNGCompressSettings *)
```

Definition at line 295 of file lodepng.h.

3.8.2.5 lazymatching

```
unsigned LodePNGCompressSettings::lazymatching
```

Definition at line 292 of file lodepng.h.

3.8.2.6 minmatch

```
unsigned LodePNGCompressSettings::minmatch
```

Definition at line 290 of file lodepng.h.

3.8.2.7 nicematch

```
unsigned LodePNGCompressSettings::nicematch
```

Definition at line 291 of file lodepng.h.

3.8.2.8 use_lz77

```
unsigned LodePNGCompressSettings::use_lz77
```

Definition at line 288 of file lodepng.h.

3.8.2.9 windowsize

```
unsigned LodePNGCompressSettings::windowsize
```

Definition at line 289 of file lodepng.h.

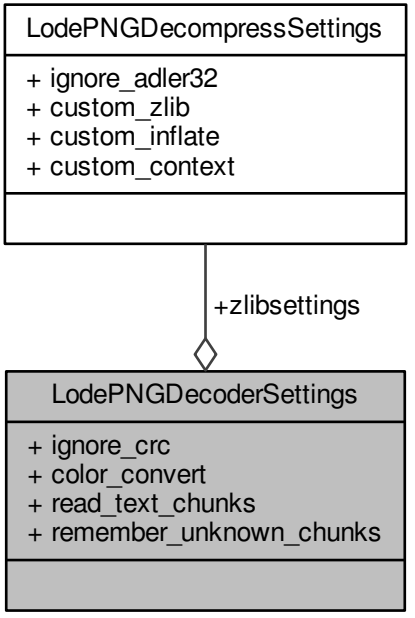
The documentation for this struct was generated from the following file:

- lodepng.h

3.9 LodePNGDecoderSettings Struct Reference

`#include <lodepng.h>`

Collaboration diagram for LodePNGDecoderSettings:



Public Attributes

- LodePNGDecompressSettings zlibsettings

- unsigned ignore_crc
- unsigned color_convert
- unsigned read_text_chunks
- unsigned remember_unknown_chunks

3.9.1 Detailed Description

Definition at line 519 of file lodepng.h.

3.9.2 Member Data Documentation

3.9.2.1 color_convert

```
unsigned LodePNGDecoderSettings::color_convert
```

Definition at line 525 of file lodepng.h.

3.9.2.2 ignore_crc

```
unsigned LodePNGDecoderSettings::ignore_crc
```

Definition at line 523 of file lodepng.h.

3.9.2.3 read_text_chunks

```
unsigned LodePNGDecoderSettings::read_text_chunks
```

Definition at line 528 of file lodepng.h.

3.9.2.4 remember_unknown_chunks

```
unsigned LodePNGDecoderSettings::remember_unknown_chunks
```

Definition at line 530 of file lodepng.h.

3.9.2.5 zlibsettings

```
LodePNGDecompressSettings LodePNGDecoderSettings::zlibsettings
```

Definition at line 521 of file lodepng.h.

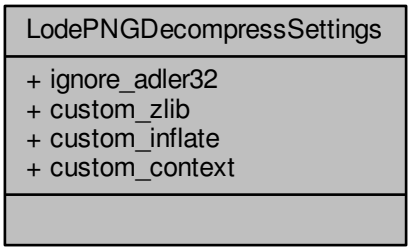
The documentation for this struct was generated from the following file:

- lodepng.h

3.10 LodePNGDecompressSettings Struct Reference

`#include <lodepng.h>`

Collaboration diagram for LodePNGDecompressSettings:



Public Attributes

- unsigned ignore_adler32
- unsigned(∗ custom_zlib)(unsigned char ∗∗, size_t ∗, const unsigned char ∗, size_t, const LodePNGDecompressSettings ∗)
- unsigned(∗ custom_inflate)(unsigned char ∗∗, size_t ∗, const unsigned char ∗, size_t, const LodePNGDecompressSettings ∗)
- const void ∗ custom_context

3.10.1 Detailed Description

Definition at line 256 of file lodepng.h.

3.10.2 Member Data Documentation

3.10.2.1 custom_context

```
const void* LodePNGDecompressSettings::custom_context
```

Definition at line 271 of file lodepng.h.

3.10.2.2 custom_inflate

```
unsigned(* LodePNGDecompressSettings::custom_inflate) (unsigned char **, size_t *, const unsigned char *, size_t,
const LodePNGDecompressSettings *)
```

Definition at line 267 of file lodepng.h.

3.10.2.3 custom_zlib

```
unsigned(* LodePNGDecompressSettings::custom_zlib) (unsigned char **, size_t *, const unsigned char *, size_t, const
LodePNGDecompressSettings *)
```

Definition at line 261 of file lodepng.h.

3.10.2.4 ignore_adler32

```
unsigned LodePNGDecompressSettings::ignore_adler32
```

Definition at line 258 of file lodepng.h.

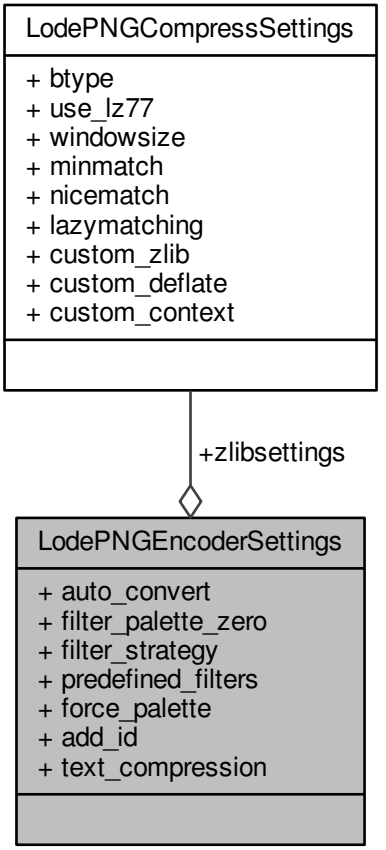
The documentation for this struct was generated from the following file:

- lodepng.h

3.11 LodePNGEncoderSettings Struct Reference

```
#include <lodepng.h>
```

Collaboration diagram for LodePNGEncoderSettings:



Public Attributes

- LodePNGCompressSettings zlibsettings
- unsigned auto_convert
- unsigned filter_palette_zero
- LodePNGFilterStrategy filter_strategy
- const unsigned char * predefined_filters
- unsigned force_palette
- unsigned add_id
- unsigned text_compression

3.11.1 Detailed Description

Definition at line 585 of file lodepng.h.

3.11.2 Member Data Documentation

3.11.2.1 add_id

```
unsigned LodePNGEncoderSettings::add_id
```

Definition at line 610 of file lodepng.h.

3.11.2.2 auto_convert

```
unsigned LodePNGEncoderSettings::auto_convert
```

Definition at line 589 of file lodepng.h.

3.11.2.3 filter_palette_zero

```
unsigned LodePNGEncoderSettings::filter_palette_zero
```

Definition at line 595 of file lodepng.h.

3.11.2.4 filter_strategy

```
LodePNGFilterStrategy LodePNGEncoderSettings::filter_strategy
```

Definition at line 598 of file lodepng.h.

3.11.2.5 force_palette

```
unsigned LodePNGEncoderSettings::force_palette
```

Definition at line 607 of file lodepng.h.

3.11.2.6 predefined_filters

```
const unsigned char* LodePNGEncoderSettings::predefined_filters
```

Definition at line 603 of file lodepng.h.

3.11.2.7 text_compression

```
unsigned LodePNGEncoderSettings::text_compression
```

Definition at line 612 of file lodepng.h.

3.11.2.8 zlibsettings

```
LodePNGCompressSettings LodePNGEncoderSettings::zlibsettings
```

Definition at line 587 of file lodepng.h.

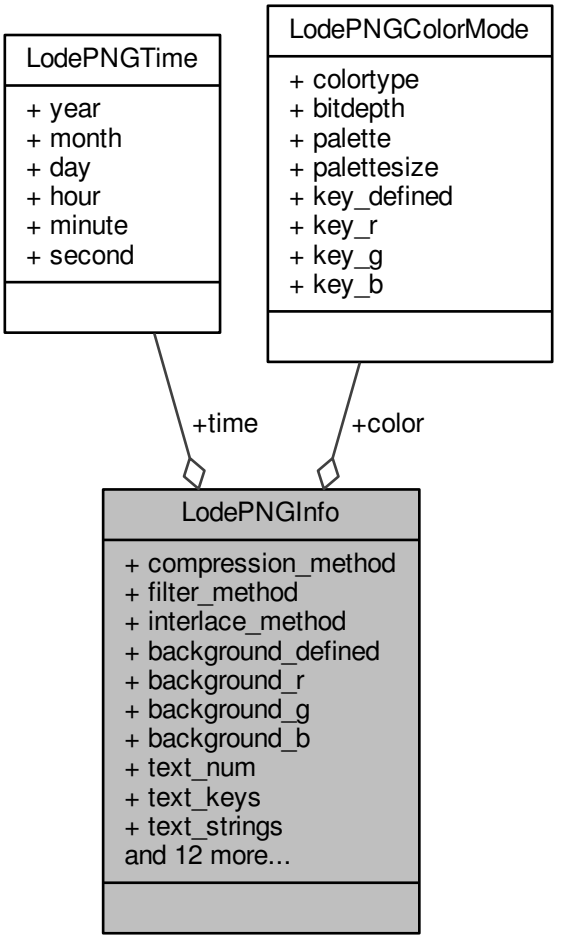
The documentation for this struct was generated from the following file:

- lodepng.h

3.12 LodePNGInfo Struct Reference

```
#include <lodepng.h>
```

Collaboration diagram for LodePNGInfo:



Public Attributes

- unsigned compression_method
- unsigned filter_method
- unsigned interlace_method
- LodePNGColorMode color
- unsigned background_defined
- unsigned background_r
- unsigned background_g
- unsigned background_b
- size_t text_num
- char ** text_keys
- char ** text_strings
- size_t itext_num
- char ** itext_keys
- char ** itext_langtags
- char ** itext_transkeys
- char ** itext_strings
- unsigned time_defined
- LodePNGTime time
- unsigned phys_defined
- unsigned phys_x
- unsigned phys_y
- unsigned phys_unit
- unsigned char * unknown_chunks_data [3]
- size_t unknown_chunks_size [3]

3.12.1 Detailed Description

Definition at line 407 of file lodepng.h.

3.12.2 Member Data Documentation

3.12.2.1 background_b

```
unsigned LodePNGInfo::background_b
```

Definition at line 430 of file lodepng.h.

3.12.2.2 background_defined

```
unsigned LodePNGInfo::background_defined
```

Definition at line 427 of file lodepng.h.

3.12.2.3 background_g

```
unsigned LodePNGInfo::background_g
```

Definition at line 429 of file lodepng.h.

3.12.2.4 background_r

```
unsigned LodePNGInfo::background_r
```

Definition at line 428 of file lodepng.h.

3.12.2.5 color

```
LodePNGColorMode LodePNGInfo::color
```

Definition at line 413 of file lodepng.h.

3.12.2.6 compression_method

```
unsigned LodePNGInfo::compression_method
```

Definition at line 410 of file lodepng.h.

3.12.2.7 filter_method

```
unsigned LodePNGInfo::filter_method
```

Definition at line 411 of file lodepng.h.

3.12.2.8 interlace_method

```
unsigned LodePNGInfo::interlace_method
```

Definition at line 412 of file lodepng.h.

3.12.2.9 itext_keys

```
char** LodePNGInfo::itext_keys
```

Definition at line 455 of file lodepng.h.

3.12.2.10 itext_langtags

```
char** LodePNGInfo::itext_langtags
```

Definition at line 456 of file lodepng.h.

3.12.2.11 itext_num

```
size_t LodePNGInfo::itext_num
```

Definition at line 454 of file lodepng.h.

3.12.2.12 itext_strings

```
char** LodePNGInfo::itext_strings
```

Definition at line 458 of file lodepng.h.

3.12.2.13 itext_transkeys

```
char** LodePNGInfo::itext_transkeys
```

Definition at line 457 of file lodepng.h.

3.12.2.14 phys_defined

```
unsigned LodePNGInfo::phys_defined
```

Definition at line 465 of file lodepng.h.

3.12.2.15 phys_unit

```
unsigned LodePNGInfo::phys_unit
```

Definition at line 468 of file lodepng.h.

3.12.2.16 phys_x

```
unsigned LodePNGInfo::phys_x
```

Definition at line 466 of file lodepng.h.

3.12.2.17 phys_y

```
unsigned LodePNGInfo::phys_y
```

Definition at line 467 of file lodepng.h.

3.12.2.18 text_keys

```
char** LodePNGInfo::text_keys
```

Definition at line 446 of file lodepng.h.

3.12.2.19 text_num

```
size_t LodePNGInfo::text_num
```

Definition at line 445 of file lodepng.h.

3.12.2.20 text_strings

```
char** LodePNGInfo::text_strings
```

Definition at line 447 of file lodepng.h.

3.12.2.21 time

```
LodePNGTime LodePNGInfo::time
```

Definition at line 462 of file lodepng.h.

3.12.2.22 time_defined

```
unsigned LodePNGInfo::time_defined
```

Definition at line 461 of file lodepng.h.

3.12.2.23 unknown_chunks_data

```
unsigned char* LodePNGInfo::unknown_chunks_data[3]
```

Definition at line 479 of file lodepng.h.

3.12.2.24 unknown_chunks_size

```
size_t LodePNGInfo::unknown_chunks_size[3]
```

Definition at line 480 of file lodepng.h.

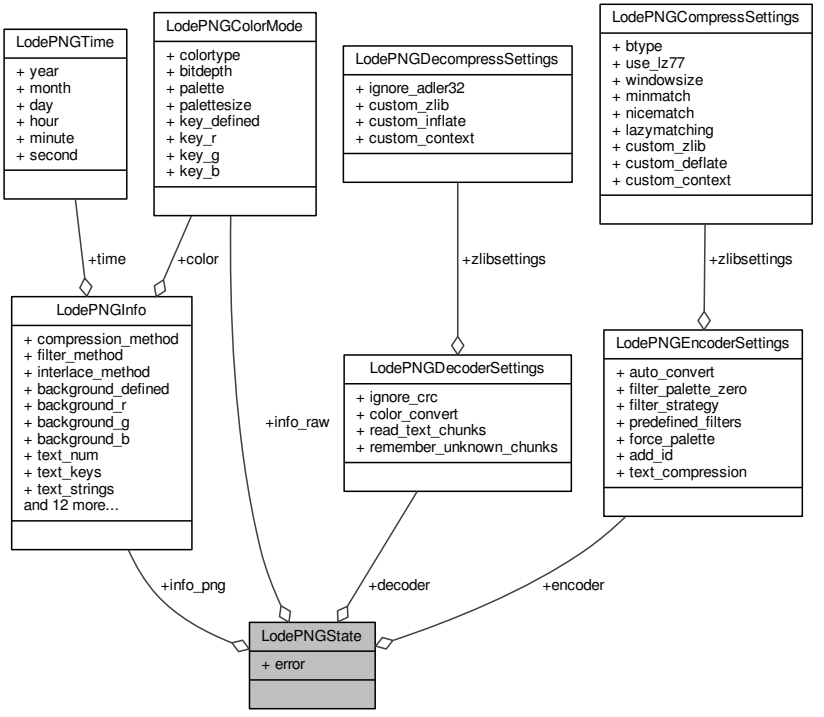
The documentation for this struct was generated from the following file:

- lodepng.h

3.13 LodePNGState Struct Reference

`#include <lodepng.h>`

Collaboration diagram for LodePNGState:



Public Attributes

- LodePNGDecoderSettings decoder

- LodePNGEncoderSettings encoder
- LodePNGColorMode info_raw
- LodePNGInfo info_png
- unsigned error

3.13.1 Detailed Description

Definition at line 622 of file lodepng.h.

3.13.2 Member Data Documentation

3.13.2.1 decoder

```
LodePNGDecoderSettings LodePNGState::decoder
```

Definition at line 625 of file lodepng.h.

3.13.2.2 encoder

```
LodePNGEncoderSettings LodePNGState::encoder
```

Definition at line 628 of file lodepng.h.

3.13.2.3 error

```
unsigned LodePNGState::error
```

Definition at line 632 of file lodepng.h.

3.13.2.4 info_png

```
LodePNGInfo LodePNGState::info_png
```

Definition at line 631 of file lodepng.h.

3.13.2.5 info_raw

```
LodePNGColorMode LodePNGState::info_raw
```

Definition at line 630 of file lodepng.h.

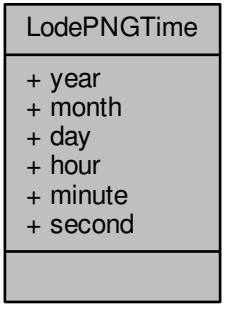
The documentation for this struct was generated from the following file:

- lodepng.h

3.14 LodePNGTime Struct Reference

```
#include <lodepng.h>
```

Collaboration diagram for LodePNGTime:



Public Attributes

- unsigned year
- unsigned month
- unsigned day
- unsigned hour
- unsigned minute
- unsigned second

3.14.1 Detailed Description

Definition at line 395 of file lodepng.h.

3.14.2 Member Data Documentation

3.14.2.1 day

```
unsigned LodePNGTime::day
```

Definition at line 399 of file lodepng.h.

3.14.2.2 hour

```
unsigned LodePNGTime::hour
```

Definition at line 400 of file lodepng.h.

3.14.2.3 minute

```
unsigned LodePNGTime::minute
```

Definition at line 401 of file lodepng.h.

3.14.2.4 month

```
unsigned LodePNGTime::month
```

Definition at line 398 of file lodepng.h.

3.14.2.5 second

```
unsigned LodePNGTime::second
```

Definition at line 402 of file lodepng.h.

3.14.2.6 year

```
unsigned LodePNGTime::year
```

Definition at line 397 of file lodepng.h.

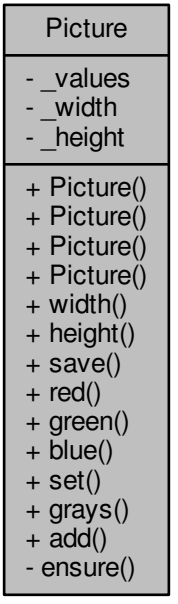
The documentation for this struct was generated from the following file:

- lodepng.h

3.15 Picture Class Reference

`#include <picture.h>`

Collaboration diagram for Picture:



Public Member Functions

- Picture ()

- Picture (string filename)
- Picture (int width, int height, int red=255, int green=255, int blue=255)
- Picture (const vector< vector< int > > &grays)
- int width () const
- int height () const
- void save (string filename) const
- int red (int x, int y) const
- int green (int x, int y) const
- int blue (int x, int y) const
- void set (int x, int y, int red, int green, int blue)
- vector< vector< int > > grays () const
- void add (const Picture &other, int x=0, int y=0)

Private Member Functions

- void ensure (int x, int y)

Private Attributes

- vector< unsigned char > _values
- int _width
- int _height

3.15.1 Detailed Description

Definition at line 10 of file picture.h.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 Picture() [1/4]

```
Picture::Picture ( )
```

Constructs a picture with width and height zero.

Definition at line 5 of file picture.cpp.

```
6 {
7     _width = 0;
8     _height = 0;
9 }
```

3.15.2.2 Picture() [2/4]

```
Picture::Picture (
            string filename )
```

Constructs a picture from the given PNG file.

Parameters

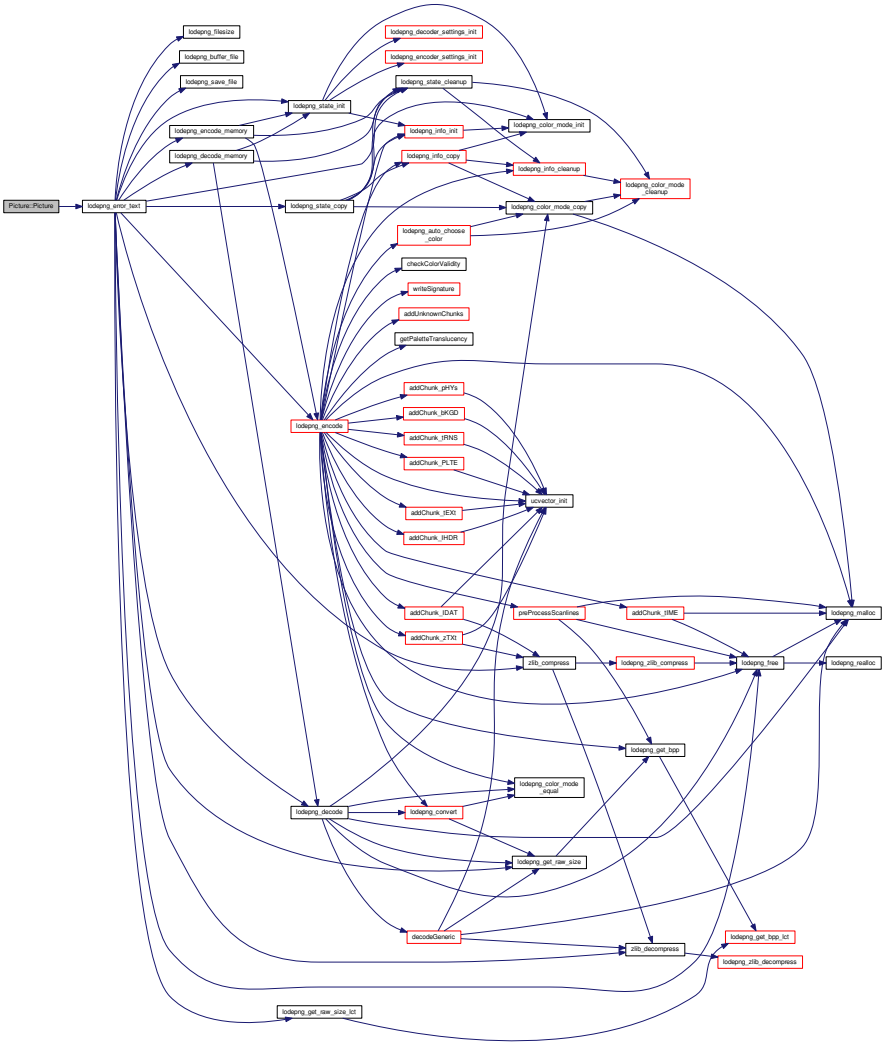
filename	a file name that should specify a PNG file.

Definition at line 51 of file picture.cpp.

```
52 {
53     unsigned int w, h;
```

```
54     unsigned error = lodepng::decode(_values, w, h, filename.c_str());
55     if (error != 0) throw runtime_error(lodepng_error_text(error));
56     _width = w;
57     _height = h;
58 }
```

Here is the call graph for this function:



3.15.2.3 Picture() [3/4]

```
Picture::Picture (
            int width,
            int height,
            int red = 255,
            int green = 255,
            int blue = 255 )
```

Constructs a picture with pixels in a single color (by default, white).

Parameters

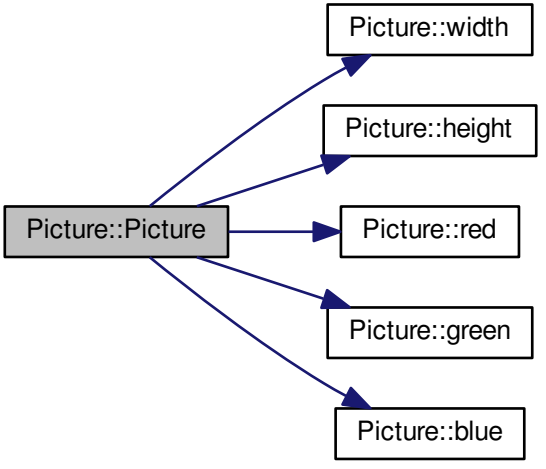
red	the red value of the pixels (between 0 and 255)
green	the green value of the pixels (between 0 and 255)
blue	the blue value of the pixels (between 0 and 255)
width	the width of the picture
height	the height of the picture

Definition at line 11 of file picture.cpp.

```
12     : _values(width * height * 4)
13 {
14     _width = width;
15     _height = height;
16     for (int k = 0; k < _values.size(); k += 4)
17     {
18        _values[k] = red;
19        _values[k + 1] = green;
```

```
20          _values[k + 2] = blue;
21          _values[k + 3] = 255;
22      }
23  }
```

Here is the call graph for this function:



3.15.2.4 Picture() [4/4]

```
Picture::Picture (
            const vector< vector< int > > & grays )
```

Constructs a picture from a two-dimensional vector of gray levels.

Parameters

grays	the gray levels

Definition at line 25 of file picture.cpp.

```
26 {
27    if (grays.size() == 0 || grays[0].size() == 0)
28    {
29       _width = 0;
30       _height = 0;
31    }
32    else
33    {
34       _values = vector<unsigned char>(grays[0].size() * grays.size() * 4);
35       _width = grays[0].size();
36       _height = grays.size();
37       int k = 0;
38       for (int y = 0; y < _height; y++)
39          for (int x = 0; x < _width; x++)
40          {
41             int gray = grays[y][x];
42             _values[k] = gray;
43             _values[k + 1] = gray;
44             _values[k + 2] = gray;
45             _values[k + 3] = 255;
46             k += 4;
47          }
48    }
49 }
```

Here is the call graph for this function:

3.15.3 Member Function Documentation

3.15.3.1 add()

```
void Picture::add (
            const Picture & other,
            int x = 0,
            int y = 0 )
```

Adds a picture to this picture at a given position, expanding the picture if necessary.

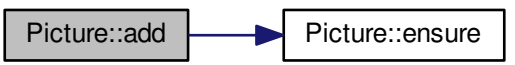
Parameters

other	the picture to add
x	the x-coordinate (column) of the top left corner
y	the y-coordinate (row) of the top left corner

Definition at line 103 of file picture.cpp.

```
104 {
105    ensure(x + other._width - 1, y + other._height - 1);
106    int k = 0;
107    for (int dy = 0; dy < other._height; dy++)
108       for (int dx = 0; dx < other._width; dx++)
109       {
110          set(x + dx, y + dy, other._values[k],
111             other._values[k + 1], other._values[k + 2]);
112          k += 4;
113       }
114 }
```

Here is the call graph for this function:



3.15.3.2 blue()

```
int Picture::blue (
            int x,
            int y ) const
```

Yields the red value at the given position.

Parameters

x	the x-coordinate (column)
y	the y-coordinate (row)

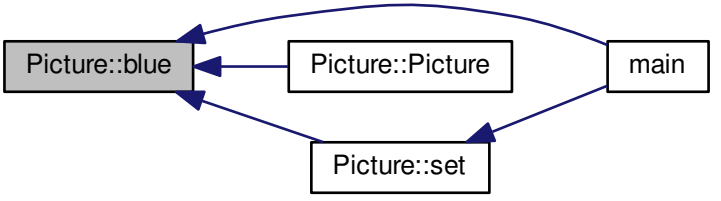
Returns

the blue value of the pixel (between 0 and 255), or 0 if the given point is not in the picture.

Definition at line 82 of file picture.cpp.

```
83 {
84    if (0 <= x && x < _width && 0 <= y && y < _height)
85       return _values[4 * (y * _width + x) + 2];
86    else
87       return 0;
88 }
```

Here is the caller graph for this function:



3.15.3.3 ensure()

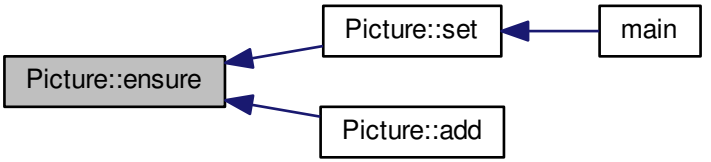
```
void Picture::ensure (
            int x,
            int y )  [private]
```

Ensures that the given point exists.

Definition at line 134 of file picture.cpp.

```
135 {
136    if (x >= _width || y >= _height)
137    {
138       int new_width = max(x + 1, _width);
139       int new_height = max(y + 1, _height);
140       vector<unsigned char> new_values(4 * new_width * new_height);
141       fill(new_values.begin(), new_values.end(), 255); // fill with white
142       int j = 0;
143       for (int dy = 0; dy < _height; dy++)
144          for (int dx = 0; dx < _width; dx++)
145             for (int k = 0; k < 4; k++, j++)
146                new_values[4 * (dy * new_width + dx) + k] = _values[j];
147       _values.swap(new_values);
148       _width = new_width;
149       _height = new_height;
150    }
151 }
```

Here is the caller graph for this function:



3.15.3.4 grays()

```
vector< vector< int > > Picture::grays ( ) const
```

Yields the gray levels of all pixels of this image.

Returns

a 2D array of gray values (between 0 and 255)

Definition at line 116 of file picture.cpp.

```
117 {
118   vector<vector<int> > result(_height);
119   for (int y = 0; y < _height; y++) {
```

```
120       result[y] = vector<int>(_width);
121       for (int x = 0; x < _width; x++) {
122          int k = 4 * (y * _width + x);
123          result[y][x] = (int)(0.2126 * _values[k]
124             + 0.7152 * _values[k + 1]
125             + 0.0722 * _values[k + 2]);
126       }
127    }
128    return result;
129 }
```

Here is the caller graph for this function:

3.15.3.5 green()

```
int Picture::green (
            int x,
            int y ) const
```

Yields the green value at the given position.

Parameters

x	the x-coordinate (column)
y	the y-coordinate (row)

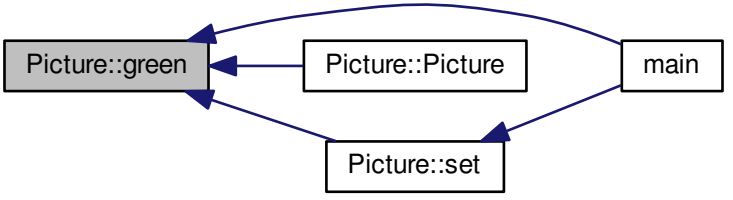
Returns

the green value of the pixel (between 0 and 255), or 0 if the given point is not in the picture.

Definition at line 74 of file picture.cpp.

```
75 {
76    if (0 <= x && x < _width && 0 <= y && y < _height)
77       return _values[4 * (y * _width + x) + 1];
78    else
79       return 0;
80 }
```

Here is the caller graph for this function:



3.15.3.6 height()

```
int Picture::height ( ) const  [inline]
```

Returns the height of this picture.

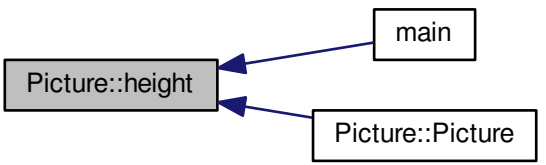
Returns

the height

Definition at line 51 of file picture.h.

```
51 { return _height; }
```

Here is the caller graph for this function:



3.15.3.7 red()

```
int Picture::red (
            int x,
            int y ) const
```

Yields the red value at the given position.

Parameters

x	the x-coordinate (column)
y	the y-coordinate (row)

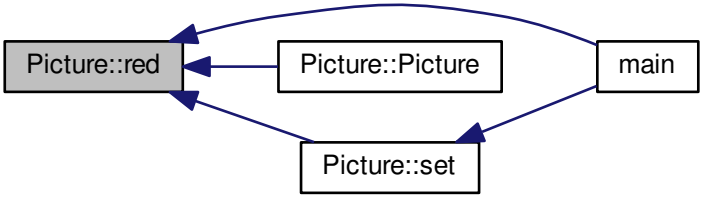
Returns

the red value of the pixel (between 0 and 255), or 0 if the given point is not in the picture.

Definition at line 66 of file picture.cpp.

```
67 {
68    if (0 <= x && x < _width && 0 <= y && y < _height)
69       return _values[4 * (y * _width + x)];
70    else
71       return 0;
72 }
```

Here is the caller graph for this function:



3.15.3.8 save()

```
void Picture::save (
            string filename ) const
```

Saves this picture to the given file.

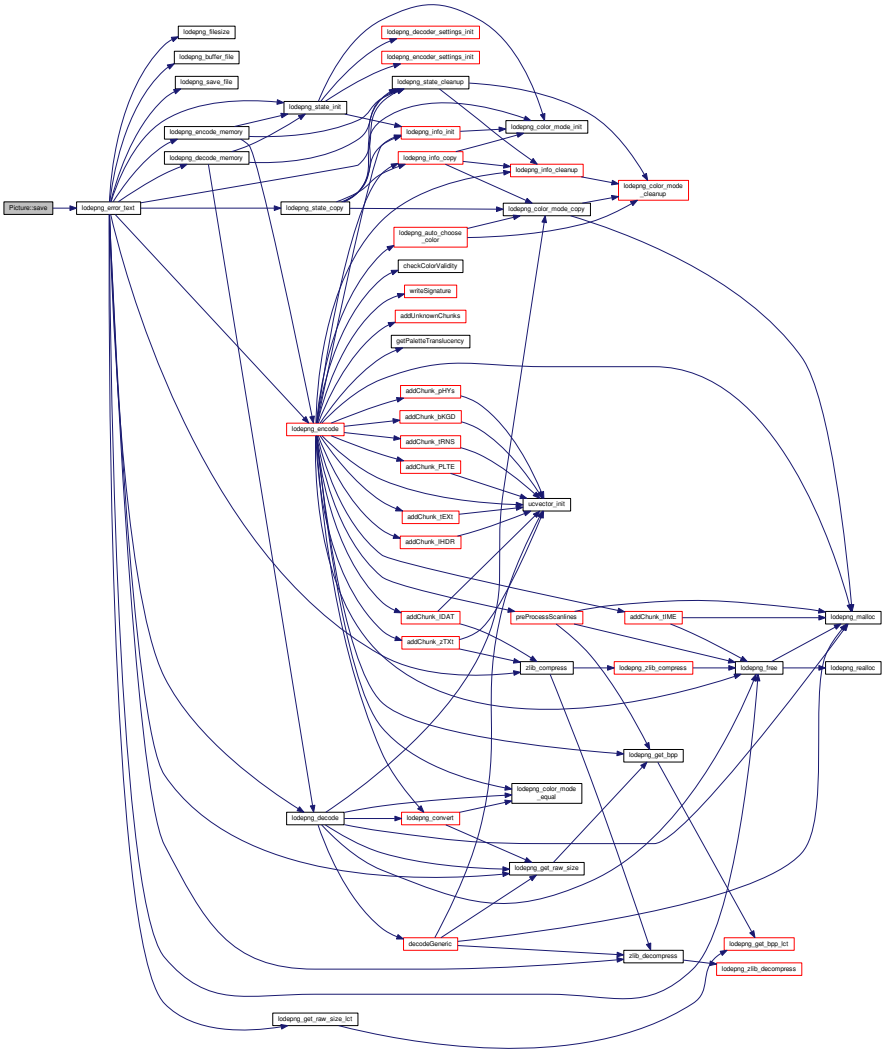
Parameters

filename	a file name that should specify a PNG file.

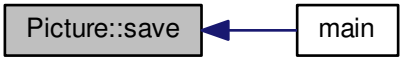
Definition at line 60 of file picture.cpp.

```
61 {
62     unsigned error = lodepng::encode(filename.c_str(), _values, _width,
       _height);
63     if (error != 0) throw runtime_error(lodepng_error_text(error));
64 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.15.3.9 set()

```
void Picture::set (
            int x,
            int y,
            int red,
            int green,
            int blue )
```

Sets a pixel to a given color, expanding the picture if necessary.

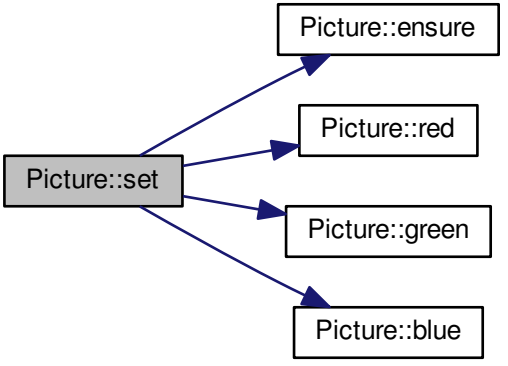
Parameters

x	the x-coordinate (column)
y	the y-coordinate (row)
red	the red value of the pixel (between 0 and 255)
green	the green value of the pixel (between 0 and 255)
blue	the blue value of the pixel (between 0 and 255)

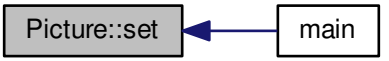
Definition at line 90 of file picture.cpp.

```
91 {
92     if (x >= 0 && y >= 0)
93     {
94         ensure(x, y);
95         int k = 4 * (y * _width + x);
96         _values[k] = red;
97         _values[k + 1] = green;
98         _values[k + 2] = blue;
99         _values[k + 3] = 255;
100     }
101 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.15.3.10 width()

```
int Picture::width ( ) const  [inline]
```

Returns the width of this picture.

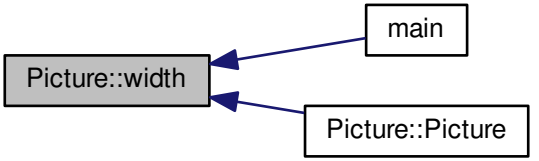
Returns

the width

Definition at line 45 of file picture.h.

```
45 { return _width; }
```

Here is the caller graph for this function:



3.15.4 Member Data Documentation

3.15.4.1 _height

```
int Picture::_height  [private]
```

Definition at line 117 of file picture.h.

3.15.4.2 _values

```
vector<unsigned char> Picture::_values  [private]
```

Definition at line 115 of file picture.h.

3.15.4.3 _width

```
int Picture::_width  [private]
```

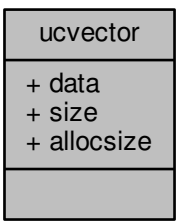
Definition at line 116 of file picture.h.

The documentation for this class was generated from the following files:

- picture.h
- picture.cpp

3.16 ucvector Struct Reference

Collaboration diagram for ucvector:



Public Attributes

- unsigned char * data
- size_t size
- size_t allocsize

3.16.1 Detailed Description

Definition at line 205 of file lodepng.cpp.

3.16.2 Member Data Documentation

3.16.2.1 allocsize

```
size_t ucvector::allocsize
```

Definition at line 209 of file lodepng.cpp.

3.16.2.2 data

```
unsigned char* ucvector::data
```

Definition at line 207 of file lodepng.cpp.

3.16.2.3 size

```
size_t ucvector::size
```

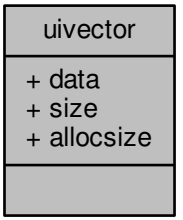
Definition at line 208 of file lodepng.cpp.

The documentation for this struct was generated from the following file:

- lodepng.cpp

3.17 uivector Struct Reference

Collaboration diagram for uivector:



Public Attributes

- unsigned * data
- size_t size
- size_t allocsize

3.17.1 Detailed Description

Definition at line 137 of file lodepng.cpp.

3.17.2 Member Data Documentation

3.17.2.1 allocsize

```
size_t uivector::allocsize
```

Definition at line 141 of file lodepng.cpp.

3.17.2.2 data

```
unsigned* uivector::data
```

Definition at line 139 of file lodepng.cpp.

3.17.2.3 size

```
size_t uivector::size
```

Definition at line 140 of file lodepng.cpp.

The documentation for this struct was generated from the following file:

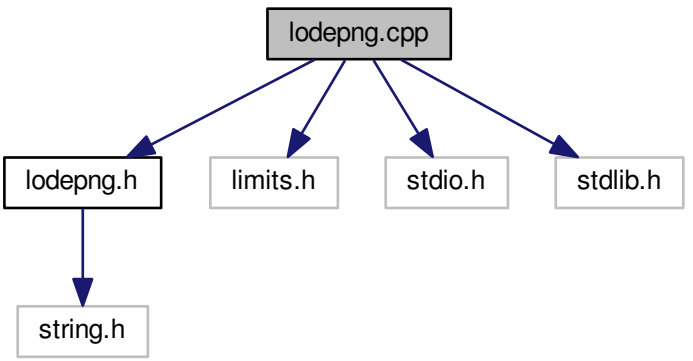
- lodepng.cpp

4 File Documentation

4.1 lodepng.cpp File Reference

```
#include "lodepng.h"
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for lodepng.cpp:



Classes

- struct uivector

- struct ucvector
- struct HuffmanTree
- struct BPMNode
- struct BPMLists
- struct Hash
- struct ColorTree

Macros

- #define CERROR_BREAK(errorvar, code)
- #define ERROR_BREAK(code) CERROR_BREAK(error, code)
- #define CERROR_RETURN_ERROR(errorvar, code)
- #define CERROR_TRY_RETURN(call)
- #define CERROR_RETURN(errorvar, code)
- #define addBitToStream(bitpointer, bitstream, bit)
- #define READBIT(bitpointer, bitstream) ((bitstream[bitpointer >> 3] >> (bitpointer & 0x7)) & (unsigned char)1)
- #define FIRST_LENGTH_CODE_INDEX 257
- #define LAST_LENGTH_CODE_INDEX 285
- #define NUM_DEFLATE_CODE_SYMBOLS 288
- #define NUM_DISTANCE_SYMBOLS 32
- #define NUM_CODE_LENGTH_CODES 19
- #define DEFAULT_WINDOWSIZE 2048

Typedefs

- typedef struct uivector uivector
- typedef struct ucvector ucvector
- typedef struct HuffmanTree HuffmanTree
- typedef struct BPMNode BPMNode
- typedef struct BPMLists BPMLists
- typedef struct Hash Hash
- typedef struct ColorTree ColorTree

Functions

- static void * lodepng_malloc (size_t size)
- static void * lodepng_realloc (void *ptr, size_t new_size)
- static void lodepng_free (void *ptr)
- static void uivector_cleanup (void *p)
- static unsigned uivector_reserve (uivector *p, size_t allocsize)
- static unsigned uivector_resize (uivector *p, size_t size)
- static unsigned uivector_resizev (uivector *p, size_t size, unsigned value)
- static void uivector_init (uivector *p)
- static unsigned uivector_push_back (uivector *p, unsigned c)
- static unsigned ucvector_reserve (ucvector *p, size_t allocsize)
- static unsigned ucvector_resize (ucvector *p, size_t size)
- static void ucvector_cleanup (void *p)
- static void ucvector_init (ucvector *p)
- static void ucvector_init_buffer (ucvector *p, unsigned char *buffer, size_t size)
- static unsigned ucvector_push_back (ucvector *p, unsigned char c)
- static unsigned string_resize (char **out, size_t size)
- static void string_init (char **out)
- static void string_cleanup (char **out)
- static void string_set (char **out, const char *in)
- unsigned lodepng_read32bitInt (const unsigned char *buffer)
- static void lodepng_set32bitInt (unsigned char *buffer, unsigned value)
- static void lodepng_add32bitInt (ucvector *buffer, unsigned value)
- static long lodepng_filesize (const char *filename)
- static unsigned lodepng_buffer_file (unsigned char *out, size_t size, const char *filename)
- unsigned lodepng_load_file (unsigned char **out, size_t *outsize, const char *filename)
- unsigned lodepng_save_file (const unsigned char *buffer, size_t buffersize, const char *filename)
- static void addBitsToStream (size_t *bitpointer, ucvector *bitstream, unsigned value, size_t nbits)
- static void addBitsToStreamReversed (size_t *bitpointer, ucvector *bitstream, unsigned value, size_t nbits)
- static unsigned char readBitFromStream (size_t *bitpointer, const unsigned char *bitstream)
- static unsigned readBitsFromStream (size_t *bitpointer, const unsigned char *bitstream, size_t nbits)
- static void HuffmanTree_init (HuffmanTree *tree)
- static void HuffmanTree_cleanup (HuffmanTree *tree)

- static unsigned HuffmanTree_make2DTree (HuffmanTree *tree)
- static unsigned HuffmanTree_makeFromLengths2 (HuffmanTree *tree)
- static unsigned HuffmanTree_makeFromLengths (HuffmanTree *tree, const unsigned *bitlen, size_t numcodes, unsigned maxbitlen)
- static BPMNode * bpmnode_create (BPMLists *lists, int weight, unsigned index, BPMNode *tail)
- static void bpmnode_sort (BPMNode *leaves, size_t num)
- static void boundaryPM (BPMLists *lists, BPMNode *leaves, size_t numpresent, int c, int num)
- unsigned lodepng_huffman_code_lengths (unsigned *lengths, const unsigned *frequencies, size_t numcodes, unsigned maxbitlen)
- static unsigned HuffmanTree_makeFromFrequencies (HuffmanTree *tree, const unsigned *frequencies, size_t mincodes, size_t numcodes, unsigned maxbitlen)
- static unsigned HuffmanTree_getCode (const HuffmanTree *tree, unsigned index)
- static unsigned HuffmanTree_getLength (const HuffmanTree *tree, unsigned index)
- static unsigned generateFixedLitLenTree (HuffmanTree *tree)
- static unsigned generateFixedDistanceTree (HuffmanTree *tree)
- static unsigned huffmanDecodeSymbol (const unsigned char *in, size_t *bp, const HuffmanTree *codetree, size_t inbitlength)
- static void getTreeInflateFixed (HuffmanTree *tree_ll, HuffmanTree *tree_d)
- static unsigned getTreeInflateDynamic (HuffmanTree *tree_ll, HuffmanTree *tree_d, const unsigned char *in, size_t *bp, size_t inlength)
- static unsigned inflateHuffmanBlock (ucvector *out, const unsigned char *in, size_t *bp, size_t *pos, size_t inlength, unsigned btype)
- static unsigned inflateNoCompression (ucvector *out, const unsigned char *in, size_t *bp, size_t *pos, size_t inlength)
- static unsigned lodepng_inflatev (ucvector *out, const unsigned char *in, size_t insize, const LodePNGDecompressSettings *settings)
- unsigned lodepng_inflate (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGDecompressSettings *settings)
- static unsigned inflate (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGDecompressSettings *settings)
- static void addHuffmanSymbol (size_t *bp, ucvector *compressed, unsigned code, unsigned bitlen)
- static size_t searchCodeIndex (const unsigned *array, size_t array_size, size_t value)
- static void addLengthDistance (uivector *values, size_t length, size_t distance)
- static unsigned hash_init (Hash *hash, unsigned windowsize)
- static void hash_cleanup (Hash *hash)
- static unsigned getHash (const unsigned char *data, size_t size, size_t pos)
- static unsigned countZeros (const unsigned char *data, size_t size, size_t pos)
- static void updateHashChain (Hash *hash, size_t wpos, unsigned hashval, unsigned short numzeros)
- static unsigned encodeLZ77 (uivector *out, Hash *hash, const unsigned char *in, size_t inpos, size_t insize, unsigned windowsize, unsigned minmatch, unsigned nicematch, unsigned lazymatching)
- static unsigned deflateNoCompression (ucvector *out, const unsigned char *data, size_t datasize)

- static void writeLZ77data (size_t *bp, ucvector *out, const uivector *lz77_encoded, const HuffmanTree *tree_ll, const HuffmanTree *tree_d)
- static unsigned deflateDynamic (ucvector *out, size_t *bp, Hash *hash, const unsigned char *data, size_t datapos, size_t dataend, const LodePNGCompressSettings *settings, unsigned final)
- static unsigned deflateFixed (ucvector *out, size_t *bp, Hash *hash, const unsigned char *data, size_t datapos, size_t dataend, const LodePNGCompressSettings *settings, unsigned final)
- static unsigned lodepng_deflatev (ucvector *out, const unsigned char *in, size_t insize, const LodePNGCompressSettings *settings)
- unsigned lodepng_deflate (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGCompressSettings *settings)
- static unsigned deflate (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGCompressSettings *settings)
- static unsigned update_adler32 (unsigned adler, const unsigned char *data, unsigned len)
- static unsigned adler32 (const unsigned char *data, unsigned len)
- unsigned lodepng_zlib_decompress (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGDecompressSettings *settings)
- static unsigned zlib_decompress (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGDecompressSettings *settings)
- unsigned lodepng_zlib_compress (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGCompressSettings *settings)
- static unsigned zlib_compress (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGCompressSettings *settings)
- void lodepng_compress_settings_init (LodePNGCompressSettings *settings)
- void lodepng_decompress_settings_init (LodePNGDecompressSettings *settings)
- unsigned lodepng_crc32 (const unsigned char *data, size_t length)
- static unsigned char readBitFromReversedStream (size_t *bitpointer, const unsigned char *bitstream)
- static unsigned readBitsFromReversedStream (size_t *bitpointer, const unsigned char *bitstream, size_t nbits)
- static void setBitOfReversedStream0 (size_t *bitpointer, unsigned char *bitstream, unsigned char bit)
- static void setBitOfReversedStream (size_t *bitpointer, unsigned char *bitstream, unsigned char bit)
- unsigned lodepng_chunk_length (const unsigned char *chunk)
- void lodepng_chunk_type (char type[5], const unsigned char *chunk)
- unsigned char lodepng_chunk_type_equals (const unsigned char *chunk, const char *type)
- unsigned char lodepng_chunk_ancillary (const unsigned char *chunk)
- unsigned char lodepng_chunk_private (const unsigned char *chunk)
- unsigned char lodepng_chunk_safetocopy (const unsigned char *chunk)
- unsigned char * lodepng_chunk_data (unsigned char *chunk)

- const unsigned char * lodepng_chunk_data_const (const unsigned char *chunk)
- unsigned lodepng_chunk_check_crc (const unsigned char *chunk)
- void lodepng_chunk_generate_crc (unsigned char *chunk)
- unsigned char * lodepng_chunk_next (unsigned char *chunk)
- const unsigned char * lodepng_chunk_next_const (const unsigned char *chunk)
- unsigned lodepng_chunk_append (unsigned char **out, size_t *outlength, const unsigned char *chunk)
- unsigned lodepng_chunk_create (unsigned char **out, size_t *outlength, unsigned length, const char *type, const unsigned char *data)
- static unsigned checkColorValidity (LodePNGColorType colortype, unsigned bd)
- static unsigned getNumColorChannels (LodePNGColorType colortype)
- static unsigned lodepng_get_bpp_lct (LodePNGColorType colortype, unsigned bitdepth)
- void lodepng_color_mode_init (LodePNGColorMode *info)
- void lodepng_color_mode_cleanup (LodePNGColorMode *info)
- unsigned lodepng_color_mode_copy (LodePNGColorMode *dest, const LodePNGColorMode *source)
- static int lodepng_color_mode_equal (const LodePNGColorMode *a, const LodePNGColorMode *b)
- void lodepng_palette_clear (LodePNGColorMode *info)
- unsigned lodepng_palette_add (LodePNGColorMode *info, unsigned char r, unsigned char g, unsigned char b, unsigned char a)
- unsigned lodepng_get_bpp (const LodePNGColorMode *info)
- unsigned lodepng_get_channels (const LodePNGColorMode *info)
- unsigned lodepng_is_greyscale_type (const LodePNGColorMode *info)
- unsigned lodepng_is_alpha_type (const LodePNGColorMode *info)
- unsigned lodepng_is_palette_type (const LodePNGColorMode *info)
- unsigned lodepng_has_palette_alpha (const LodePNGColorMode *info)
- unsigned lodepng_can_have_alpha (const LodePNGColorMode *info)
- size_t lodepng_get_raw_size (unsigned w, unsigned h, const LodePNGColorMode *color)
- size_t lodepng_get_raw_size_lct (unsigned w, unsigned h, LodePNGColorType colortype, unsigned bitdepth)
- static size_t lodepng_get_raw_size_idat (unsigned w, unsigned h, const LodePNGColorMode *color)
- static void LodePNGUnknownChunks_init (LodePNGInfo *info)
- static void LodePNGUnknownChunks_cleanup (LodePNGInfo *info)
- static unsigned LodePNGUnknownChunks_copy (LodePNGInfo *dest, const LodePNGInfo *src)
- static void LodePNGText_init (LodePNGInfo *info)
- static void LodePNGText_cleanup (LodePNGInfo *info)
- static unsigned LodePNGText_copy (LodePNGInfo *dest, const LodePNGInfo *source)
- void lodepng_clear_text (LodePNGInfo *info)
- unsigned lodepng_add_text (LodePNGInfo *info, const char *key, const char *str)

- static void LodePNGIText_init (LodePNGInfo *info)
- static void LodePNGIText_cleanup (LodePNGInfo *info)
- static unsigned LodePNGIText_copy (LodePNGInfo *dest, const LodePNGInfo *source)
- void lodepng_clear_itext (LodePNGInfo *info)
- unsigned lodepng_add_itext (LodePNGInfo *info, const char *key, const char *langtag, const char *transkey, const char *str)
- void lodepng_info_init (LodePNGInfo *info)
- void lodepng_info_cleanup (LodePNGInfo *info)
- unsigned lodepng_info_copy (LodePNGInfo *dest, const LodePNGInfo *source)
- void lodepng_info_swap (LodePNGInfo *a, LodePNGInfo *b)
- static void addColorBits (unsigned char *out, size_t index, unsigned bits, unsigned in)
- static void color_tree_init (ColorTree *tree)
- static void color_tree_cleanup (ColorTree *tree)
- static int color_tree_get (ColorTree *tree, unsigned char r, unsigned char g, unsigned char b, unsigned char a)
- static int color_tree_has (ColorTree *tree, unsigned char r, unsigned char g, unsigned char b, unsigned char a)
- static void color_tree_add (ColorTree *tree, unsigned char r, unsigned char g, unsigned char b, unsigned char a, unsigned index)
- static unsigned rgba8ToPixel (unsigned char *out, size_t i, const LodePNGColorMode *mode, ColorTree *tree, unsigned char r, unsigned char g, unsigned char b, unsigned char a)
- static void rgba16ToPixel (unsigned char *out, size_t i, const LodePNGColorMode *mode, unsigned short r, unsigned short g, unsigned short b, unsigned short a)
- static void getPixelColorRGBA8 (unsigned char *r, unsigned char *g, unsigned char *b, unsigned char *a, const unsigned char *in, size_t i, const LodePNGColorMode *mode)
- static void getPixelColorsRGBA8 (unsigned char *buffer, size_t numpixels, unsigned has_alpha, const unsigned char *in, const LodePNGColorMode *mode)
- static void getPixelColorRGBA16 (unsigned short *r, unsigned short *g, unsigned short *b, unsigned short *a, const unsigned char *in, size_t i, const LodePNGColorMode *mode)
- unsigned lodepng_convert (unsigned char *out, const unsigned char *in, const LodePNGColorMode *mode_out, const LodePNGColorMode *mode_in, unsigned w, unsigned h)
- void lodepng_color_profile_init (LodePNGColorProfile *profile)
- static unsigned getValueRequiredBits (unsigned char value)
- unsigned lodepng_get_color_profile (LodePNGColorProfile *profile, const unsigned char *in, unsigned w, unsigned h, const LodePNGColorMode *mode)
- unsigned lodepng_auto_choose_color (LodePNGColorMode *mode_out, const unsigned char *image, unsigned w, unsigned h, const LodePNGColorMode *mode_in)
- static unsigned char paethPredictor (short a, short b, short c)

- static void Adam7_getpassvalues (unsigned passw[7], unsigned passh[7], size_t filter_passstart[8], size_t padded_passstart[8], size_t passstart[8], unsigned w, unsigned h, unsigned bpp)
- unsigned lodepng_inspect (unsigned *w, unsigned *h, LodePNGState *state, const unsigned char *in, size_t insize)
- static unsigned unfilterScanline (unsigned char *recon, const unsigned char *scanline, const unsigned char *precon, size_t bytewidth, unsigned char filterType, size_t length)
- static unsigned unfilter (unsigned char *out, const unsigned char *in, unsigned w, unsigned h, unsigned bpp)
- static void Adam7_deinterlace (unsigned char *out, const unsigned char *in, unsigned w, unsigned h, unsigned bpp)
- static void removePaddingBits (unsigned char *out, const unsigned char *in, size_t olinebits, size_t ilinebits, unsigned h)
- static unsigned postProcessScanlines (unsigned char *out, unsigned char *in, unsigned w, unsigned h, const LodePNGInfo *info_png)
- static unsigned readChunk_PLTE (LodePNGColorMode *color, const unsigned char *data, size_t chunkLength)
- static unsigned readChunk_tRNS (LodePNGColorMode *color, const unsigned char *data, size_t chunkLength)
- static unsigned readChunk_bKGD (LodePNGInfo *info, const unsigned char *data, size_t chunkLength)
- static unsigned readChunk_tEXt (LodePNGInfo *info, const unsigned char *data, size_t chunkLength)
- static unsigned readChunk_zTXt (LodePNGInfo *info, const LodePNGDecompressSettings *zlibsettings, const unsigned char *data, size_t chunkLength)
- static unsigned readChunk_iTXt (LodePNGInfo *info, const LodePNGDecompressSettings *zlibsettings, const unsigned char *data, size_t chunkLength)
- static unsigned readChunk_tIME (LodePNGInfo *info, const unsigned char *data, size_t chunkLength)
- static unsigned readChunk_pHYs (LodePNGInfo *info, const unsigned char *data, size_t chunkLength)
- static void decodeGeneric (unsigned char **out, unsigned *w, unsigned *h, LodePNGState *state, const unsigned char *in, size_t insize)
- unsigned lodepng_decode (unsigned char **out, unsigned *w, unsigned *h, LodePNGState *state, const unsigned char *in, size_t insize)
- unsigned lodepng_decode_memory (unsigned char **out, unsigned *w, unsigned *h, const unsigned char *in, size_t insize, LodePNGColorType colortype, unsigned bitdepth)
- unsigned lodepng_decode32 (unsigned char **out, unsigned *w, unsigned *h, const unsigned char *in, size_t insize)
- unsigned lodepng_decode24 (unsigned char **out, unsigned *w, unsigned *h, const unsigned char *in, size_t insize)
- unsigned lodepng_decode_file (unsigned char **out, unsigned *w, unsigned *h, const char *filename, LodePNGColorType colortype, unsigned bitdepth)
- unsigned lodepng_decode32_file (unsigned char **out, unsigned *w, unsigned *h, const char *filename)
- unsigned lodepng_decode24_file (unsigned char **out, unsigned *w, unsigned *h, const char *filename)
- void lodepng_decoder_settings_init (LodePNGDecoderSettings *settings)
- void lodepng_state_init (LodePNGState *state)
- void lodepng_state_cleanup (LodePNGState *state)
- void lodepng_state_copy (LodePNGState *dest, const LodePNGState *source)

- static unsigned addChunk (ucvector *out, const char *chunkName, const unsigned char *data, size_t length)
- static void writeSignature (ucvector *out)
- static unsigned addChunk_IHDR (ucvector *out, unsigned w, unsigned h, LodePNGColorType colortype, unsigned bitdepth, unsigned interlace_method)
- static unsigned addChunk_PLTE (ucvector *out, const LodePNGColorMode *info)
- static unsigned addChunk_tRNS (ucvector *out, const LodePNGColorMode *info)
- static unsigned addChunk_IDAT (ucvector *out, const unsigned char *data, size_t datasize, LodePNGCompressSettings *zlibsettings)
- static unsigned addChunk_IEND (ucvector *out)
- static unsigned addChunk_tEXt (ucvector *out, const char *keyword, const char *textstring)
- static unsigned addChunk_zTXt (ucvector *out, const char *keyword, const char *textstring, LodePNGCompressSettings *zlibsettings)
- static unsigned addChunk_iTXt (ucvector *out, unsigned compressed, const char *keyword, const char *langtag, const char *transkey, const char *textstring, LodePNGCompressSettings *zlibsettings)
- static unsigned addChunk_bKGD (ucvector *out, const LodePNGInfo *info)
- static unsigned addChunk_tIME (ucvector *out, const LodePNGTime *time)
- static unsigned addChunk_pHYs (ucvector *out, const LodePNGInfo *info)
- static void filterScanline (unsigned char *out, const unsigned char *scanline, const unsigned char *prevline, size_t length, size_t bytewidth, unsigned char filterType)
- static float flog2 (float f)
- static unsigned filter (unsigned char *out, const unsigned char *in, unsigned w, unsigned h, const LodePNGColorMode *info, const LodePNGEncoderSettings *settings)
- static void addPaddingBits (unsigned char *out, const unsigned char *in, size_t olinebits, size_t ilinebits, unsigned h)
- static void Adam7_interlace (unsigned char *out, const unsigned char *in, unsigned w, unsigned h, unsigned bpp)
- static unsigned preProcessScanlines (unsigned char **out, size_t *outsize, const unsigned char *in, unsigned w, unsigned h, const LodePNGInfo *info_png, const LodePNGEncoderSettings *settings)
- static unsigned getPaletteTranslucency (const unsigned char *palette, size_t palettesize)
- static unsigned addUnknownChunks (ucvector *out, unsigned char *data, size_t datasize)
- unsigned lodepng_encode (unsigned char **out, size_t *outsize, const unsigned char *image, unsigned w, unsigned h, LodePNGState *state)
- unsigned lodepng_encode_memory (unsigned char **out, size_t *outsize, const unsigned char *image, unsigned w, unsigned h, LodePNGColorType colortype, unsigned bitdepth)
- unsigned lodepng_encode32 (unsigned char **out, size_t *outsize, const unsigned char *image, unsigned w, unsigned h)
- unsigned lodepng_encode24 (unsigned char **out, size_t *outsize, const unsigned char *image, unsigned w, unsigned h)
- unsigned lodepng_encode_file (const char *filename, const unsigned char *image, unsigned w, unsigned h, LodePNGColorType colortype, unsigned bitdepth)
- unsigned lodepng_encode32_file (const char *filename, const unsigned char *image, unsigned w, unsigned h)

- unsigned lodepng_encode24_file (const char *filename, const unsigned char *image, unsigned w, unsigned h)
- void lodepng_encoder_settings_init (LodePNGEncoderSettings *settings)
- const char * lodepng_error_text (unsigned code)

Variables

- const char * LODEPNG_VERSION_STRING = "20161127"
- static const unsigned LENGTHBASE [29]
- static const unsigned LENGTHEXTRA [29]
- static const unsigned DISTANCEBASE [30]
- static const unsigned DISTANCEEXTRA [30]
- static const unsigned CLCL_ORDER [NUM_CODE_LENGTH_CODES] = {16, 17, 18, 0, 8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 1, 15}
- static const size_t MAX_SUPPORTED_DEFLATE_LENGTH = 258
- static const unsigned HASH_NUM_VALUES = 65536
- static const unsigned HASH_BIT_MASK = 65535
- const LodePNGCompressSettings lodepng_default_compress_settings = {2, 1, DEFAULT_WINDOWSIZE, 3, 128, 1, 0, 0, 0}
- const LodePNGDecompressSettings lodepng_default_decompress_settings = {0, 0, 0, 0}
- static unsigned lodepng_crc32_table [256]
- static const unsigned ADAM7_IX [7] = { 0, 4, 0, 2, 0, 1, 0 }
- static const unsigned ADAM7_IY [7] = { 0, 0, 4, 0, 2, 0, 1 }
- static const unsigned ADAM7_DX [7] = { 8, 8, 4, 4, 2, 2, 1 }
- static const unsigned ADAM7_DY [7] = { 8, 8, 8, 4, 4, 2, 2 }

4.1.1 Macro Definition Documentation

4.1.1.1 addBitToStream

```
#define addBitToStream(
            bitpointer,
            bitstream,
            bit )
```

Value:

```
{\
  /*add a new byte at the end*/\
  if(((*bitpointer) & 7) == 0) ucvector_push_back(bitstream, (unsigned char)0);\
  /*earlier bit of huffman code is in a lesser significant bit of an earlier byte*/\
  (bitstream->data[bitstream->size - 1]) |= (bit << ((*bitpointer) & 0x7));\
  ++(*bitpointer);\
}
```

Definition at line 421 of file lodepng.cpp.

4.1.1.2 CERROR_BREAK

```
#define CERROR_BREAK(
            errorvar,
            code )
```

Value:

```
{\
  errorvar = code;\
  break;\
}
```

Definition at line 96 of file lodepng.cpp.

4.1.1.3 CERROR_RETURN

```
#define CERROR_RETURN(
            errorvar,
            code )
```

Value:

```
{\
  errorvar = code;\
  return;\
}
```

Definition at line 120 of file lodepng.cpp.

4.1.1.4 CERROR_RETURN_ERROR

```
#define CERROR_RETURN_ERROR(
            errorvar,
            code )
```

Value:

```
{\
  errorvar = code;\
  return code;\
}
```

Definition at line 106 of file lodepng.cpp.

4.1.1.5 CERROR_TRY_RETURN

```
#define CERROR_TRY_RETURN(
            call )
```

Value:

```
{\
  unsigned error = call;\
  if(error) return error;\
}
```

Definition at line 113 of file lodepng.cpp.

4.1.1.6 DEFAULT_WINDOWSIZE

```
#define DEFAULT_WINDOWSIZE 2048
```

Definition at line 2271 of file lodepng.cpp.

4.1.1.7 ERROR_BREAK

```
#define ERROR_BREAK(
            code ) CERROR_BREAK(error, code)
```

Definition at line 103 of file lodepng.cpp.

4.1.1.8 FIRST_LENGTH_CODE_INDEX

```
#define FIRST_LENGTH_CODE_INDEX 257
```

Definition at line 470 of file lodepng.cpp.

4.1.1.9 LAST_LENGTH_CODE_INDEX

```
#define LAST_LENGTH_CODE_INDEX 285
```

Definition at line 471 of file lodepng.cpp.

4.1.1.10 NUM_CODE_LENGTH_CODES

```
#define NUM_CODE_LENGTH_CODES 19
```

Definition at line 477 of file lodepng.cpp.

4.1.1.11 NUM_DEFLATE_CODE_SYMBOLS

```
#define NUM_DEFLATE_CODE_SYMBOLS 288
```

Definition at line 473 of file lodepng.cpp.

4.1.1.12 NUM_DISTANCE_SYMBOLS

```
#define NUM_DISTANCE_SYMBOLS 32
```

Definition at line 475 of file lodepng.cpp.

4.1.1.13 READBIT

```
#define READBIT(
              bitpointer,
              bitstream ) ((bitstream[bitpointer >> 3] >> (bitpointer & 0x7)) & (unsigned char)1)
```

Definition at line 445 of file lodepng.cpp.

4.1.2 Typedef Documentation

4.1.2.1 BPMLists

```
typedef struct BPMLists BPMLists
```

4.1.2.2 BPMNode

```
typedef struct BPMNode BPMNode
```

4.1.2.3 ColorTree

```
typedef struct ColorTree ColorTree
```

Definition at line 2992 of file lodepng.cpp.

4.1.2.4 Hash

```
typedef struct Hash Hash
```

4.1.2.5 HuffmanTree

```
typedef struct HuffmanTree HuffmanTree
```

4.1.2.6 ucvector

```
typedef struct ucvector ucvector
```

4.1.2.7 uivector

```
typedef struct uivector uivector
```

4.1.3 Function Documentation

4.1.3.1 Adam7_deinterlace()

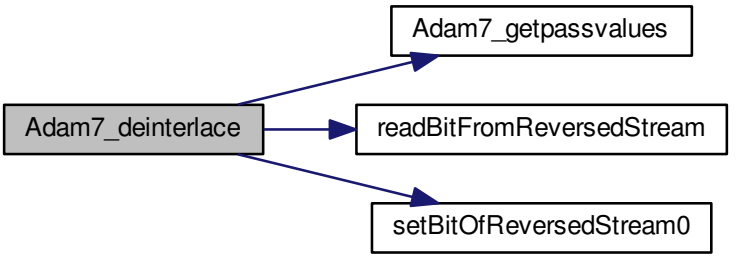
```
static void Adam7_deinterlace (
            unsigned char * out,
            const unsigned char * in,
            unsigned w,
            unsigned h,
            unsigned bpp )  [static]
```

Definition at line 4085 of file lodepng.cpp.

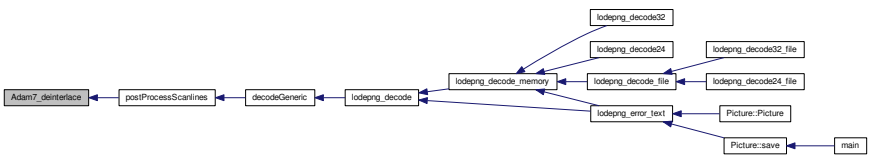
```
4086 {
4087   unsigned passw[7], passh[7];
4088   size_t filter_passstart[8], padded_passstart[8], passstart[8];
4089   unsigned i;
4090
4091   Adam7_getpassvalues(passw, passh, filter_passstart, padded_passstart, passsta
       bpp);
4092
4093   if(bpp >= 8)
4094   {
4095     for(i = 0; i != 7; ++i)
4096     {
4097       unsigned x, y, b;
4098       size_t bytewidth = bpp / 8;
4099       for(y = 0; y < passh[i]; ++y)
4100       for(x = 0; x < passw[i]; ++x)
4101       {
4102         size_t pixelinstart = passstart[i] + (y * passw[i] + x) * bytewidth;
4103         size_t pixeloutstart = ((ADAM7_IY[i] + y * ADAM7_DY[i]) * w +
      ADAM7_IX[i] + x * ADAM7_DX[i]) * bytewidth;
4104         for(b = 0; b < bytewidth; ++b)
```

```
4105           {
4106             out[pixeloutstart + b] = in[pixelinstart + b];
4107           }
4108         }
4109       }
4110     }
4111     else /*bpp < 8: Adam7 with pixels < 8 bit is a bit trickier: with bit pointer
4112     {
4113       for(i = 0; i != 7; ++i)
4114       {
4115         unsigned x, y, b;
4116         unsigned ilinebits = bpp * passw[i];
4117         unsigned olinebits = bpp * w;
4118         size_t obp, ibp; /*bit pointers (for out and in buffer)*/
4119         for(y = 0; y < passh[i]; ++y)
4120         for(x = 0; x < passw[i]; ++x)
4121         {
4122           ibp = (8 * passstart[i]) + (y * ilinebits + x * bpp);
4123           obp = (ADAM7_IY[i] + y * ADAM7_DY[i]) * olinebits + (
      ADAM7_IX[i] + x * ADAM7_DX[i]) * bpp;
4124           for(b = 0; b < bpp; ++b)
4125           {
4126             unsigned char bit = readBitFromReversedStream(&ibp, in);
4127             /*note that this function assumes the out buffer is completely 0, use
       otherwise*/
4128             setBitOfReversedStream0(&obp, out, bit);
4129           }
4130         }
4131       }
4132     }
4133 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.2 Adam7_getpassvalues()

```
static void Adam7_getpassvalues (
            unsigned passw[7],
```

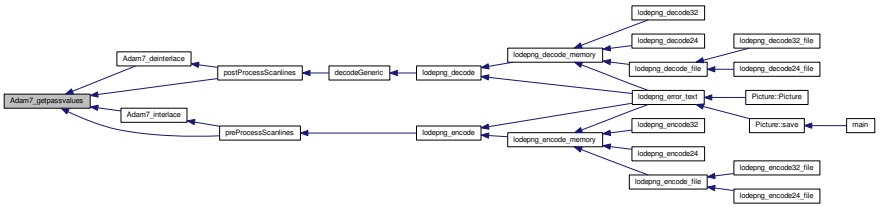
```
unsigned passh[7],
size_t filter_passstart[8],
size_t padded_passstart[8],
size_t passstart[8],
unsigned w,
unsigned h,
unsigned bpp ) [static]
```

Definition at line 3868 of file lodepng.cpp.

```
3870 {
3871   /*the passstart values have 8 values: the 8th one indicates the byte after th
       pass*/
3872   unsigned i;
3873
3874   /*calculate width and height in pixels of each pass*/
3875   for(i = 0; i != 7; ++i)
3876   {
3877     passw[i] = (w + ADAM7_DX[i] - ADAM7_IX[i] - 1) / ADAM7_DX[i];
3878     passh[i] = (h + ADAM7_DY[i] - ADAM7_IY[i] - 1) / ADAM7_DY[i];
3879     if(passw[i] == 0) passh[i] = 0;
3880     if(passh[i] == 0) passw[i] = 0;
3881   }
3882
3883   filter_passstart[0] = padded_passstart[0] = passstart[0] = 0;
3884   for(i = 0; i != 7; ++i)
3885   {
3886     /*if passw[i] is 0, it's 0 bytes, not 1 (no filtertype-byte)*/
3887     filter_passstart[i + 1] = filter_passstart[i]
3888                               + ((passw[i] && passh[i]) ? passh[i] * (1 + (passw[
3889     /*bits padded if needed to fill full byte at end of each scanline*/
3890     padded_passstart[i + 1] = padded_passstart[i] + passh[i] * ((passw[i] * bpp
```

```
3891     /*only padded at end of reduced image*/
3892     passstart[i + 1] = passstart[i] + (passh[i] * passw[i] * bpp + 7) / 8;
3893   }
3894 }
```

Here is the caller graph for this function:



4.1.3.3 Adam7_interlace()

```
static void Adam7_interlace (
            unsigned char * out,
            const unsigned char * in,
            unsigned w,
            unsigned h,
            unsigned bpp )  [static]
```

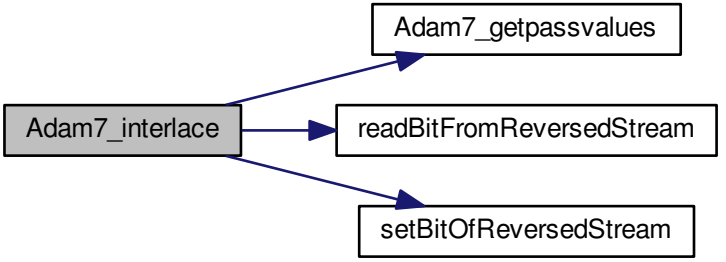
Definition at line 5462 of file lodepng.cpp.

```
5463 {
5464   unsigned passw[7], passh[7];
```

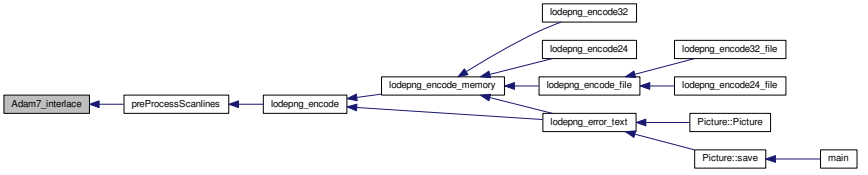
```
5465   size_t filter_passstart[8], padded_passstart[8], passstart[8];
5466   unsigned i;
5467
5468   Adam7_getpassvalues(passw, passh, filter_passstart, padded_passstart, passsta
       bpp);
5469
5470   if(bpp >= 8)
5471   {
5472     for(i = 0; i != 7; ++i)
5473     {
5474       unsigned x, y, b;
5475       size_t bytewidth = bpp / 8;
5476       for(y = 0; y < passh[i]; ++y)
5477       for(x = 0; x < passw[i]; ++x)
5478       {
5479         size_t pixelinstart = ((ADAM7_IY[i] + y * ADAM7_DY[i]) * w +
     ADAM7_IX[i] + x * ADAM7_DX[i]) * bytewidth;
5480         size_t pixeloutstart = passstart[i] + (y * passw[i] + x) * bytewidth;
5481         for(b = 0; b < bytewidth; ++b)
5482         {
5483           out[pixeloutstart + b] = in[pixelinstart + b];
5484         }
5485       }
5486     }
5487   }
5488   else /*bpp < 8: Adam7 with pixels < 8 bit is a bit trickier: with bit pointer
5489   {
5490     for(i = 0; i != 7; ++i)
5491     {
5492       unsigned x, y, b;
5493       unsigned ilinebits = bpp * passw[i];
```

```
5494       unsigned olinebits = bpp * w;
5495       size_t obp, ibp; /*bit pointers (for out and in buffer)*/
5496       for(y = 0; y < passh[i]; ++y)
5497       for(x = 0; x < passw[i]; ++x)
5498       {
5499         ibp = (ADAM7_IY[i] + y * ADAM7_DY[i]) * olinebits + (
      ADAM7_IX[i] + x * ADAM7_DX[i]) * bpp;
5500         obp = (8 * passstart[i]) + (y * ilinebits + x * bpp);
5501         for(b = 0; b < bpp; ++b)
5502         {
5503           unsigned char bit = readBitFromReversedStream(&ibp, in);
5504           setBitOfReversedStream(&obp, out, bit);
5505         }
5506       }
5507     }
5508   }
5509 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.4 addBitsToStream()

```
static void addBitsToStream (
            size_t * bitpointer,
```

```
            ucvector * bitstream,
            unsigned value,
            size_t nbits )  [static]
```

Definition at line 430 of file lodepng.cpp.

```
431 {
432   size_t i;
433   for(i = 0; i != nbits; ++i) addBitToStream(bitpointer, bitstream, (unsigned ch
       i) & 1));
434 }
```

Here is the caller graph for this function:

4.1.3.5 addBitsToStreamReversed()

```
static void addBitsToStreamReversed (
            size_t * bitpointer,
            ucvector * bitstream,
            unsigned value,
            size_t nbits )  [static]
```

Definition at line 436 of file lodepng.cpp.

```
437 {
438   size_t i;
439   for(i = 0; i != nbits; ++i) addBitToStream(bitpointer, bitstream, (unsigned ch
       (nbits - 1 - i)) & 1));
440 }
```

Here is the caller graph for this function:

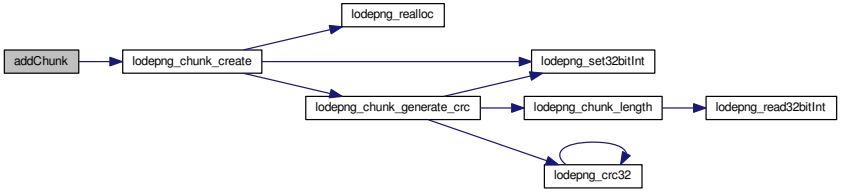
4.1.3.6 addChunk()

```
static unsigned addChunk (
            ucvector * out,
            const char * chunkName,
            const unsigned char * data,
            size_t length )  [static]
```

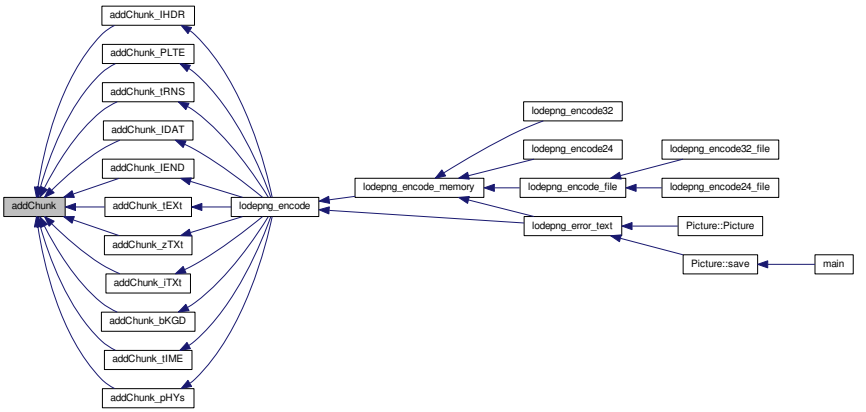
Definition at line 4868 of file lodepng.cpp.

```
4869 {
4870   CERROR_TRY_RETURN(lodepng_chunk_create(&out->
      data, &out->size, (unsigned)length, chunkName, data));
4871   out->allocsize = out->size; /*fix the allocsize again*/
4872   return 0;
4873 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.7 addChunk_bKGD()

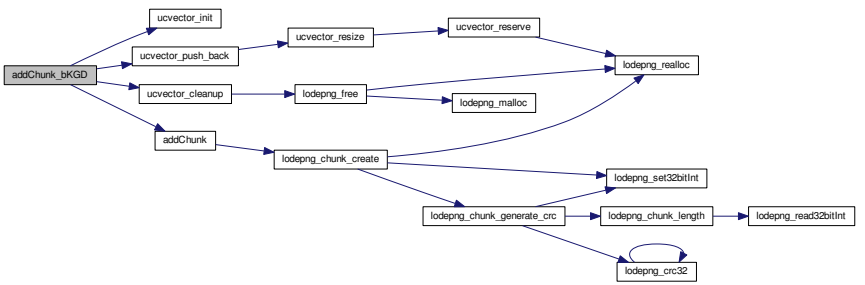
```
static unsigned addChunk_bKGD (
            ucvector * out,
            const LodePNGInfo * info )  [static]
```

Definition at line 5079 of file lodepng.cpp.

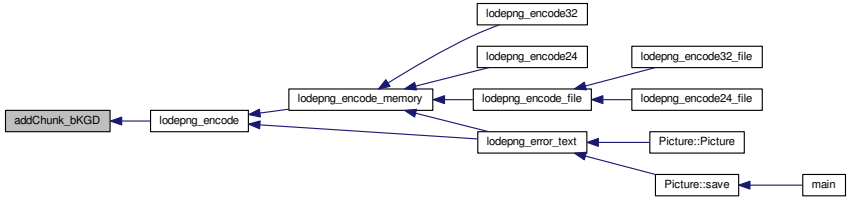
```
5080 {
5081   unsigned error = 0;
5082   ucvector bKGD;
5083   ucvector_init(&bKGD);
5084   if(info->color.colortype == LCT_GREY || info->color.
      colortype == LCT_GREY_ALPHA)
5085   {
5086     ucvector_push_back(&bKGD, (unsigned char)(info->
      background_r >> 8));
5087     ucvector_push_back(&bKGD, (unsigned char)(info->
      background_r & 255));
5088   }
5089   else if(info->color.colortype == LCT_RGB || info->color.
      colortype == LCT_RGBA)
5090   {
5091     ucvector_push_back(&bKGD, (unsigned char)(info->
      background_r >> 8));
5092     ucvector_push_back(&bKGD, (unsigned char)(info->
      background_r & 255));
5093     ucvector_push_back(&bKGD, (unsigned char)(info->
      background_g >> 8));
5094     ucvector_push_back(&bKGD, (unsigned char)(info->
      background_g & 255));
5095     ucvector_push_back(&bKGD, (unsigned char)(info->
```

```
    background_b >> 8));
5096    ucvector_push_back(&bKGD, (unsigned char)(info->
    background_b & 255));
5097   }
5098   else if(info->color.colortype == LCT_PALETTE)
5099   {
5100    ucvector_push_back(&bKGD, (unsigned char)(info->
    background_r & 255)); /*palette index*/
5101   }
5102
5103   error = addChunk(out, "bKGD", bKGD.data, bKGD.size);
5104   ucvector_cleanup(&bKGD);
5105
5106   return error;
5107 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.8 addChunk_IDAT()

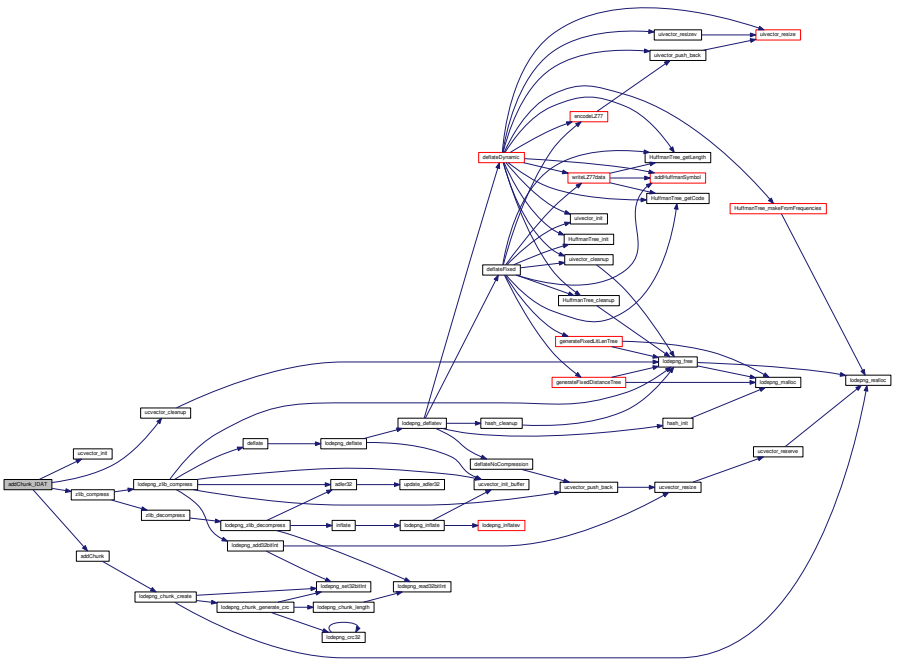
```
static unsigned addChunk_IDAT (
            ucvector * out,
            const unsigned char * data,
            size_t datasize,
            LodePNGCompressSettings * zlibsettings )  [static]
```

Definition at line 4971 of file lodepng.cpp.

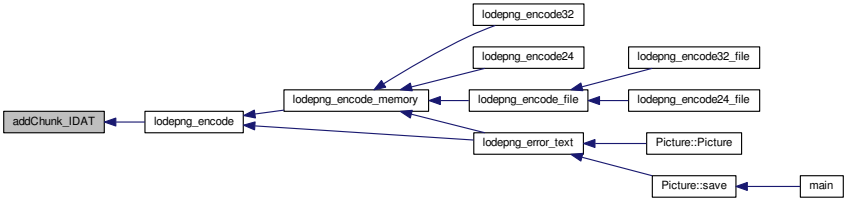
```
4973 {
4974   ucvector zlibdata;
4975   unsigned error = 0;
4976
4977   /*compress with the Zlib compressor*/
4978   ucvector_init(&zlibdata);
4979   error = zlib_compress(&zlibdata.data, &zlibdata.size, data, datasize, zlibset
4980   if(!error) error = addChunk(out, "IDAT", zlibdata.data, zlibdata.
```

```
      size);
4981   ucvector_cleanup(&zlibdata);
4982
4983   return error;
4984 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



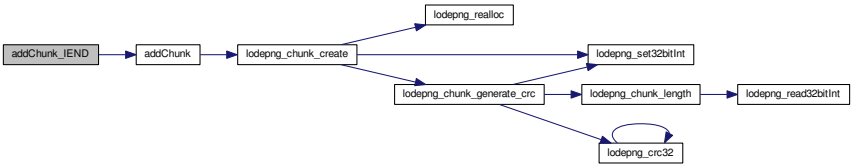
4.1.3.9 addChunk_IEND()

```
static unsigned addChunk_IEND (
            ucvector * out )  [static]
```

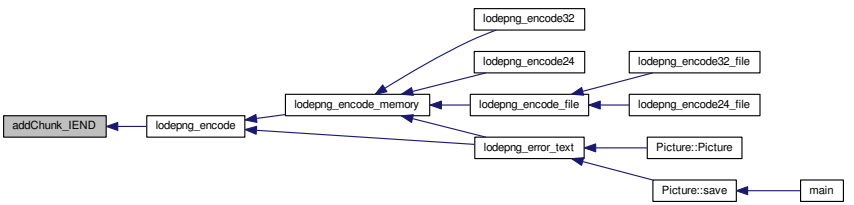
Definition at line 4986 of file lodepng.cpp.

```
4987 {
4988   unsigned error = 0;
4989   error = addChunk(out, "IEND", 0, 0);
4990   return error;
4991 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.10 addChunk_IHDR()

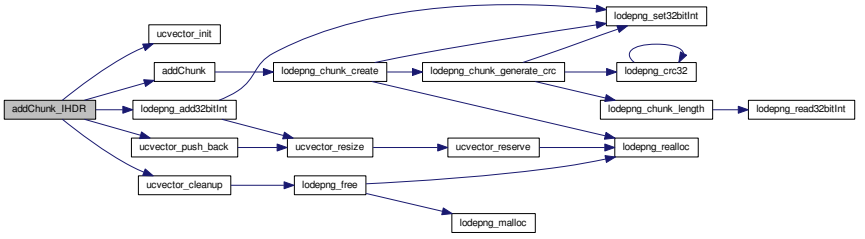
```
static unsigned addChunk_IHDR (
            ucvector * out,
            unsigned w,
            unsigned h,
            LodePNGColorType colortype,
```

unsigned *bitdepth*,
unsigned *interlace_method*) [static]

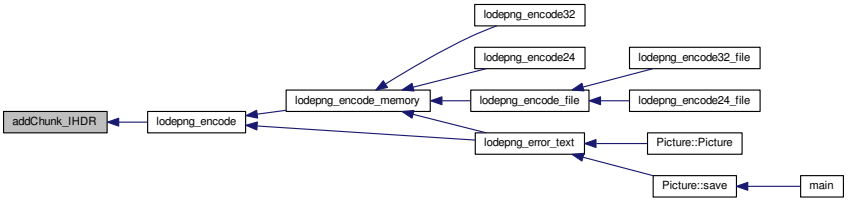
Definition at line 4888 of file lodepng.cpp.

```
4890 {
4891   unsigned error = 0;
4892   ucvector header;
4893   ucvector_init(&header);
4894
4895   lodepng_add32bitInt(&header, w); /*width*/
4896   lodepng_add32bitInt(&header, h); /*height*/
4897   ucvector_push_back(&header, (unsigned char)bitdepth); /*bit depth*/
4898   ucvector_push_back(&header, (unsigned char)colortype); /*color type*/
4899   ucvector_push_back(&header, 0); /*compression method*/
4900   ucvector_push_back(&header, 0); /*filter method*/
4901   ucvector_push_back(&header, interlace_method); /*interlace method*/
4902
4903   error = addChunk(out, "IHDR", header.data, header.size);
4904   ucvector_cleanup(&header);
4905
4906   return error;
4907 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.11 addChunk_iTXt()

```
static unsigned addChunk_iTXt (
            ucvector * out,
            unsigned compressed,
```

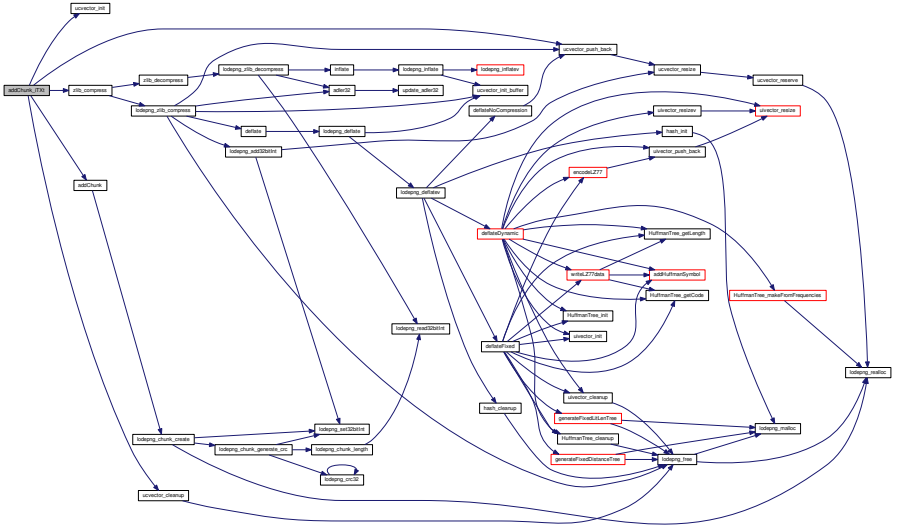
```
    const char * keyword,
    const char * langtag,
    const char * transkey,
    const char * textstring,
    LodePNGCompressSettings * zlibsettings )  [static]
```

Definition at line 5038 of file lodepng.cpp.

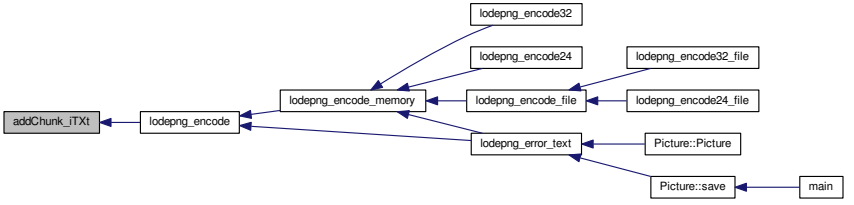
```
5040 {
5041   unsigned error = 0;
5042   ucvector data;
5043   size_t i, textsize = strlen(textstring);
5044
5045   ucvector_init(&data);
5046
5047   for(i = 0; keyword[i] != 0; ++i) ucvector_push_back(&data, (unsigned char)key
5048   if(i < 1 || i > 79) return 89; /*error: invalid keyword size*/
5049   ucvector_push_back(&data, 0); /*null termination char*/
5050   ucvector_push_back(&data, compressed ? 1 : 0); /*compression flag*/
5051   ucvector_push_back(&data, 0); /*compression method*/
5052   for(i = 0; langtag[i] != 0; ++i) ucvector_push_back(&data, (unsigned char)lan
5053   ucvector_push_back(&data, 0); /*null termination char*/
5054   for(i = 0; transkey[i] != 0; ++i) ucvector_push_back(&data, (unsigned char)tr
     ;
5055   ucvector_push_back(&data, 0); /*null termination char*/
5056
5057   if(compressed)
5058   {
5059     ucvector compressed_data;
5060     ucvector_init(&compressed_data);
5061     error = zlib_compress(&compressed_data.data, &compressed_data.
     size,
```

```
5062                                (unsigned char*)textstring, textsize, zlibsettings);
5063     if(!error)
5064     {
5065       for(i = 0; i != compressed_data.size; ++i) ucvector_push_back(&data,
     compressed_data.data[i]);
5066     }
5067     ucvector_cleanup(&compressed_data);
5068   }
5069   else /*not compressed*/
5070   {
5071     for(i = 0; textstring[i] != 0; ++i) ucvector_push_back(&data, (unsigned cha
     textstring[i]);
5072   }
5073
5074   if(!error) error = addChunk(out, "iTXt", data.data, data.size);
5075   ucvector_cleanup(&data);
5076   return error;
5077 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



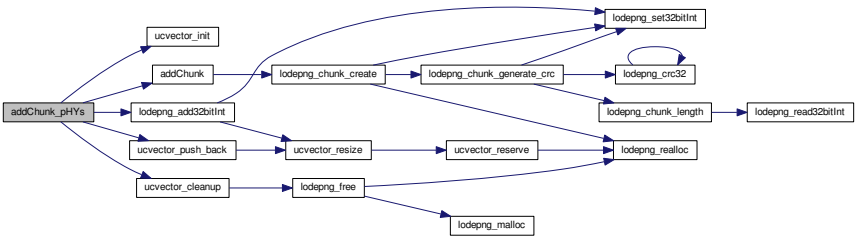
4.1.3.12 addChunk_pHYs()

```
static unsigned addChunk_pHYs (
            ucvector * out,
            const LodePNGInfo * info )  [static]
```

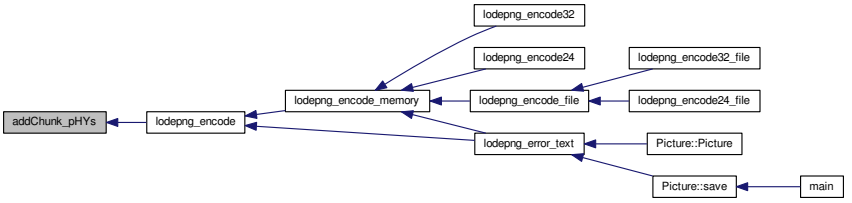
Definition at line 5126 of file lodepng.cpp.

```
5127 {
5128   unsigned error = 0;
5129   ucvector data;
5130   ucvector_init(&data);
5131
5132   lodepng_add32bitInt(&data, info->phys_x);
5133   lodepng_add32bitInt(&data, info->phys_y);
5134   ucvector_push_back(&data, info->phys_unit);
5135
5136   error = addChunk(out, "pHYs", data.data, data.size);
5137   ucvector_cleanup(&data);
5138
5139   return error;
5140 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.13 addChunk_PLTE()

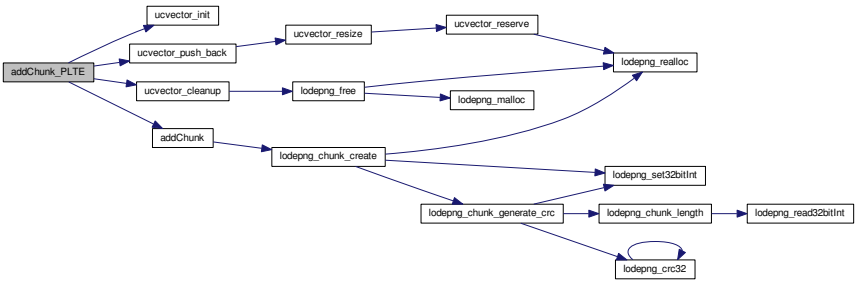
```
static unsigned addChunk_PLTE (
            ucvector * out,
            const LodePNGColorMode * info )  [static]
```

Definition at line 4909 of file lodepng.cpp.

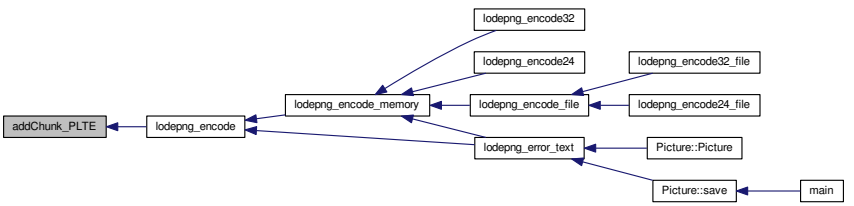
```
4910 {
4911   unsigned error = 0;
4912   size_t i;
4913   ucvector PLTE;
4914   ucvector_init(&PLTE);
4915   for(i = 0; i != info->palettesize * 4; ++i)
4916   {
4917     /*add all channels except alpha channel*/
4918     if(i % 4 != 3) ucvector_push_back(&PLTE, info->palette[i]);
```

```
4919   }
4920   error = addChunk(out, "PLTE", PLTE.data, PLTE.size);
4921   ucvector_cleanup(&PLTE);
4922
4923   return error;
4924 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



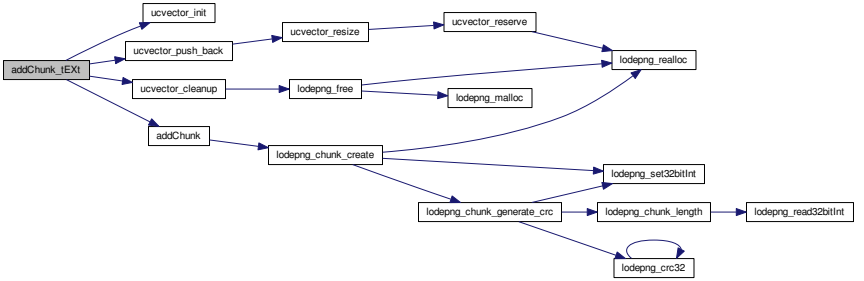
4.1.3.14 addChunk_tEXt()

```
static unsigned addChunk_tEXt (
            ucvector * out,
            const char * keyword,
            const char * textstring )  [static]
```

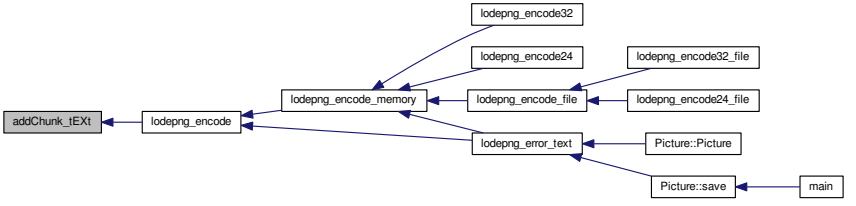
Definition at line 4995 of file lodepng.cpp.

```
4996 {
4997   unsigned error = 0;
4998   size_t i;
4999   ucvector text;
5000   ucvector_init(&text);
5001   for(i = 0; keyword[i] != 0; ++i) ucvector_push_back(&text, (unsigned char)key
5002   if(i < 1 || i > 79) return 89; /*error: invalid keyword size*/
5003   ucvector_push_back(&text, 0); /*0 termination char*/
5004   for(i = 0; textstring[i] != 0; ++i) ucvector_push_back(&text, (unsigned char)
      [i]);
5005   error = addChunk(out, "tEXt", text.data, text.size);
5006   ucvector_cleanup(&text);
5007
5008   return error;
5009 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



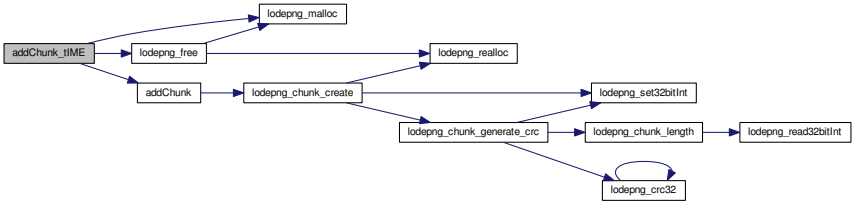
4.1.3.15 addChunk_tIME()

```
static unsigned addChunk_tIME (
            ucvector * out,
            const LodePNGTime * time )  [static]
```

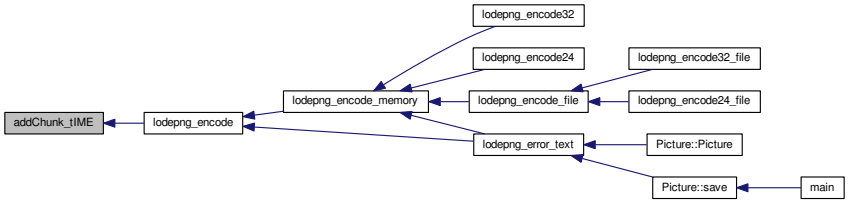
Definition at line 5109 of file lodepng.cpp.

```
5110 {
5111   unsigned error = 0;
5112   unsigned char* data = (unsigned char*)lodepng_malloc(7);
5113   if(!data) return 83; /*alloc fail*/
5114   data[0] = (unsigned char)(time->year >> 8);
5115   data[1] = (unsigned char)(time->year & 255);
5116   data[2] = (unsigned char)time->month;
5117   data[3] = (unsigned char)time->day;
5118   data[4] = (unsigned char)time->hour;
5119   data[5] = (unsigned char)time->minute;
5120   data[6] = (unsigned char)time->second;
5121   error = addChunk(out, "tIME", data, 7);
5122   lodepng_free(data);
5123   return error;
5124 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.16 addChunk_tRNS()

```
static unsigned addChunk_tRNS (
            ucvector * out,
            const LodePNGColorMode * info )  [static]
```

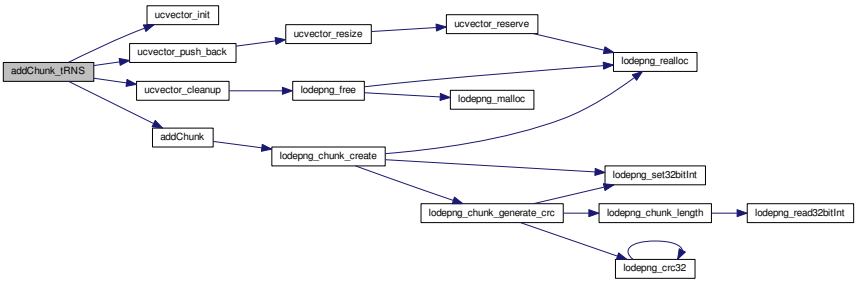
Definition at line 4926 of file lodepng.cpp.

```
4927 {
4928   unsigned error = 0;
4929   size_t i;
4930   ucvector tRNS;
4931   ucvector_init(&tRNS);
4932   if(info->colortype == LCT_PALETTE)
4933   {
4934     size_t amount = info->palettesize;
4935     /*the tail of palette values that all have 255 as alpha, does not have to b
```

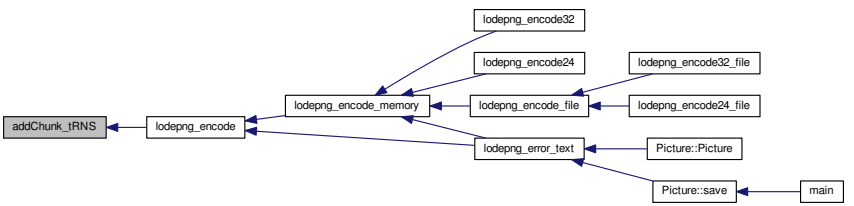
```
4936     for(i = info->palettesize; i != 0; --i)
4937     {
4938       if(info->palette[4 * (i - 1) + 3] == 255) --amount;
4939       else break;
4940     }
4941     /*add only alpha channel*/
4942     for(i = 0; i != amount; ++i) ucvector_push_back(&tRNS, info->
     palette[4 * i + 3]);
4943   }
4944   else if(info->colortype == LCT_GREY)
4945   {
4946     if(info->key_defined)
4947     {
4948       ucvector_push_back(&tRNS, (unsigned char)(info->key_r >> 8));
4949       ucvector_push_back(&tRNS, (unsigned char)(info->key_r & 255));
4950     }
4951   }
4952   else if(info->colortype == LCT_RGB)
4953   {
4954     if(info->key_defined)
4955     {
4956       ucvector_push_back(&tRNS, (unsigned char)(info->key_r >> 8));
4957       ucvector_push_back(&tRNS, (unsigned char)(info->key_r & 255));
4958       ucvector_push_back(&tRNS, (unsigned char)(info->key_g >> 8));
4959       ucvector_push_back(&tRNS, (unsigned char)(info->key_g & 255));
4960       ucvector_push_back(&tRNS, (unsigned char)(info->key_b >> 8));
4961       ucvector_push_back(&tRNS, (unsigned char)(info->key_b & 255));
4962     }
4963   }
4964
4965   error = addChunk(out, "tRNS", tRNS.data, tRNS.size);
```

```
4966    ucvector_cleanup(&tRNS);
4967
4968    return error;
4969 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.17 addChunk_zTXt()

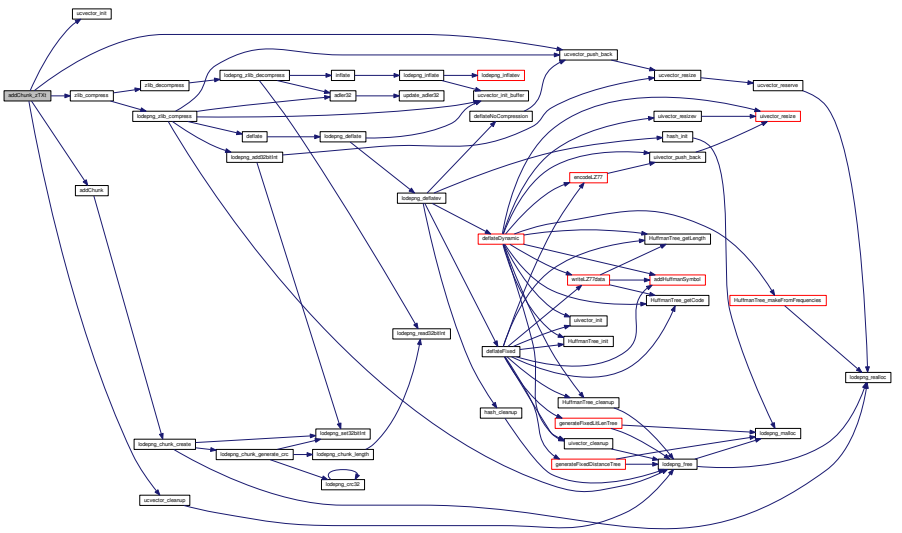
```
static unsigned addChunk_zTXt (
            ucvector * out,
            const char * keyword,
            const char * textstring,
            LodePNGCompressSettings * zlibsettings )  [static]
```

Definition at line 5011 of file lodepng.cpp.

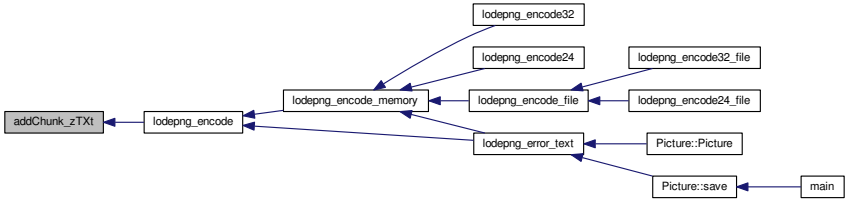
```
5013 {
5014   unsigned error = 0;
5015   ucvector data, compressed;
5016   size_t i, textsize = strlen(textstring);
5017
5018   ucvector_init(&data);
5019   ucvector_init(&compressed);
5020   for(i = 0; keyword[i] != 0; ++i) ucvector_push_back(&data, (unsigned char)key
5021   if(i < 1 || i > 79) return 89; /*error: invalid keyword size*/
5022   ucvector_push_back(&data, 0); /*0 termination char*/
5023   ucvector_push_back(&data, 0); /*compression method: 0*/
5024
5025   error = zlib_compress(&compressed.data, &compressed.size,
5026                         (unsigned char*)textstring, textsize, zlibsettings);
5027   if(!error)
5028   {
5029     for(i = 0; i != compressed.size; ++i) ucvector_push_back(&data, compressed.
      data[i]);
5030     error = addChunk(out, "zTXt", data.data, data.size);
5031   }
5032
5033   ucvector_cleanup(&compressed);
```

```
5034   ucvector_cleanup(&data);
5035   return error;
5036 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



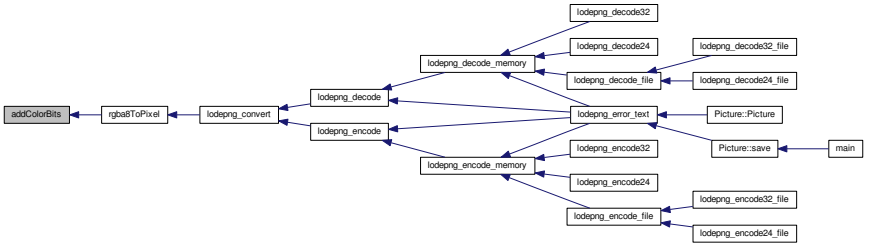
4.1.3.18 addColorBits()

```
static void addColorBits (
            unsigned char * out,
            size_t index,
            unsigned bits,
            unsigned in )  [static]
```

Definition at line 2981 of file lodepng.cpp.

```
2982 {
2983   unsigned m = bits == 1 ? 7 : bits == 2 ? 3 : 1; /*8 / bits - 1*/
2984   /*p = the partial index in the byte, e.g. with 4 palettebits it is 0 for firs
      */
2985   unsigned p = index & m;
2986   in &= (1u << bits) - 1u; /*filter out any other bits of the input value*/
2987   in = in << (bits * (m - p));
2988   if(p == 0) out[index * bits / 8] = in;
2989   else out[index * bits / 8] |= in;
2990 }
```

Here is the caller graph for this function:



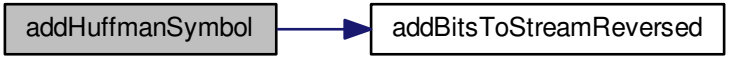
4.1.3.19 addHuffmanSymbol()

```
static void addHuffmanSymbol (
            size_t * bp,
            ucvector * compressed,
            unsigned code,
            unsigned bitlen )  [static]
```

Definition at line 1321 of file lodepng.cpp.

```
1322 {
1323   addBitsToStreamReversed(bp, compressed, code, bitlen);
1324 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.3.20 addLengthDistance()

```
static void addLengthDistance (
            uivector * values,
            size_t length,
            size_t distance )  [static]
```

Definition at line 1343 of file lodepng.cpp.

```
1344 {
1345   /*values in encoded vector are those used by deflate:
1346   0-255: literal bytes
1347   256: end
1348   257-285: length/distance pair (length code, followed by extra length bits, di
       bits)
1349   286-287: invalid*/
1350
1351   unsigned length_code = (unsigned)searchCodeIndex(LENGTHBASE, 29, length);
1352   unsigned extra_length = (unsigned)(length - LENGTHBASE[length_code]);
1353   unsigned dist_code = (unsigned)searchCodeIndex(DISTANCEBASE, 30, distance);
1354   unsigned extra_distance = (unsigned)(distance - DISTANCEBASE[dist_code]);
1355
1356   uivector_push_back(values, length_code +
      FIRST_LENGTH_CODE_INDEX);
1357   uivector_push_back(values, extra_length);
1358   uivector_push_back(values, dist_code);
1359   uivector_push_back(values, extra_distance);
1360 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.1.3.21 addPaddingBits()

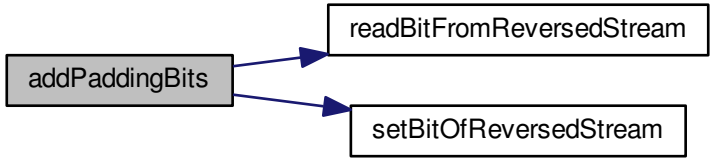
```
static void addPaddingBits (
            unsigned char * out,
            const unsigned char * in,
            size_t olinebits,
            size_t ilinebits,
            unsigned h )  [static]
```

Definition at line 5429 of file lodepng.cpp.

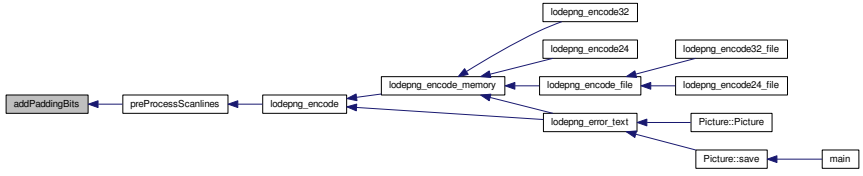
```
5431 {
5432   /*The opposite of the removePaddingBits function
5433   olinebits must be >= ilinebits*/
5434   unsigned y;
5435   size_t diff = olinebits - ilinebits;
5436   size_t obp = 0, ibp = 0; /*bit pointers*/
5437   for(y = 0; y != h; ++y)
5438   {
5439     size_t x;
5440     for(x = 0; x < ilinebits; ++x)
5441     {
```

```
5442         unsigned char bit = readBitFromReversedStream(&ibp, in);
5443         setBitOfReversedStream(&obp, out, bit);
5444       }
5445       /*obp += diff; --> no, fill in some value in the padding bits too, to avoid
5446       "Use of uninitialised value of size ###" warning from valgrind*/
5447       for(x = 0; x != diff; ++x) setBitOfReversedStream(&obp, out, 0);
5448     }
5449 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



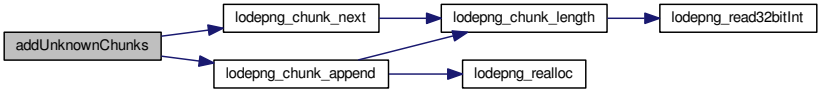
4.1.3.22 addUnknownChunks()

```
static unsigned addUnknownChunks (
            ucvector * out,
            unsigned char * data,
            size_t datasize )  [static]
```

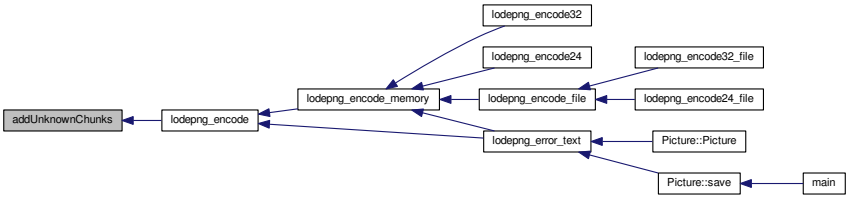
Definition at line 5627 of file lodepng.cpp.

```
5628 {
5629   unsigned char* inchunk = data;
5630   while((size_t)(inchunk - data) < datasize)
5631   {
5632     CERROR_TRY_RETURN(lodepng_chunk_append(&out->
      data, &out->size, inchunk));
5633     out->allocsize = out->size; /*fix the allocsize again*/
5634     inchunk = lodepng_chunk_next(inchunk);
5635   }
5636   return 0;
5637 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



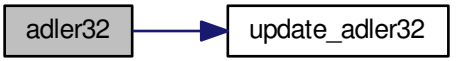
4.1.3.23 adler32()

```
static unsigned adler32 (
            const unsigned char * data,
            unsigned len )  [static]
```

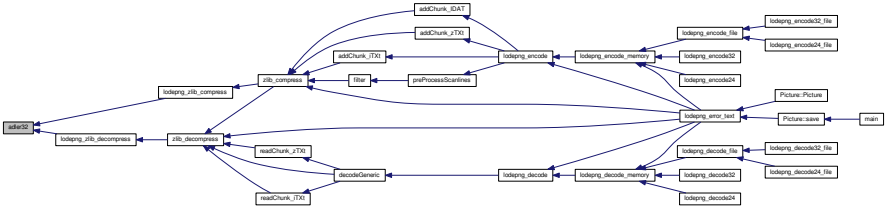
Definition at line 2115 of file lodepng.cpp.

```
2116 {
2117   return update_adler32(1L, data, len);
2118 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.24 boundaryPM()

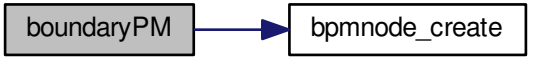
```
static void boundaryPM (
            BPMLists * lists,
            BPMNode * leaves,
            size_t numpresent,
            int c,
            int num )  [static]
```

Definition at line 758 of file lodepng.cpp.

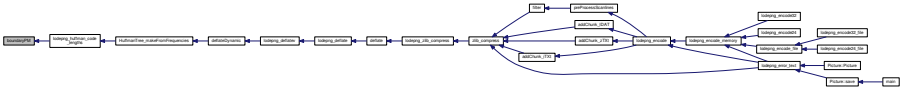
```
759 {
760   unsigned lastindex = lists->chains1[c]->index;
761
762   if(c == 0)
763   {
764     if(lastindex >= numpresent) return;
765     lists->chains0[c] = lists->chains1[c];
766     lists->chains1[c] = bpmnode_create(lists, leaves[lastindex].weight, lastinde
       0);
767   }
768   else
769   {
770     /*sum of the weights of the head nodes of the previous lookahead chains.*/
771     int sum = lists->chains0[c - 1]->weight + lists->chains1[c - 1]->
    weight;
772     lists->chains0[c] = lists->chains1[c];
773     if(lastindex < numpresent && sum > leaves[lastindex].weight)
774     {
775       lists->chains1[c] = bpmnode_create(lists, leaves[lastindex].weight, lastin
    1, lists->chains1[c]->tail);
776       return;
777     }
778     lists->chains1[c] = bpmnode_create(lists, sum, lastindex, lists->
    chains1[c - 1]);
779     /*in the end we are only interested in the chain of the last list, so no
780     need to recurse if we're at the last one (this gives measurable speedup)*/
781     if(num + 1 < (int)(2 * numpresent - 2))
782     {
783       boundaryPM(lists, leaves, numpresent, c - 1, num);
784       boundaryPM(lists, leaves, numpresent, c - 1, num);
785     }
```

```
786     }
787 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.25 bpmnode_create()

```
static BPMNode* bpmnode_create (
            BPMLists * lists,
            int weight,
            unsigned index,
            BPMNode * tail )  [static]
```

Definition at line 698 of file lodepng.cpp.

```
699 {
700   unsigned i;
701   BPMNode* result;
702 
703   /*memory full, so garbage collect*/
704   if(lists->nextfree >= lists->numfree)
705   {
706     /*mark only those that are in use*/
707     for(i = 0; i != lists->memsize; ++i) lists->memory[i].in_use = 0;
708     for(i = 0; i != lists->listsize; ++i)
709     {
710       BPMNode* node;
711       for(node = lists->chains0[i]; node != 0; node = node->tail) node->
    in_use = 1;
712       for(node = lists->chains1[i]; node != 0; node = node->tail) node->
    in_use = 1;
713     }
714     /*collect those that are free*/
715     lists->numfree = 0;
716     for(i = 0; i != lists->memsize; ++i)
717     {
718       if(!lists->memory[i].in_use) lists->freelist[lists->
    numfree++] = &lists->memory[i];
719     }
720     lists->nextfree = 0;
721   }
722 
723   result = lists->freelist[lists->nextfree++];
724   result->weight = weight;
725   result->index = index;
726   result->tail = tail;
```

```
727   return result;
728 }
```

Here is the caller graph for this function:

4.1.3.26 bpmnode_sort()

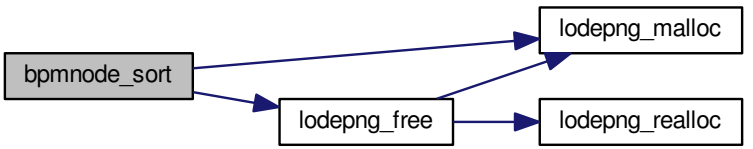
```
static void bpmnode_sort (
            BPMNode * leaves,
            size_t num )  [static]
```

Definition at line 731 of file lodepng.cpp.

```
732 {
733   BPMNode* mem = (BPMNode*)lodepng_malloc(sizeof(*leaves) * num);
734   size_t width, counter = 0;
735   for(width = 1; width < num; width *= 2)
736   {
737     BPMNode* a = (counter & 1) ? mem : leaves;
738     BPMNode* b = (counter & 1) ? leaves : mem;
739     size_t p;
740     for(p = 0; p < num; p += 2 * width)
741     {
742       size_t q = (p + width > num) ? num : (p + width);
```

```
743       size_t r = (p + 2 * width > num) ? num : (p + 2 * width);
744       size_t i = p, j = q, k;
745       for(k = p; k < r; k++)
746       {
747         if(i < q && (j >= r || a[i].weight <= a[j].weight)) b[k] = a[i++];
748         else b[k] = a[j++];
749       }
750     }
751     counter++;
752   }
753   if(counter & 1) memcpy(leaves, mem, sizeof(*leaves) * num);
754   lodepng_free(mem);
755 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

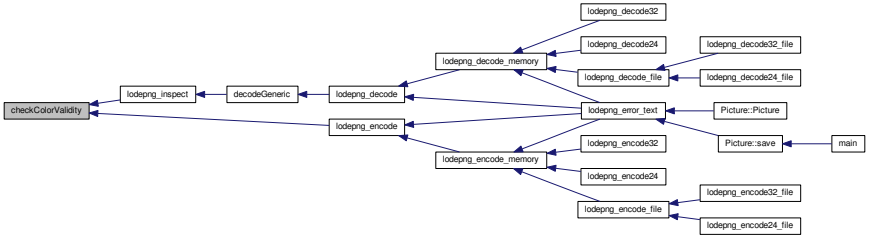
4.1.3.27 checkColorValidity()

```
static unsigned checkColorValidity (
            LodePNGColorType colortype,
            unsigned bd )  [static]
```

Definition at line 2548 of file lodepng.cpp.

```
2549 {
2550   switch(colortype)
2551   {
2552     case 0: if(!(bd == 1 || bd == 2 || bd == 4 || bd == 8 || bd == 16)) return
2553     case 2: if(!(                                 bd == 8 || bd == 16)) return
2554     case 3: if(!(bd == 1 || bd == 2 || bd == 4 || bd == 8             )) return
2555     case 4: if(!(                                 bd == 8 || bd == 16)) return
2556     case 6: if(!(                                 bd == 8 || bd == 16)) return
2557     default: return 31;
2558   }
2559   return 0; /*allowed color type / bits combination*/
2560 }
```

Here is the caller graph for this function:



4.1.3.28 color_tree_add()

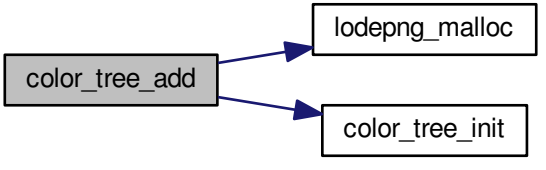
```
static void color_tree_add (
            ColorTree * tree,
            unsigned char r,
            unsigned char g,
            unsigned char b,
            unsigned char a,
            unsigned index )  [static]
```

Definition at line 3048 of file lodepng.cpp.

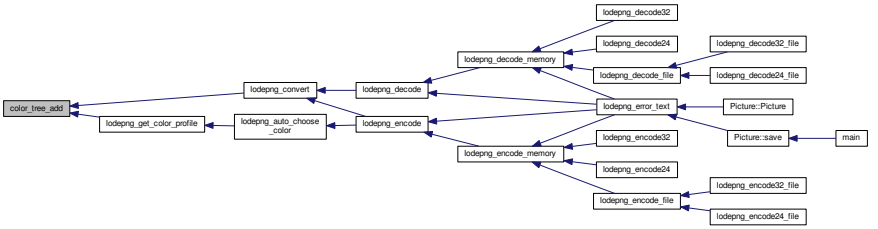
```
3050 {
3051   int bit;
3052   for(bit = 0; bit < 8; ++bit)
3053   {
3054     int i = 8 * ((r >> bit) & 1) + 4 * ((g >> bit) & 1) + 2 * ((b >> bit) & 1)
3055     if(!tree->children[i])
```

```
3056     {
3057       tree->children[i] = (ColorTree*)lodepng_malloc(sizeof(
     ColorTree));
3058       color_tree_init(tree->children[i]);
3059     }
3060     tree = tree->children[i];
3061   }
3062   tree->index = (int)index;
3063 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



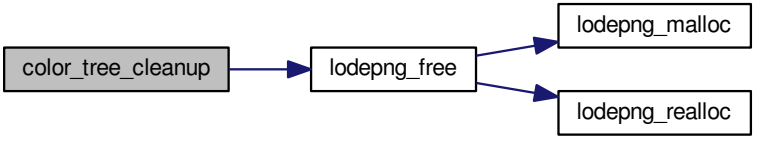
4.1.3.29 color_tree_cleanup()

```
static void color_tree_cleanup (
            ColorTree * tree )  [static]
```

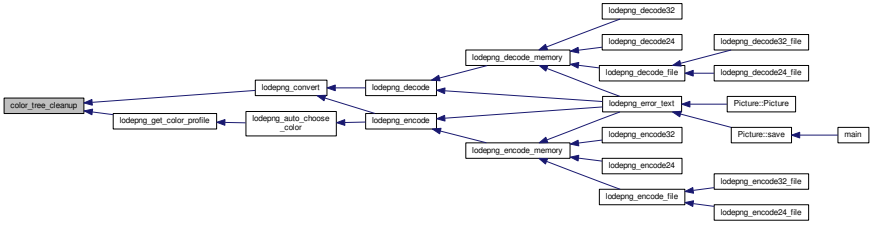
Definition at line 3013 of file lodepng.cpp.

```
3014 {
3015   int i;
3016   for(i = 0; i != 16; ++i)
3017   {
3018     if(tree->children[i])
3019     {
3020       color_tree_cleanup(tree->children[i]);
3021       lodepng_free(tree->children[i]);
3022     }
3023   }
3024 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.30 color_tree_get()

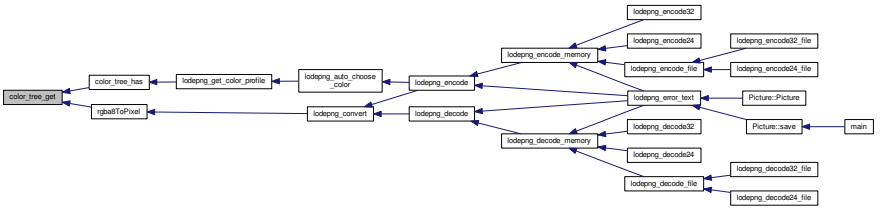
```
static int color_tree_get (
            ColorTree * tree,
            unsigned char r,
```

```
        unsigned char g,
        unsigned char b,
        unsigned char a )  [static]
```

Definition at line 3027 of file lodepng.cpp.

```
3028 {
3029   int bit = 0;
3030   for(bit = 0; bit < 8; ++bit)
3031   {
3032     int i = 8 * ((r >> bit) & 1) + 4 * ((g >> bit) & 1) + 2 * ((b >> bit) & 1)
3033     if(!tree->children[i]) return -1;
3034     else tree = tree->children[i];
3035   }
3036   return tree ? tree->index : -1;
3037 }
```

Here is the caller graph for this function:



4.1.3.31 color_tree_has()

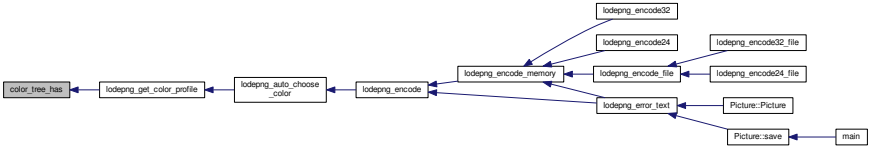
```
static int color_tree_has (
            ColorTree * tree,
            unsigned char r,
            unsigned char g,
            unsigned char b,
            unsigned char a )  [static]
```

Definition at line 3040 of file lodepng.cpp.

```
3041 {
3042   return color_tree_get(tree, r, g, b, a) >= 0;
3043 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



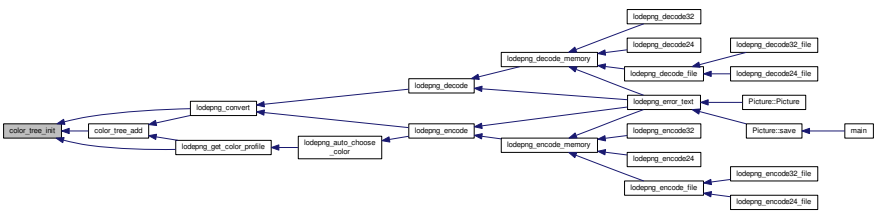
4.1.3.32 color_tree_init()

```
static void color_tree_init (
            ColorTree * tree )  [static]
```

Definition at line 3006 of file lodepng.cpp.

```
3007 {
3008   int i;
3009   for(i = 0; i != 16; ++i) tree->children[i] = 0;
3010   tree->index = -1;
3011 }
```

Here is the caller graph for this function:



4.1.3.33 countZeros()

```
static unsigned countZeros (
            const unsigned char * data,
            size_t size,
            size_t pos )  [static]
```

Definition at line 1442 of file lodepng.cpp.

```
1443 {
1444   const unsigned char* start = data + pos;
1445   const unsigned char* end = start + MAX_SUPPORTED_DEFLATE_LENGTH;
1446   if(end > data + size) end = data + size;
1447   data = start;
1448   while(data != end && *data == 0) ++data;
1449   /*subtracting two addresses returned as 32-bit number (max value is MAX_SUPPO
1450   return (unsigned)(data - start);
1451 }
```

Here is the caller graph for this function:

4.1.3.34 decodeGeneric()

```
static void decodeGeneric (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            LodePNGState * state,
            const unsigned char * in,
            size_t insize )  [static]
```

Definition at line 4522 of file lodepng.cpp.

```
4525 {
4526   unsigned char IEND = 0;
4527   const unsigned char* chunk;
4528   size_t i;
4529   ucvector idat; /*the data from idat chunks*/
4530   ucvector scanlines;
4531   size_t predict;
4532   size_t numpixels;
4533   size_t outsize = 0;
4534
4535   /*for unknown chunk order*/
4536   unsigned unknown = 0;
4537 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
4538   unsigned critical_pos = 1; /*1 = after IHDR, 2 = after PLTE, 3 = after IDAT*/
4539 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
4540
4541   /*provide some proper output values if error will happen*/
4542   *out = 0;
4543
4544   state->error = lodepng_inspect(w, h, state, in, insize); /*reads header and r
```

```
      other parameters in state->info_png*/
4545  if(state->error) return;
4546
4547  numpixels = *w * *h;
4548
4549  /*multiplication overflow*/
4550  if(*h != 0 && numpixels / *h != *w) CERROR_RETURN(state->error, 92);
4551  /*multiplication overflow possible further below. Allows up to 2^31-1 pixel
4552  bytes with 16-bit RGBA, the rest is room for filter bytes.*/
4553  if(numpixels > 268435455) CERROR_RETURN(state->error, 92);
4554
4555  ucvector_init(&idat);
4556  chunk = &in[33]; /*first byte of the first chunk after the header*/
4557
4558  /*loop through the chunks, ignoring unknown chunks and stopping at IEND chunk
4559  IDAT data is put at the start of the in buffer*/
4560  while(!IEND && !state->error)
4561  {
4562    unsigned chunkLength;
4563    const unsigned char* data; /*the data in the chunk*/
4564
4565    /*error: size of the in buffer too small to contain next chunk*/
4566    if((size_t)((chunk - in) + 12) > insize || chunk < in) CERROR_BREAK(state->
      error, 30);
4567
4568    /*length of the data of the chunk, excluding the length bytes, chunk type a
4569    chunkLength = lodepng_chunk_length(chunk);
4570    /*error: chunk length larger than the max PNG chunk size*/
4571    if(chunkLength > 2147483647) CERROR_BREAK(state->error, 63);
4572
4573    if((size_t)((chunk - in) + chunkLength + 12) > insize || (chunk + chunkLeng
```

```
4574     {
4575       CERROR_BREAK(state->error, 64); /*error: size of the in buffer too small
      next chunk*/
4576     }
4577
4578     data = lodepng_chunk_data_const(chunk);
4579
4580     /*IDAT chunk, containing compressed image data*/
4581     if(lodepng_chunk_type_equals(chunk, "IDAT"))
4582     {
4583       size_t oldsize = idat.size;
4584       if(!ucvector_resize(&idat, oldsize + chunkLength))
      CERROR_BREAK(state->error, 83 /*alloc fail*/);
4585       for(i = 0; i != chunkLength; ++i) idat.data[oldsize + i] = data[i];
4586 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
4587       critical_pos = 3;
4588 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
4589     }
4590     /*IEND chunk*/
4591     else if(lodepng_chunk_type_equals(chunk, "IEND"))
4592     {
4593       IEND = 1;
4594     }
4595     /*palette chunk (PLTE)*/
4596     else if(lodepng_chunk_type_equals(chunk, "PLTE"))
4597     {
4598       state->error = readChunk_PLTE(&state->info_png.
      color, data, chunkLength);
4599       if(state->error) break;
4600 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
4601       critical_pos = 2;
```

```
4602 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
4603     }
4604     /*palette transparency chunk (tRNS)*/
4605     else if(lodepng_chunk_type_equals(chunk, "tRNS"))
4606     {
4607       state->error = readChunk_tRNS(&state->info_png.
      color, data, chunkLength);
4608       if(state->error) break;
4609     }
4610 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
4611     /*background color chunk (bKGD)*/
4612     else if(lodepng_chunk_type_equals(chunk, "bKGD"))
4613     {
4614       state->error = readChunk_bKGD(&state->info_png, data, chunkLength);
4615       if(state->error) break;
4616     }
4617     /*text chunk (tEXt)*/
4618     else if(lodepng_chunk_type_equals(chunk, "tEXt"))
4619     {
4620       if(state->decoder.read_text_chunks)
4621       {
4622         state->error = readChunk_tEXt(&state->info_png, data, chunkLength);
4623         if(state->error) break;
4624       }
4625     }
4626     /*compressed text chunk (zTXt)*/
4627     else if(lodepng_chunk_type_equals(chunk, "zTXt"))
4628     {
4629       if(state->decoder.read_text_chunks)
4630       {
4631         state->error = readChunk_zTXt(&state->info_png, &state->
```

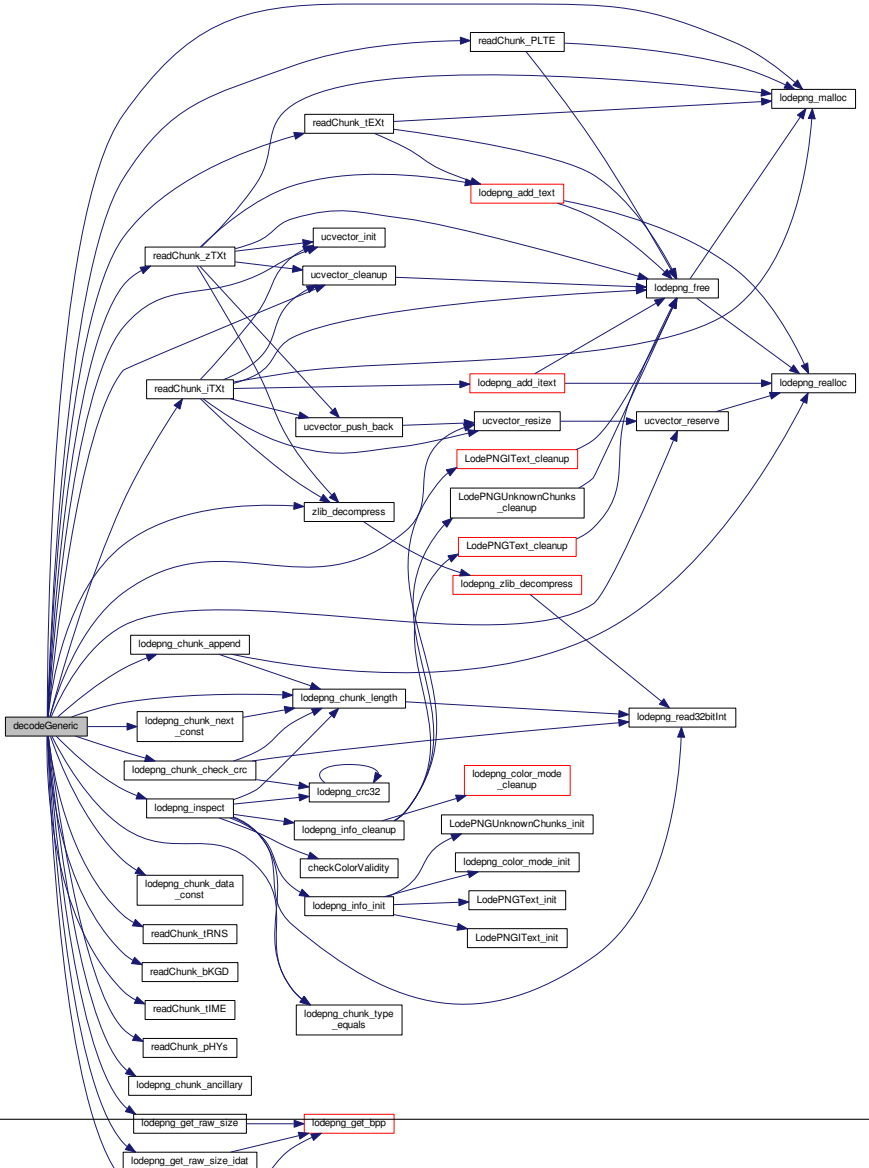
```
    decoder.zlibsettings, data, chunkLength);
4632         if(state->error) break;
4633       }
4634     }
4635     /*international text chunk (iTXt)*/
4636     else if(lodepng_chunk_type_equals(chunk, "iTXt"))
4637     {
4638       if(state->decoder.read_text_chunks)
4639       {
4640         state->error = readChunk_iTXt(&state->info_png, &state->
    decoder.zlibsettings, data, chunkLength);
4641         if(state->error) break;
4642       }
4643     }
4644     else if(lodepng_chunk_type_equals(chunk, "tIME"))
4645     {
4646       state->error = readChunk_tIME(&state->info_png, data, chunkLength);
4647       if(state->error) break;
4648     }
4649     else if(lodepng_chunk_type_equals(chunk, "pHYs"))
4650     {
4651       state->error = readChunk_pHYs(&state->info_png, data, chunkLength);
4652       if(state->error) break;
4653     }
4654 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
4655     else /*it's not an implemented chunk type, so ignore it: skip over the data
4656     {
4657       /*error: unknown critical chunk (5th bit of first byte of chunk type is 0
4658       if(!lodepng_chunk_ancillary(chunk)) CERROR_BREAK(state->
    error, 69);
4659
```

```
4660       unknown = 1;
4661 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
4662       if(state->decoder.remember_unknown_chunks)
4663       {
4664         state->error = lodepng_chunk_append(&state->
     info_png.unknown_chunks_data[critical_pos - 1],
4665                                             &state->info_png.
     unknown_chunks_size[critical_pos - 1], chunk);
4666         if(state->error) break;
4667       }
4668 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
4669     }
4670
4671     if(!state->decoder.ignore_crc && !unknown) /*check CRC if wanted, only on k
     types*/
4672     {
4673       if(lodepng_chunk_check_crc(chunk)) CERROR_BREAK(state->
     error, 57); /*invalid CRC*/
4674     }
4675
4676     if(!IEND) chunk = lodepng_chunk_next_const(chunk);
4677   }
4678
4679   ucvector_init(&scanlines);
4680   /*predict output size, to allocate exact size for output buffer to avoid more
4681   If the decompressed size does not match the prediction, the image must be cor
4682   if(state->info_png.interlace_method == 0)
4683   {
4684     /*The extra *h is added because this are the filter bytes every scanline st
4685     predict = lodepng_get_raw_size_idat(*w, *h, &state->
     info_png.color) + *h;
```

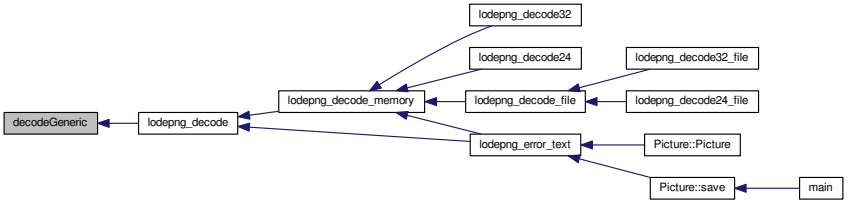
```
4686   }
4687   else
4688   {
4689     /*Adam-7 interlaced: predicted size is the sum of the 7 sub-images sizes*/
4690     const LodePNGColorMode* color = &state->info_png.
       color;
4691     predict = 0;
4692     predict += lodepng_get_raw_size_idat((*w + 7) >> 3, (*h + 7) >> 3, color) +
       *h + 7) >> 3);
4693     if(*w > 4) predict += lodepng_get_raw_size_idat((*w + 3) >> 3, (*h + 7) >>
       color) + ((*h + 7) >> 3);
4694     predict += lodepng_get_raw_size_idat((*w + 3) >> 2, (*h + 3) >> 3, color) +
       *h + 3) >> 3);
4695     if(*w > 2) predict += lodepng_get_raw_size_idat((*w + 1) >> 2, (*h + 3) >>
       color) + ((*h + 3) >> 2);
4696     predict += lodepng_get_raw_size_idat((*w + 1) >> 1, (*h + 1) >> 2, color) +
       *h + 1) >> 2);
4697     if(*w > 1) predict += lodepng_get_raw_size_idat((*w + 0) >> 1, (*h + 1) >>
       color) + ((*h + 1) >> 1);
4698     predict += lodepng_get_raw_size_idat((*w + 0), (*h + 0) >> 1, color) + ((*h
       0) >> 1);
4699   }
4700   if(!state->error && !ucvector_reserve(&scanlines, predict)) state->
       error = 83; /*alloc fail*/
4701   if(!state->error)
4702   {
4703     state->error = zlib_decompress(&scanlines.data, &scanlines.
       size, idat.data,
4704                                    idat.size, &state->decoder.
       zlibsettings);
4705     if(!state->error && scanlines.size != predict) state->error = 91; /*decompr
```

```
      doesn't match prediction*/
4706   }
4707   ucvector_cleanup(&idat);
4708
4709   if(!state->error)
4710   {
4711     outsize = lodepng_get_raw_size(*w, *h, &state->info_png.
      color);
4712     *out = (unsigned char*)lodepng_malloc(outsize);
4713     if(!*out) state->error = 83; /*alloc fail*/
4714   }
4715   if(!state->error)
4716   {
4717     for(i = 0; i < outsize; i++) (*out)[i] = 0;
4718     state->error = postProcessScanlines(*out, scanlines.
      data, *w, *h, &state->info_png);
4719   }
4720   ucvector_cleanup(&scanlines);
4721 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.35 deflate()

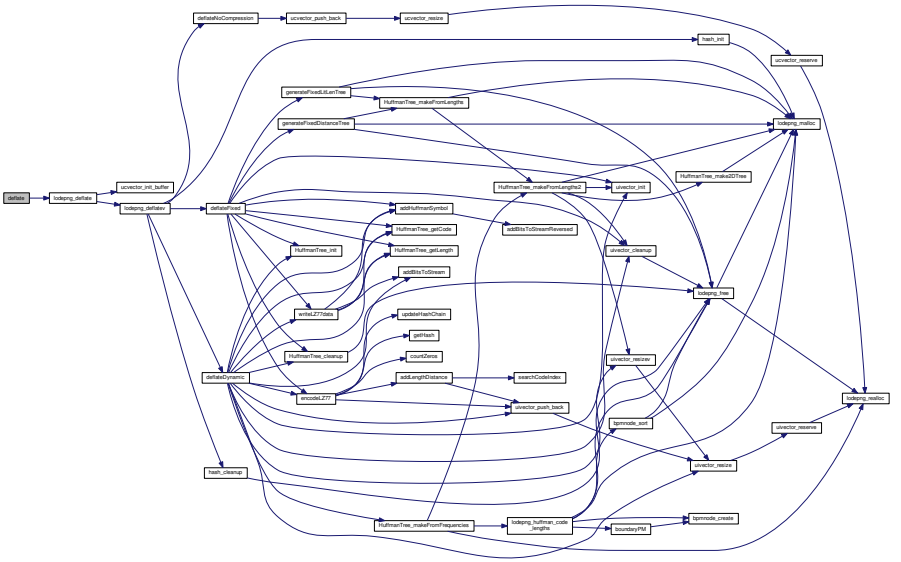
```
static unsigned deflate (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGCompressSettings * settings )  [static]
```

Definition at line 2071 of file lodepng.cpp.

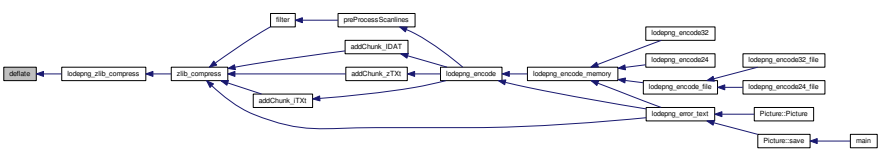
```
2074 {
2075   if(settings->custom_deflate)
2076   {
2077     return settings->custom_deflate(out, outsize, in, insize, settings);
2078   }
2079   else
2080   {
```

```
2081     return lodepng_deflate(out, outsize, in, insize, settings);
2082   }
2083 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.36 deflateDynamic()

```
static unsigned deflateDynamic (
            ucvector * out,
            size_t * bp,
            Hash * hash,
            const unsigned char * data,
            size_t datapos,
            size_t dataend,
            const LodePNGCompressSettings * settings,
            unsigned final ) [static]
```

Definition at line 1724 of file lodepng.cpp.

```
1727 {
1728   unsigned error = 0;
1729
1730   /*
1731   A block is compressed as follows: The PNG data is lz77 encoded, resulting in
1732   literal bytes and length/distance pairs. This is then huffman compressed with
1733   two huffman trees. One huffman tree is used for the lit and len values ("ll").
1734   another huffman tree is used for the dist values ("d"). These two trees are
1735   stored using their code lengths, and to compress even more these code lengths
1736   are also run-length encoded and huffman compressed. This gives a huffman tree
1737   of code lengths "cl". The code lenghts used to describe this third tree are
1738   the code length code lengths ("clcl").
1739   */
1740
1741   /*The lz77 encoded data, represented with integers since there will also be l
       it*/
```

```
1742   uivector lz77_encoded;
1743   HuffmanTree tree_ll; /*tree for lit,len values*/
1744   HuffmanTree tree_d; /*tree for distance codes*/
1745   HuffmanTree tree_cl; /*tree for encoding the code lengths representing tree_l
1746   uivector frequencies_ll; /*frequency of lit,len codes*/
1747   uivector frequencies_d; /*frequency of dist codes*/
1748   uivector frequencies_cl; /*frequency of code length codes*/
1749   uivector bitlen_lld; /*lit,len,dist code lenghts (int bits), literally (witho
1750   uivector bitlen_lld_e; /*bitlen_lld encoded with repeat codes (this is a rude
      compression)*/
1751   /*bitlen_cl is the code length code lengths ("clcl"). The bit lengths of code
1752   (these are written as is in the file, it would be crazy to compress these usi
1753   tree that needs to be represented by yet another set of code lengths)*/
1754   uivector bitlen_cl;
1755   size_t datasize = dataend - datapos;
1756
1757   /*
1758   Due to the huffman compression of huffman tree representations ("two levels")
1759   bitlen_lld is to tree_cl what data is to tree_ll and tree_d.
1760   bitlen_lld_e is to bitlen_lld what lz77_encoded is to data.
1761   bitlen_cl is to bitlen_lld_e what bitlen_lld is to lz77_encoded.
1762   */
1763
1764   unsigned BFINAL = final;
1765   size_t numcodes_ll, numcodes_d, i;
1766   unsigned HLIT, HDIST, HCLEN;
1767
1768   uivector_init(&lz77_encoded);
1769   HuffmanTree_init(&tree_ll);
1770   HuffmanTree_init(&tree_d);
1771   HuffmanTree_init(&tree_cl);
```

```
1772   uivector_init(&frequencies_ll);
1773   uivector_init(&frequencies_d);
1774   uivector_init(&frequencies_cl);
1775   uivector_init(&bitlen_lld);
1776   uivector_init(&bitlen_lld_e);
1777   uivector_init(&bitlen_cl);
1778
1779   /*This while loop never loops due to a break at the end, it is here to
1780   allow breaking out of it to the cleanup phase on error conditions.*/
1781   while(!error)
1782   {
1783     if(settings->use_lz77)
1784     {
1785       error = encodeLZ77(&lz77_encoded, hash, data, datapos, dataend, settings-
      windowsize,
1786                          settings->minmatch, settings->nicematch, settings->
      lazymatching);
1787       if(error) break;
1788     }
1789     else
1790     {
1791       if(!uivector_resize(&lz77_encoded, datasize)) ERROR_BREAK(83 /*alloc fail
      );
1792       for(i = datapos; i < dataend; ++i) lz77_encoded.data[i - datapos] = data[
       will be Huffman compressed*/
1793     }
1794
1795     if(!uivector_resizev(&frequencies_ll, 286, 0)) ERROR_BREAK(83 /*alloc fail*
      );
1796     if(!uivector_resizev(&frequencies_d, 30, 0)) ERROR_BREAK(83 /*alloc fail*/)
1797
```

```
1798     /*Count the frequencies of lit, len and dist codes*/
1799     for(i = 0; i != lz77_encoded.size; ++i)
1800     {
1801       unsigned symbol = lz77_encoded.data[i];
1802       ++frequencies_ll.data[symbol];
1803       if(symbol > 256)
1804       {
1805         unsigned dist = lz77_encoded.data[i + 2];
1806         ++frequencies_d.data[dist];
1807         i += 3;
1808       }
1809     }
1810     frequencies_ll.data[256] = 1; /*there will be exactly 1 end code, at the en
1811
1812     /*Make both huffman trees, one for the lit and len codes, one for the dist
1813     error = HuffmanTree_makeFromFrequencies(&tree_ll, frequencies_ll.
     data, 257, frequencies_ll.size, 15);
1814     if(error) break;
1815     /*2, not 1, is chosen for mincodes: some buggy PNG decoders require at leas
     */
1816     error = HuffmanTree_makeFromFrequencies(&tree_d, frequencies_d.
     data, 2, frequencies_d.size, 15);
1817     if(error) break;
1818
1819     numcodes_ll = tree_ll.numcodes; if(numcodes_ll > 286) numcodes_ll = 286;
1820     numcodes_d = tree_d.numcodes; if(numcodes_d > 30) numcodes_d = 30;
1821     /*store the code lengths of both generated trees in bitlen_lld*/
1822     for(i = 0; i != numcodes_ll; ++i) uivector_push_back(&bitlen_lld,
     HuffmanTree_getLength(&tree_ll, (unsigned)i));
1823     for(i = 0; i != numcodes_d; ++i) uivector_push_back(&bitlen_lld,
     HuffmanTree_getLength(&tree_d, (unsigned)i));
```

```
1824
1825     /*run-length compress bitlen_ldd into bitlen_lld_e by using repeat codes 16
1826     17 (3-10 zeroes), 18 (11-138 zeroes)*/
1827     for(i = 0; i != (unsigned)bitlen_lld.size; ++i)
1828     {
1829       unsigned j = 0; /*amount of repititions*/
1830       while(i + j + 1 < (unsigned)bitlen_lld.size && bitlen_lld.data[i + j + 1]
      data[i]) ++j;
1831
1832       if(bitlen_lld.data[i] == 0 && j >= 2) /*repeat code for zeroes*/
1833       {
1834         ++j; /*include the first zero*/
1835         if(j <= 10) /*repeat code 17 supports max 10 zeroes*/
1836         {
1837           uivector_push_back(&bitlen_lld_e, 17);
1838           uivector_push_back(&bitlen_lld_e, j - 3);
1839         }
1840         else /*repeat code 18 supports max 138 zeroes*/
1841         {
1842           if(j > 138) j = 138;
1843           uivector_push_back(&bitlen_lld_e, 18);
1844           uivector_push_back(&bitlen_lld_e, j - 11);
1845         }
1846         i += (j - 1);
1847       }
1848       else if(j >= 3) /*repeat code for value other than zero*/
1849       {
1850         size_t k;
1851         unsigned num = j / 6, rest = j % 6;
1852         uivector_push_back(&bitlen_lld_e, bitlen_lld.data[i]);
1853         for(k = 0; k < num; ++k)
```

```
1854         {
1855           uivector_push_back(&bitlen_lld_e, 16);
1856           uivector_push_back(&bitlen_lld_e, 6 - 3);
1857         }
1858         if(rest >= 3)
1859         {
1860           uivector_push_back(&bitlen_lld_e, 16);
1861           uivector_push_back(&bitlen_lld_e, rest - 3);
1862         }
1863         else j -= rest;
1864         i += j;
1865       }
1866       else /*too short to benefit from repeat code*/
1867       {
1868         uivector_push_back(&bitlen_lld_e, bitlen_lld.data[i]);
1869       }
1870     }
1871
1872     /*generate tree_cl, the huffmantree of huffmantrees*/
1873
1874     if(!uivector_resizev(&frequencies_cl, NUM_CODE_LENGTH_CODES, 0))
     ERROR_BREAK(83 /*alloc fail*/);
1875     for(i = 0; i != bitlen_lld_e.size; ++i)
1876     {
1877       ++frequencies_cl.data[bitlen_lld_e.data[i]];
1878       /*after a repeat code come the bits that specify the number of repetition
1879       those don't need to be in the frequencies_cl calculation*/
1880       if(bitlen_lld_e.data[i] >= 16) ++i;
1881     }
1882
1883     error = HuffmanTree_makeFromFrequencies(&tree_cl, frequencies_cl.
```

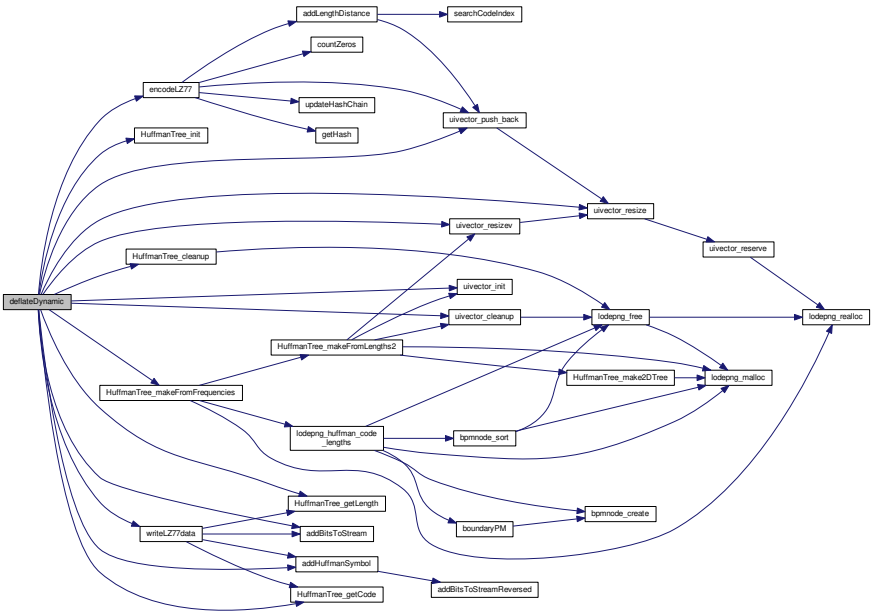
```
      data,
1884                                                frequencies_cl.size, frequencies_cl
      size, 7);
1885    if(error) break;
1886
1887    if(!uivector_resize(&bitlen_cl, tree_cl.numcodes))
      ERROR_BREAK(83 /*alloc fail*/);
1888    for(i = 0; i != tree_cl.numcodes; ++i)
1889    {
1890      /*lenghts of code length tree is in the order as specified by deflate*/
1891      bitlen_cl.data[i] = HuffmanTree_getLength(&tree_cl,
      CLCL_ORDER[i]);
1892    }
1893    while(bitlen_cl.data[bitlen_cl.size - 1] == 0 && bitlen_cl.size > 4)
1894    {
1895      /*remove zeros at the end, but minimum size must be 4*/
1896      if(!uivector_resize(&bitlen_cl, bitlen_cl.size - 1))
      ERROR_BREAK(83 /*alloc fail*/);
1897    }
1898    if(error) break;
1899
1900    /*
1901    Write everything into the output
1902
1903    After the BFINAL and BTYPE, the dynamic block consists out of the following
1904    - 5 bits HLIT, 5 bits HDIST, 4 bits HCLEN
1905    - (HCLEN+4)*3 bits code lengths of code length alphabet
1906    - HLIT + 257 code lenghts of lit/length alphabet (encoded using the code le
1907      alphabet, + possible repetition codes 16, 17, 18)
1908    - HDIST + 1 code lengths of distance alphabet (encoded using the code lengt
1909      alphabet, + possible repetition codes 16, 17, 18)
```

```
1910     - compressed data
1911     - 256 (end code)
1912     */
1913
1914     /*Write block type*/
1915     addBitToStream(bp, out, BFINAL);
1916     addBitToStream(bp, out, 0); /*first bit of BTYPE "dynamic"*/
1917     addBitToStream(bp, out, 1); /*second bit of BTYPE "dynamic"*/
1918
1919     /*write the HLIT, HDIST and HCLEN values*/
1920     HLIT = (unsigned)(numcodes_ll - 257);
1921     HDIST = (unsigned)(numcodes_d - 1);
1922     HCLEN = (unsigned)bitlen_cl.size - 4;
1923     /*trim zeroes for HCLEN. HLIT and HDIST were already trimmed at tree creati
1924     while(!bitlen_cl.data[HCLEN + 4 - 1] && HCLEN > 0) --HCLEN;
1925     addBitsToStream(bp, out, HLIT, 5);
1926     addBitsToStream(bp, out, HDIST, 5);
1927     addBitsToStream(bp, out, HCLEN, 4);
1928
1929     /*write the code lenghts of the code length alphabet*/
1930     for(i = 0; i != HCLEN + 4; ++i) addBitsToStream(bp, out, bitlen_cl.
     data[i], 3);
1931
1932     /*write the lenghts of the lit/len AND the dist alphabet*/
1933     for(i = 0; i != bitlen_lld_e.size; ++i)
1934     {
1935       addHuffmanSymbol(bp, out, HuffmanTree_getCode(&tree_cl,
     bitlen_lld_e.data[i]),
1936                        HuffmanTree_getLength(&tree_cl, bitlen_lld_e.
     data[i]));
1937       /*extra bits of repeat codes*/
```

```
1938       if(bitlen_lld_e.data[i] == 16) addBitsToStream(bp, out, bitlen_lld_e.
     data[++i], 2);
1939       else if(bitlen_lld_e.data[i] == 17) addBitsToStream(bp, out, bitlen_lld_e
     data[++i], 3);
1940       else if(bitlen_lld_e.data[i] == 18) addBitsToStream(bp, out, bitlen_lld_e
     data[++i], 7);
1941     }
1942
1943     /*write the compressed data symbols*/
1944     writeLZ77data(bp, out, &lz77_encoded, &tree_ll, &tree_d);
1945     /*error: the length of the end code 256 must be larger than 0*/
1946     if(HuffmanTree_getLength(&tree_ll, 256) == 0)
     ERROR_BREAK(64);
1947
1948     /*write the end code*/
1949     addHuffmanSymbol(bp, out, HuffmanTree_getCode(&tree_ll, 256),
     HuffmanTree_getLength(&tree_ll, 256));
1950
1951     break; /*end of error-while*/
1952   }
1953
1954   /*cleanup*/
1955   uivector_cleanup(&lz77_encoded);
1956   HuffmanTree_cleanup(&tree_ll);
1957   HuffmanTree_cleanup(&tree_d);
1958   HuffmanTree_cleanup(&tree_cl);
1959   uivector_cleanup(&frequencies_ll);
1960   uivector_cleanup(&frequencies_d);
1961   uivector_cleanup(&frequencies_cl);
1962   uivector_cleanup(&bitlen_lld_e);
1963   uivector_cleanup(&bitlen_lld);
```

```
1964   uivector_cleanup(&bitlen_cl);
1965
1966   return error;
1967 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.3.37 deflateFixed()

```
static unsigned deflateFixed (
            ucvector * out,
            size_t * bp,
            Hash * hash,
            const unsigned char * data,
            size_t datapos,
            size_t dataend,
            const LodePNGCompressSettings * settings,
            unsigned final )  [static]
```

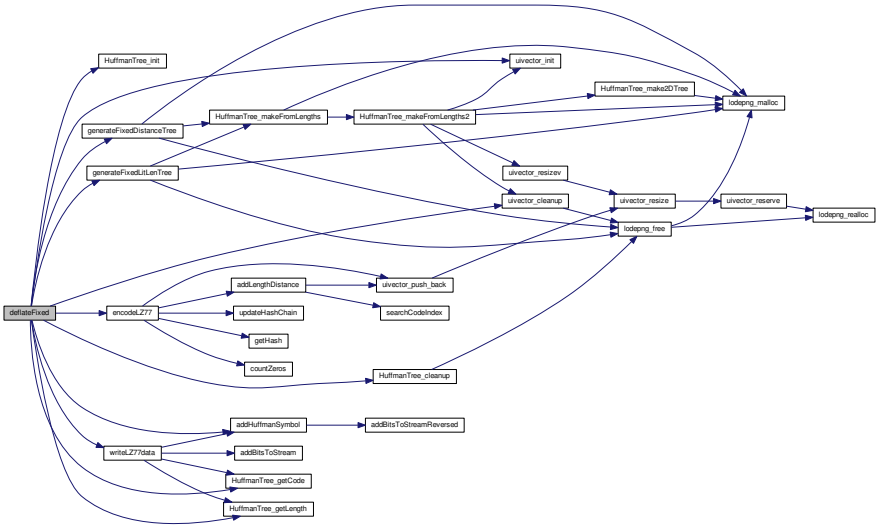
Definition at line 1969 of file lodepng.cpp.

```
1973 {
1974   HuffmanTree tree_ll; /*tree for literal values and length codes*/
1975   HuffmanTree tree_d; /*tree for distance codes*/
1976
1977   unsigned BFINAL = final;
1978   unsigned error = 0;
1979   size_t i;
1980
```

```
1981   HuffmanTree_init(&tree_ll);
1982   HuffmanTree_init(&tree_d);
1983
1984   generateFixedLitLenTree(&tree_ll);
1985   generateFixedDistanceTree(&tree_d);
1986
1987   addBitToStream(bp, out, BFINAL);
1988   addBitToStream(bp, out, 1); /*first bit of BTYPE*/
1989   addBitToStream(bp, out, 0); /*second bit of BTYPE*/
1990
1991   if(settings->use_lz77) /*LZ77 encoded*/
1992   {
1993     uivector lz77_encoded;
1994     uivector_init(&lz77_encoded);
1995     error = encodeLZ77(&lz77_encoded, hash, data, datapos, dataend, settings->
      windowsize,
1996                        settings->minmatch, settings->nicematch, settings->
      lazymatching);
1997     if(!error) writeLZ77data(bp, out, &lz77_encoded, &tree_ll, &tree_d);
1998     uivector_cleanup(&lz77_encoded);
1999   }
2000   else /*no LZ77, but still will be Huffman compressed*/
2001   {
2002     for(i = datapos; i < dataend; ++i)
2003     {
2004       addHuffmanSymbol(bp, out, HuffmanTree_getCode(&tree_ll, data[i]),
      HuffmanTree_getLength(&tree_ll, data[i]));
2005     }
2006   }
2007   /*add END code*/
2008   if(!error) addHuffmanSymbol(bp, out, HuffmanTree_getCode(&tree_ll, 256
```

```
    ), HuffmanTree_getLength(&tree_ll, 256));
2009
2010   /*cleanup*/
2011   HuffmanTree_cleanup(&tree_ll);
2012   HuffmanTree_cleanup(&tree_d);
2013
2014   return error;
2015 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.3.38 deflateNoCompression()

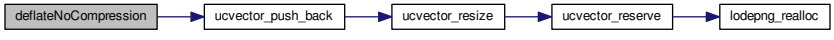
```
static unsigned deflateNoCompression (
            ucvector * out,
            const unsigned char * data,
            size_t datasize )  [static]
```

Definition at line 1653 of file lodepng.cpp.

```
1654 {
1655   /*non compressed deflate block data: 1 bit BFINAL,2 bits BTYPE,(5 bits): it j
1656   2 bytes LEN, 2 bytes NLEN, LEN bytes literal DATA*/
1657
1658   size_t i, j, numdeflateblocks = (datasize + 65534) / 65535;
1659   unsigned datapos = 0;
1660   for(i = 0; i != numdeflateblocks; ++i)
1661   {
1662     unsigned BFINAL, BTYPE, LEN, NLEN;
1663     unsigned char firstbyte;
1664
1665     BFINAL = (i == numdeflateblocks - 1);
```

```
1666     BTYPE = 0;
1667
1668     firstbyte = (unsigned char)(BFINAL + ((BTYPE & 1) << 1) + ((BTYPE & 2) << 1
1669     ucvector_push_back(out, firstbyte);
1670
1671     LEN = 65535;
1672     if(datasize - datapos < 65535) LEN = (unsigned)datasize - datapos;
1673     NLEN = 65535 - LEN;
1674
1675     ucvector_push_back(out, (unsigned char)(LEN & 255));
1676     ucvector_push_back(out, (unsigned char)(LEN >> 8));
1677     ucvector_push_back(out, (unsigned char)(NLEN & 255));
1678     ucvector_push_back(out, (unsigned char)(NLEN >> 8));
1679
1680     /*Decompressed data*/
1681     for(j = 0; j < 65535 && datapos < datasize; ++j)
1682     {
1683       ucvector_push_back(out, data[datapos++]);
1684     }
1685   }
1686
1687   return 0;
1688 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.3.39 encodeLZ77()

```
static unsigned encodeLZ77 (
            uivector * out,
            Hash * hash,
            const unsigned char * in,
            size_t inpos,
            size_t insize,
            unsigned windowsize,
            unsigned minmatch,
            unsigned nicematch,
            unsigned lazymatching )  [static]
```

Definition at line 1474 of file lodepng.cpp.

```
1477 {
1478   size_t pos;
1479   unsigned i, error = 0;
1480   /*for large window lengths, assume the user wants no compression loss. Otherw
       speedup.*/
1481   unsigned maxchainlength = windowsize >= 8192 ? windowsize : windowsize / 8;
1482   unsigned maxlazymatch = windowsize >= 8192 ? MAX_SUPPORTED_DEFLATE_LENGTH :
```

```
      64;
1483
1484   unsigned usezeros = 1; /*not sure if setting it to false for windowsize < 819
1485   unsigned numzeros = 0;
1486
1487   unsigned offset; /*the offset represents the distance in LZ77 terminology*/
1488   unsigned length;
1489   unsigned lazy = 0;
1490   unsigned lazylength = 0, lazyoffset = 0;
1491   unsigned hashval;
1492   unsigned current_offset, current_length;
1493   unsigned prev_offset;
1494   const unsigned char *lastptr, *foreptr, *backptr;
1495   unsigned hashpos;
1496
1497   if(windowsize == 0 || windowsize > 32768) return 60; /*error: windowsize smal
1498   if((windowsize & (windowsize - 1)) != 0) return 90; /*error: must be power of
1499
1500   if(nicematch > MAX_SUPPORTED_DEFLATE_LENGTH) nicematch =
      MAX_SUPPORTED_DEFLATE_LENGTH;
1501
1502   for(pos = inpos; pos < insize; ++pos)
1503   {
1504     size_t wpos = pos & (windowsize - 1); /*position for in 'circular' hash buf
1505     unsigned chainlength = 0;
1506
1507     hashval = getHash(in, insize, pos);
1508
1509     if(usezeros && hashval == 0)
1510     {
1511       if(numzeros == 0) numzeros = countZeros(in, insize, pos);
```

```
1512        else if(pos + numzeros > insize || in[pos + numzeros - 1] != 0) --numzero
1513      }
1514      else
1515      {
1516        numzeros = 0;
1517      }
1518
1519      updateHashChain(hash, wpos, hashval, numzeros);
1520
1521      /*the length and offset found for the current position*/
1522      length = 0;
1523      offset = 0;
1524
1525      hashpos = hash->chain[wpos];
1526
1527      lastptr = &in[insize < pos + MAX_SUPPORTED_DEFLATE_LENGTH ? insize : pos +
    MAX_SUPPORTED_DEFLATE_LENGTH];
1528
1529      /*search for the longest string*/
1530      prev_offset = 0;
1531      for(;;)
1532      {
1533        if(chainlength++ >= maxchainlength) break;
1534        current_offset = hashpos <= wpos ? wpos - hashpos : wpos - hashpos + wind
1535
1536        if(current_offset < prev_offset) break; /*stop when went completely aroun
1537        prev_offset = current_offset;
1538        if(current_offset > 0)
1539        {
1540          /*test the next characters*/
1541          foreptr = &in[pos];
```

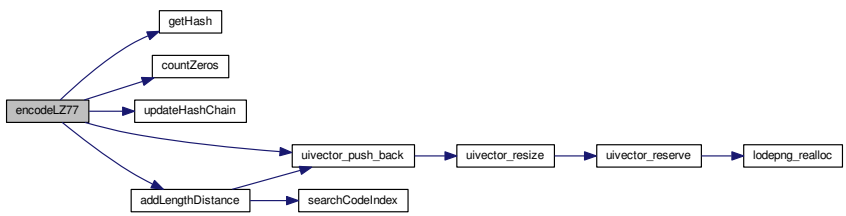
```
1542         backptr = &in[pos - current_offset];
1543
1544         /*common case in PNGs is lots of zeros. Quickly skip over them as a spe
1545         if(numzeros >= 3)
1546         {
1547           unsigned skip = hash->zeros[hashpos];
1548           if(skip > numzeros) skip = numzeros;
1549           backptr += skip;
1550           foreptr += skip;
1551         }
1552
1553         while(foreptr != lastptr && *backptr == *foreptr) /*maximum supported l
     length*/
1554         {
1555           ++backptr;
1556           ++foreptr;
1557         }
1558         current_length = (unsigned)(foreptr - &in[pos]);
1559
1560         if(current_length > length)
1561         {
1562           length = current_length; /*the longest length*/
1563           offset = current_offset; /*the offset that is related to this longest
1564           /*jump out once a length of max length is found (speed gain). This al
1565           out if length is MAX_SUPPORTED_DEFLATE_LENGTH*/
1566           if(current_length >= nicematch) break;
1567         }
1568       }
1569
1570       if(hashpos == hash->chain[hashpos]) break;
1571
```

```
1572       if(numzeros >= 3 && length > numzeros)
1573       {
1574         hashpos = hash->chainz[hashpos];
1575         if(hash->zeros[hashpos] != numzeros) break;
1576       }
1577       else
1578       {
1579         hashpos = hash->chain[hashpos];
1580         /*outdated hash value, happens if particular value was not encountered
1581         if(hash->val[hashpos] != (int)hashval) break;
1582       }
1583     }
1584
1585     if(lazymatching)
1586     {
1587       if(!lazy && length >= 3 && length <= maxlazymatch && length <
     MAX_SUPPORTED_DEFLATE_LENGTH)
1588       {
1589         lazy = 1;
1590         lazylength = length;
1591         lazyoffset = offset;
1592         continue; /*try the next byte*/
1593       }
1594       if(lazy)
1595       {
1596         lazy = 0;
1597         if(pos == 0) ERROR_BREAK(81);
1598         if(length > lazylength + 1)
1599         {
1600           /*push the previous character as literal*/
1601           if(!uivector_push_back(out, in[pos - 1]))
```

```
      ERROR_BREAK(83 /*alloc fail*/);
1602         }
1603         else
1604         {
1605           length = lazylength;
1606           offset = lazyoffset;
1607           hash->head[hashval] = -1; /*the same hashchain update will be done, t
       alteration*/
1608           hash->headz[numzeros] = -1; /*idem*/
1609           --pos;
1610         }
1611       }
1612     }
1613     if(length >= 3 && offset > windowsize) ERROR_BREAK(86 /*too big (or overflo
       offset*/);
1614
1615     /*encode it as length/distance pair or literal value*/
1616     if(length < 3) /*only lengths of 3 or higher are supported as length/distan
1617     {
1618       if(!uivector_push_back(out, in[pos])) ERROR_BREAK(83 /*alloc fail*/);
1619     }
1620     else if(length < minmatch || (length == 3 && offset > 4096))
1621     {
1622       /*compensate for the fact that longer offsets have more extra bits, a
1623       length of only 3 may be not worth it then*/
1624       if(!uivector_push_back(out, in[pos])) ERROR_BREAK(83 /*alloc fail*/);
1625     }
1626     else
1627     {
1628       addLengthDistance(out, length, offset);
1629       for(i = 1; i < length; ++i)
```

```
1630       {
1631         ++pos;
1632         wpos = pos & (windowsize - 1);
1633         hashval = getHash(in, insize, pos);
1634         if(usezeros && hashval == 0)
1635         {
1636           if(numzeros == 0) numzeros = countZeros(in, insize, pos);
1637           else if(pos + numzeros > insize || in[pos + numzeros - 1] != 0) --num
1638         }
1639         else
1640         {
1641           numzeros = 0;
1642         }
1643         updateHashChain(hash, wpos, hashval, numzeros);
1644       }
1645     }
1646   } /*end of the loop through each character of input*/
1647
1648   return error;
1649 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.3.40 filter()

```
static unsigned filter (
            unsigned char * out,
            const unsigned char * in,
            unsigned w,
            unsigned h,
            const LodePNGColorMode * info,
            const LodePNGEncoderSettings * settings )  [static]
```

Definition at line 5209 of file lodepng.cpp.

```
5211 {
5212   /*
5213   For PNG filter method 0
5214   out must be a buffer with as size: h + (w * h * bpp + 7) / 8, because there a
5215   the scanlines with 1 extra byte per scanline
5216   */
5217
5218   unsigned bpp = lodepng_get_bpp(info);
5219   /*the width of a scanline in bytes, not including the filter type*/
5220   size_t linebytes = (w * bpp + 7) / 8;
```

```
5221   /*bytewidth is used for filtering, is 1 when bpp < 8, number of bytes per pix
5222   size_t bytewidth = (bpp + 7) / 8;
5223   const unsigned char* prevline = 0;
5224   unsigned x, y;
5225   unsigned error = 0;
5226   LodePNGFilterStrategy strategy = settings->
       filter_strategy;
5227
5228   /*
5229   There is a heuristic called the minimum sum of absolute differences heuristic
       standard:
5230    *  If the image type is Palette, or the bit depth is smaller than 8, then do
5231       use fixed filtering, with the filter None).
5232    * (The other case) If the image type is Grayscale or RGB (with or without Al
5233      not smaller than 8, then use adaptive filtering heuristic as follows: inde
       apply
5234      all five filters and select the filter that produces the smallest sum of a
5235   This heuristic is used if filter strategy is LFS_MINSUM and filter_palette_ze
5236
5237   If filter_palette_zero is true and filter_strategy is not LFS_MINSUM, the abo
5238   but for "the other case", whatever strategy filter_strategy is set to instead
5239   heuristic is used.
5240   */
5241   if(settings->filter_palette_zero &&
5242      (info->colortype == LCT_PALETTE || info->bitdepth < 8)) strategy =
       LFS_ZERO;
5243
5244   if(bpp == 0) return 31; /*error: invalid color type*/
5245
5246   if(strategy == LFS_ZERO)
5247   {
```

```
5248     for(y = 0; y != h; ++y)
5249     {
5250       size_t outindex = (1 + linebytes) * y; /*the extra filterbyte added to ea
5251       size_t inindex = linebytes * y;
5252       out[outindex] = 0; /*filter type byte*/
5253       filterScanline(&out[outindex + 1], &in[inindex], prevline, linebytes, byt
5254       prevline = &in[inindex];
5255     }
5256   }
5257   else if(strategy == LFS_MINSUM)
5258   {
5259     /*adaptive filtering*/
5260     size_t sum[5];
5261     unsigned char* attempt[5]; /*five filtering attempts, one for each filter t
5262     size_t smallest = 0;
5263     unsigned char type, bestType = 0;
5264
5265     for(type = 0; type != 5; ++type)
5266     {
5267       attempt[type] = (unsigned char*)lodepng_malloc(linebytes);
5268       if(!attempt[type]) return 83; /*alloc fail*/
5269     }
5270
5271     if(!error)
5272     {
5273       for(y = 0; y != h; ++y)
5274       {
5275         /*try the 5 filter types*/
5276         for(type = 0; type != 5; ++type)
5277         {
5278           filterScanline(attempt[type], &in[y * linebytes], prevline, linebytes
```

```
      type);
5279
5280          /*calculate the sum of the result*/
5281          sum[type] = 0;
5282          if(type == 0)
5283          {
5284            for(x = 0; x != linebytes; ++x) sum[type] += (unsigned char)(attemp
5285          }
5286          else
5287          {
5288            for(x = 0; x != linebytes; ++x)
5289            {
5290              /*For differences, each byte should be treated as signed, values
5291              (converted to signed char). Filtertype 0 isn't a difference thoug
5292              This means filtertype 0 is almost never chosen, but that is justi
5293              unsigned char s = attempt[type][x];
5294              sum[type] += s < 128 ? s : (255U - s);
5295            }
5296          }
5297
5298          /*check if this is smallest sum (or if type == 0 it's the first case
      */
5299          if(type == 0 || sum[type] < smallest)
5300          {
5301            bestType = type;
5302            smallest = sum[type];
5303          }
5304        }
5305
5306        prevline = &in[y * linebytes];
5307
```

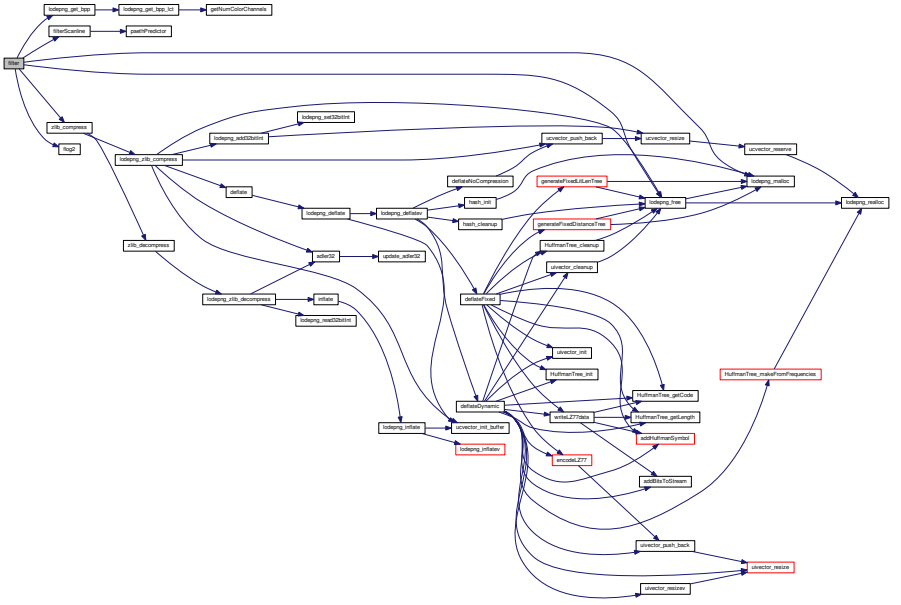
```
5308         /*now fill the out values*/
5309         out[y * (linebytes + 1)] = bestType; /*the first byte of a scanline wil
5310         for(x = 0; x != linebytes; ++x) out[y * (linebytes + 1) + 1 + x] = atte
5311       }
5312     }
5313
5314     for(type = 0; type != 5; ++type) lodepng_free(attempt[type]);
5315   }
5316   else if(strategy == LFS_ENTROPY)
5317   {
5318     float sum[5];
5319     unsigned char* attempt[5]; /*five filtering attempts, one for each filter t
5320     float smallest = 0;
5321     unsigned type, bestType = 0;
5322     unsigned count[256];
5323
5324     for(type = 0; type != 5; ++type)
5325     {
5326       attempt[type] = (unsigned char*)lodepng_malloc(linebytes);
5327       if(!attempt[type]) return 83; /*alloc fail*/
5328     }
5329
5330     for(y = 0; y != h; ++y)
5331     {
5332       /*try the 5 filter types*/
5333       for(type = 0; type != 5; ++type)
5334       {
5335         filterScanline(attempt[type], &in[y * linebytes], prevline, linebytes,
     type);
5336         for(x = 0; x != 256; ++x) count[x] = 0;
5337         for(x = 0; x != linebytes; ++x) ++count[attempt[type][x]];
```

```
5338          ++count[type]; /*the filter type itself is part of the scanline*/
5339          sum[type] = 0;
5340          for(x = 0; x != 256; ++x)
5341          {
5342            float p = count[x] / (float)(linebytes + 1);
5343            sum[type] += count[x] == 0 ? 0 : flog2(1 / p) * p;
5344          }
5345          /*check if this is smallest sum (or if type == 0 it's the first case so
5346          if(type == 0 || sum[type] < smallest)
5347          {
5348            bestType = type;
5349            smallest = sum[type];
5350          }
5351        }
5352
5353        prevline = &in[y * linebytes];
5354
5355        /*now fill the out values*/
5356        out[y * (linebytes + 1)] = bestType; /*the first byte of a scanline will
5357        for(x = 0; x != linebytes; ++x) out[y * (linebytes + 1) + 1 + x] = attemp
5358      }
5359
5360      for(type = 0; type != 5; ++type) lodepng_free(attempt[type]);
5361    }
5362    else if(strategy == LFS_PREDEFINED)
5363    {
5364      for(y = 0; y != h; ++y)
5365      {
5366        size_t outindex = (1 + linebytes) * y; /*the extra filterbyte added to ea
5367        size_t inindex = linebytes * y;
5368        unsigned char type = settings->predefined_filters[y];
```

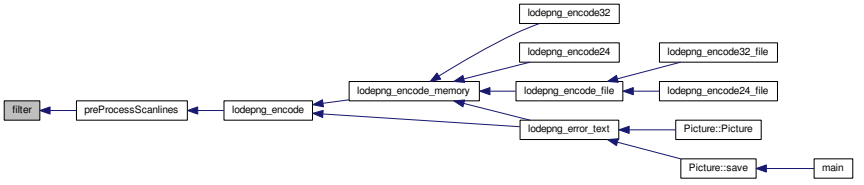
```
5369       out[outindex] = type; /*filter type byte*/
5370       filterScanline(&out[outindex + 1], &in[inindex], prevline, linebytes, byt
5371       prevline = &in[inindex];
5372     }
5373   }
5374   else if(strategy == LFS_BRUTE_FORCE)
5375   {
5376     /*brute force filter chooser.
5377     deflate the scanline after every filter attempt to see which one deflates b
5378     This is very slow and gives only slightly smaller, sometimes even larger, r
5379     size_t size[5];
5380     unsigned char* attempt[5]; /*five filtering attempts, one for each filter t
5381     size_t smallest = 0;
5382     unsigned type = 0, bestType = 0;
5383     unsigned char* dummy;
5384     LodePNGCompressSettings zlibsettings = settings->
     zlibsettings;
5385     /*use fixed tree on the attempts so that the tree is not adapted to the fil
5386     to simulate the true case where the tree is the same for the whole image. S
5387     better result with dynamic tree anyway. Using the fixed tree sometimes give
5388     cases better compression. It does make this a bit less slow, so it's worth
5389     zlibsettings.btype = 1;
5390     /*a custom encoder likely doesn't read the btype setting and is optimized f
5391     images only, so disable it*/
5392     zlibsettings.custom_zlib = 0;
5393     zlibsettings.custom_deflate = 0;
5394     for(type = 0; type != 5; ++type)
5395     {
5396       attempt[type] = (unsigned char*)lodepng_malloc(linebytes);
5397       if(!attempt[type]) return 83; /*alloc fail*/
5398     }
```

```
5399     for(y = 0; y != h; ++y) /*try the 5 filter types*/
5400     {
5401       for(type = 0; type != 5; ++type)
5402       {
5403         unsigned testsize = linebytes;
5404         /*if(testsize > 8) testsize /= 8;*/ /*it already works good enough by t
5405
5406         filterScanline(attempt[type], &in[y * linebytes], prevline, linebytes,
     type);
5407         size[type] = 0;
5408         dummy = 0;
5409         zlib_compress(&dummy, &size[type], attempt[type], testsize, &zlibsettin
5410         lodepng_free(dummy);
5411         /*check if this is smallest size (or if type == 0 it's the first case s
5412         if(type == 0 || size[type] < smallest)
5413         {
5414           bestType = type;
5415           smallest = size[type];
5416         }
5417       }
5418       prevline = &in[y * linebytes];
5419       out[y * (linebytes + 1)] = bestType; /*the first byte of a scanline will
5420       for(x = 0; x != linebytes; ++x) out[y * (linebytes + 1) + 1 + x] = attemp
5421     }
5422     for(type = 0; type != 5; ++type) lodepng_free(attempt[type]);
5423   }
5424   else return 88; /* unknown filter strategy */
5425
5426   return error;
5427 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.41 filterScanline()

```
static void filterScanline (
            unsigned char * out,
            const unsigned char * scanline,
            const unsigned char * prevline,
            size_t length,
            size_t bytewidth,
            unsigned char filterType )  [static]
```

Definition at line 5144 of file lodepng.cpp.

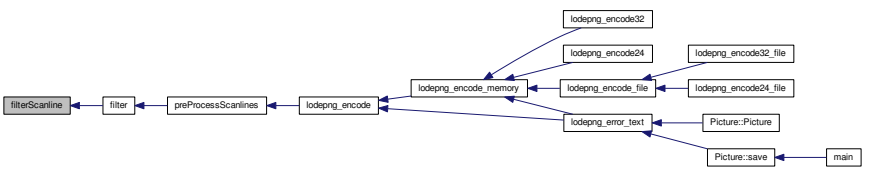
```
5146 {
5147   size_t i;
5148   switch(filterType)
5149   {
5150     case 0: /*None*/
5151       for(i = 0; i != length; ++i) out[i] = scanline[i];
5152       break;
5153     case 1: /*Sub*/
5154       for(i = 0; i != bytewidth; ++i) out[i] = scanline[i];
5155       for(i = bytewidth; i < length; ++i) out[i] = scanline[i] - scanline[i - b
5156       break;
5157     case 2: /*Up*/
5158       if(prevline)
5159       {
5160         for(i = 0; i != length; ++i) out[i] = scanline[i] - prevline[i];
5161       }
5162       else
5163       {
```

```
5164          for(i = 0; i != length; ++i) out[i] = scanline[i];
5165        }
5166        break;
5167      case 3: /*Average*/
5168        if(prevline)
5169        {
5170          for(i = 0; i != bytewidth; ++i) out[i] = scanline[i] - (prevline[i] >>
5171          for(i = bytewidth; i < length; ++i) out[i] = scanline[i] - ((scanline[i
      >> 1);
5172        }
5173        else
5174        {
5175          for(i = 0; i != bytewidth; ++i) out[i] = scanline[i];
5176          for(i = bytewidth; i < length; ++i) out[i] = scanline[i] - (scanline[i
5177        }
5178        break;
5179      case 4: /*Paeth*/
5180        if(prevline)
5181        {
5182          /*paethPredictor(0, prevline[i], 0) is always prevline[i]*/
5183          for(i = 0; i != bytewidth; ++i) out[i] = (scanline[i] - prevline[i]);
5184          for(i = bytewidth; i < length; ++i)
5185          {
5186            out[i] = (scanline[i] - paethPredictor(scanline[i - bytewidth], prevl
      prevline[i - bytewidth]));
5187          }
5188        }
5189        else
5190        {
5191          for(i = 0; i != bytewidth; ++i) out[i] = scanline[i];
5192          /*paethPredictor(scanline[i - bytewidth], 0, 0) is always scanline[i -
```

```
5193         for(i = bytewidth; i < length; ++i) out[i] = (scanline[i] - scanline[i
5194       }
5195       break;
5196     default: return; /*unexisting filter type given*/
5197   }
5198 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



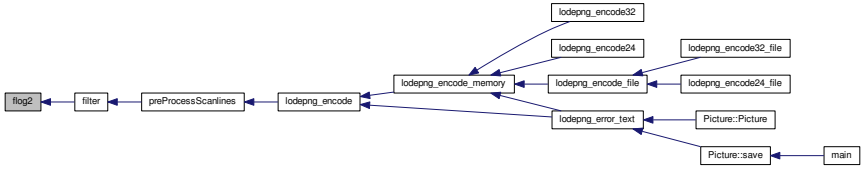
4.1.3.42 flog2()

```
static float flog2 (
            float f )  [static]
```

Definition at line 5201 of file lodepng.cpp.

```
5202 {
5203   float result = 0;
5204   while(f > 32) { result += 4; f /= 16; }
5205   while(f > 2) { ++result; f /= 2; }
5206   return result + 1.442695f * (f * f * f / 3 - 3 * f * f / 2 + 3 * f - 1.83333f
5207 }
```

Here is the caller graph for this function:



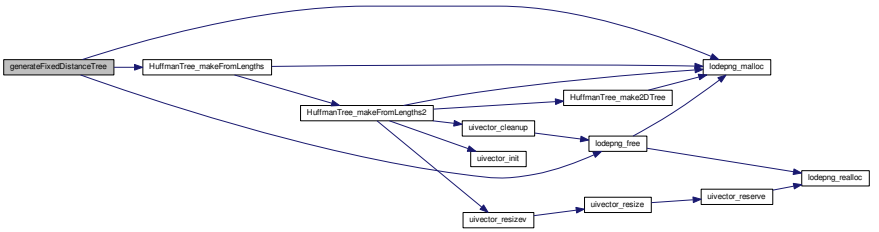
4.1.3.43 generateFixedDistanceTree()

```
static unsigned generateFixedDistanceTree (
            HuffmanTree * tree )  [static]
```

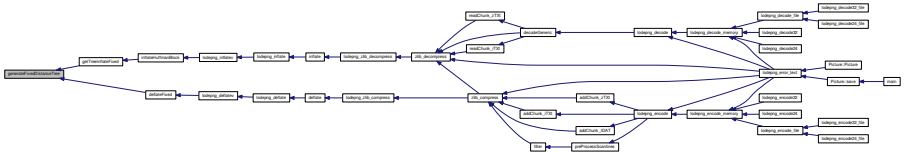
Definition at line 927 of file lodepng.cpp.

```
928 {
929   unsigned i, error = 0;
930   unsigned* bitlen = (unsigned*)lodepng_malloc(
      NUM_DISTANCE_SYMBOLS * sizeof(unsigned));
931   if(!bitlen) return 83; /*alloc fail*/
932
933   /*there are 32 distance codes, but 30-31 are unused*/
934   for(i = 0; i != NUM_DISTANCE_SYMBOLS; ++i) bitlen[i] = 5;
935   error = HuffmanTree_makeFromLengths(tree, bitlen,
      NUM_DISTANCE_SYMBOLS, 15);
936
937   lodepng_free(bitlen);
938   return error;
939 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.44 generateFixedLitLenTree()

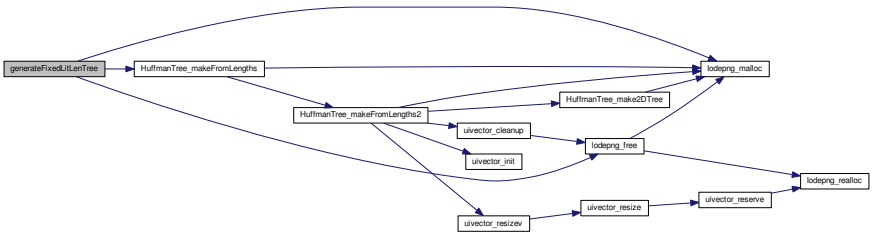
```
static unsigned generateFixedLitLenTree (
            HuffmanTree * tree ) [static]
```

Definition at line 908 of file lodepng.cpp.

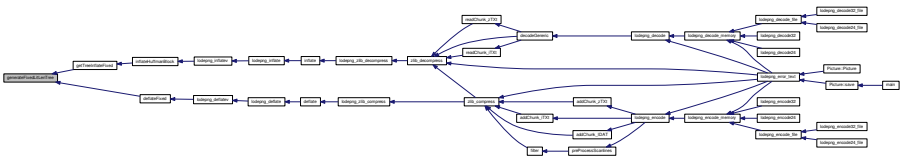
```
909 {
910   unsigned i, error = 0;
911   unsigned* bitlen = (unsigned*)lodepng_malloc(
      NUM_DEFLATE_CODE_SYMBOLS * sizeof(unsigned));
912   if(!bitlen) return 83; /*alloc fail*/
913
914   /*288 possible codes: 0-255=literals, 256=endcode, 257-285=lengthcodes, 286-28
915   for(i =   0; i <= 143; ++i) bitlen[i] = 8;
916   for(i = 144; i <= 255; ++i) bitlen[i] = 9;
917   for(i = 256; i <= 279; ++i) bitlen[i] = 7;
918   for(i = 280; i <= 287; ++i) bitlen[i] = 8;
919
920   error = HuffmanTree_makeFromLengths(tree, bitlen,
```

```
      NUM_DEFLATE_CODE_SYMBOLS, 15);
921
922   lodepng_free(bitlen);
923   return error;
924 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.45 getHash()

```
static unsigned getHash (
            const unsigned char * data,
            size_t size,
            size_t pos )  [static]
```

Definition at line 1421 of file lodepng.cpp.

```
1422 {
1423   unsigned result = 0;
1424   if(pos + 2 < size)
1425   {
1426     /*A simple shift and xor hash is used. Since the data of PNGs is dominated
1427     by zeroes due to the filters, a better hash does not have a significant
1428     effect on speed in traversing the chain, and causes more time spend on
1429     calculating the hash.*/
1430     result ^= (unsigned)(data[pos + 0] << 0u);
1431     result ^= (unsigned)(data[pos + 1] << 4u);
1432     result ^= (unsigned)(data[pos + 2] << 8u);
1433   } else {
1434     size_t amount, i;
1435     if(pos >= size) return 0;
1436     amount = size - pos;
1437     for(i = 0; i != amount; ++i) result ^= (unsigned)(data[pos + i] << (i * 8u)
1438   }
1439   return result & HASH_BIT_MASK;
1440 }
```

Here is the caller graph for this function:

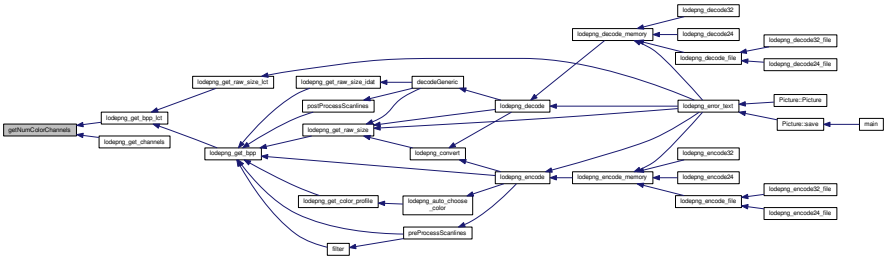
4.1.3.46 getNumColorChannels()

```
static unsigned getNumColorChannels (
            LodePNGColorType colortype )  [static]
```

Definition at line 2562 of file lodepng.cpp.

```
2563 {
2564   switch(colortype)
2565   {
2566     case 0: return 1; /*grey*/
2567     case 2: return 3; /*RGB*/
2568     case 3: return 1; /*palette*/
2569     case 4: return 2; /*grey + alpha*/
2570     case 6: return 4; /*RGBA*/
2571   }
2572   return 0; /*unexisting color type*/
2573 }
```

Here is the caller graph for this function:



4.1.3.47 getPaletteTranslucency()

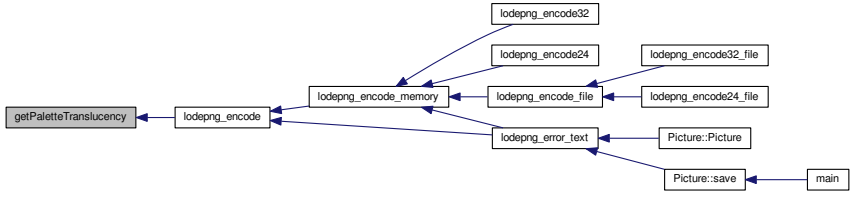
```
static unsigned getPaletteTranslucency (
            const unsigned char * palette,
            size_t palettesize )  [static]
```

Definition at line 5606 of file lodepng.cpp.

```
5607 {
5608   size_t i;
5609   unsigned key = 0;
5610   unsigned r = 0, g = 0, b = 0; /*the value of the color with alpha 0, so long
      */
5611   for(i = 0; i != palettesize; ++i)
5612   {
5613     if(!key && palette[4 * i + 3] == 0)
5614     {
```

```
5615       r = palette[4 * i + 0]; g = palette[4 * i + 1]; b = palette[4 * i + 2];
5616       key = 1;
5617       i = (size_t)(-1); /*restart from beginning, to detect earlier opaque colo
5618     }
5619     else if(palette[4 * i + 3] != 255) return 2;
5620     /*when key, no opaque RGB may have key's RGB*/
5621     else if(key && r == palette[i * 4 + 0] && g == palette[i * 4 + 1] && b == p
5622   }
5623   return key;
5624 }
```

Here is the caller graph for this function:



4.1.3.48 getPixelColorRGBA16()

```
static void getPixelColorRGBA16 (
            unsigned short * r,
            unsigned short * g,
            unsigned short * b,
            unsigned short * a,
```

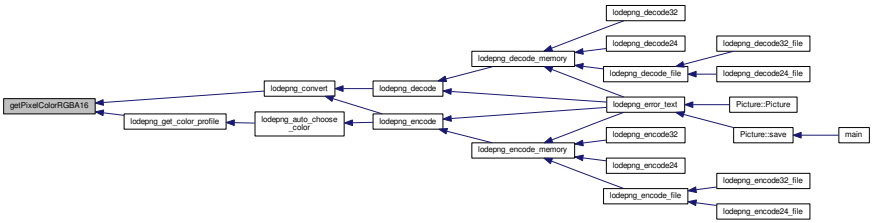
```
    const unsigned char * in,
    size_t i,
    const LodePNGColorMode * mode )  [static]
```

Definition at line 3425 of file lodepng.cpp.

```
3427 {
3428   if(mode->colortype == LCT_GREY)
3429   {
3430     *r = *g = *b = 256 * in[i * 2 + 0] + in[i * 2 + 1];
3431     if(mode->key_defined && 256U * in[i * 2 + 0] + in[i * 2 + 1] == mode->
      key_r) *a = 0;
3432     else *a = 65535;
3433   }
3434   else if(mode->colortype == LCT_RGB)
3435   {
3436     *r = 256u * in[i * 6 + 0] + in[i * 6 + 1];
3437     *g = 256u * in[i * 6 + 2] + in[i * 6 + 3];
3438     *b = 256u * in[i * 6 + 4] + in[i * 6 + 5];
3439     if(mode->key_defined
3440        && 256u * in[i * 6 + 0] + in[i * 6 + 1] == mode->key_r
3441        && 256u * in[i * 6 + 2] + in[i * 6 + 3] == mode->key_g
3442        && 256u * in[i * 6 + 4] + in[i * 6 + 5] == mode->key_b) *a = 0;
3443     else *a = 65535;
3444   }
3445   else if(mode->colortype == LCT_GREY_ALPHA)
3446   {
3447     *r = *g = *b = 256u * in[i * 4 + 0] + in[i * 4 + 1];
3448     *a = 256u * in[i * 4 + 2] + in[i * 4 + 3];
3449   }
3450   else if(mode->colortype == LCT_RGBA)
3451   {
```

```
3452       *r = 256u * in[i * 8 + 0] + in[i * 8 + 1];
3453       *g = 256u * in[i * 8 + 2] + in[i * 8 + 3];
3454       *b = 256u * in[i * 8 + 4] + in[i * 8 + 5];
3455       *a = 256u * in[i * 8 + 6] + in[i * 8 + 7];
3456     }
3457 }
```

Here is the caller graph for this function:



4.1.3.49 getPixelColorRGBA8()

```
static void getPixelColorRGBA8 (
            unsigned char * r,
            unsigned char * g,
            unsigned char * b,
            unsigned char * a,
            const unsigned char * in,
            size_t i,
            const LodePNGColorMode * mode )  [static]
```

Definition at line 3181 of file lodepng.cpp.

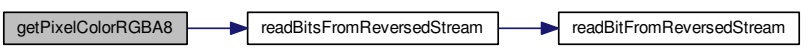
```
3185 {
3186   if(mode->colortype == LCT_GREY)
3187   {
3188     if(mode->bitdepth == 8)
3189     {
3190       *r = *g = *b = in[i];
3191       if(mode->key_defined && *r == mode->key_r) *a = 0;
3192       else *a = 255;
3193     }
3194     else if(mode->bitdepth == 16)
3195     {
3196       *r = *g = *b = in[i * 2 + 0];
3197       if(mode->key_defined && 256U * in[i * 2 + 0] + in[i * 2 + 1] == mode->
     key_r) *a = 0;
3198       else *a = 255;
3199     }
3200     else
3201     {
3202       unsigned highest = ((1U << mode->bitdepth) - 1U); /*highest possible valu
     */
3203       size_t j = i * mode->bitdepth;
3204       unsigned value = readBitsFromReversedStream(&j, in, mode->
     bitdepth);
3205       *r = *g = *b = (value * 255) / highest;
3206       if(mode->key_defined && value == mode->key_r) *a = 0;
3207       else *a = 255;
3208     }
3209   }
3210   else if(mode->colortype == LCT_RGB)
3211   {
3212     if(mode->bitdepth == 8)
```

```
3213     {
3214       *r = in[i * 3 + 0]; *g = in[i * 3 + 1]; *b = in[i * 3 + 2];
3215       if(mode->key_defined && *r == mode->key_r && *g == mode->
     key_g && *b == mode->key_b) *a = 0;
3216       else *a = 255;
3217     }
3218     else
3219     {
3220       *r = in[i * 6 + 0];
3221       *g = in[i * 6 + 2];
3222       *b = in[i * 6 + 4];
3223       if(mode->key_defined && 256U * in[i * 6 + 0] + in[i * 6 + 1] == mode->
     key_r
3224          && 256U * in[i * 6 + 2] + in[i * 6 + 3] == mode->key_g
3225          && 256U * in[i * 6 + 4] + in[i * 6 + 5] == mode->key_b) *a = 0;
3226       else *a = 255;
3227     }
3228   }
3229   else if(mode->colortype == LCT_PALETTE)
3230   {
3231     unsigned index;
3232     if(mode->bitdepth == 8) index = in[i];
3233     else
3234     {
3235       size_t j = i * mode->bitdepth;
3236       index = readBitsFromReversedStream(&j, in, mode->
     bitdepth);
3237     }
3238
3239     if(index >= mode->palettesize)
3240     {
```

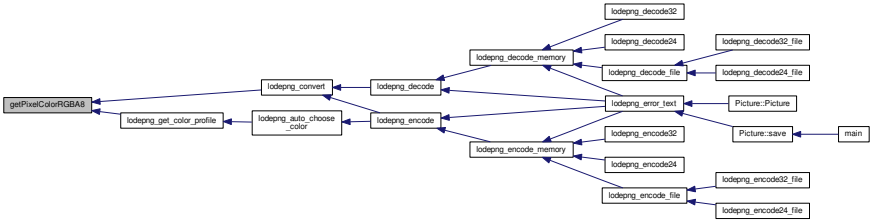
```
3241       /*This is an error according to the PNG spec, but common PNG decoders mak
3242       Done here too, slightly faster due to no error handling needed.*/
3243       *r = *g = *b = 0;
3244       *a = 255;
3245     }
3246     else
3247     {
3248       *r = mode->palette[index * 4 + 0];
3249       *g = mode->palette[index * 4 + 1];
3250       *b = mode->palette[index * 4 + 2];
3251       *a = mode->palette[index * 4 + 3];
3252     }
3253   }
3254   else if(mode->colortype == LCT_GREY_ALPHA)
3255   {
3256     if(mode->bitdepth == 8)
3257     {
3258       *r = *g = *b = in[i * 2 + 0];
3259       *a = in[i * 2 + 1];
3260     }
3261     else
3262     {
3263       *r = *g = *b = in[i * 4 + 0];
3264       *a = in[i * 4 + 2];
3265     }
3266   }
3267   else if(mode->colortype == LCT_RGBA)
3268   {
3269     if(mode->bitdepth == 8)
3270     {
3271       *r = in[i * 4 + 0];
```

```
3272         *g = in[i * 4 + 1];
3273         *b = in[i * 4 + 2];
3274         *a = in[i * 4 + 3];
3275       }
3276       else
3277       {
3278         *r = in[i * 8 + 0];
3279         *g = in[i * 8 + 2];
3280         *b = in[i * 8 + 4];
3281         *a = in[i * 8 + 6];
3282       }
3283     }
3284 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.50 getPixelColorsRGBA8()

```
static void getPixelColorsRGBA8 (
            unsigned char * buffer,
            size_t numpixels,
            unsigned has_alpha,
            const unsigned char * in,
            const LodePNGColorMode * mode )  [static]
```

Definition at line 3291 of file lodepng.cpp.

```
3294 {
3295   unsigned num_channels = has_alpha ? 4 : 3;
3296   size_t i;
3297   if(mode->colortype == LCT_GREY)
3298   {
3299     if(mode->bitdepth == 8)
3300     {
```

```
3301       for(i = 0; i != numpixels; ++i, buffer += num_channels)
3302       {
3303         buffer[0] = buffer[1] = buffer[2] = in[i];
3304         if(has_alpha) buffer[3] = mode->key_defined && in[i] == mode->
       key_r ? 0 : 255;
3305       }
3306     }
3307     else if(mode->bitdepth == 16)
3308     {
3309       for(i = 0; i != numpixels; ++i, buffer += num_channels)
3310       {
3311         buffer[0] = buffer[1] = buffer[2] = in[i * 2];
3312         if(has_alpha) buffer[3] = mode->key_defined && 256U * in[i * 2 + 0] + i
       mode->key_r ? 0 : 255;
3313       }
3314     }
3315     else
3316     {
3317       unsigned highest = ((1U << mode->bitdepth) - 1U); /*highest possible valu
       */
3318       size_t j = 0;
3319       for(i = 0; i != numpixels; ++i, buffer += num_channels)
3320       {
3321         unsigned value = readBitsFromReversedStream(&j, in, mode->
       bitdepth);
3322         buffer[0] = buffer[1] = buffer[2] = (value * 255) / highest;
3323         if(has_alpha) buffer[3] = mode->key_defined && value == mode->
       key_r ? 0 : 255;
3324       }
3325     }
3326   }
```

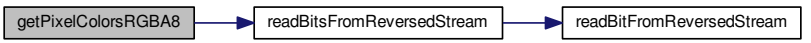
```
3327   else if(mode->colortype == LCT_RGB)
3328   {
3329     if(mode->bitdepth == 8)
3330     {
3331       for(i = 0; i != numpixels; ++i, buffer += num_channels)
3332       {
3333         buffer[0] = in[i * 3 + 0];
3334         buffer[1] = in[i * 3 + 1];
3335         buffer[2] = in[i * 3 + 2];
3336         if(has_alpha) buffer[3] = mode->key_defined && buffer[0] == mode->
     key_r
3337             && buffer[1]== mode->key_g && buffer[2] == mode->key_b ? 0 : 255;
3338       }
3339     }
3340     else
3341     {
3342       for(i = 0; i != numpixels; ++i, buffer += num_channels)
3343       {
3344         buffer[0] = in[i * 6 + 0];
3345         buffer[1] = in[i * 6 + 2];
3346         buffer[2] = in[i * 6 + 4];
3347         if(has_alpha) buffer[3] = mode->key_defined
3348             && 256U * in[i * 6 + 0] + in[i * 6 + 1] == mode->key_r
3349             && 256U * in[i * 6 + 2] + in[i * 6 + 3] == mode->key_g
3350             && 256U * in[i * 6 + 4] + in[i * 6 + 5] == mode->key_b ? 0 : 255;
3351       }
3352     }
3353   }
3354   else if(mode->colortype == LCT_PALETTE)
3355   {
3356     unsigned index;
```

```
3357     size_t j = 0;
3358     for(i = 0; i != numpixels; ++i, buffer += num_channels)
3359     {
3360       if(mode->bitdepth == 8) index = in[i];
3361       else index = readBitsFromReversedStream(&j, in, mode->
     bitdepth);
3362 
3363       if(index >= mode->palettesize)
3364       {
3365         /*This is an error according to the PNG spec, but most PNG decoders mak
3366         Done here too, slightly faster due to no error handling needed.*/
3367         buffer[0] = buffer[1] = buffer[2] = 0;
3368         if(has_alpha) buffer[3] = 255;
3369       }
3370       else
3371       {
3372         buffer[0] = mode->palette[index * 4 + 0];
3373         buffer[1] = mode->palette[index * 4 + 1];
3374         buffer[2] = mode->palette[index * 4 + 2];
3375         if(has_alpha) buffer[3] = mode->palette[index * 4 + 3];
3376       }
3377     }
3378   }
3379   else if(mode->colortype == LCT_GREY_ALPHA)
3380   {
3381     if(mode->bitdepth == 8)
3382     {
3383       for(i = 0; i != numpixels; ++i, buffer += num_channels)
3384       {
3385         buffer[0] = buffer[1] = buffer[2] = in[i * 2 + 0];
3386         if(has_alpha) buffer[3] = in[i * 2 + 1];
```

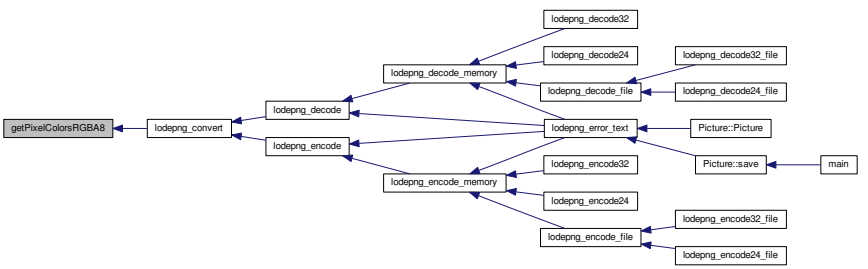
```
3387         }
3388       }
3389       else
3390       {
3391         for(i = 0; i != numpixels; ++i, buffer += num_channels)
3392         {
3393           buffer[0] = buffer[1] = buffer[2] = in[i * 4 + 0];
3394           if(has_alpha) buffer[3] = in[i * 4 + 2];
3395         }
3396       }
3397     }
3398     else if(mode->colortype == LCT_RGBA)
3399     {
3400       if(mode->bitdepth == 8)
3401       {
3402         for(i = 0; i != numpixels; ++i, buffer += num_channels)
3403         {
3404           buffer[0] = in[i * 4 + 0];
3405           buffer[1] = in[i * 4 + 1];
3406           buffer[2] = in[i * 4 + 2];
3407           if(has_alpha) buffer[3] = in[i * 4 + 3];
3408         }
3409       }
3410       else
3411       {
3412         for(i = 0; i != numpixels; ++i, buffer += num_channels)
3413         {
3414           buffer[0] = in[i * 8 + 0];
3415           buffer[1] = in[i * 8 + 2];
3416           buffer[2] = in[i * 8 + 4];
3417           if(has_alpha) buffer[3] = in[i * 8 + 6];
```

```
3418         }
3419       }
3420     }
3421 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.51 getTreeInflateDynamic()

```
static unsigned getTreeInflateDynamic (
            HuffmanTree * tree_ll,
            HuffmanTree * tree_d,
            const unsigned char * in,
            size_t * bp,
            size_t inlength )  [static]
```

Definition at line 983 of file lodepng.cpp.

```
985 {
986   /*make sure that length values that aren't filled in will be 0, or a wrong tre
987   unsigned error = 0;
988   unsigned n, HLIT, HDIST, HCLEN, i;
989   size_t inbitlength = inlength * 8;
990
991   /*see comments in deflateDynamic for explanation of the context and these vari
992   unsigned* bitlen_ll = 0; /*lit,len code lengths*/
993   unsigned* bitlen_d = 0; /*dist code lengths*/
994   /*code length code lengths ("clcl"), the bit lengths of the huffman tree used
       bitlen_d*/
995   unsigned* bitlen_cl = 0;
996   HuffmanTree tree_cl; /*the code tree for code length codes (the huffman tree f
       huffman trees)*/
997
998   if((*bp) + 14 > (inlength << 3)) return 49; /*error: the bit pointer is or wil
999
1000   /*number of literal/length codes + 257. Unlike the spec, the value 257 is add
1001   HLIT =  readBitsFromStream(bp, in, 5) + 257;
1002   /*number of distance codes. Unlike the spec, the value 1 is added to it here
1003   HDIST = readBitsFromStream(bp, in, 5) + 1;
```

```
1004   /*number of code length codes. Unlike the spec, the value 4 is added to it he
1005   HCLEN = readBitsFromStream(bp, in, 4) + 4;
1006
1007   if((*bp) + HCLEN * 3 > (inlength << 3)) return 50; /*error: the bit pointer i
       */
1008
1009   HuffmanTree_init(&tree_cl);
1010
1011   while(!error)
1012   {
1013     /*read the code length codes out of 3 * (amount of code length codes) bits*
1014
1015     bitlen_cl = (unsigned*)lodepng_malloc(NUM_CODE_LENGTH_CODES * sizeof
       (unsigned));
1016     if(!bitlen_cl) ERROR_BREAK(83 /*alloc fail*/);
1017
1018     for(i = 0; i != NUM_CODE_LENGTH_CODES; ++i)
1019     {
1020       if(i < HCLEN) bitlen_cl[CLCL_ORDER[i]] = readBitsFromStream(bp, in, 3);
1021       else bitlen_cl[CLCL_ORDER[i]] = 0; /*if not, it must stay 0*/
1022     }
1023
1024     error = HuffmanTree_makeFromLengths(&tree_cl, bitlen_cl,
       NUM_CODE_LENGTH_CODES, 7);
1025     if(error) break;
1026
1027     /*now we can use this tree to read the lengths for the tree that this funct
1028     bitlen_ll = (unsigned*)lodepng_malloc(NUM_DEFLATE_CODE_SYMBOLS *
       sizeof(unsigned));
1029     bitlen_d = (unsigned*)lodepng_malloc(NUM_DISTANCE_SYMBOLS * sizeof(
       unsigned));
```

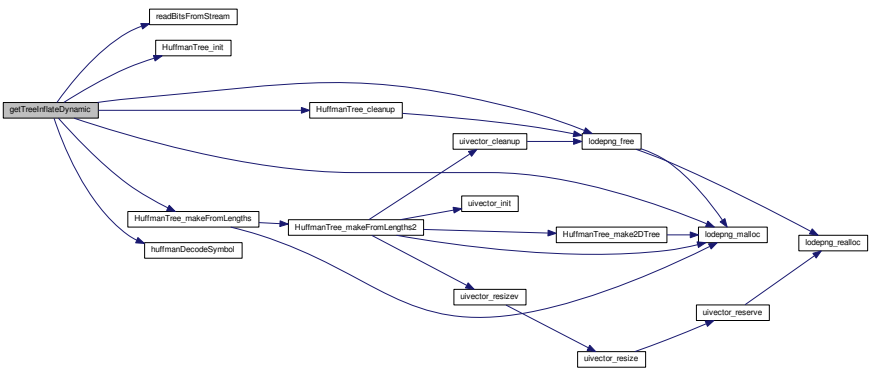
```
1030     if(!bitlen_ll || !bitlen_d) ERROR_BREAK(83 /*alloc fail*/);
1031     for(i = 0; i != NUM_DEFLATE_CODE_SYMBOLS; ++i) bitlen_ll[i] = 0;
1032     for(i = 0; i != NUM_DISTANCE_SYMBOLS; ++i) bitlen_d[i] = 0;
1033
1034     /*i is the current symbol we're reading in the part that contains the code
       codes*/
1035     i = 0;
1036     while(i < HLIT + HDIST)
1037     {
1038       unsigned code = huffmanDecodeSymbol(in, bp, &tree_cl, inbitlength);
1039       if(code <= 15) /*a length code*/
1040       {
1041         if(i < HLIT) bitlen_ll[i] = code;
1042         else bitlen_d[i - HLIT] = code;
1043         ++i;
1044       }
1045       else if(code == 16) /*repeat previous*/
1046       {
1047         unsigned replength = 3; /*read in the 2 bits that indicate repeat lengt
1048         unsigned value; /*set value to the previous code*/
1049
1050         if(i == 0) ERROR_BREAK(54); /*can't repeat previous if i is 0*/
1051
1052         if((*bp + 2) > inbitlength) ERROR_BREAK(50); /*error, bit pointer jumps
1053         replength += readBitsFromStream(bp, in, 2);
1054
1055         if(i < HLIT + 1) value = bitlen_ll[i - 1];
1056         else value = bitlen_d[i - HLIT - 1];
1057         /*repeat this value in the next lengths*/
1058         for(n = 0; n < replength; ++n)
1059         {
```

```
1060            if(i >= HLIT + HDIST) ERROR_BREAK(13); /*error: i is larger than the
1061            if(i < HLIT) bitlen_ll[i] = value;
1062            else bitlen_d[i - HLIT] = value;
1063            ++i;
1064          }
1065        }
1066        else if(code == 17) /*repeat "0" 3-10 times*/
1067        {
1068          unsigned replength = 3; /*read in the bits that indicate repeat length*
1069          if((*bp + 3) > inbitlength) ERROR_BREAK(50); /*error, bit pointer jumps
1070          replength += readBitsFromStream(bp, in, 3);
1071
1072          /*repeat this value in the next lengths*/
1073          for(n = 0; n < replength; ++n)
1074          {
1075            if(i >= HLIT + HDIST) ERROR_BREAK(14); /*error: i is larger than the
1076
1077            if(i < HLIT) bitlen_ll[i] = 0;
1078            else bitlen_d[i - HLIT] = 0;
1079            ++i;
1080          }
1081        }
1082        else if(code == 18) /*repeat "0" 11-138 times*/
1083        {
1084          unsigned replength = 11; /*read in the bits that indicate repeat length
1085          if((*bp + 7) > inbitlength) ERROR_BREAK(50); /*error, bit pointer jumps
1086          replength += readBitsFromStream(bp, in, 7);
1087
1088          /*repeat this value in the next lengths*/
1089          for(n = 0; n < replength; ++n)
1090          {
```

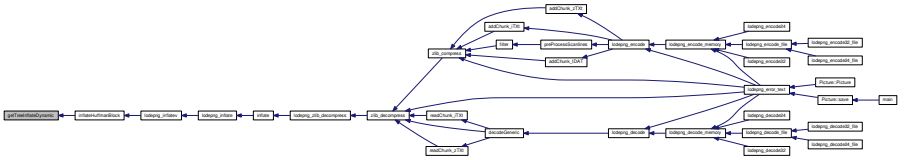
```
1091           if(i >= HLIT + HDIST) ERROR_BREAK(15); /*error: i is larger than the
1092
1093           if(i < HLIT) bitlen_ll[i] = 0;
1094           else bitlen_d[i - HLIT] = 0;
1095           ++i;
1096         }
1097       }
1098       else /*if(code == (unsigned)(-1))*/ /*huffmanDecodeSymbol returns (unsign
1099       {
1100         if(code == (unsigned)(-1))
1101         {
1102           /*return error code 10 or 11 depending on the situation that happened
1103           (10=no endcode, 11=wrong jump outside of tree)*/
1104           error = (*bp) > inbitlength ? 10 : 11;
1105         }
1106         else error = 16; /*unexisting code, this can never happen*/
1107         break;
1108       }
1109     }
1110     if(error) break;
1111
1112     if(bitlen_ll[256] == 0) ERROR_BREAK(64); /*the length of the end code 256 m
       than 0*/
1113
1114     /*now we've finally got HLIT and HDIST, so generate the code trees, and the
1115     error = HuffmanTree_makeFromLengths(tree_ll, bitlen_ll,
      NUM_DEFLATE_CODE_SYMBOLS, 15);
1116     if(error) break;
1117     error = HuffmanTree_makeFromLengths(tree_d, bitlen_d,
      NUM_DISTANCE_SYMBOLS, 15);
1118
```

```
1119     break; /*end of error-while*/
1120   }
1121
1122   lodepng_free(bitlen_cl);
1123   lodepng_free(bitlen_ll);
1124   lodepng_free(bitlen_d);
1125   HuffmanTree_cleanup(&tree_cl);
1126
1127   return error;
1128 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



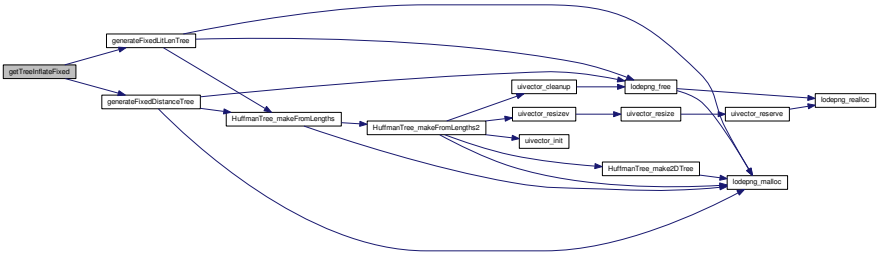
4.1.3.52 getTreeInflateFixed()

```
static void getTreeInflateFixed (
            HuffmanTree * tree_ll,
            HuffmanTree * tree_d )  [static]
```

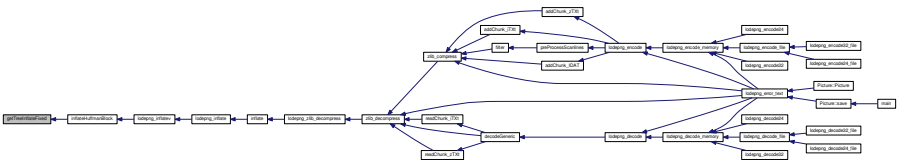
Definition at line 975 of file lodepng.cpp.

```
976 {
977   /*TODO: check for out of memory errors*/
978   generateFixedLitLenTree(tree_ll);
979   generateFixedDistanceTree(tree_d);
980 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



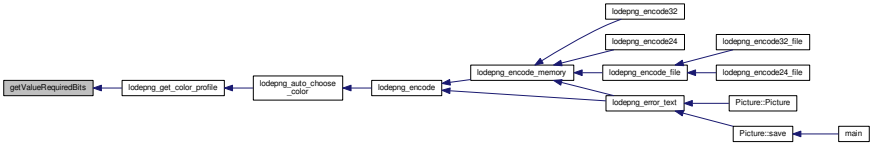
4.1.3.53 getValueRequiredBits()

```
static unsigned getValueRequiredBits (
            unsigned char value )  [static]
```

Definition at line 3557 of file lodepng.cpp.

```
3558 {
3559   if(value == 0 || value == 255) return 1;
3560   /*The scaling of 2-bit and 4-bit values uses multiples of 85 and 17*/
3561   if(value % 17 == 0) return value % 85 == 0 ? 2 : 4;
3562   return 8;
3563 }
```

Here is the caller graph for this function:



4.1.3.54 hash_cleanup()

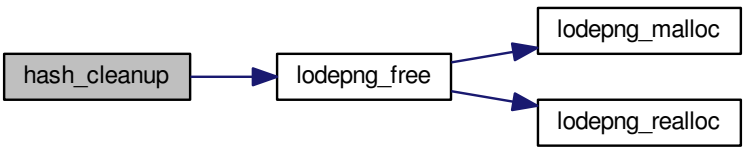
```
static void hash_cleanup (
            Hash * hash )  [static]
```

Definition at line 1408 of file lodepng.cpp.

```
1409 {
1410   lodepng_free(hash->head);
1411   lodepng_free(hash->val);
1412   lodepng_free(hash->chain);
1413
```

```
1414   lodepng_free(hash->zeros);
1415   lodepng_free(hash->headz);
1416   lodepng_free(hash->chainz);
1417 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.3.55 hash_init()

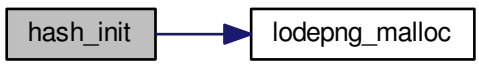
```
static unsigned hash_init (
            Hash * hash,
            unsigned windowsize )  [static]
```

Definition at line 1381 of file lodepng.cpp.

```
1382 {
1383   unsigned i;
1384   hash->head = (int*)lodepng_malloc(sizeof(int) *
      HASH_NUM_VALUES);
1385   hash->val = (int*)lodepng_malloc(sizeof(int) * windowsize);
1386   hash->chain = (unsigned short*)lodepng_malloc(sizeof(unsigned short) * window
1387
1388   hash->zeros = (unsigned short*)lodepng_malloc(sizeof(unsigned short) * window
1389   hash->headz = (int*)lodepng_malloc(sizeof(int) * (
      MAX_SUPPORTED_DEFLATE_LENGTH + 1));
1390   hash->chainz = (unsigned short*)lodepng_malloc(sizeof(unsigned short) * windo
1391
1392   if(!hash->head || !hash->chain || !hash->val  || !hash->headz|| !hash->
      chainz || !hash->zeros)
1393   {
1394     return 83; /*alloc fail*/
1395   }
1396
1397   /*initialize hash table*/
1398   for(i = 0; i != HASH_NUM_VALUES; ++i) hash->head[i] = -1;
1399   for(i = 0; i != windowsize; ++i) hash->val[i] = -1;
1400   for(i = 0; i != windowsize; ++i) hash->chain[i] = i; /*same value as index in
1401
1402   for(i = 0; i <= MAX_SUPPORTED_DEFLATE_LENGTH; ++i) hash->
```

```
    headz[i] = -1;
1403   for(i = 0; i != windowsize; ++i) hash->chainz[i] = i; /*same value as index i
    */
1404
1405   return 0;
1406 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

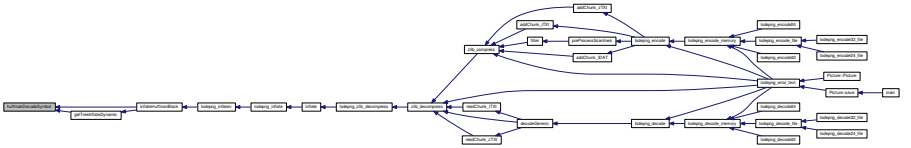
4.1.3.56 huffmanDecodeSymbol()

```
static unsigned huffmanDecodeSymbol (
            const unsigned char * in,
            size_t * bp,
            const HuffmanTree * codetree,
            size_t inbitlength )  [static]
```

Definition at line 947 of file lodepng.cpp.

```
949 {
950   unsigned treepos = 0, ct;
951   for(;;)
952   {
953     if(*bp >= inbitlength) return (unsigned)(-1); /*error: end of input memory r
954     /*
955     decode the symbol from the tree. The "readBitFromStream" code is inlined in
956     the expression below because this is the biggest bottleneck while decoding
957     */
958     ct = codetree->tree2d[(treepos << 1) + READBIT(*bp, in)];
959     ++(*bp);
960     if(ct < codetree->numcodes) return ct; /*the symbol is decoded, return it*/
961     else treepos = ct - codetree->numcodes; /*symbol not yet decoded, instead mo
962
963     if(treepos >= codetree->numcodes) return (unsigned)(-1); /*error: it appeare
       codetree*/
964   }
965 }
```

Here is the caller graph for this function:



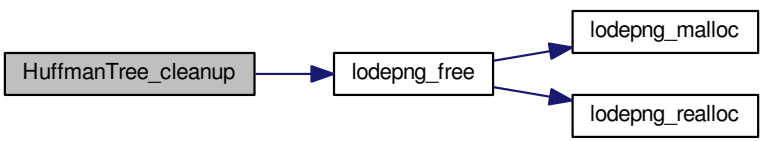
4.1.3.57 HuffmanTree_cleanup()

```
static void HuffmanTree_cleanup (
            HuffmanTree * tree )  [static]
```

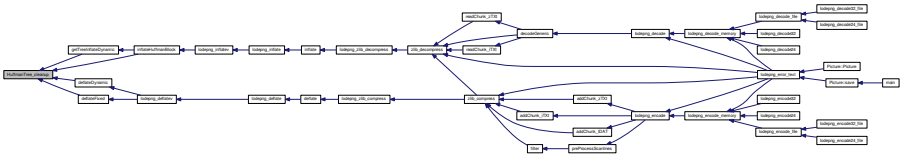
Definition at line 538 of file lodepng.cpp.

```
539 {
540   lodepng_free(tree->tree2d);
541   lodepng_free(tree->tree1d);
542   lodepng_free(tree->lengths);
543 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.58 HuffmanTree_getCode()

```
static unsigned HuffmanTree_getCode (
            const HuffmanTree * tree,
            unsigned index )  [static]
```

Definition at line 896 of file lodepng.cpp.

```
897 {
898   return tree->tree1d[index];
899 }
```

Here is the caller graph for this function:

4.1.3.59 HuffmanTree_getLength()

```
static unsigned HuffmanTree_getLength (
            const HuffmanTree * tree,
            unsigned index )  [static]
```

Definition at line 901 of file lodepng.cpp.

```
902 {
903   return tree->lengths[index];
904 }
```

Here is the caller graph for this function:

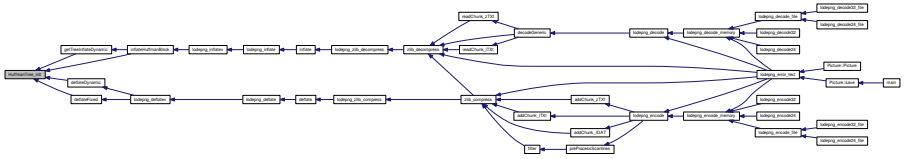
4.1.3.60 HuffmanTree_init()

```
static void HuffmanTree_init (
            HuffmanTree * tree )  [static]
```

Definition at line 531 of file lodepng.cpp.

```
532 {
533   tree->tree2d = 0;
534   tree->tree1d = 0;
535   tree->lengths = 0;
536 }
```

Here is the caller graph for this function:



4.1.3.61 HuffmanTree_make2DTree()

```
static unsigned HuffmanTree_make2DTree (
            HuffmanTree * tree )  [static]
```

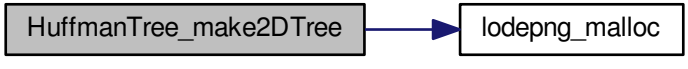
Definition at line 546 of file lodepng.cpp.

```
547 {
548   unsigned nodefilled = 0; /*up to which node it is filled*/
549   unsigned treepos = 0; /*position in the tree (1 of the numcodes columns)*/
550   unsigned n, i;
551
552   tree->tree2d = (unsigned*)lodepng_malloc(tree->numcodes * 2 * sizeof(unsigned
      ));
553   if(!tree->tree2d) return 83; /*alloc fail*/
554
555   /*
556   convert tree1d[] to tree2d[][]. In the 2D array, a value of 32767 means
557   uninited, a value >= numcodes is an address to another bit, a value < numcodes
558   is a code. The 2 rows are the 2 possible bit values (0 or 1), there are as
```

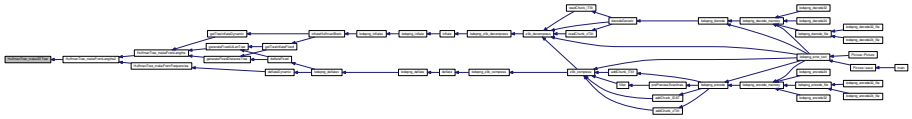
```
559   many columns as codes - 1.
560   A good huffman tree has N * 2 - 1 nodes, of which N - 1 are internal nodes.
561   Here, the internal nodes are stored (what their 0 and 1 option point to).
562   There is only memory for such good tree currently, if there are more nodes
563   (due to too long length codes), error 55 will happen
564   */
565   for(n = 0; n < tree->numcodes * 2; ++n)
566   {
567     tree->tree2d[n] = 32767; /*32767 here means the tree2d isn't filled there ye
568   }
569
570   for(n = 0; n < tree->numcodes; ++n) /*the codes*/
571   {
572     for(i = 0; i != tree->lengths[n]; ++i) /*the bits for this code*/
573     {
574       unsigned char bit = (unsigned char)((tree->tree1d[n] >> (tree->
      lengths[n] - i - 1)) & 1);
575       /*oversubscribed, see comment in lodepng_error_text*/
576       if(treepos > 2147483647 || treepos + 2 > tree->numcodes) return 55;
577       if(tree->tree2d[2 * treepos + bit] == 32767) /*not yet filled in*/
578       {
579         if(i + 1 == tree->lengths[n]) /*last bit*/
580         {
581           tree->tree2d[2 * treepos + bit] = n; /*put the current code in it*/
582           treepos = 0;
583         }
584         else
585         {
586           /*put address of the next step in here, first that address has to be f
587           (it's just nodefilled + 1)...*/
588           ++nodefilled;
```

```
589           /*addresses encoded with numcodes added to it*/
590           tree->tree2d[2 * treepos + bit] = nodefilled + tree->numcodes;
591           treepos = nodefilled;
592         }
593       }
594       else treepos = tree->tree2d[2 * treepos + bit] - tree->numcodes;
595     }
596   }
597
598   for(n = 0; n < tree->numcodes * 2; ++n)
599   {
600     if(tree->tree2d[n] == 32767) tree->tree2d[n] = 0; /*remove possible remainin
601   }
602
603   return 0;
604 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.62 HuffmanTree_makeFromFrequencies()

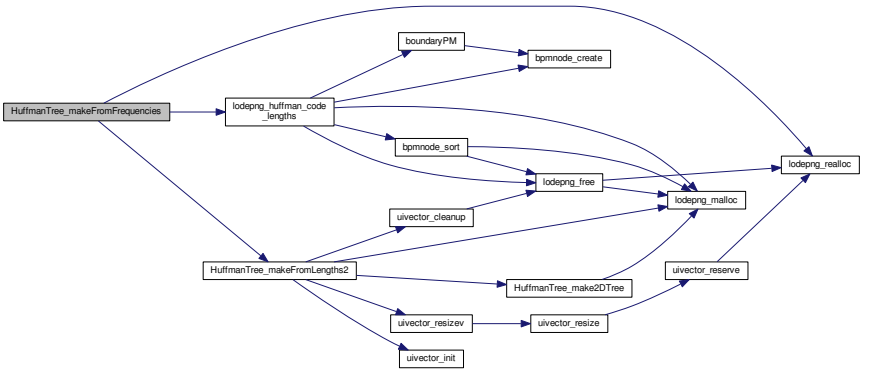
```
static unsigned HuffmanTree_makeFromFrequencies (
            HuffmanTree * tree,
            const unsigned * frequencies,
            size_t mincodes,
            size_t numcodes,
            unsigned maxbitlen )  [static]
```

Definition at line 879 of file lodepng.cpp.

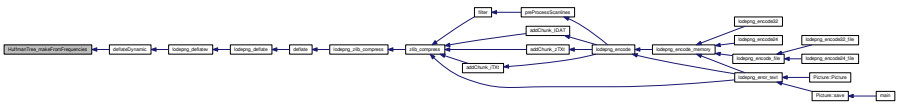
```
881 {
882   unsigned error = 0;
883   while(!frequencies[numcodes - 1] && numcodes > mincodes) --numcodes; /*trim ze
884   tree->maxbitlen = maxbitlen;
885   tree->numcodes = (unsigned)numcodes; /*number of symbols*/
886   tree->lengths = (unsigned*)lodepng_realloc(tree->lengths, numcodes * sizeof(
      unsigned));
887   if(!tree->lengths) return 83; /*alloc fail*/
888   /*initialize all lengths to 0*/
889   memset(tree->lengths, 0, numcodes * sizeof(unsigned));
890
```

```
891   error = lodepng_huffman_code_lengths(tree->lengths, frequencies,
      numcodes, maxbitlen);
892   if(!error) error = HuffmanTree_makeFromLengths2(tree);
893   return error;
894 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



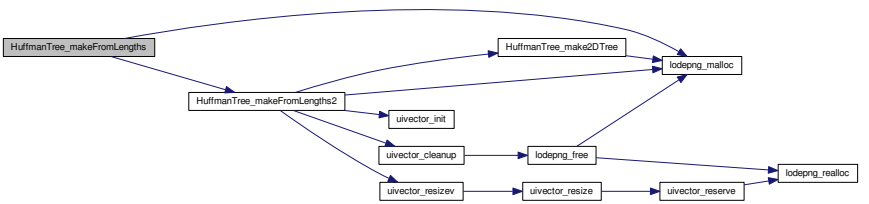
4.1.3.63 HuffmanTree_makeFromLengths()

```
static unsigned HuffmanTree_makeFromLengths (
            HuffmanTree * tree,
            const unsigned * bitlen,
            size_t numcodes,
            unsigned maxbitlen )  [static]
```

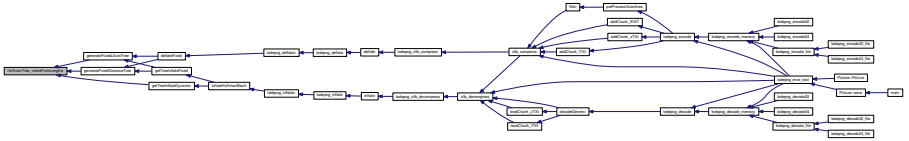
Definition at line 656 of file lodepng.cpp.

```
658 {
659   unsigned i;
660   tree->lengths = (unsigned*)lodepng_malloc(numcodes * sizeof(unsigned));
661   if(!tree->lengths) return 83; /*alloc fail*/
662   for(i = 0; i != numcodes; ++i) tree->lengths[i] = bitlen[i];
663   tree->numcodes = (unsigned)numcodes; /*number of symbols*/
664   tree->maxbitlen = maxbitlen;
665   return HuffmanTree_makeFromLengths2(tree);
666 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.64 HuffmanTree_makeFromLengths2()

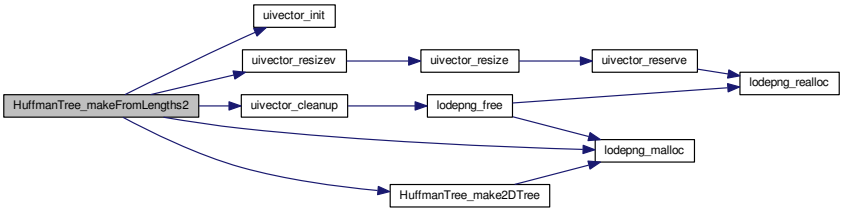
```
static unsigned HuffmanTree_makeFromLengths2 (
            HuffmanTree * tree )  [static]
```

Definition at line 611 of file lodepng.cpp.

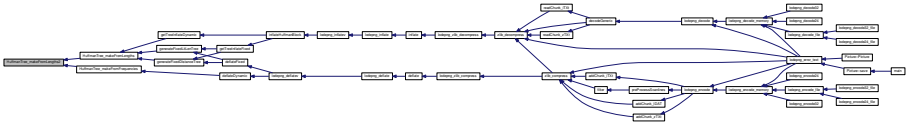
```
612 {
613   uivector blcount;
614   uivector nextcode;
615   unsigned error = 0;
616   unsigned bits, n;
617
618   uivector_init(&blcount);
619   uivector_init(&nextcode);
620
621   tree->tree1d = (unsigned*)lodepng_malloc(tree->numcodes * sizeof(unsigned));
622   if(!tree->tree1d) error = 83; /*alloc fail*/
623
624   if(!uivector_resizev(&blcount, tree->maxbitlen + 1, 0)
625   || !uivector_resizev(&nextcode, tree->maxbitlen + 1, 0))
```

```
626     error = 83; /*alloc fail*/
627
628   if(!error)
629   {
630     /*step 1: count number of instances of each code length*/
631     for(bits = 0; bits != tree->numcodes; ++bits) ++blcount.data[tree->
      lengths[bits]];
632     /*step 2: generate the nextcode values*/
633     for(bits = 1; bits <= tree->maxbitlen; ++bits)
634     {
635       nextcode.data[bits] = (nextcode.data[bits - 1] + blcount.data[bits - 1]) <
636     }
637     /*step 3: generate all the codes*/
638     for(n = 0; n != tree->numcodes; ++n)
639     {
640       if(tree->lengths[n] != 0) tree->tree1d[n] = nextcode.data[tree->
      lengths[n]]++;
641     }
642   }
643
644   uivector_cleanup(&blcount);
645   uivector_cleanup(&nextcode);
646
647   if(!error) return HuffmanTree_make2DTree(tree);
648   else return error;
649 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



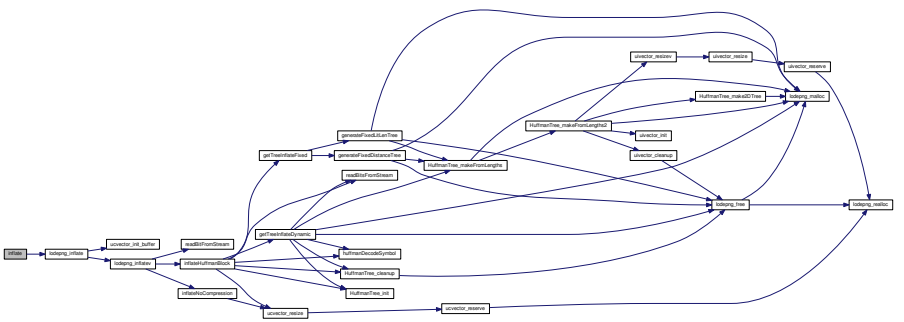
4.1.3.65 inflate()

```
static unsigned inflate (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGDecompressSettings * settings )  [static]
```

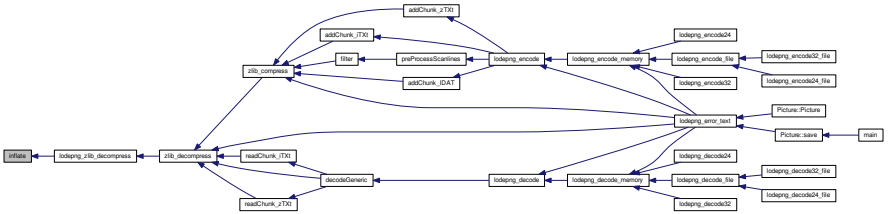
Definition at line 1296 of file lodepng.cpp.

```
1299 {
1300   if(settings->custom_inflate)
1301   {
1302     return settings->custom_inflate(out, outsize, in, insize, settings);
1303   }
1304   else
1305   {
1306     return lodepng_inflate(out, outsize, in, insize, settings);
1307   }
1308 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.66 inflateHuffmanBlock()

```
static unsigned inflateHuffmanBlock (
            ucvector * out,
            const unsigned char * in,
            size_t * bp,
            size_t * pos,
            size_t inlength,
            unsigned btype )  [static]
```

Definition at line 1131 of file lodepng.cpp.

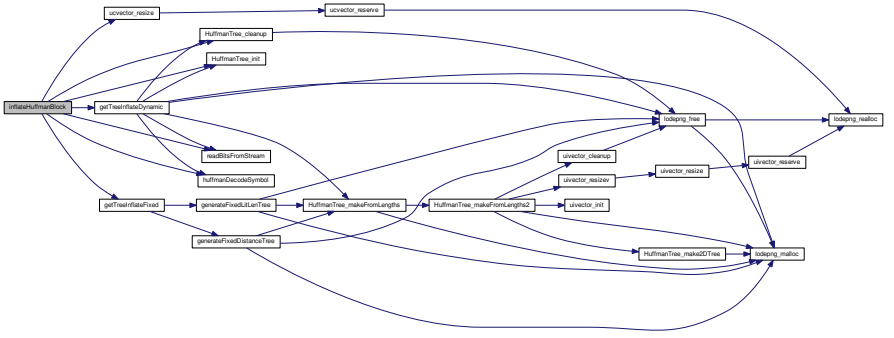
```
1133 {
1134   unsigned error = 0;
1135   HuffmanTree tree_ll; /*the huffman tree for literal and length codes*/
1136   HuffmanTree tree_d; /*the huffman tree for distance codes*/
1137   size_t inbitlength = inlength * 8;
1138
1139   HuffmanTree_init(&tree_ll);
```

```
1140   HuffmanTree_init(&tree_d);
1141
1142   if(btype == 1) getTreeInflateFixed(&tree_ll, &tree_d);
1143   else if(btype == 2) error = getTreeInflateDynamic(&tree_ll, &tree_d, in, bp,
       inlength);
1144
1145   while(!error) /*decode all symbols until end reached, breaks at end code*/
1146   {
1147     /*code_ll is literal, length or end code*/
1148     unsigned code_ll = huffmanDecodeSymbol(in, bp, &tree_ll, inbitlength);
1149     if(code_ll <= 255) /*literal symbol*/
1150     {
1151       /*ucvector_push_back would do the same, but for some reason the two lines
1152       if(!ucvector_resize(out, (*pos) + 1)) ERROR_BREAK(83 /*alloc fail*/);
1153       out->data[*pos] = (unsigned char)code_ll;
1154       ++(*pos);
1155     }
1156     else if(code_ll >= FIRST_LENGTH_CODE_INDEX && code_ll <=
       LAST_LENGTH_CODE_INDEX) /*length code*/
1157     {
1158       unsigned code_d, distance;
1159       unsigned numextrabits_l, numextrabits_d; /*extra bits for length and dist
1160       size_t start, forward, backward, length;
1161
1162       /*part 1: get length base*/
1163       length = LENGTHBASE[code_ll - FIRST_LENGTH_CODE_INDEX];
1164
1165       /*part 2: get extra bits and add the value of that to length*/
1166       numextrabits_l = LENGTHEXTRA[code_ll - FIRST_LENGTH_CODE_INDEX];
1167       if((*bp + numextrabits_l) > inbitlength) ERROR_BREAK(51); /*error, bit po
        past memory*/
```

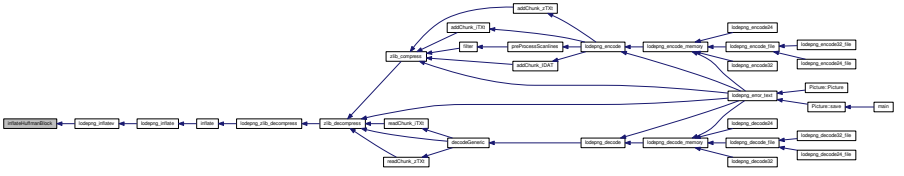
```
1168        length += readBitsFromStream(bp, in, numextrabits_l);
1169
1170        /*part 3: get distance code*/
1171        code_d = huffmanDecodeSymbol(in, bp, &tree_d, inbitlength);
1172        if(code_d > 29)
1173        {
1174          if(code_ll == (unsigned)(-1)) /*huffmanDecodeSymbol returns (unsigned)(
1175          {
1176            /*return error code 10 or 11 depending on the situation that happened
1177            (10=no endcode, 11=wrong jump outside of tree)*/
1178            error = (*bp) > inlength * 8 ? 10 : 11;
1179          }
1180          else error = 18; /*error: invalid distance code (30-31 are never used)*
1181          break;
1182        }
1183        distance = DISTANCEBASE[code_d];
1184
1185        /*part 4: get extra bits from distance*/
1186        numextrabits_d = DISTANCEEXTRA[code_d];
1187        if((*bp + numextrabits_d) > inbitlength) ERROR_BREAK(51); /*error, bit po
      past memory*/
1188        distance += readBitsFromStream(bp, in, numextrabits_d);
1189
1190        /*part 5: fill in all the out[n] values based on the length and dist*/
1191        start = (*pos);
1192        if(distance > start) ERROR_BREAK(52); /*too long backward distance*/
1193        backward = start - distance;
1194
1195        if(!ucvector_resize(out, (*pos) + length)) ERROR_BREAK(83 /*alloc fail*/)
1196        if (distance < length) {
1197          for(forward = 0; forward < length; ++forward)
```

```
1198         {
1199           out->data[(*pos)++] = out->data[backward++];
1200         }
1201       } else {
1202         memcpy(out->data + *pos, out->data + backward, length);
1203         *pos += length;
1204       }
1205     }
1206     else if(code_ll == 256)
1207     {
1208       break; /*end code, break the loop*/
1209     }
1210     else /*if(code == (unsigned)(-1))*/ /*huffmanDecodeSymbol returns (unsigned
1211     {
1212       /*return error code 10 or 11 depending on the situation that happened in
1213       (10=no endcode, 11=wrong jump outside of tree)*/
1214       error = ((*bp) > inlength * 8) ? 10 : 11;
1215       break;
1216     }
1217   }
1218
1219   HuffmanTree_cleanup(&tree_ll);
1220   HuffmanTree_cleanup(&tree_d);
1221
1222   return error;
1223 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.67 inflateNoCompression()

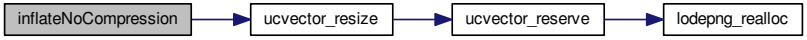
```
static unsigned inflateNoCompression (
            ucvector * out,
            const unsigned char * in,
```

```
    size_t * bp,
    size_t * pos,
    size_t inlength )  [static]
```

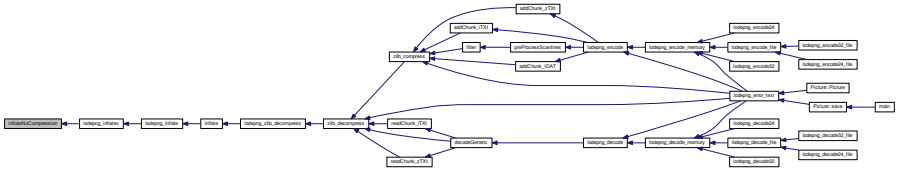
Definition at line 1225 of file lodepng.cpp.

```
1226 {
1227   size_t p;
1228   unsigned LEN, NLEN, n, error = 0;
1229
1230   /*go to first boundary of byte*/
1231   while(((*bp) & 0x7) != 0) ++(*bp);
1232   p = (*bp) / 8; /*byte position*/
1233
1234   /*read LEN (2 bytes) and NLEN (2 bytes)*/
1235   if(p + 4 >= inlength) return 52; /*error, bit pointer will jump past memory*/
1236   LEN = in[p] + 256u * in[p + 1]; p += 2;
1237   NLEN = in[p] + 256u * in[p + 1]; p += 2;
1238
1239   /*check if 16-bit NLEN is really the one's complement of LEN*/
1240   if(LEN + NLEN != 65535) return 21; /*error: NLEN is not one's complement of L
1241
1242   if(!ucvector_resize(out, (*pos) + LEN)) return 83; /*alloc fail*/
1243
1244   /*read the literal data: LEN bytes are now stored in the out buffer*/
1245   if(p + LEN > inlength) return 23; /*error: reading outside of in buffer*/
1246   for(n = 0; n < LEN; ++n) out->data[(*pos)++] = in[p++];
1247
1248   (*bp) = p * 8;
1249
1250   return error;
1251 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



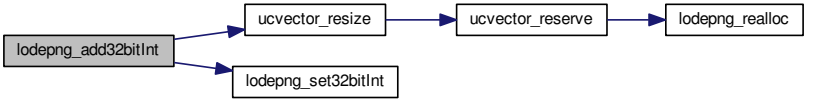
4.1.3.68 lodepng_add32bitInt()

```
static void lodepng_add32bitInt (
            ucvector * buffer,
            unsigned value )  [static]
```

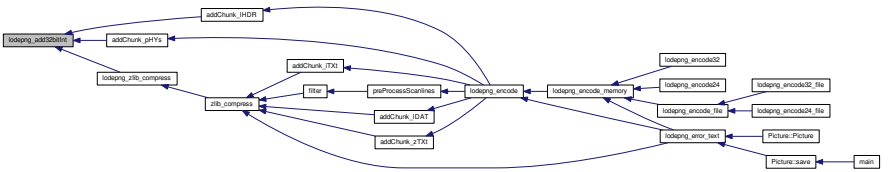
Definition at line 337 of file lodepng.cpp.

```
338 {
339   ucvector_resize(buffer, buffer->size + 4); /*todo: give error if resize failed
340   lodepng_set32bitInt(&buffer->data[buffer->size - 4], value);
341 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.69 lodepng_add_itext()

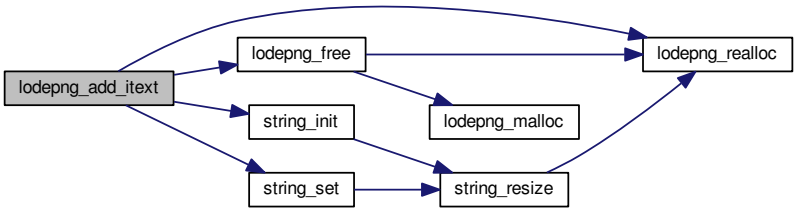
```
unsigned lodepng_add_itext (
            LodePNGInfo * info,
            const char * key,
            const char * langtag,
            const char * transkey,
            const char * str )
```

Definition at line 2885 of file lodepng.cpp.

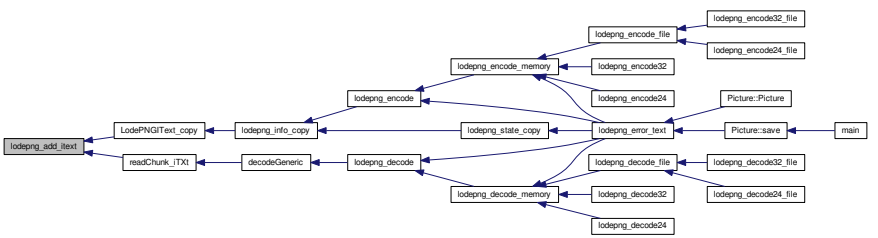
```
2887 {
2888   char** new_keys = (char**)(lodepng_realloc(info->itext_keys, sizeof(char*) *
     info->itext_num + 1)));
2889   char** new_langtags = (char**)(lodepng_realloc(info->
     itext_langtags, sizeof(char*) * (info->itext_num + 1)));
2890   char** new_transkeys = (char**)(lodepng_realloc(info->
     itext_transkeys, sizeof(char*) * (info->itext_num + 1)));
2891   char** new_strings = (char**)(lodepng_realloc(info->
     itext_strings, sizeof(char*) * (info->itext_num + 1)));
2892   if(!new_keys || !new_langtags || !new_transkeys || !new_strings)
2893   {
2894     lodepng_free(new_keys);
2895     lodepng_free(new_langtags);
2896     lodepng_free(new_transkeys);
2897     lodepng_free(new_strings);
2898     return 83; /*alloc fail*/
2899   }
2900
2901   ++info->itext_num;
2902   info->itext_keys = new_keys;
2903   info->itext_langtags = new_langtags;
2904   info->itext_transkeys = new_transkeys;
2905   info->itext_strings = new_strings;
2906
2907   string_init(&info->itext_keys[info->itext_num - 1]);
2908   string_set(&info->itext_keys[info->itext_num - 1], key);
2909
2910   string_init(&info->itext_langtags[info->itext_num - 1]);
2911   string_set(&info->itext_langtags[info->itext_num - 1], langtag);
2912
2913   string_init(&info->itext_transkeys[info->itext_num - 1]);
```

```
2914   string_set(&info->itext_transkeys[info->itext_num - 1], transkey);
2915
2916   string_init(&info->itext_strings[info->itext_num - 1]);
2917   string_set(&info->itext_strings[info->itext_num - 1], str);
2918
2919   return 0;
2920 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.70 lodepng_add_text()

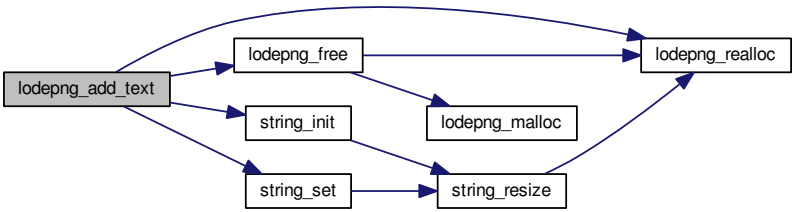
```
unsigned lodepng_add_text (
            LodePNGInfo * info,
            const char * key,
            const char * str )
```

Definition at line 2813 of file lodepng.cpp.

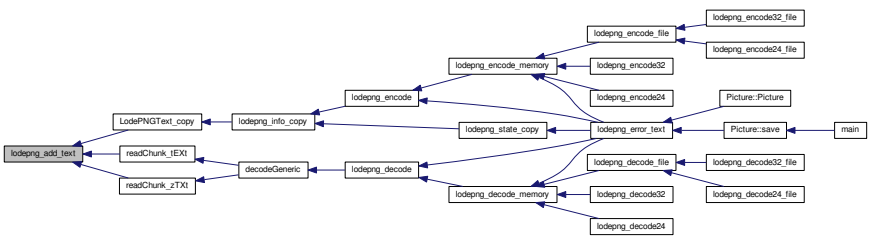
```
2814 {
2815   char** new_keys = (char**)(lodepng_realloc(info->text_keys, sizeof(char*) * (
      ->text_num + 1)));
2816   char** new_strings = (char**)(lodepng_realloc(info->
      text_strings, sizeof(char*) * (info->text_num + 1)));
2817   if(!new_keys || !new_strings)
2818   {
2819     lodepng_free(new_keys);
2820     lodepng_free(new_strings);
2821     return 83; /*alloc fail*/
2822   }
2823
2824   ++info->text_num;
2825   info->text_keys = new_keys;
2826   info->text_strings = new_strings;
2827
2828   string_init(&info->text_keys[info->text_num - 1]);
2829   string_set(&info->text_keys[info->text_num - 1], key);
2830
2831   string_init(&info->text_strings[info->text_num - 1]);
2832   string_set(&info->text_strings[info->text_num - 1], str);
```

```
2833
2834   return 0;
2835 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.71 lodepng_auto_choose_color()

```
unsigned lodepng_auto_choose_color (
            LodePNGColorMode * mode_out,
            const unsigned char * image,
            unsigned w,
            unsigned h,
            const LodePNGColorMode * mode_in )
```

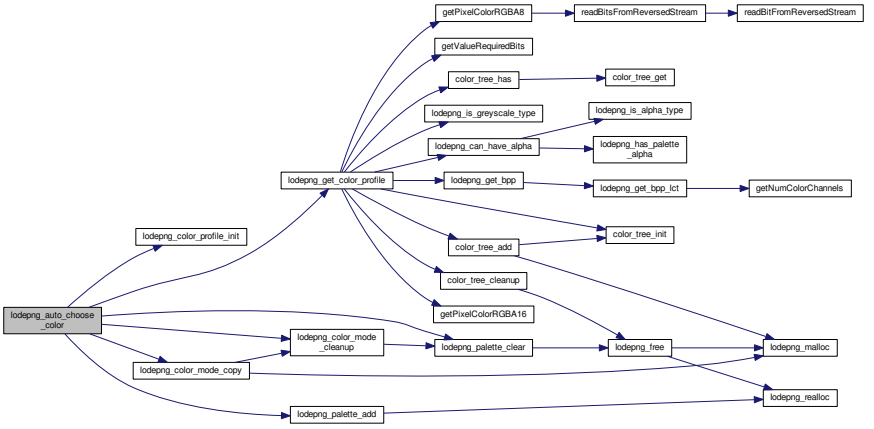
Definition at line 3763 of file lodepng.cpp.

```
3766 {
3767   LodePNGColorProfile prof;
3768   unsigned error = 0;
3769   unsigned i, n, palettebits, palette_ok;
3770
3771   lodepng_color_profile_init(&prof);
3772   error = lodepng_get_color_profile(&prof, image, w, h, mode_in);
3773   if(error) return error;
3774   mode_out->key_defined = 0;
3775
3776   if(prof.key && w * h <= 16)
3777   {
3778     prof.alpha = 1; /*too few pixels to justify tRNS chunk overhead*/
3779     prof.key = 0;
3780     if(prof.bits < 8) prof.bits = 8; /*PNG has no alphachannel modes with less
      channel*/
3781   }
3782   n = prof.numcolors;
3783   palettebits = n <= 2 ? 1 : (n <= 4 ? 2 : (n <= 16 ? 4 : 8));
3784   palette_ok = n <= 256 && prof.bits <= 8;
3785   if(w * h < n * 2) palette_ok = 0; /*don't add palette overhead if image has o
```

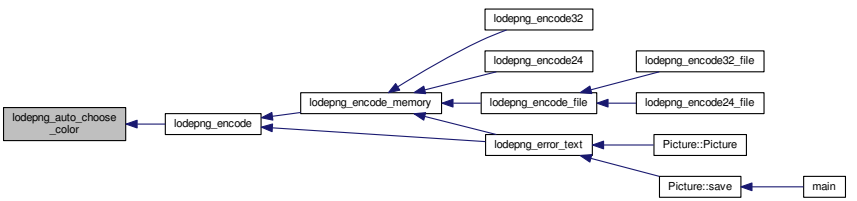
```
3786   if(!prof.colored && prof.bits <= palettebits) palette_ok = 0; /*grey is less
3787
3788   if(palette_ok)
3789   {
3790     unsigned char* p = prof.palette;
3791     lodepng_palette_clear(mode_out); /*remove potential earlier palette*/
3792     for(i = 0; i != prof.numcolors; ++i)
3793     {
3794       error = lodepng_palette_add(mode_out, p[i * 4 + 0], p[i * 4 + 1], p[i * 4
    i * 4 + 3]);
3795       if(error) break;
3796     }
3797
3798     mode_out->colortype = LCT_PALETTE;
3799     mode_out->bitdepth = palettebits;
3800
3801     if(mode_in->colortype == LCT_PALETTE && mode_in->
    palettesize >= mode_out->palettesize
3802         && mode_in->bitdepth == mode_out->bitdepth)
3803     {
3804       /*If input should have same palette colors, keep original to preserve its
     conversion*/
3805       lodepng_color_mode_cleanup(mode_out);
3806       lodepng_color_mode_copy(mode_out, mode_in);
3807     }
3808   }
3809   else /*8-bit or 16-bit per channel*/
3810   {
3811     mode_out->bitdepth = prof.bits;
3812     mode_out->colortype = prof.alpha ? (prof.colored ?
    LCT_RGBA : LCT_GREY_ALPHA)
```

```
3813                                        : (prof.colored ? LCT_RGB :
     LCT_GREY);
3814
3815     if(prof.key)
3816     {
3817       unsigned mask = (1u << mode_out->bitdepth) - 1u; /*profile always uses 16
      it*/
3818       mode_out->key_r = prof.key_r & mask;
3819       mode_out->key_g = prof.key_g & mask;
3820       mode_out->key_b = prof.key_b & mask;
3821       mode_out->key_defined = 1;
3822     }
3823   }
3824
3825   return error;
3826 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



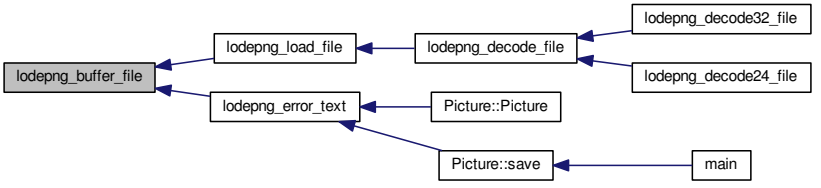
4.1.3.72 lodepng_buffer_file()

```
static unsigned lodepng_buffer_file (
            unsigned char * out,
            size_t size,
            const char * filename )  [static]
```

Definition at line 373 of file lodepng.cpp.

```
374 {
375   FILE* file;
376   size_t readsize;
377   file = fopen(filename, "rb");
378   if(!file) return 78;
379
380   readsize = fread(out, 1, size, file);
381   fclose(file);
382
383   if (readsize != size) return 78;
384   return 0;
385 }
```

Here is the caller graph for this function:



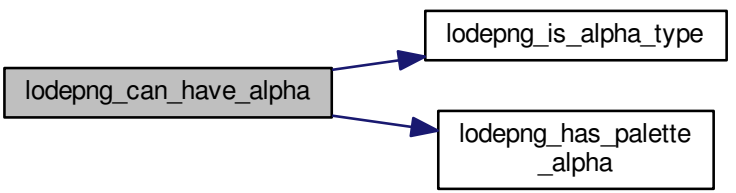
4.1.3.73 lodepng_can_have_alpha()

```
unsigned lodepng_can_have_alpha (
            const LodePNGColorMode * info )
```

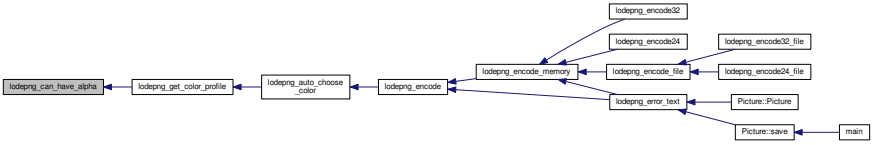
Definition at line 2701 of file lodepng.cpp.

```
2702 {
2703   return info->key_defined
2704       || lodepng_is_alpha_type(info)
2705       || lodepng_has_palette_alpha(info);
2706 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



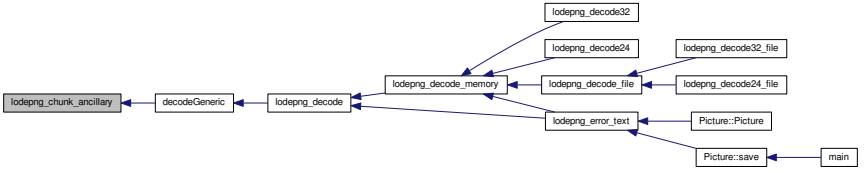
4.1.3.74 lodepng_chunk_ancillary()

```
unsigned char lodepng_chunk_ancillary (
            const unsigned char * chunk )
```

Definition at line 2439 of file lodepng.cpp.

```
2440 {
2441   return((chunk[4] & 32) != 0);
2442 }
```

Here is the caller graph for this function:



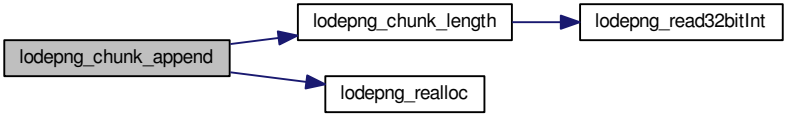
4.1.3.75 lodepng_chunk_append()

```
unsigned lodepng_chunk_append (
            unsigned char ** out,
            size_t * outlength,
            const unsigned char * chunk )
```

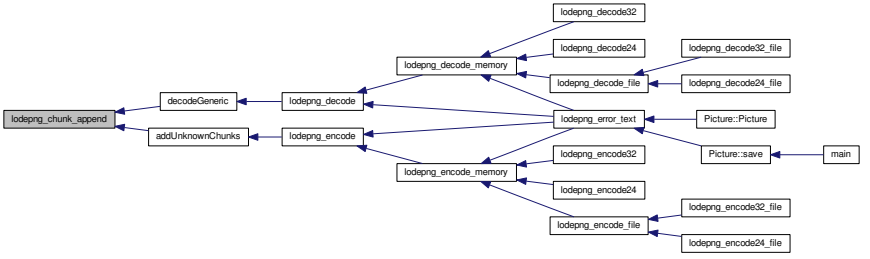
Definition at line 2493 of file lodepng.cpp.

```
2494 {
2495   unsigned i;
2496   unsigned total_chunk_length = lodepng_chunk_length(chunk) + 12;
2497   unsigned char *chunk_start, *new_buffer;
2498   size_t new_length = (*outlength) + total_chunk_length;
2499   if(new_length < total_chunk_length || new_length < (*outlength)) return 77; /
2500
2501   new_buffer = (unsigned char*)lodepng_realloc(*out, new_length);
2502   if(!new_buffer) return 83; /*alloc fail*/
2503   (*out) = new_buffer;
2504   (*outlength) = new_length;
2505   chunk_start = &(*out)[new_length - total_chunk_length];
2506
2507   for(i = 0; i != total_chunk_length; ++i) chunk_start[i] = chunk[i];
2508
2509   return 0;
2510 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



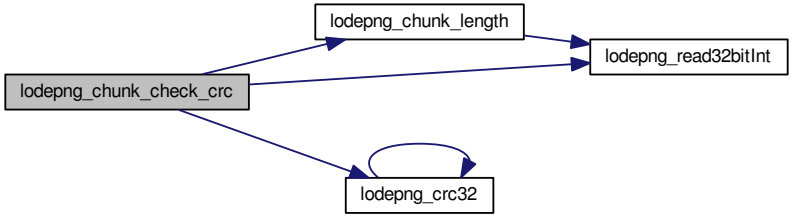
4.1.3.76 lodepng_chunk_check_crc()

```
unsigned lodepng_chunk_check_crc (
            const unsigned char * chunk )
```

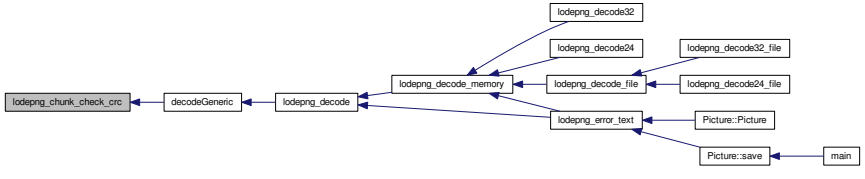
Definition at line 2464 of file lodepng.cpp.

```
2465 {
2466   unsigned length = lodepng_chunk_length(chunk);
2467   unsigned CRC = lodepng_read32bitInt(&chunk[length + 8]);
2468   /*the CRC is taken of the data and the 4 chunk type letters, not the length*/
2469   unsigned checksum = lodepng_crc32(&chunk[4], length + 4);
2470   if(CRC != checksum) return 1;
2471   else return 0;
2472 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.77 lodepng_chunk_create()

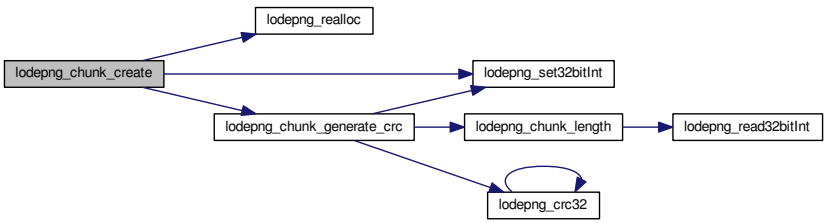
```
unsigned lodepng_chunk_create (
            unsigned char ** out,
            size_t * outlength,
            unsigned length,
            const char * type,
            const unsigned char * data )
```

Definition at line 2512 of file lodepng.cpp.

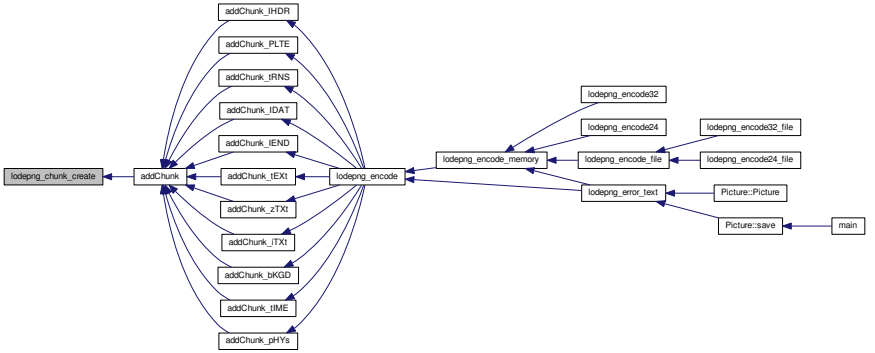
```
2514 {
2515   unsigned i;
2516   unsigned char *chunk, *new_buffer;
2517   size_t new_length = (*outlength) + length + 12;
2518   if(new_length < length + 12 || new_length < (*outlength)) return 77; /*intege
2519   new_buffer = (unsigned char*)lodepng_realloc(*out, new_length);
2520   if(!new_buffer) return 83; /*alloc fail*/
2521   (*out) = new_buffer;
2522   (*outlength) = new_length;
2523   chunk = &(*out)[(*outlength) - length - 12];
2524
2525   /*1: length*/
2526   lodepng_set32bitInt(chunk, (unsigned)length);
2527
2528   /*2: chunk name (4 letters)*/
2529   chunk[4] = (unsigned char)type[0];
2530   chunk[5] = (unsigned char)type[1];
2531   chunk[6] = (unsigned char)type[2];
2532   chunk[7] = (unsigned char)type[3];
```

```
2533
2534   /*3: the data*/
2535   for(i = 0; i != length; ++i) chunk[8 + i] = data[i];
2536
2537   /*4: CRC (of the chunkname characters and the data)*/
2538   lodepng_chunk_generate_crc(chunk);
2539
2540   return 0;
2541 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.78 lodepng_chunk_data()

```
unsigned char* lodepng_chunk_data (
            unsigned char * chunk )
```

Definition at line 2454 of file lodepng.cpp.

```
2455 {
2456   return &chunk[8];
2457 }
```

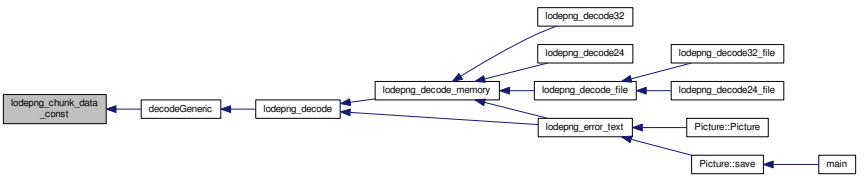
4.1.3.79 lodepng_chunk_data_const()

```
const unsigned char* lodepng_chunk_data_const (
            const unsigned char * chunk )
```

Definition at line 2459 of file lodepng.cpp.

```
2460 {
2461   return &chunk[8];
2462 }
```

Here is the caller graph for this function:



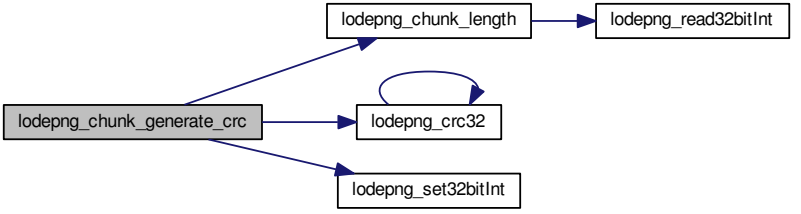
4.1.3.80 lodepng_chunk_generate_crc()

```
void lodepng_chunk_generate_crc (
            unsigned char * chunk )
```

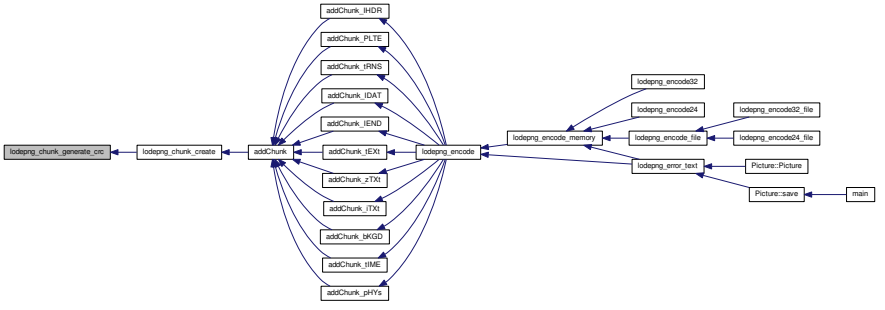
Definition at line 2474 of file lodepng.cpp.

```
2475 {
2476   unsigned length = lodepng_chunk_length(chunk);
2477   unsigned CRC = lodepng_crc32(&chunk[4], length + 4);
2478   lodepng_set32bitInt(chunk + 8 + length, CRC);
2479 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.81 lodepng_chunk_length()

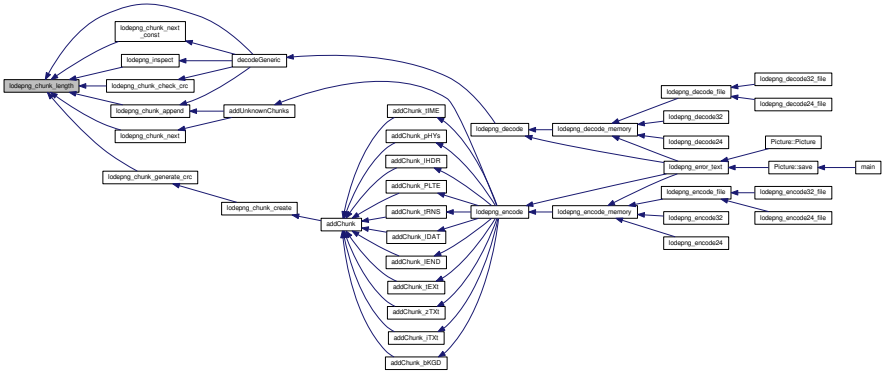
```
unsigned lodepng_chunk_length (
            const unsigned char * chunk )
```

Definition at line 2421 of file lodepng.cpp.

```
2422 {
2423   return lodepng_read32bitInt(&chunk[0]);
2424 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



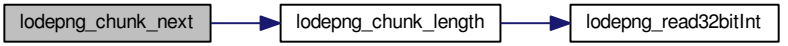
4.1.3.82 lodepng_chunk_next()

```
unsigned char* lodepng_chunk_next (
            unsigned char * chunk )
```

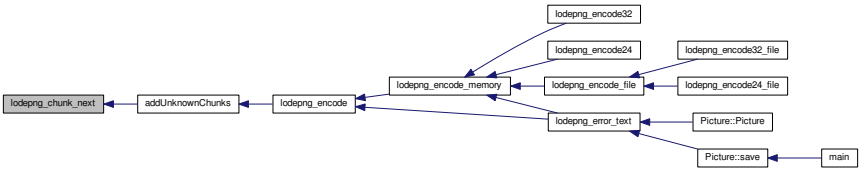
Definition at line 2481 of file lodepng.cpp.

```
2482 {
2483   unsigned total_chunk_length = lodepng_chunk_length(chunk) + 12;
2484   return &chunk[total_chunk_length];
2485 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



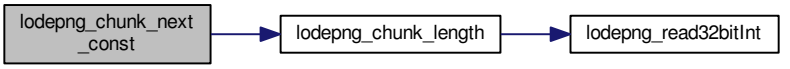
4.1.3.83 lodepng_chunk_next_const()

```
const unsigned char* lodepng_chunk_next_const (
            const unsigned char * chunk )
```

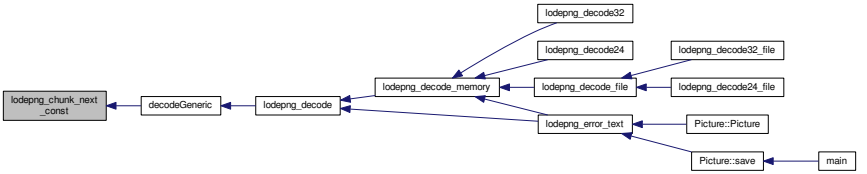
Definition at line 2487 of file lodepng.cpp.

```
2488 {
2489   unsigned total_chunk_length = lodepng_chunk_length(chunk) + 12;
2490   return &chunk[total_chunk_length];
2491 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.84 lodepng_chunk_private()

```
unsigned char lodepng_chunk_private (
            const unsigned char * chunk )
```

Definition at line 2444 of file lodepng.cpp.

```
2445 {
2446   return((chunk[6] & 32) != 0);
2447 }
```

4.1.3.85 lodepng_chunk_safetocopy()

```
unsigned char lodepng_chunk_safetocopy (
            const unsigned char * chunk )
```

Definition at line 2449 of file lodepng.cpp.

```
2450 {
2451   return((chunk[7] & 32) != 0);
2452 }
```

4.1.3.86 lodepng_chunk_type()

```
void lodepng_chunk_type (
            char type[5],
            const unsigned char * chunk )
```

Definition at line 2426 of file lodepng.cpp.

```
2427 {
2428   unsigned i;
2429   for(i = 0; i != 4; ++i) type[i] = (char)chunk[4 + i];
2430   type[4] = 0; /*null termination char*/
2431 }
```

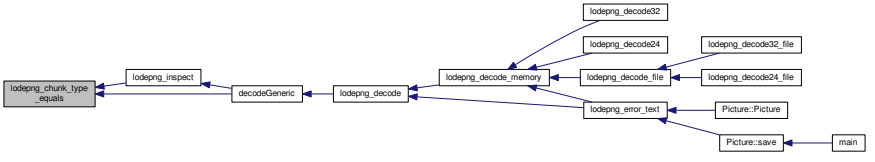
4.1.3.87 lodepng_chunk_type_equals()

```
unsigned char lodepng_chunk_type_equals (
            const unsigned char * chunk,
            const char * type )
```

Definition at line 2433 of file lodepng.cpp.

```
2434 {
2435   if(strlen(type) != 4) return 0;
2436   return (chunk[4] == type[0] && chunk[5] == type[1] && chunk[6] == type[2] &&
2437 }
```

Here is the caller graph for this function:



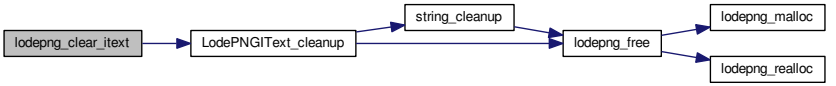
4.1.3.88 lodepng_clear_itext()

```
void lodepng_clear_itext (
            LodePNGInfo * info )
```

Definition at line 2880 of file lodepng.cpp.

```
2881 {
2882   LodePNGIText_cleanup(info);
2883 }
```

Here is the call graph for this function:



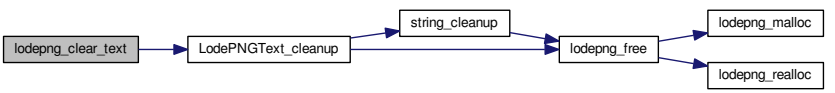
4.1.3.89 lodepng_clear_text()

```
void lodepng_clear_text (
            LodePNGInfo * info )
```

Definition at line 2808 of file lodepng.cpp.

```
2809 {
2810   LodePNGText_cleanup(info);
2811 }
```

Here is the call graph for this function:



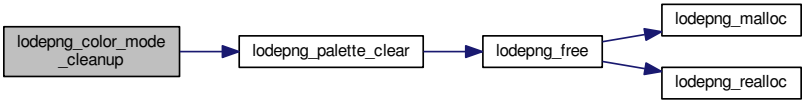
4.1.3.90 lodepng_color_mode_cleanup()

```
void lodepng_color_mode_cleanup (
            LodePNGColorMode * info )
```

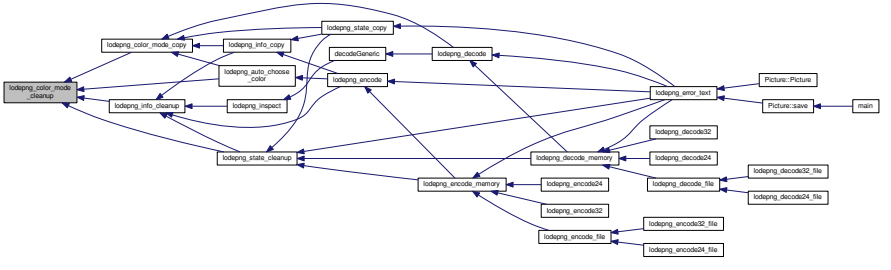
Definition at line 2593 of file lodepng.cpp.

```
2594 {
2595   lodepng_palette_clear(info);
2596 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



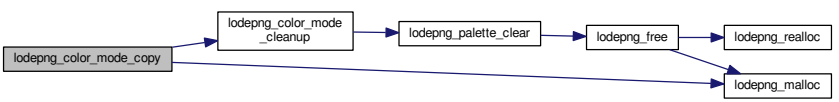
4.1.3.91 lodepng_color_mode_copy()

```
unsigned lodepng_color_mode_copy (
            LodePNGColorMode * dest,
            const LodePNGColorMode * source )
```

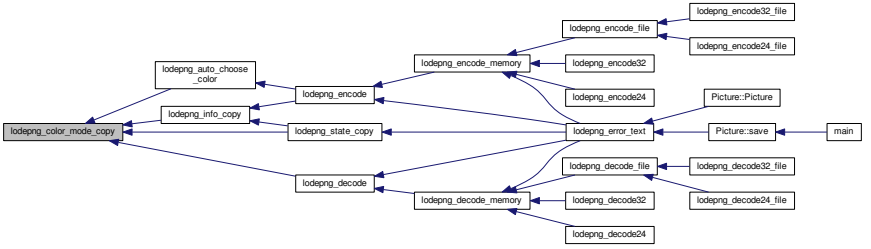
Definition at line 2598 of file lodepng.cpp.

```
2599 {
2600   size_t i;
2601   lodepng_color_mode_cleanup(dest);
2602   *dest = *source;
2603   if(source->palette)
2604   {
2605     dest->palette = (unsigned char*)lodepng_malloc(1024);
2606     if(!dest->palette && source->palettesize) return 83; /*alloc fail*/
2607     for(i = 0; i != source->palettesize * 4; ++i) dest->palette[i] = source->
     palette[i];
2608   }
2609   return 0;
2610 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.92 lodepng_color_mode_equal()

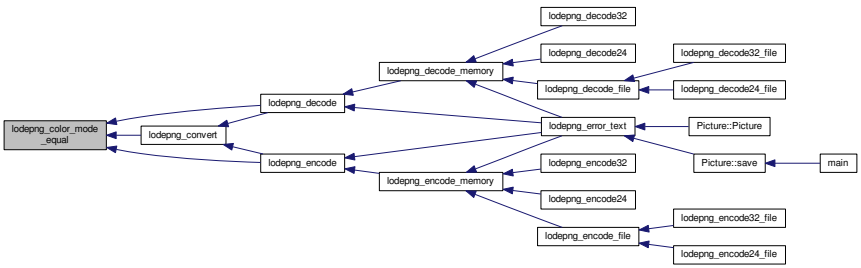
```
static int lodepng_color_mode_equal (
            const LodePNGColorMode * a,
            const LodePNGColorMode * b )  [static]
```

Definition at line 2612 of file lodepng.cpp.

```
2613 {
2614   size_t i;
2615   if(a->colortype != b->colortype) return 0;
2616   if(a->bitdepth != b->bitdepth) return 0;
2617   if(a->key_defined != b->key_defined) return 0;
2618   if(a->key_defined)
2619   {
2620     if(a->key_r != b->key_r) return 0;
2621     if(a->key_g != b->key_g) return 0;
```

```
2622     if(a->key_b != b->key_b) return 0;
2623   }
2624   /*if one of the palette sizes is 0, then we consider it to be the same as the
2625   other: it means that e.g. the palette was not given by the user and should be
2626   considered the same as the palette inside the PNG.*/
2627   if(1/*a->palettesize != 0 && b->palettesize != 0*/) {
2628     if(a->palettesize != b->palettesize) return 0;
2629     for(i = 0; i != a->palettesize * 4; ++i)
2630     {
2631       if(a->palette[i] != b->palette[i]) return 0;
2632     }
2633   }
2634   return 1;
2635 }
```

Here is the caller graph for this function:



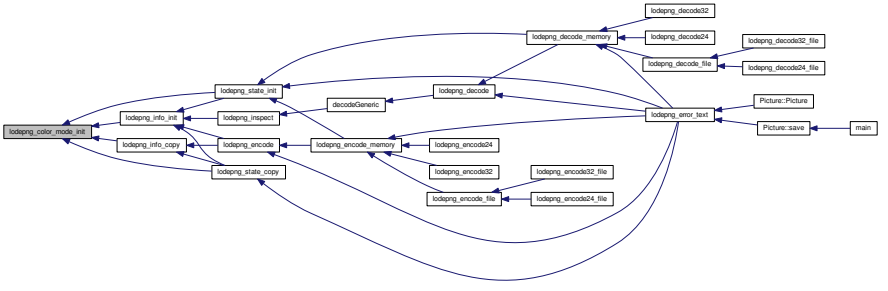
4.1.3.93 lodepng_color_mode_init()

```
void lodepng_color_mode_init (
            LodePNGColorMode * info )
```

Definition at line 2583 of file lodepng.cpp.

```
2584 {
2585   info->key_defined = 0;
2586   info->key_r = info->key_g = info->key_b = 0;
2587   info->colortype = LCT_RGBA;
2588   info->bitdepth = 8;
2589   info->palette = 0;
2590   info->palettesize = 0;
2591 }
```

Here is the caller graph for this function:



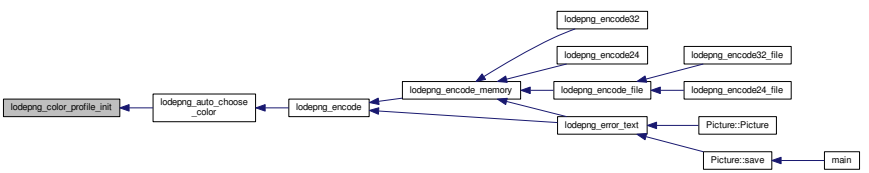
4.1.3.94 lodepng_color_profile_init()

```
void lodepng_color_profile_init (
            LodePNGColorProfile * profile )
```

Definition at line 3533 of file lodepng.cpp.

```
3534 {
3535   profile->colored = 0;
3536   profile->key = 0;
3537   profile->key_r = profile->key_g = profile->key_b = 0;
3538   profile->alpha = 0;
3539   profile->numcolors = 0;
3540   profile->bits = 1;
3541 }
```

Here is the caller graph for this function:



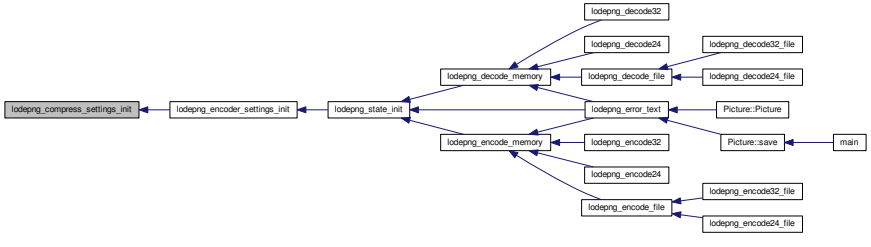
4.1.3.95 lodepng_compress_settings_init()

```
void lodepng_compress_settings_init (
            LodePNGCompressSettings * settings )
```

Definition at line 2273 of file lodepng.cpp.

```
2274 {
2275   /*compress with dynamic huffman tree (not in the mathematical sense, just not
2276   settings->btype = 2;
2277   settings->use_lz77 = 1;
2278   settings->windowsize = DEFAULT_WINDOWSIZE;
2279   settings->minmatch = 3;
2280   settings->nicematch = 128;
2281   settings->lazymatching = 1;
2282
2283   settings->custom_zlib = 0;
2284   settings->custom_deflate = 0;
2285   settings->custom_context = 0;
2286 }
```

Here is the caller graph for this function:



4.1.3.96 lodepng_convert()

```
unsigned lodepng_convert (
            unsigned char * out,
            const unsigned char * in,
            const LodePNGColorMode * mode_out,
            const LodePNGColorMode * mode_in,
            unsigned w,
            unsigned h )
```

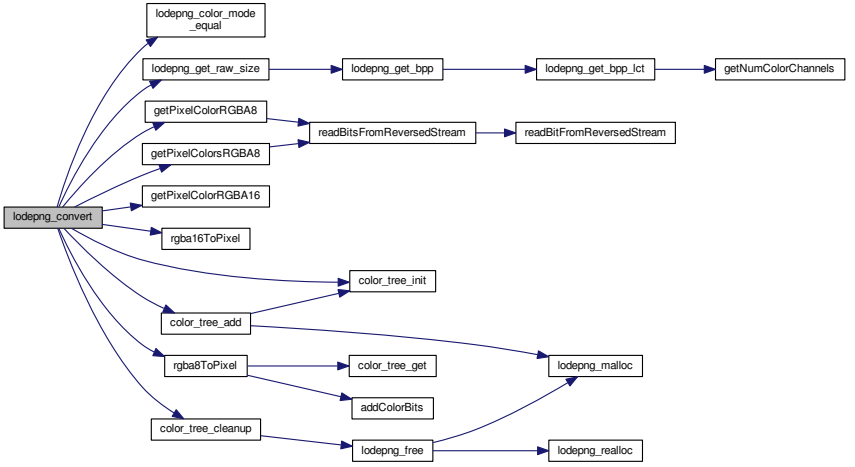
Definition at line 3459 of file lodepng.cpp.

```
3462 {
3463   size_t i;
3464   ColorTree tree;
3465   size_t numpixels = w * h;
3466
3467   if(lodepng_color_mode_equal(mode_out, mode_in))
3468   {
3469     size_t numbytes = lodepng_get_raw_size(w, h, mode_in);
3470     for(i = 0; i != numbytes; ++i) out[i] = in[i];
3471     return 0;
3472   }
3473
3474   if(mode_out->colortype == LCT_PALETTE)
3475   {
3476     size_t palettesize = mode_out->palettesize;
3477     const unsigned char* palette = mode_out->palette;
3478     size_t palsize = 1u << mode_out->bitdepth;
3479     /*if the user specified output palette but did not give the values, assume
3480     they want the values of the input color type (assuming that one is palette)
3481     Note that we never create a new palette ourselves.*/
```

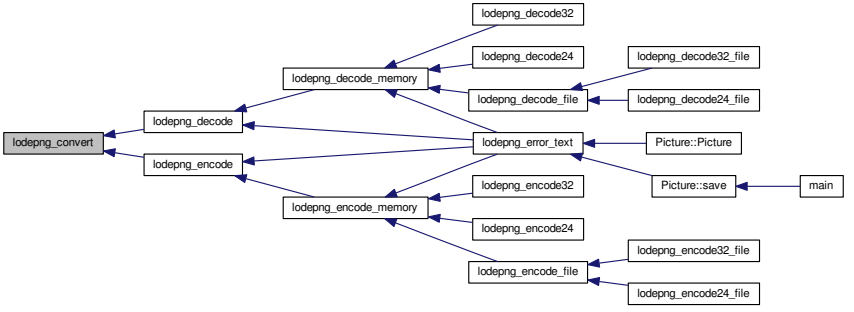
```
3482     if(palettesize == 0)
3483     {
3484       palettesize = mode_in->palettesize;
3485       palette = mode_in->palette;
3486     }
3487     if(palettesize < palsize) palsize = palettesize;
3488     color_tree_init(&tree);
3489     for(i = 0; i != palsize; ++i)
3490     {
3491       const unsigned char* p = &palette[i * 4];
3492       color_tree_add(&tree, p[0], p[1], p[2], p[3], i);
3493     }
3494   }
3495
3496   if(mode_in->bitdepth == 16 && mode_out->bitdepth == 16)
3497   {
3498     for(i = 0; i != numpixels; ++i)
3499     {
3500       unsigned short r = 0, g = 0, b = 0, a = 0;
3501       getPixelColorRGBA16(&r, &g, &b, &a, in, i, mode_in);
3502       rgba16ToPixel(out, i, mode_out, r, g, b, a);
3503     }
3504   }
3505   else if(mode_out->bitdepth == 8 && mode_out->colortype ==
     LCT_RGBA)
3506   {
3507     getPixelColorsRGBA8(out, numpixels, 1, in, mode_in);
3508   }
3509   else if(mode_out->bitdepth == 8 && mode_out->colortype ==
     LCT_RGB)
3510   {
```

```
3511     getPixelColorsRGBA8(out, numpixels, 0, in, mode_in);
3512   }
3513   else
3514   {
3515     unsigned char r = 0, g = 0, b = 0, a = 0;
3516     for(i = 0; i != numpixels; ++i)
3517     {
3518       getPixelColorRGBA8(&r, &g, &b, &a, in, i, mode_in);
3519       CERROR_TRY_RETURN(rgba8ToPixel(out, i, mode_out, &tree, r, g, b, a));
3520     }
3521   }
3522
3523   if(mode_out->colortype == LCT_PALETTE)
3524   {
3525     color_tree_cleanup(&tree);
3526   }
3527
3528   return 0; /*no error*/
3529 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



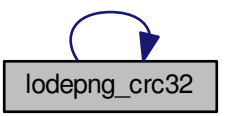
4.1.3.97 lodepng_crc32()

```
unsigned lodepng_crc32 (
            const unsigned char * data,
            size_t length )
```

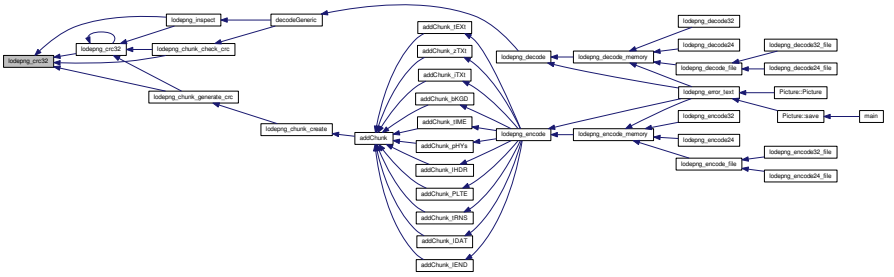
Definition at line 2359 of file lodepng.cpp.

```
2360 {
2361   unsigned r = 0xffffffffu;
2362   size_t i;
2363   for(i = 0; i < length; ++i)
2364   {
2365     r = lodepng_crc32_table[(r ^ data[i]) & 0xff] ^ (r >> 8);
2366   }
2367   return r ^ 0xffffffffu;
2368 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.98 lodepng_decode()

```
unsigned lodepng_decode (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            LodePNGState * state,
            const unsigned char * in,
            size_t insize )
```

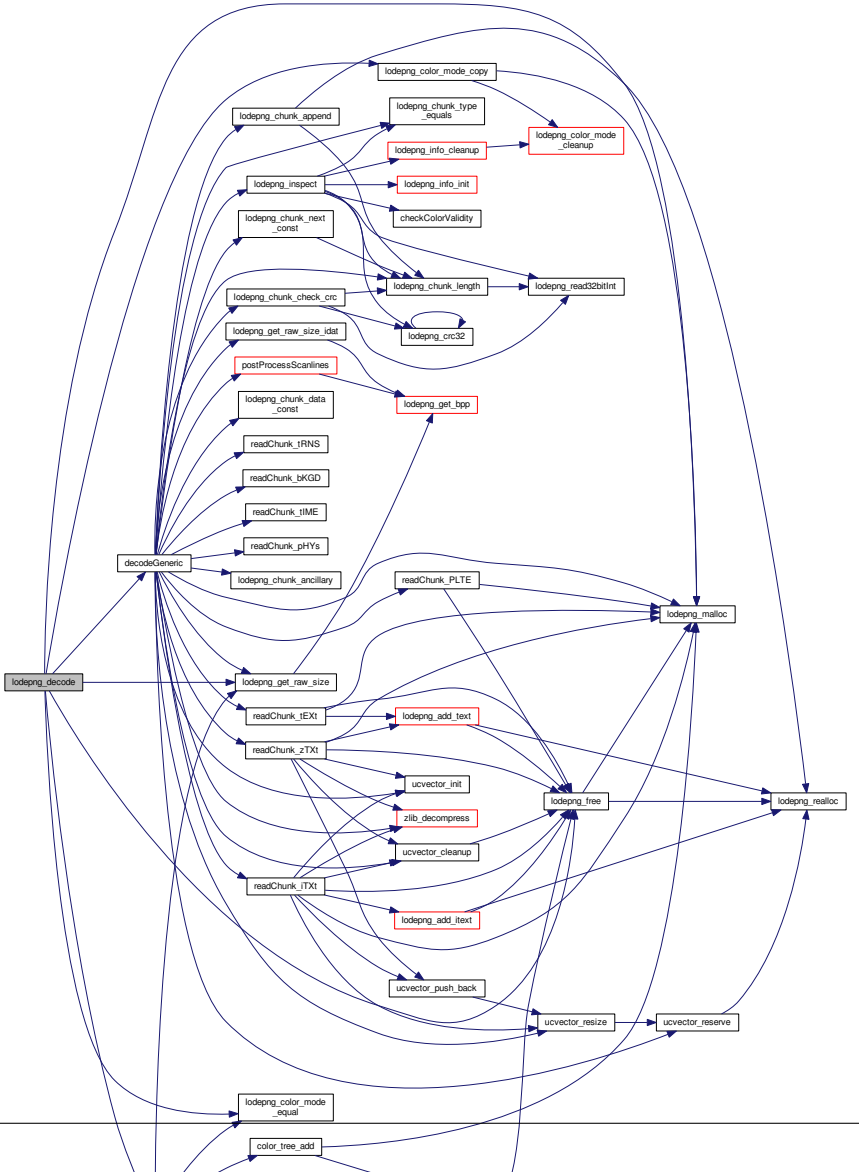
Definition at line 4723 of file lodepng.cpp.

```
4726 {
4727   *out = 0;
4728   decodeGeneric(out, w, h, state, in, insize);
4729   if(state->error) return state->error;
4730   if(!state->decoder.color_convert ||
```

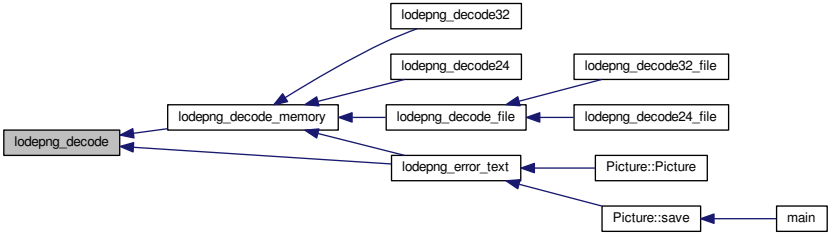
```
    lodepng_color_mode_equal(&state->info_raw, &state->
    info_png.color))
4731  {
4732    /*same color type, no copying or converting of data needed*/
4733    /*store the info_png color settings on the info_raw so that the info_raw st
4734    the raw image has to the end user*/
4735    if(!state->decoder.color_convert)
4736    {
4737      state->error = lodepng_color_mode_copy(&state->
    info_raw, &state->info_png.color);
4738      if(state->error) return state->error;
4739    }
4740  }
4741  else
4742  {
4743    /*color conversion needed; sort of copy of the data*/
4744    unsigned char* data = *out;
4745    size_t outsize;
4746
4747    /*TODO: check if this works according to the statement in the documentation
4748    from greyscale input color type, to 8-bit greyscale or greyscale with alpha
4749    if(!(state->info_raw.colortype == LCT_RGB || state->
    info_raw.colortype == LCT_RGBA)
4750       && !(state->info_raw.bitdepth == 8))
4751    {
4752      return 56; /*unsupported color mode conversion*/
4753    }
4754
4755    outsize = lodepng_get_raw_size(*w, *h, &state->info_raw);
4756    *out = (unsigned char*)lodepng_malloc(outsize);
4757    if(!(*out))
```

```
4758     {
4759       state->error = 83; /*alloc fail*/
4760     }
4761     else state->error = lodepng_convert(*out, data, &state->
      info_raw,
4762                                         &state->info_png.color, *w, *h);
4763     lodepng_free(data);
4764   }
4765   return state->error;
4766 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



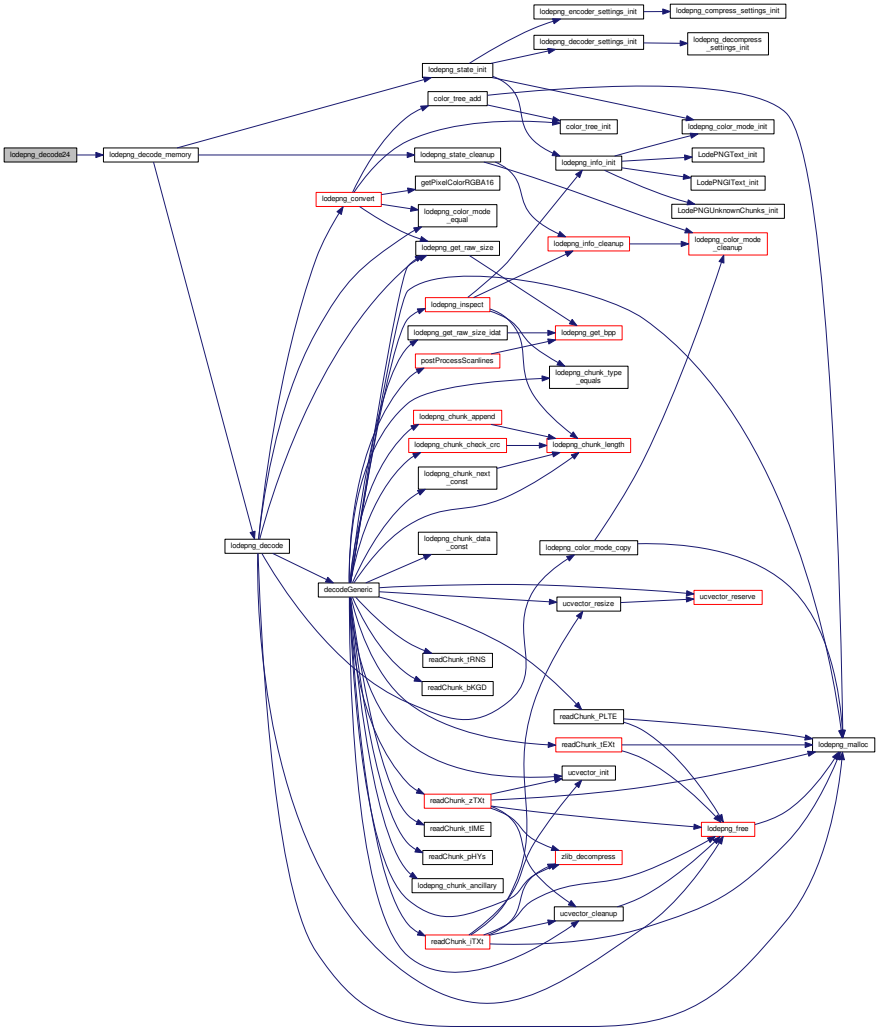
4.1.3.99 lodepng_decode24()

```
unsigned lodepng_decode24 (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const unsigned char * in,
            size_t insize )
```

Definition at line 4786 of file lodepng.cpp.

```
4787 {
4788   return lodepng_decode_memory(out, w, h, in, insize,
      LCT_RGB, 8);
4789 }
```

Here is the call graph for this function:



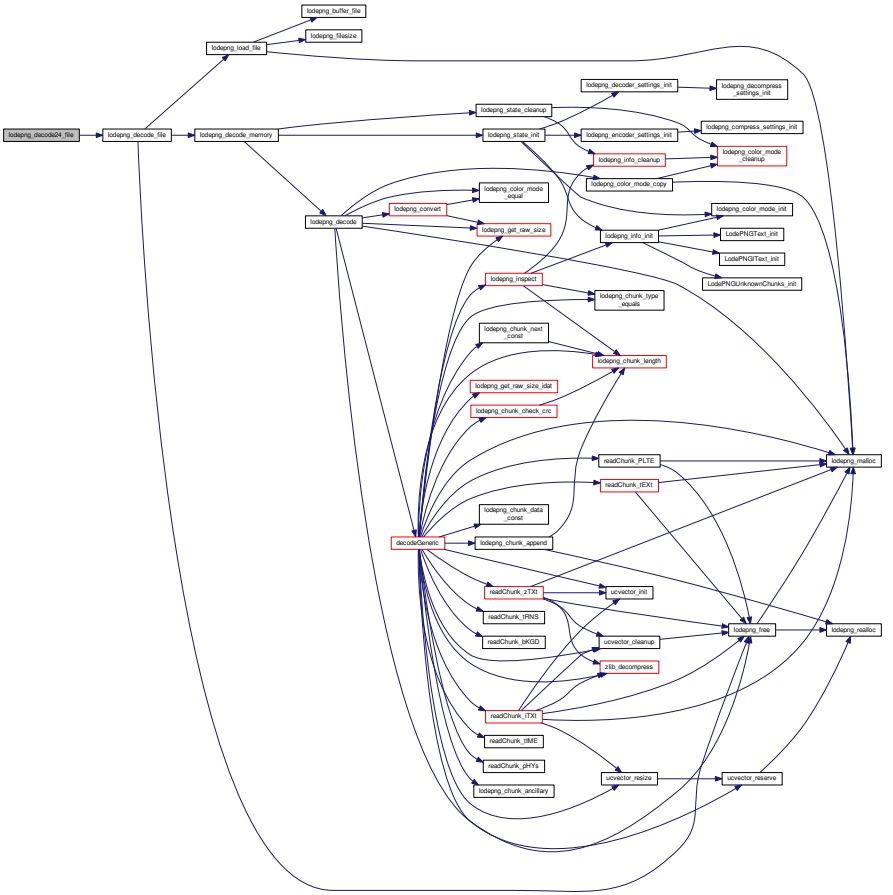
4.1.3.100 lodepng_decode24_file()

```
unsigned lodepng_decode24_file (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const char * filename )
```

Definition at line 4809 of file lodepng.cpp.

```
4810 {
4811   return lodepng_decode_file(out, w, h, filename, LCT_RGB, 8);
4812 }
```

Here is the call graph for this function:



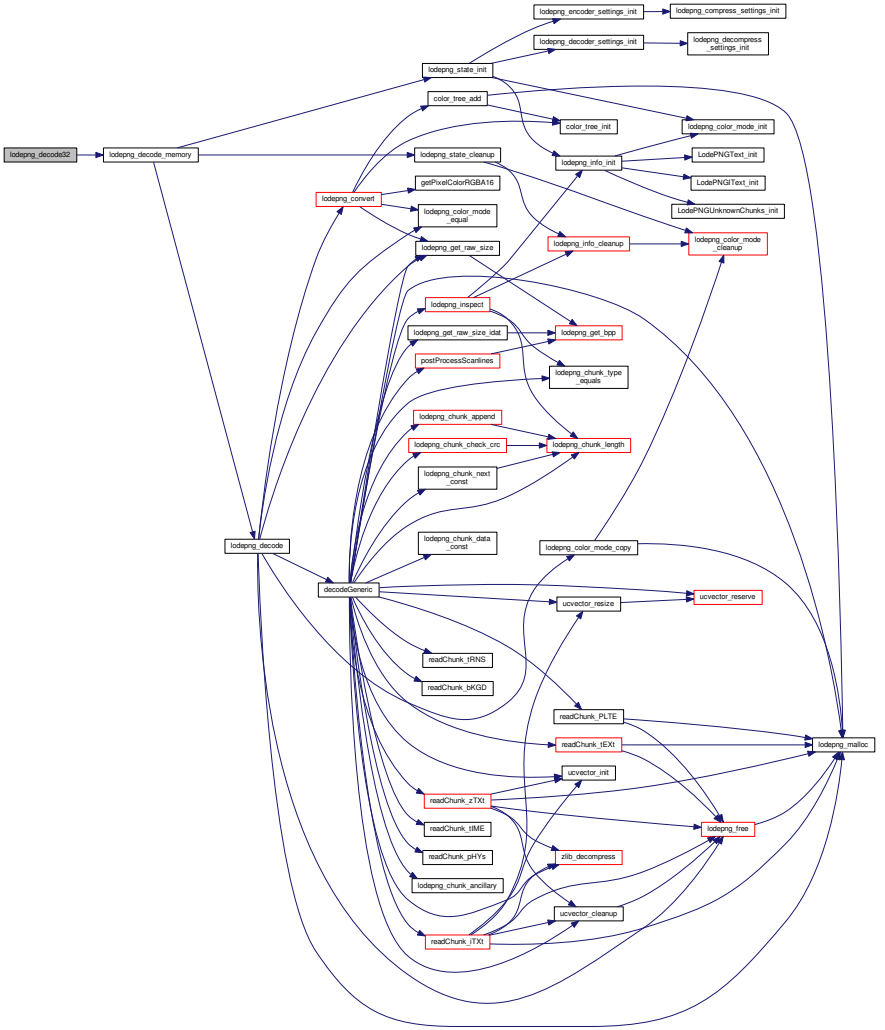
4.1.3.101 lodepng_decode32()

```
unsigned lodepng_decode32 (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const unsigned char * in,
            size_t insize )
```

Definition at line 4781 of file lodepng.cpp.

```
4782 {
4783   return lodepng_decode_memory(out, w, h, in, insize,
      LCT_RGBA, 8);
4784 }
```

Here is the call graph for this function:



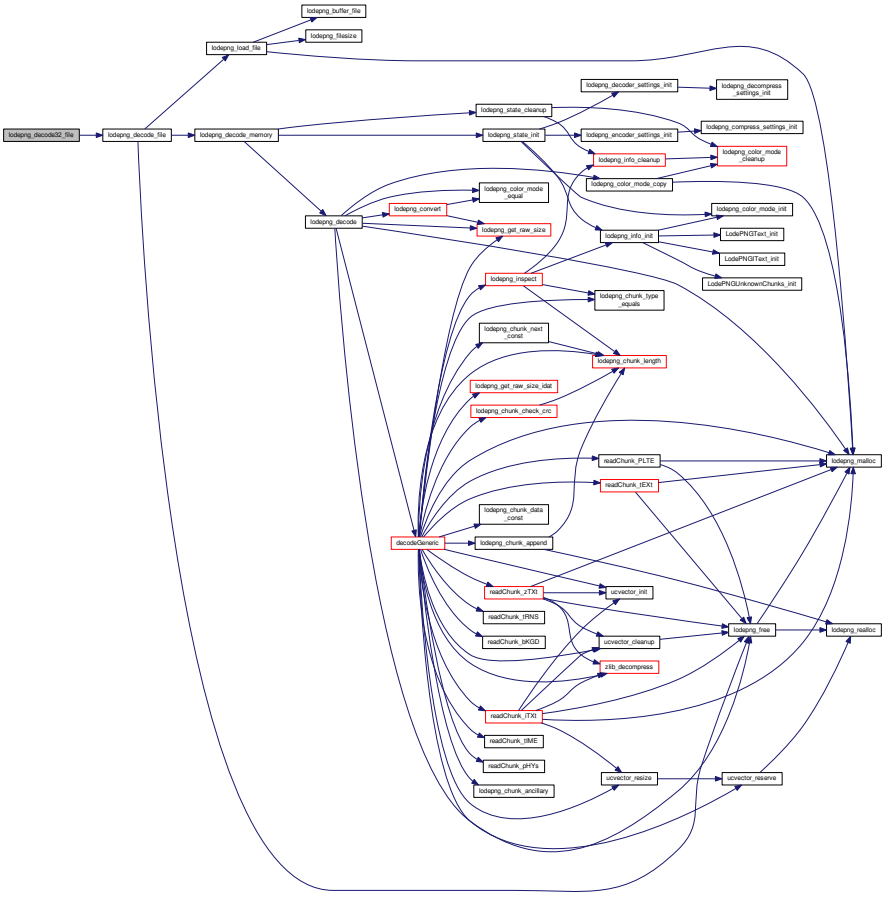
4.1.3.102 lodepng_decode32_file()

```
unsigned lodepng_decode32_file (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const char * filename )
```

Definition at line 4804 of file lodepng.cpp.

```
4805 {
4806   return lodepng_decode_file(out, w, h, filename, LCT_RGBA, 8);
4807 }
```

Here is the call graph for this function:



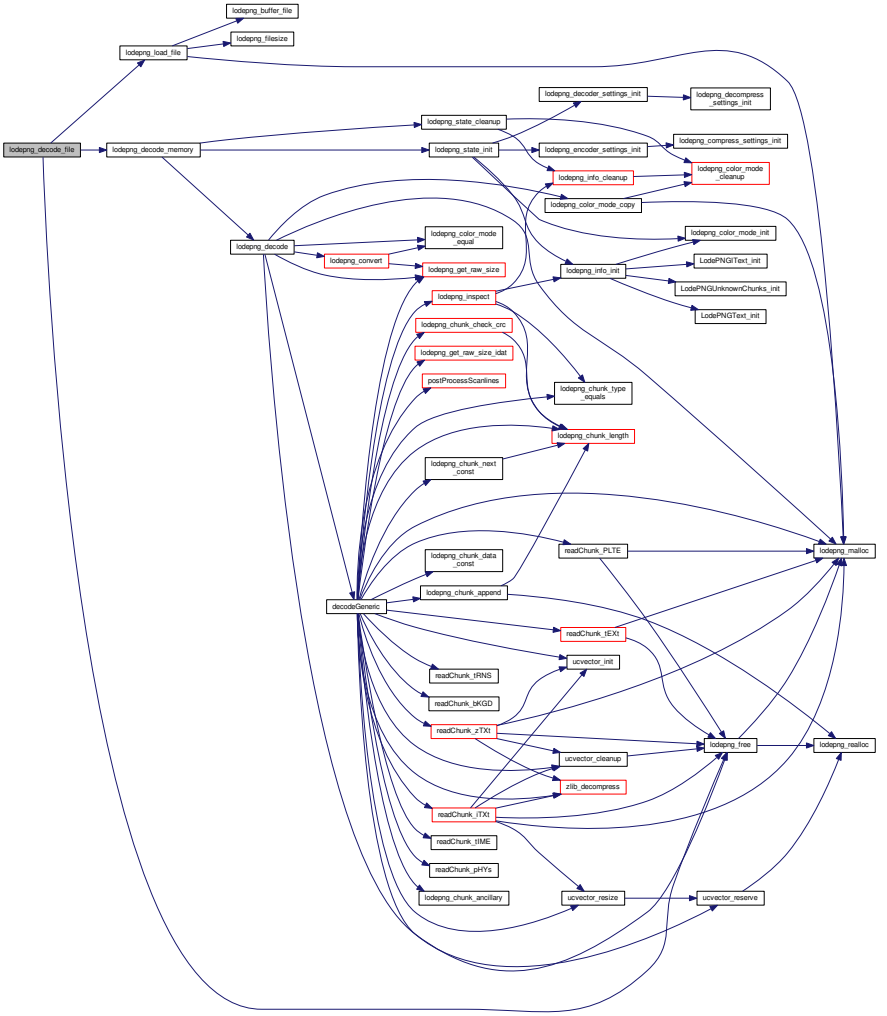
4.1.3.103 lodepng_decode_file()

```
unsigned lodepng_decode_file (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const char * filename,
            LodePNGColorType colortype,
            unsigned bitdepth )
```

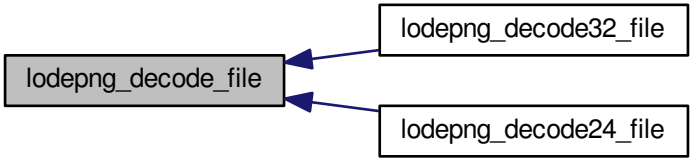
Definition at line 4792 of file lodepng.cpp.

```
4794 {
4795   unsigned char* buffer = 0;
4796   size_t buffersize;
4797   unsigned error;
4798   error = lodepng_load_file(&buffer, &buffersize, filename);
4799   if(!error) error = lodepng_decode_memory(out, w, h, buffer, buffersize, color
      bitdepth);
4800   lodepng_free(buffer);
4801   return error;
4802 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.104 lodepng_decode_memory()

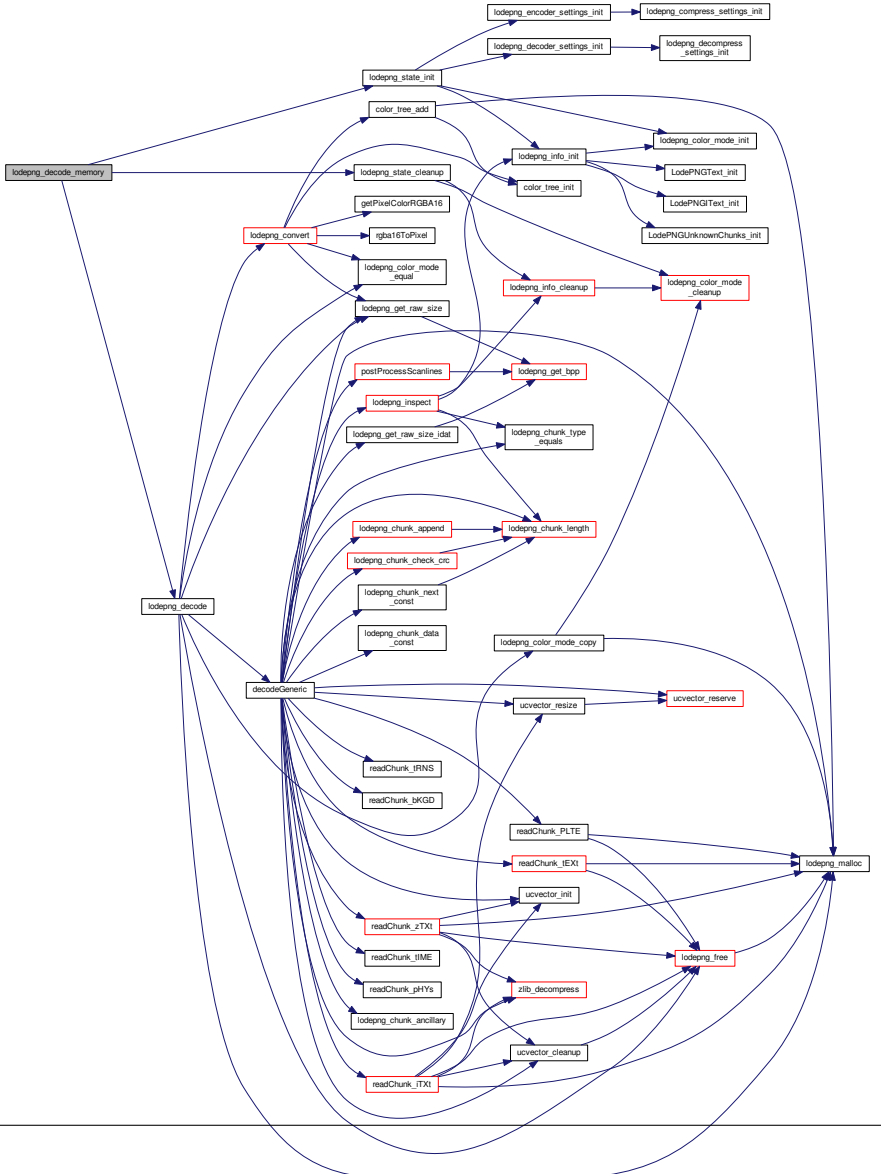
```
unsigned lodepng_decode_memory (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const unsigned char * in,
            size_t insize,
            LodePNGColorType colortype,
            unsigned bitdepth )
```

Definition at line 4768 of file lodepng.cpp.

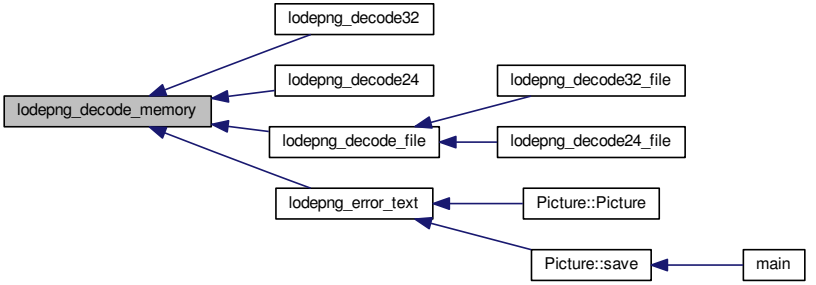
```
4770 {
4771   unsigned error;
4772   LodePNGState state;
```

```
4773   lodepng_state_init(&state);
4774   state.info_raw.colortype = colortype;
4775   state.info_raw.bitdepth = bitdepth;
4776   error = lodepng_decode(out, w, h, &state, in, insize);
4777   lodepng_state_cleanup(&state);
4778   return error;
4779 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.105 lodepng_decoder_settings_init()

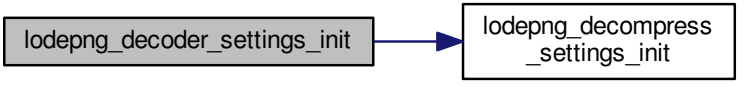
```
void lodepng_decoder_settings_init (
            LodePNGDecoderSettings * settings )
```

Definition at line 4815 of file lodepng.cpp.

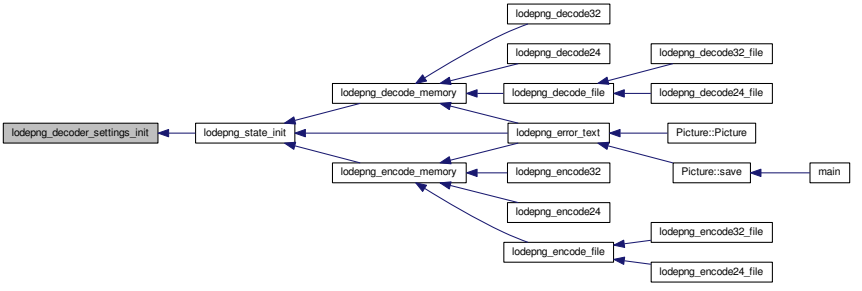
```
4816 {
4817   settings->color_convert = 1;
4818 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
4819   settings->read_text_chunks = 1;
4820   settings->remember_unknown_chunks = 0;
4821 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
4822   settings->ignore_crc = 0;
```

```
4823   lodepng_decompress_settings_init(&settings->
       zlibsettings);
4824 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



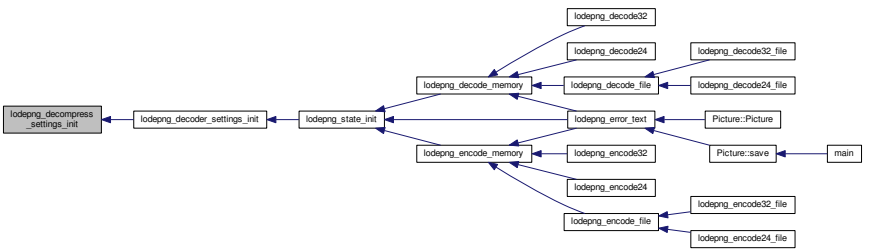
4.1.3.106 lodepng_decompress_settings_init()

```
void lodepng_decompress_settings_init (
            LodePNGDecompressSettings * settings )
```

Definition at line 2295 of file lodepng.cpp.

```
2296 {
2297   settings->ignore_adler32 = 0;
2298
2299   settings->custom_zlib = 0;
2300   settings->custom_inflate = 0;
2301   settings->custom_context = 0;
2302 }
```

Here is the caller graph for this function:



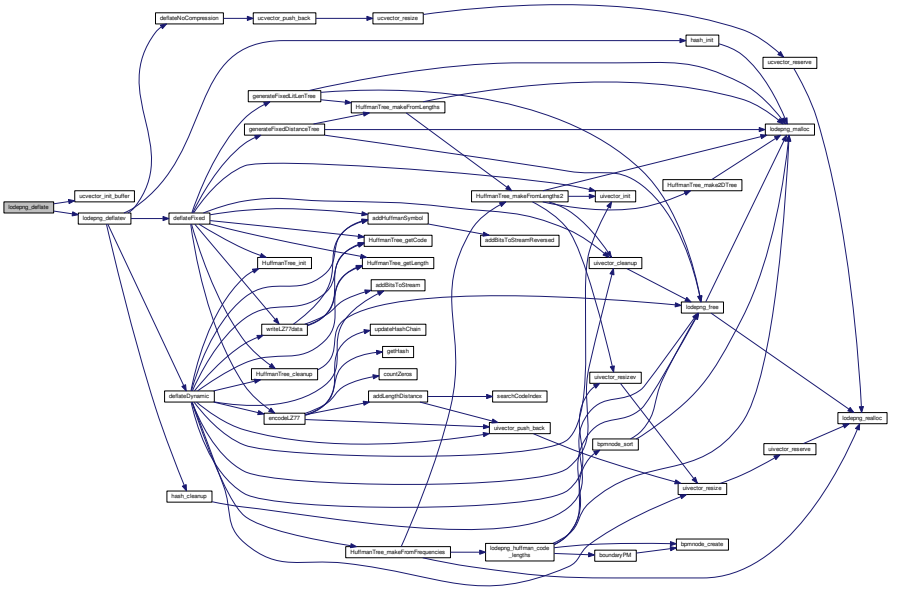
4.1.3.107 lodepng_deflate()

```
unsigned lodepng_deflate (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGCompressSettings * settings )
```

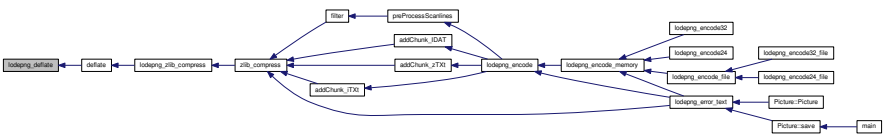
Definition at line 2058 of file lodepng.cpp.

```
2061 {
2062   unsigned error;
2063   ucvector v;
2064   ucvector_init_buffer(&v, *out, *outsize);
2065   error = lodepng_deflatev(&v, in, insize, settings);
2066   *out = v.data;
2067   *outsize = v.size;
2068   return error;
2069 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.108 lodepng_deflatev()

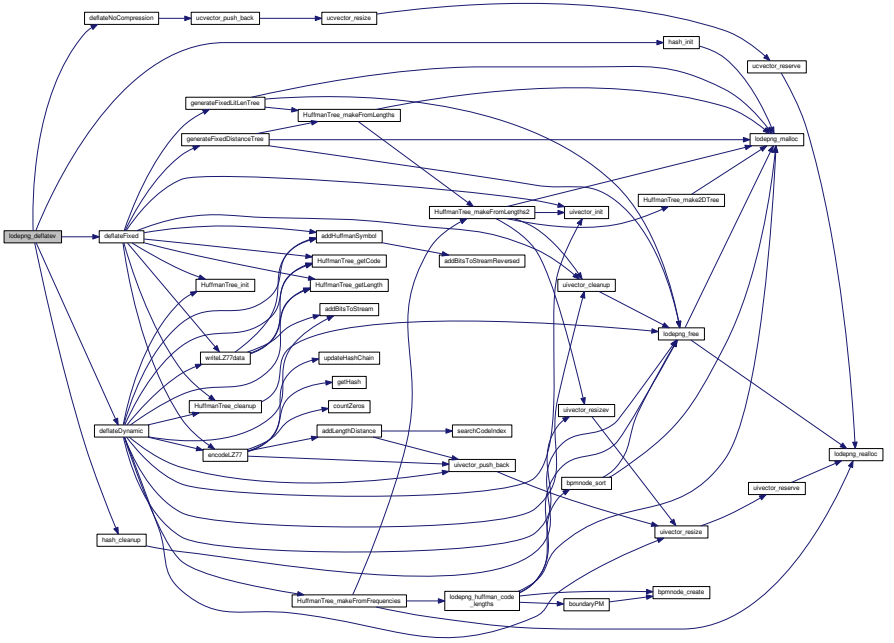
```
static unsigned lodepng_deflatev (
            ucvector * out,
            const unsigned char * in,
            size_t insize,
            const LodePNGCompressSettings * settings )  [static]
```

Definition at line 2017 of file lodepng.cpp.

```
2019 {
2020   unsigned error = 0;
2021   size_t i, blocksize, numdeflateblocks;
2022   size_t bp = 0; /*the bit pointer*/
2023   Hash hash;
2024
2025   if(settings->btype > 2) return 61;
2026   else if(settings->btype == 0) return deflateNoCompression(out, in, insize);
2027   else if(settings->btype == 1) blocksize = insize;
2028   else /*if(settings->btype == 2)*/
2029   {
2030     /*on PNGs, deflate blocks of 65-262k seem to give most dense encoding*/
2031     blocksize = insize / 8 + 8;
2032     if(blocksize < 65536) blocksize = 65536;
2033     if(blocksize > 262144) blocksize = 262144;
2034   }
2035
2036   numdeflateblocks = (insize + blocksize - 1) / blocksize;
2037   if(numdeflateblocks == 0) numdeflateblocks = 1;
2038
2039   error = hash_init(&hash, settings->windowsize);
2040   if(error) return error;
```

```
2041
2042   for(i = 0; i != numdeflateblocks && !error; ++i)
2043   {
2044     unsigned final = (i == numdeflateblocks - 1);
2045     size_t start = i * blocksize;
2046     size_t end = start + blocksize;
2047     if(end > insize) end = insize;
2048
2049     if(settings->btype == 1) error = deflateFixed(out, &bp, &hash, in, start, e
      settings, final);
2050     else if(settings->btype == 2) error = deflateDynamic(out, &bp, &hash, in, s
      , settings, final);
2051   }
2052
2053   hash_cleanup(&hash);
2054
2055   return error;
2056 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.3.109 lodepng_encode()

```
unsigned lodepng_encode (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * image,
            unsigned w,
            unsigned h,
            LodePNGState * state )
```

Definition at line 5640 of file lodepng.cpp.

```
5643 {
5644   LodePNGInfo info;
5645   ucvector outv;
5646   unsigned char* data = 0; /*uncompressed version of the IDAT chunk data*/
5647   size_t datasize = 0;
5648
5649   /*provide some proper output values if error will happen*/
5650   *out = 0;
5651   *outsize = 0;
5652   state->error = 0;
5653
5654   lodepng_info_init(&info);
5655   lodepng_info_copy(&info, &state->info_png);
5656
5657   if((info.color.colortype == LCT_PALETTE || state->
      encoder.force_palette)
5658       && (info.color.palettesize == 0 || info.color.
      palettesize > 256))
5659   {
5660     state->error = 68; /*invalid palette size, it is only allowed to be 1-256*/
```

```
5661     return state->error;
5662   }
5663
5664   if(state->encoder.auto_convert)
5665   {
5666     state->error = lodepng_auto_choose_color(&info.
      color, image, w, h, &state->info_raw);
5667   }
5668   if(state->error) return state->error;
5669
5670   if(state->encoder.zlibsettings.btype > 2)
5671   {
5672     CERROR_RETURN_ERROR(state->error, 61); /*error: unexisting btype*/
5673   }
5674   if(state->info_png.interlace_method > 1)
5675   {
5676     CERROR_RETURN_ERROR(state->error, 71); /*error: unexisting interlace mode*/
5677   }
5678
5679   state->error = checkColorValidity(info.color.
      colortype, info.color.bitdepth);
5680   if(state->error) return state->error; /*error: unexisting color type given*/
5681   state->error = checkColorValidity(state->info_raw.
      colortype, state->info_raw.bitdepth);
5682   if(state->error) return state->error; /*error: unexisting color type given*/
5683
5684   if(!lodepng_color_mode_equal(&state->info_raw, &info.
      color))
5685   {
5686     unsigned char* converted;
5687     size_t size = (w * h * (size_t)lodepng_get_bpp(&info.color) + 7) / 8;
```

```
5688
5689     converted = (unsigned char*)lodepng_malloc(size);
5690     if(!converted && size) state->error = 83; /*alloc fail*/
5691     if(!state->error)
5692     {
5693       state->error = lodepng_convert(converted, image, &info.
     color, &state->info_raw, w, h);
5694     }
5695     if(!state->error) preProcessScanlines(&data, &datasize, converted, w, h, &i
      &state->encoder);
5696     lodepng_free(converted);
5697   }
5698   else preProcessScanlines(&data, &datasize, image, w, h, &info, &state->
     encoder);
5699
5700   ucvector_init(&outv);
5701   while(!state->error) /*while only executed once, to break on error*/
5702   {
5703 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
5704     size_t i;
5705 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
5706     /*write signature and chunks*/
5707     writeSignature(&outv);
5708     /*IHDR*/
5709     addChunk_IHDR(&outv, w, h, info.color.colortype, info.
     color.bitdepth, info.interlace_method);
5710 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
5711     /*unknown chunks between IHDR and PLTE*/
5712     if(info.unknown_chunks_data[0])
5713     {
5714       state->error = addUnknownChunks(&outv, info.
```

```
      unknown_chunks_data[0], info.unknown_chunks_size[0]);
5715       if(state->error) break;
5716     }
5717 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
5718     /*PLTE*/
5719     if(info.color.colortype == LCT_PALETTE)
5720     {
5721       addChunk_PLTE(&outv, &info.color);
5722     }
5723     if(state->encoder.force_palette && (info.color.
      colortype == LCT_RGB || info.color.colortype ==
      LCT_RGBA))
5724     {
5725       addChunk_PLTE(&outv, &info.color);
5726     }
5727     /*tRNS*/
5728     if(info.color.colortype == LCT_PALETTE &&
      getPaletteTranslucency(info.color.palette, info.
      color.palettesize) != 0)
5729     {
5730       addChunk_tRNS(&outv, &info.color);
5731     }
5732     if((info.color.colortype == LCT_GREY || info.color.
      colortype == LCT_RGB) && info.color.key_defined)
5733     {
5734       addChunk_tRNS(&outv, &info.color);
5735     }
5736 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
5737     /*bKGD (must come between PLTE and the IDAt chunks*/
5738     if(info.background_defined) addChunk_bKGD(&outv, &info);
5739     /*pHYs (must come before the IDAT chunks)*/
```

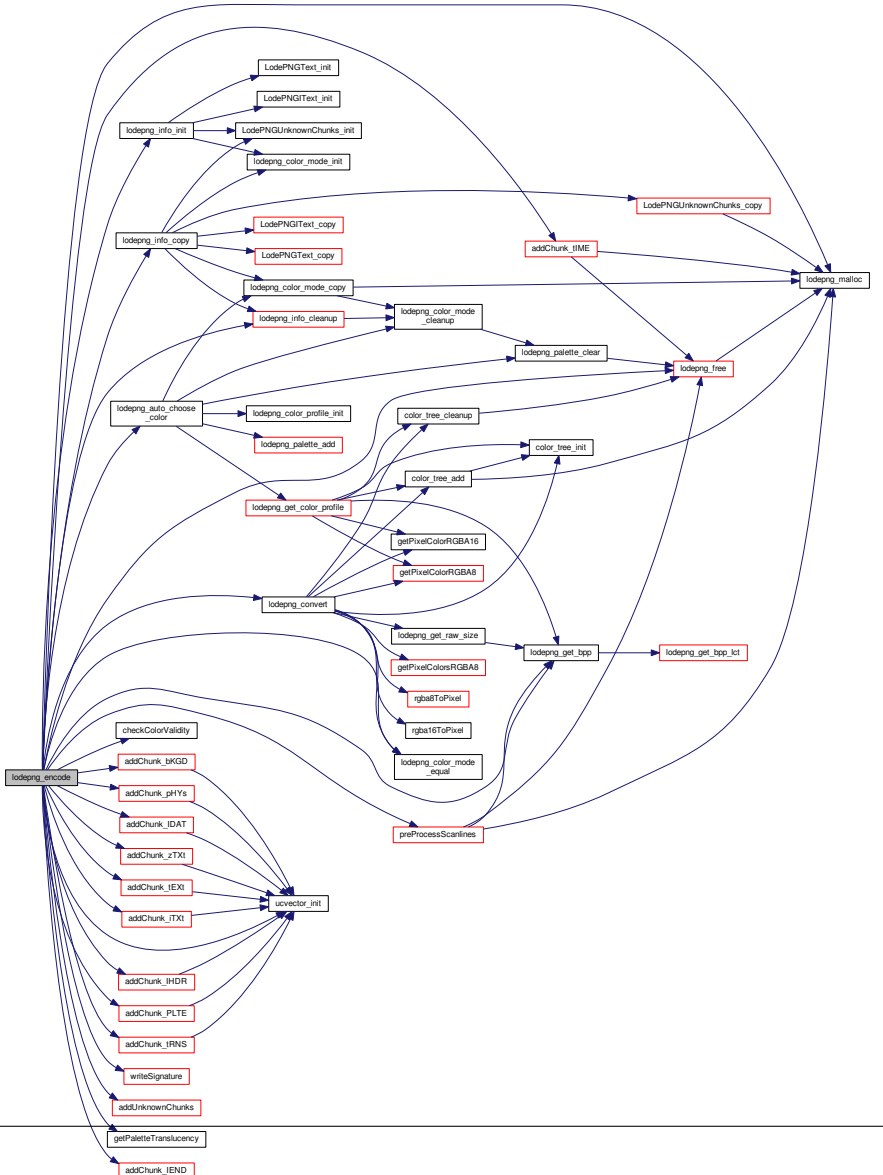
```
5740     if(info.phys_defined) addChunk_pHYs(&outv, &info);
5741
5742     /*unknown chunks between PLTE and IDAT*/
5743     if(info.unknown_chunks_data[1])
5744     {
5745       state->error = addUnknownChunks(&outv, info.
      unknown_chunks_data[1], info.unknown_chunks_size[1]);
5746       if(state->error) break;
5747     }
5748 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
5749     /*IDAT (multiple IDAT chunks must be consecutive)*/
5750     state->error = addChunk_IDAT(&outv, data, datasize, &state->
      encoder.zlibsettings);
5751     if(state->error) break;
5752 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
5753     /*tIME*/
5754     if(info.time_defined) addChunk_tIME(&outv, &info.
      time);
5755     /*tEXt and/or zTXt*/
5756     for(i = 0; i != info.text_num; ++i)
5757     {
5758       if(strlen(info.text_keys[i]) > 79)
5759       {
5760         state->error = 66; /*text chunk too large*/
5761         break;
5762       }
5763       if(strlen(info.text_keys[i]) < 1)
5764       {
5765         state->error = 67; /*text chunk too small*/
5766         break;
5767       }
```

```
5768       if(state->encoder.text_compression)
5769       {
5770         addChunk_zTXt(&outv, info.text_keys[i], info.
      text_strings[i], &state->encoder.zlibsettings);
5771       }
5772       else
5773       {
5774         addChunk_tEXt(&outv, info.text_keys[i], info.
      text_strings[i]);
5775       }
5776     }
5777     /*LodePNG version id in text chunk*/
5778     if(state->encoder.add_id)
5779     {
5780       unsigned alread_added_id_text = 0;
5781       for(i = 0; i != info.text_num; ++i)
5782       {
5783         if(!strcmp(info.text_keys[i], "LodePNG"))
5784         {
5785           alread_added_id_text = 1;
5786           break;
5787         }
5788       }
5789       if(alread_added_id_text == 0)
5790       {
5791         addChunk_tEXt(&outv, "LodePNG", LODEPNG_VERSION_STRING); /*it's
       shorter as tEXt than as zTXt chunk*/
5792       }
5793     }
5794     /*iTXt*/
5795     for(i = 0; i != info.itext_num; ++i)
```

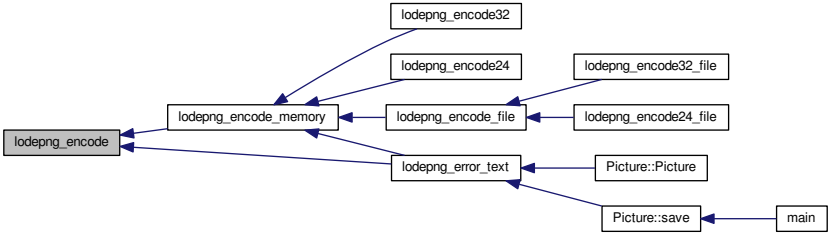
```
5796     {
5797       if(strlen(info.itext_keys[i]) > 79)
5798       {
5799         state->error = 66; /*text chunk too large*/
5800         break;
5801       }
5802       if(strlen(info.itext_keys[i]) < 1)
5803       {
5804         state->error = 67; /*text chunk too small*/
5805         break;
5806       }
5807       addChunk_iTXt(&outv, state->encoder.text_compression,
5808                     info.itext_keys[i], info.itext_langtags[i], info.
     itext_transkeys[i], info.itext_strings[i],
5809                     &state->encoder.zlibsettings);
5810     }
5811
5812     /*unknown chunks between IDAT and IEND*/
5813     if(info.unknown_chunks_data[2])
5814     {
5815       state->error = addUnknownChunks(&outv, info.
     unknown_chunks_data[2], info.unknown_chunks_size[2]);
5816       if(state->error) break;
5817     }
5818 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
5819     addChunk_IEND(&outv);
5820
5821     break; /*this isn't really a while loop; no error happened so break out now
5822   }
5823
5824   lodepng_info_cleanup(&info);
```

```
5825   lodepng_free(data);
5826   /*instead of cleaning the vector up, give it to the output*/
5827   *out = outv.data;
5828   *outsize = outv.size;
5829
5830   return state->error;
5831 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.110 lodepng_encode24()

```
unsigned lodepng_encode24 (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * image,
            unsigned w,
            unsigned h )
```

Definition at line 5854 of file lodepng.cpp.

```
5855 {
5856   return lodepng_encode_memory(out, outsize, image, w, h,
      LCT_RGB, 8);
5857 }
```

Here is the call graph for this function:

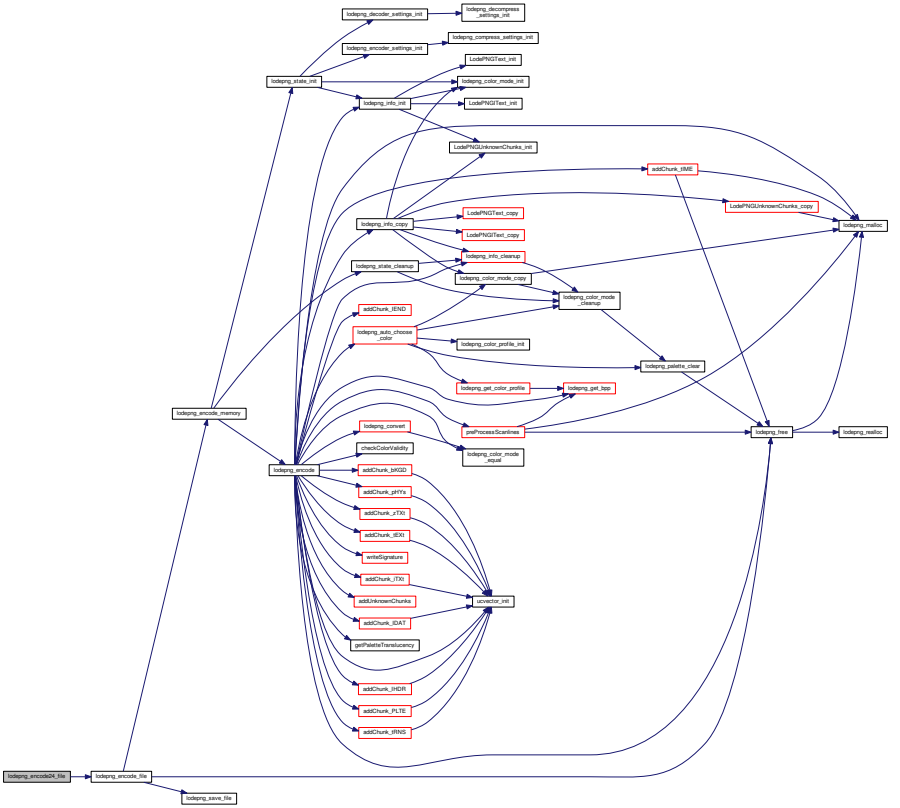
4.1.3.111 lodepng_encode24_file()

```
unsigned lodepng_encode24_file (
            const char * filename,
            const unsigned char * image,
            unsigned w,
            unsigned h )
```

Definition at line 5876 of file lodepng.cpp.

```
5877 {
5878   return lodepng_encode_file(filename, image, w, h, LCT_RGB, 8);
5879 }
```

Here is the call graph for this function:



4.1.3.112 lodepng_encode32()

```
unsigned lodepng_encode32 (
            unsigned char ** out,
```

```
    size_t * outsize,
    const unsigned char * image,
    unsigned w,
    unsigned h )
```

Definition at line 5849 of file lodepng.cpp.

```
5850 {
5851   return lodepng_encode_memory(out, outsize, image, w, h,
      LCT_RGBA, 8);
5852 }
```

Here is the call graph for this function:

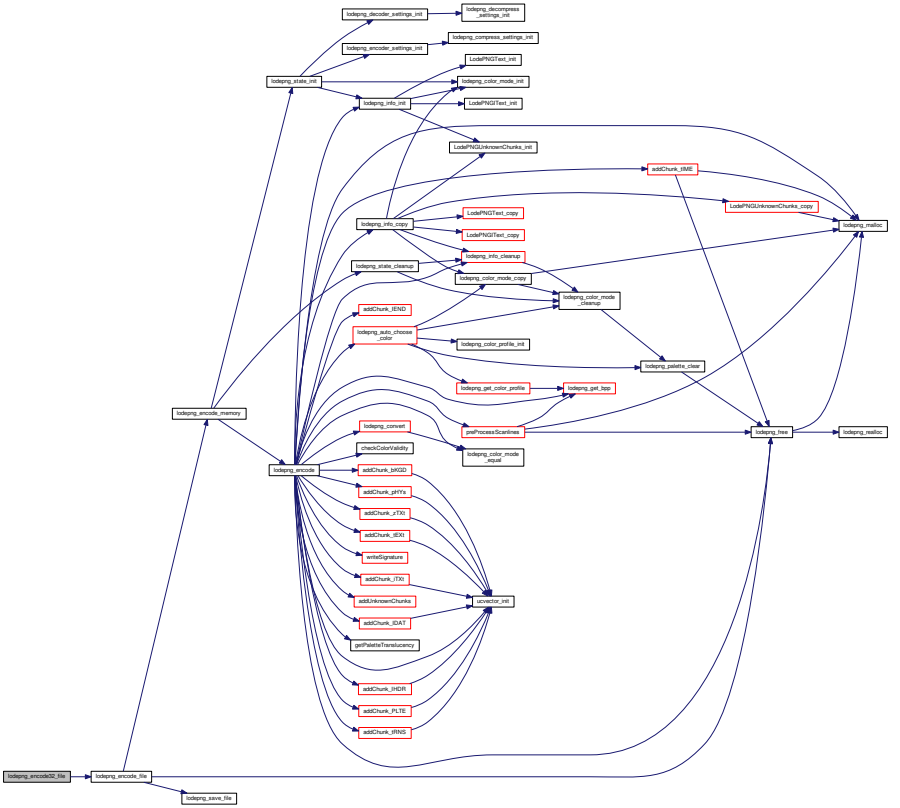
4.1.3.113 lodepng_encode32_file()

```
unsigned lodepng_encode32_file (
            const char * filename,
            const unsigned char * image,
            unsigned w,
            unsigned h )
```

Definition at line 5871 of file lodepng.cpp.

```
5872 {
5873   return lodepng_encode_file(filename, image, w, h, LCT_RGBA, 8);
5874 }
```

Here is the call graph for this function:



4.1.3.114 lodepng_encode_file()

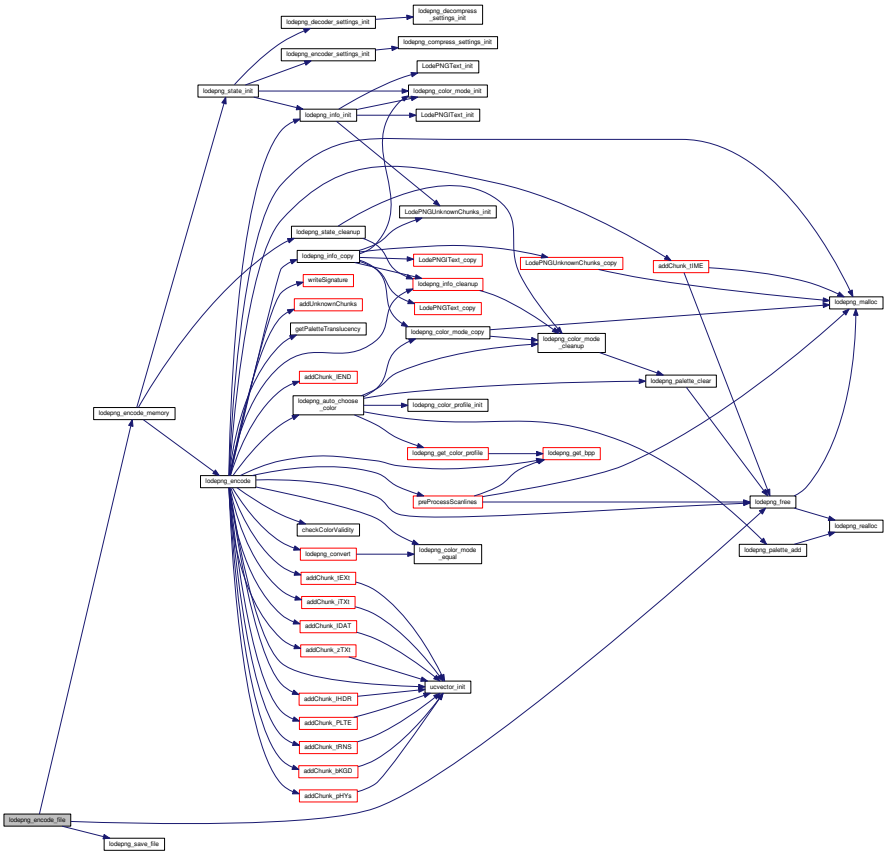
```
unsigned lodepng_encode_file (
            const char * filename,
```

```
    const unsigned char * image,
    unsigned w,
    unsigned h,
    LodePNGColorType colortype,
    unsigned bitdepth )
```

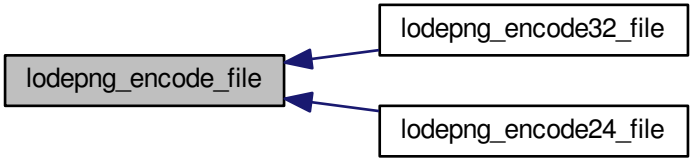
Definition at line 5860 of file lodepng.cpp.

```
5862 {
5863   unsigned char* buffer;
5864   size_t buffersize;
5865   unsigned error = lodepng_encode_memory(&buffer, &buffersize, image, w, h, col
       bitdepth);
5866   if(!error) error = lodepng_save_file(buffer, buffersize, filename);
5867   lodepng_free(buffer);
5868   return error;
5869 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.115 lodepng_encode_memory()

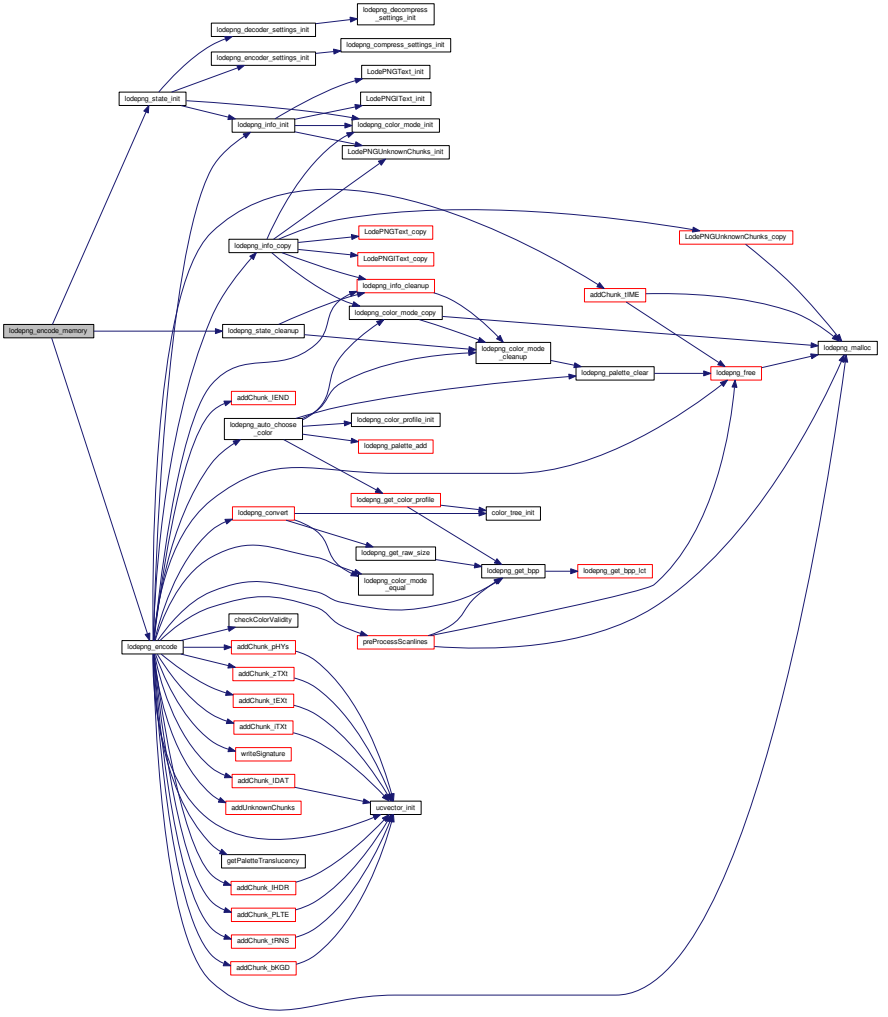
```
unsigned lodepng_encode_memory (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * image,
            unsigned w,
            unsigned h,
            LodePNGColorType colortype,
            unsigned bitdepth )
```

Definition at line 5833 of file lodepng.cpp.

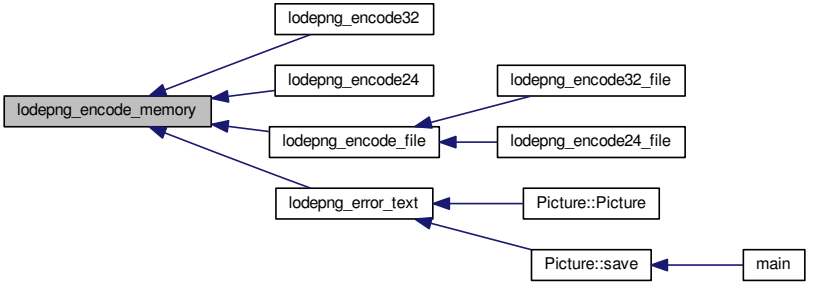
```
5835 {
5836   unsigned error;
5837   LodePNGState state;
```

```
5838   lodepng_state_init(&state);
5839   state.info_raw.colortype = colortype;
5840   state.info_raw.bitdepth = bitdepth;
5841   state.info_png.color.colortype = colortype;
5842   state.info_png.color.bitdepth = bitdepth;
5843   lodepng_encode(out, outsize, image, w, h, &state);
5844   error = state.error;
5845   lodepng_state_cleanup(&state);
5846   return error;
5847 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.116 lodepng_encoder_settings_init()

```
void lodepng_encoder_settings_init (
            LodePNGEncoderSettings * settings )
```

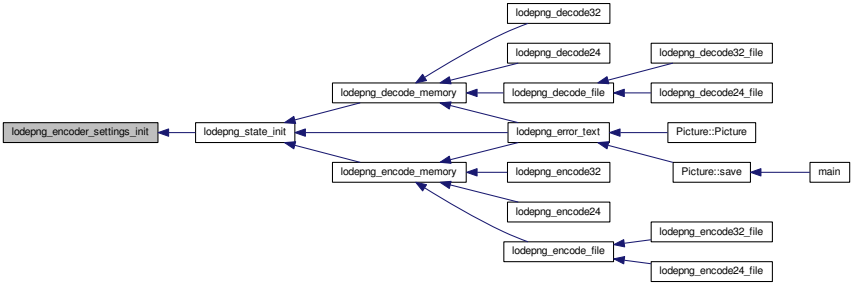
Definition at line 5882 of file lodepng.cpp.

```
5883 {
5884   lodepng_compress_settings_init(&settings->
      zlibsettings);
5885   settings->filter_palette_zero = 1;
5886   settings->filter_strategy = LFS_MINSUM;
5887   settings->auto_convert = 1;
5888   settings->force_palette = 0;
```

```
5889   settings->predefined_filters = 0;
5890 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
5891   settings->add_id = 0;
5892   settings->text_compression = 1;
5893 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
5894 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.1.3.117 lodepng_error_text()

```
const char* lodepng_error_text (
            unsigned code )
```

Definition at line 5904 of file lodepng.cpp.

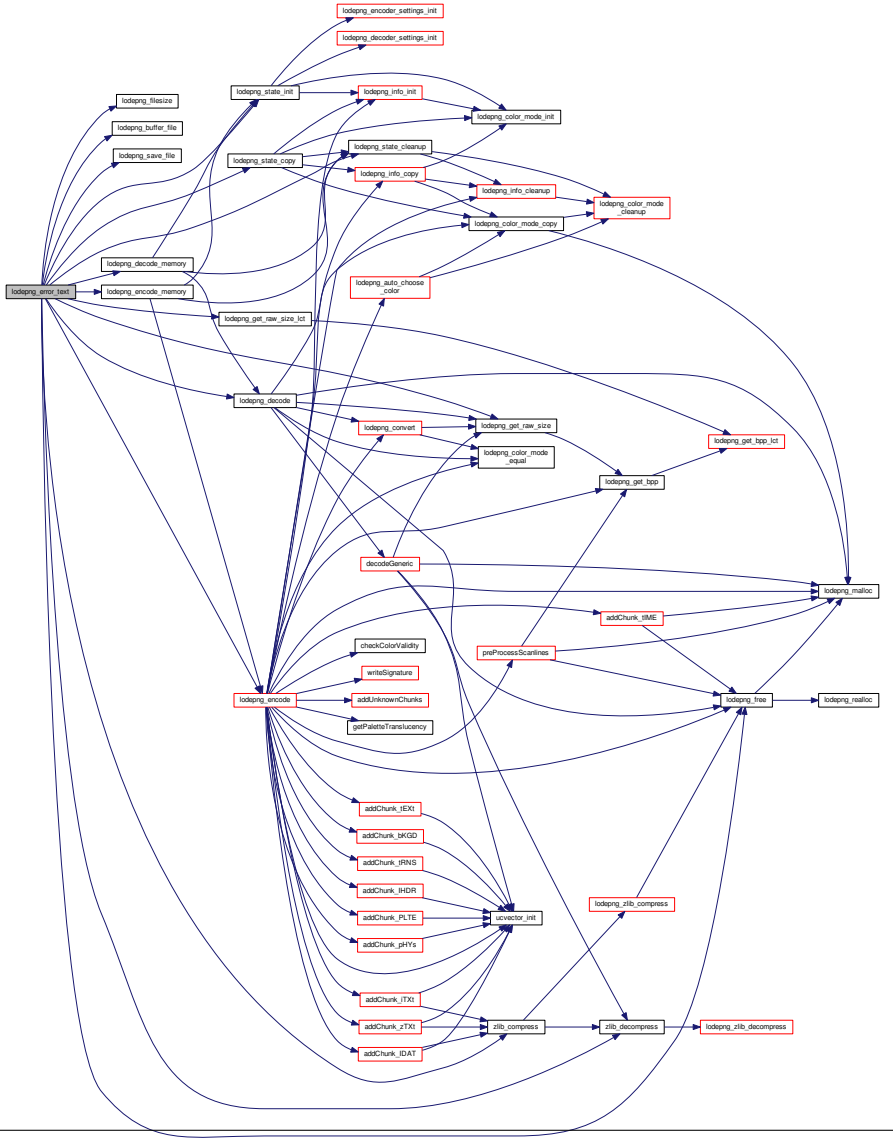
```
5905 {
5906   switch(code)
5907   {
5908     case 0: return "no error, everything went ok";
5909     case 1: return "nothing done yet"; /*the Encoder/Decoder has done nothing y
       sense yet*/
5910     case 10: return "end of input memory reached without huffman end code"; /*w
5911     case 11: return "error in code tree made it jump outside of huffman tree";
5912     case 13: return "problem while processing dynamic deflate block";
5913     case 14: return "problem while processing dynamic deflate block";
5914     case 15: return "problem while processing dynamic deflate block";
5915     case 16: return "unexisting code while processing dynamic deflate block";
5916     case 17: return "end of out buffer memory reached while inflating";
5917     case 18: return "invalid distance code while inflating";
5918     case 19: return "end of out buffer memory reached while inflating";
5919     case 20: return "invalid deflate block BTYPE encountered while decoding";
5920     case 21: return "NLEN is not ones complement of LEN in a deflate block";
5921      /*end of out buffer memory reached while inflating:
5922      This can happen if the inflated deflate data is longer than the amount of
5923      all the pixels of the image, given the color depth and image dimensions. S
5924      happen in a normal, well encoded, PNG image.*/
5925     case 22: return "end of out buffer memory reached while inflating";
5926     case 23: return "end of in buffer memory reached while inflating";
5927     case 24: return "invalid FCHECK in zlib header";
```

```
5928     case 25: return "invalid compression method in zlib header";
5929     case 26: return "FDICT encountered in zlib header while it's not used for P
5930     case 27: return "PNG file is smaller than a PNG header";
5931     /*Checks the magic file header, the first 8 bytes of the PNG file*/
5932     case 28: return "incorrect PNG signature, it's no PNG or corrupted";
5933     case 29: return "first chunk is not the header chunk";
5934     case 30: return "chunk length too large, chunk broken off at end of file";
5935     case 31: return "illegal PNG color type or bpp";
5936     case 32: return "illegal PNG compression method";
5937     case 33: return "illegal PNG filter method";
5938     case 34: return "illegal PNG interlace method";
5939     case 35: return "chunk length of a chunk is too large or the chunk too smal
5940     case 36: return "illegal PNG filter type encountered";
5941     case 37: return "illegal bit depth for this color type given";
5942     case 38: return "the palette is too big"; /*more than 256 colors*/
5943     case 39: return "more palette alpha values given in tRNS chunk than there a
5944     case 40: return "tRNS chunk has wrong size for greyscale image";
5945     case 41: return "tRNS chunk has wrong size for RGB image";
5946     case 42: return "tRNS chunk appeared while it was not allowed for this colo
5947     case 43: return "bKGD chunk has wrong size for palette image";
5948     case 44: return "bKGD chunk has wrong size for greyscale image";
5949     case 45: return "bKGD chunk has wrong size for RGB image";
5950     case 48: return "empty input buffer given to decoder. Maybe caused by non-e
5951     case 49: return "jumped past memory while generating dynamic huffman tree";
5952     case 50: return "jumped past memory while generating dynamic huffman tree";
5953     case 51: return "jumped past memory while inflating huffman block";
5954     case 52: return "jumped past memory while inflating";
5955     case 53: return "size of zlib data too small";
5956     case 54: return "repeat symbol in tree while there was no value symbol yet"
5957     /*jumped past tree while generating huffman tree, this could be when the
5958     tree will have more leaves than symbols after generating it out of the
```

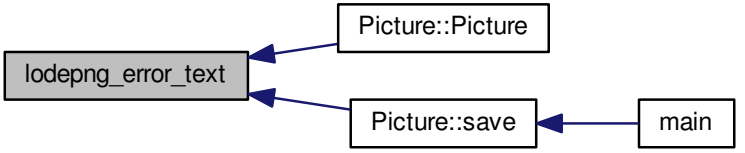
```
5959     given lenghts. They call this an oversubscribed dynamic bit lengths tree in
5960     case 55: return "jumped past tree while generating huffman tree";
5961     case 56: return "given output image colortype or bitdepth not supported for
5962     case 57: return "invalid CRC encountered (checking CRC can be disabled)";
5963     case 58: return "invalid ADLER32 encountered (checking ADLER32 can be disab
5964     case 59: return "requested color conversion not supported";
5965     case 60: return "invalid window size given in the settings of the encoder (
5966     case 61: return "invalid BTYPE given in the settings of the encoder (only 0
5967     /*LodePNG leaves the choice of RGB to greyscale conversion formula to the u
5968     case 62: return "conversion from color to greyscale not supported";
5969     case 63: return "length of a chunk too long, max allowed for PNG is 2147483
       (2^31-1)*/
5970     /*this would result in the inability of a deflated block to ever contain an
       least 1.*/
5971     case 64: return "the length of the END symbol 256 in the Huffman tree is 0"
5972     case 66: return "the length of a text chunk keyword given to the encoder is
       79 bytes";
5973     case 67: return "the length of a text chunk keyword given to the encoder is
       1 byte";
5974     case 68: return "tried to encode a PLTE chunk with a palette that has less
       colors";
5975     case 69: return "unknown chunk type with 'critical' flag encountered by the
5976     case 71: return "unexisting interlace mode given to encoder (must be 0 or 1
5977     case 72: return "while decoding, unexisting compression method encountering
       must be 0)";
5978     case 73: return "invalid tIME chunk size";
5979     case 74: return "invalid pHYs chunk size";
5980     /*length could be wrong, or data chopped off*/
5981     case 75: return "no null termination char found while decoding text chunk";
5982     case 76: return "iTXt chunk too short to contain required bytes";
5983     case 77: return "integer overflow in buffer size";
```

```
5984     case 78: return "failed to open file for reading"; /*file doesn't exist or
       reading*/
5985     case 79: return "failed to open file for writing";
5986     case 80: return "tried creating a tree of 0 symbols";
5987     case 81: return "lazy matching at pos 0 is impossible";
5988     case 82: return "color conversion to palette requested while a color isn't
5989     case 83: return "memory allocation failed";
5990     case 84: return "given image too small to contain all pixels to be encoded"
5991     case 86: return "impossible offset in lz77 encoding (internal bug)";
5992     case 87: return "must provide custom zlib function pointer if LODEPNG_COMPI
5993     case 88: return "invalid filter strategy given for LodePNGEncoderSettings.f
5994     case 89: return "text chunk keyword too short or long: must have size 1-79"
5995     /*the windowsize in the LodePNGCompressSettings. Requiring POT(==> & instea
       faster.*/
5996     case 90: return "windowsize must be a power of two";
5997     case 91: return "invalid decompressed idat size";
5998     case 92: return "too many pixels, not supported";
5999     case 93: return "zero width or height is invalid";
6000     case 94: return "header chunk must have a size of 13 bytes";
6001   }
6002   return "unknown error code";
6003 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.118 lodepng_filesize()

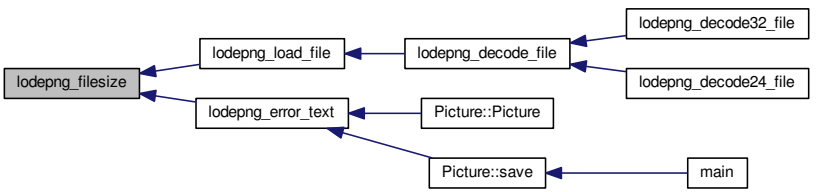
```
static long lodepng_filesize (
            const char * filename )  [static]
```

Definition at line 351 of file lodepng.cpp.

```
352 {
353   FILE* file;
354   long size;
355   file = fopen(filename, "rb");
356   if(!file) return -1;
357
358   if(fseek(file, 0, SEEK_END) != 0)
359   {
360     fclose(file);
```

```
361     return -1;
362   }
363
364   size = ftell(file);
365   /* It may give LONG_MAX as directory size, this is invalid for us. */
366   if(size == LONG_MAX) size = -1;
367
368   fclose(file);
369   return size;
370 }
```

Here is the caller graph for this function:



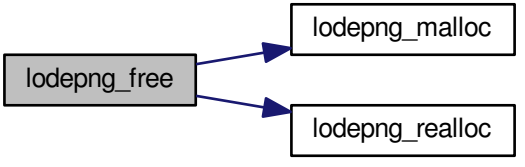
4.1.3.119 lodepng_free()

```
static void lodepng_free (
            void * ptr )  [static]
```

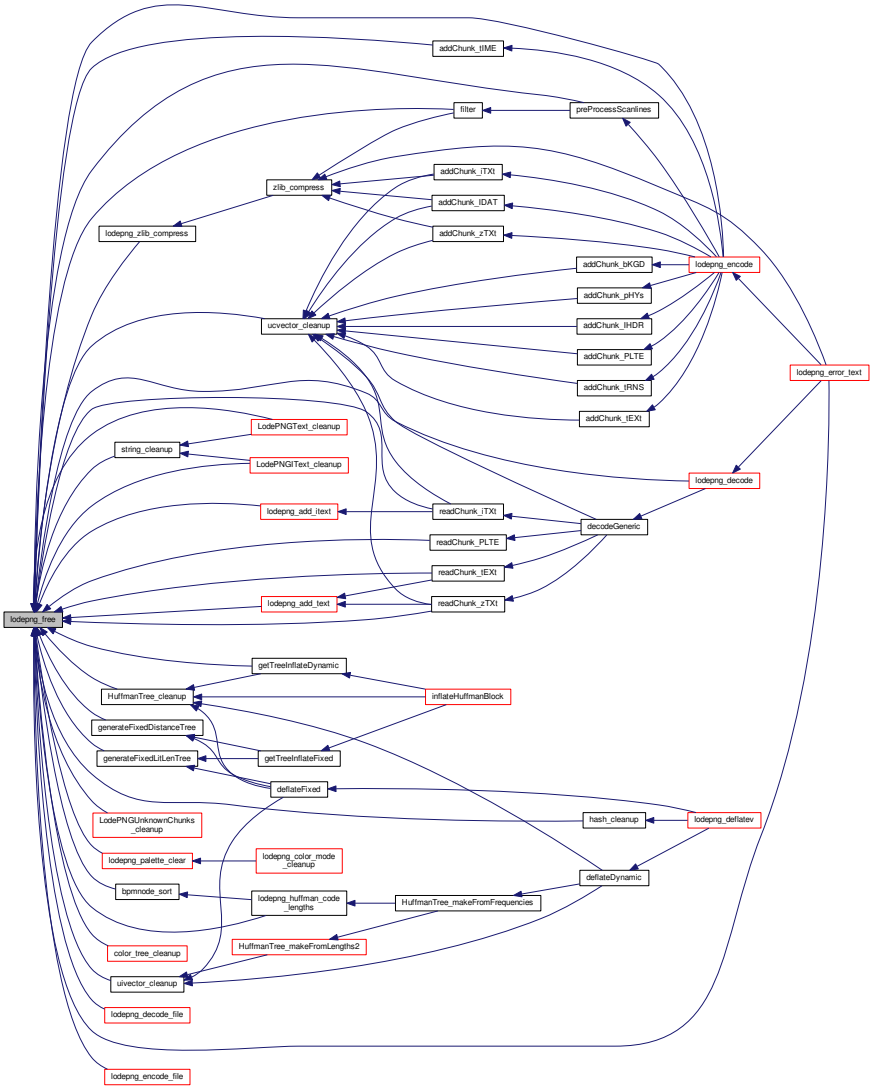
Definition at line 73 of file lodepng.cpp.

```
74 {
75   free(ptr);
76 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



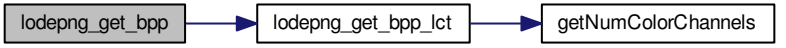
4.1.3.120 lodepng_get_bpp()

```
unsigned lodepng_get_bpp (
            const LodePNGColorMode * info )
```

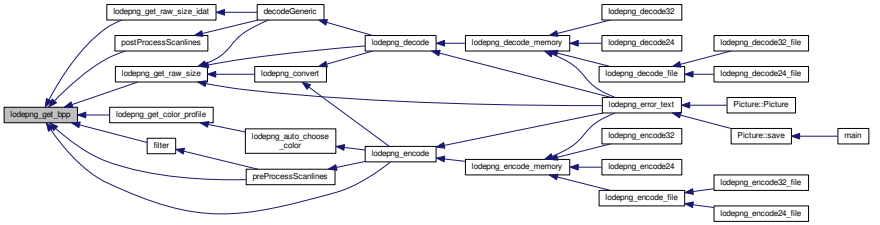
Definition at line 2665 of file lodepng.cpp.

```
2666 {
2667   /*calculate bits per pixel out of colortype and bitdepth*/
2668   return lodepng_get_bpp_lct(info->colortype, info->
      bitdepth);
2669 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



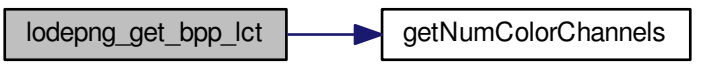
4.1.3.121 lodepng_get_bpp_lct()

```
static unsigned lodepng_get_bpp_lct (
            LodePNGColorType colortype,
            unsigned bitdepth )  [static]
```

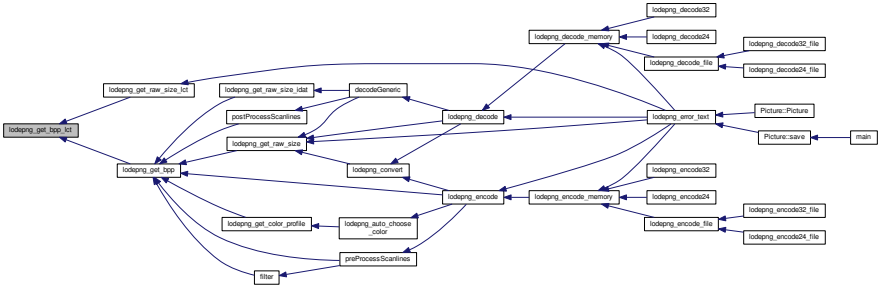
Definition at line 2575 of file lodepng.cpp.

```
2576 {
2577   /*bits per pixel is amount of channels * bits per channel*/
2578   return getNumColorChannels(colortype) * bitdepth;
2579 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.122 lodepng_get_channels()

```
unsigned lodepng_get_channels (
            const LodePNGColorMode * info )
```

Definition at line 2671 of file lodepng.cpp.

```
2672 {
2673   return getNumColorChannels(info->colortype);
2674 }
```

Here is the call graph for this function:

4.1.3.123 lodepng_get_color_profile()

```
unsigned lodepng_get_color_profile (
            LodePNGColorProfile * profile,
            const unsigned char * in,
            unsigned w,
            unsigned h,
            const LodePNGColorMode * mode )
```

Definition at line 3567 of file lodepng.cpp.

```
3570 {
3571   unsigned error = 0;
3572   size_t i;
3573   ColorTree tree;
3574   size_t numpixels = w * h;
3575
3576   unsigned colored_done = lodepng_is_greyscale_type(mode) ? 1 : 0;
```

```
3577   unsigned alpha_done = lodepng_can_have_alpha(mode) ? 0 : 1;
3578   unsigned numcolors_done = 0;
3579   unsigned bpp = lodepng_get_bpp(mode);
3580   unsigned bits_done = bpp == 1 ? 1 : 0;
3581   unsigned maxnumcolors = 257;
3582   unsigned sixteen = 0;
3583   if(bpp <= 8) maxnumcolors = bpp == 1 ? 2 : (bpp == 2 ? 4 : (bpp == 4 ? 16 : 2
3584
3585   color_tree_init(&tree);
3586
3587   /*Check if the 16-bit input is truly 16-bit*/
3588   if(mode->bitdepth == 16)
3589   {
3590     unsigned short r, g, b, a;
3591     for(i = 0; i != numpixels; ++i)
3592     {
3593       getPixelColorRGBA16(&r, &g, &b, &a, in, i, mode);
3594       if((r & 255) != ((r >> 8) & 255) || (g & 255) != ((g >> 8) & 255) ||
3595          (b & 255) != ((b >> 8) & 255) || (a & 255) != ((a >> 8) & 255)) /*firs
3596       {
3597         sixteen = 1;
3598         break;
3599       }
3600     }
3601   }
3602
3603   if(sixteen)
3604   {
3605     unsigned short r = 0, g = 0, b = 0, a = 0;
3606     profile->bits = 16;
3607     bits_done = numcolors_done = 1; /*counting colors no longer useful, palette
```

```
3608
3609     for(i = 0; i != numpixels; ++i)
3610     {
3611       getPixelColorRGBA16(&r, &g, &b, &a, in, i, mode);
3612
3613       if(!colored_done && (r != g || r != b))
3614       {
3615         profile->colored = 1;
3616         colored_done = 1;
3617       }
3618
3619       if(!alpha_done)
3620       {
3621         unsigned matchkey = (r == profile->key_r && g == profile->key_g && b ==
     key_b);
3622         if(a != 65535 && (a != 0 || (profile->key && !matchkey)))
3623         {
3624           profile->alpha = 1;
3625           profile->key = 0;
3626           alpha_done = 1;
3627         }
3628         else if(a == 0 && !profile->alpha && !profile->key)
3629         {
3630           profile->key = 1;
3631           profile->key_r = r;
3632           profile->key_g = g;
3633           profile->key_b = b;
3634         }
3635         else if(a == 65535 && profile->key && matchkey)
3636         {
3637           /* Color key cannot be used if an opaque pixel also has that RGB colo
```

```
3638           profile->alpha = 1;
3639           profile->key = 0;
3640           alpha_done = 1;
3641         }
3642       }
3643       if(alpha_done && numcolors_done && colored_done && bits_done) break;
3644     }
3645
3646     if(profile->key && !profile->alpha)
3647     {
3648       for(i = 0; i != numpixels; ++i)
3649       {
3650         getPixelColorRGBA16(&r, &g, &b, &a, in, i, mode);
3651         if(a != 0 && r == profile->key_r && g == profile->key_g && b == profile
    key_b)
3652         {
3653           /* Color key cannot be used if an opaque pixel also has that RGB colo
3654           profile->alpha = 1;
3655           profile->key = 0;
3656           alpha_done = 1;
3657         }
3658       }
3659     }
3660   }
3661   else /* < 16-bit */
3662   {
3663     unsigned char r = 0, g = 0, b = 0, a = 0;
3664     for(i = 0; i != numpixels; ++i)
3665     {
3666       getPixelColorRGBA8(&r, &g, &b, &a, in, i, mode);
3667
```

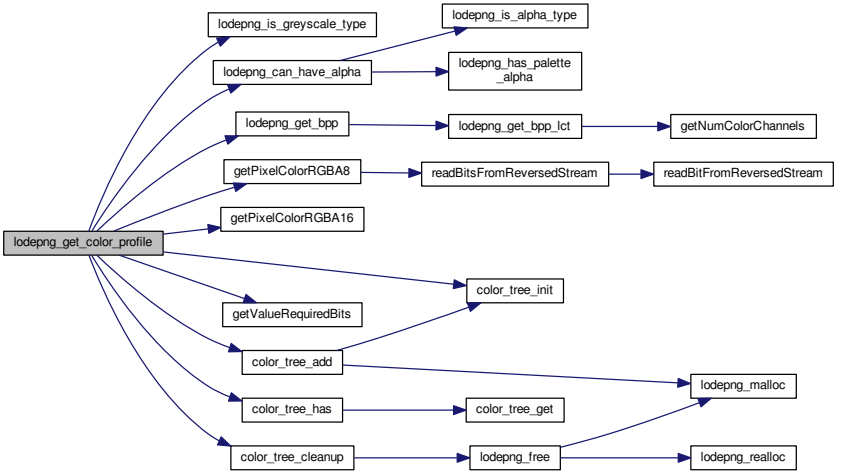
```
3668       if(!bits_done && profile->bits < 8)
3669       {
3670         /*only r is checked, < 8 bits is only relevant for greyscale*/
3671         unsigned bits = getValueRequiredBits(r);
3672         if(bits > profile->bits) profile->bits = bits;
3673       }
3674       bits_done = (profile->bits >= bpp);
3675 
3676       if(!colored_done && (r != g || r != b))
3677       {
3678         profile->colored = 1;
3679         colored_done = 1;
3680         if(profile->bits < 8) profile->bits = 8; /*PNG has no colored modes wit
     per channel*/
3681       }
3682 
3683       if(!alpha_done)
3684       {
3685         unsigned matchkey = (r == profile->key_r && g == profile->key_g && b ==
    key_b);
3686         if(a != 255 && (a != 0 || (profile->key && !matchkey)))
3687         {
3688           profile->alpha = 1;
3689           profile->key = 0;
3690           alpha_done = 1;
3691           if(profile->bits < 8) profile->bits = 8; /*PNG has no alphachannel mo
     8-bit per channel*/
3692         }
3693         else if(a == 0 && !profile->alpha && !profile->key)
3694         {
3695           profile->key = 1;
```

```
3696           profile->key_r = r;
3697           profile->key_g = g;
3698           profile->key_b = b;
3699         }
3700         else if(a == 255 && profile->key && matchkey)
3701         {
3702           /* Color key cannot be used if an opaque pixel also has that RGB colo
3703           profile->alpha = 1;
3704           profile->key = 0;
3705           alpha_done = 1;
3706           if(profile->bits < 8) profile->bits = 8; /*PNG has no alphachannel mo
       8-bit per channel*/
3707         }
3708       }
3709
3710       if(!numcolors_done)
3711       {
3712         if(!color_tree_has(&tree, r, g, b, a))
3713         {
3714           color_tree_add(&tree, r, g, b, a, profile->numcolors);
3715           if(profile->numcolors < 256)
3716           {
3717             unsigned char* p = profile->palette;
3718             unsigned n = profile->numcolors;
3719             p[n * 4 + 0] = r;
3720             p[n * 4 + 1] = g;
3721             p[n * 4 + 2] = b;
3722             p[n * 4 + 3] = a;
3723           }
3724           ++profile->numcolors;
3725           numcolors_done = profile->numcolors >= maxnumcolors;
```

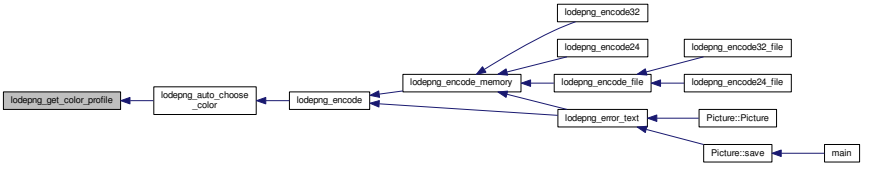
```
3726         }
3727       }
3728
3729       if(alpha_done && numcolors_done && colored_done && bits_done) break;
3730     }
3731
3732     if(profile->key && !profile->alpha)
3733     {
3734       for(i = 0; i != numpixels; ++i)
3735       {
3736         getPixelColorRGBA8(&r, &g, &b, &a, in, i, mode);
3737         if(a != 0 && r == profile->key_r && g == profile->key_g && b == profile
     key_b)
3738         {
3739           /* Color key cannot be used if an opaque pixel also has that RGB colo
3740           profile->alpha = 1;
3741           profile->key = 0;
3742           alpha_done = 1;
3743           if(profile->bits < 8) profile->bits = 8; /*PNG has no alphachannel mo
      8-bit per channel*/
3744         }
3745       }
3746     }
3747
3748     /*make the profile's key always 16-bit for consistency - repeat each byte t
3749     profile->key_r += (profile->key_r << 8);
3750     profile->key_g += (profile->key_g << 8);
3751     profile->key_b += (profile->key_b << 8);
3752   }
3753
3754   color_tree_cleanup(&tree);
```

```
3755   return error;
3756 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.124 lodepng_get_raw_size()

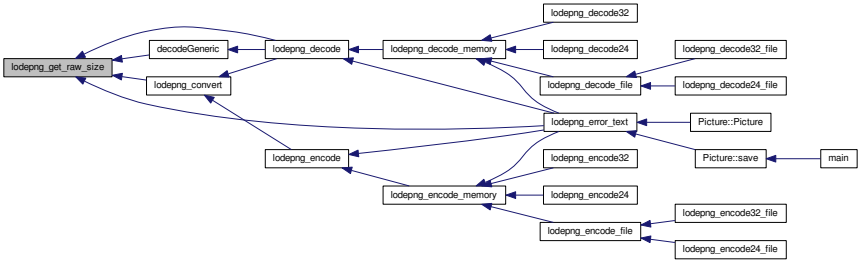
```
size_t lodepng_get_raw_size (
            unsigned w,
            unsigned h,
            const LodePNGColorMode * color )
```

Definition at line 2708 of file lodepng.cpp.

```
2709 {
2710   /*will not overflow for any color type if roughly w * h < 268435455*/
2711   size_t bpp = lodepng_get_bpp(color);
2712   size_t n = w * h;
2713   return ((n / 8) * bpp) + ((n & 7) * bpp + 7) / 8;
2714 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.1.3.125 lodepng_get_raw_size_idat()

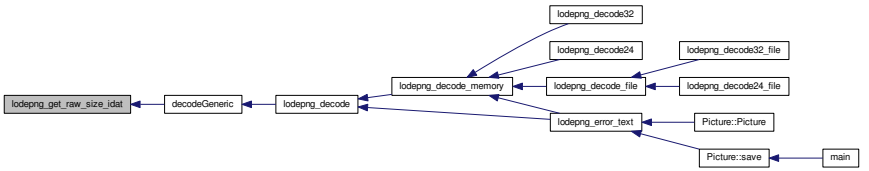
```
static size_t lodepng_get_raw_size_idat (
            unsigned w,
            unsigned h,
            const LodePNGColorMode * color )  [static]
```

Definition at line 2728 of file lodepng.cpp.

```
2729 {
2730   /*will not overflow for any color type if roughly w * h < 268435455*/
2731   size_t bpp = lodepng_get_bpp(color);
2732   size_t line = ((w / 8) * bpp) + ((w & 7) * bpp + 7) / 8;
2733   return h * line;
2734 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



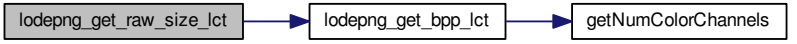
4.1.3.126 lodepng_get_raw_size_lct()

```
size_t lodepng_get_raw_size_lct (
            unsigned w,
            unsigned h,
            LodePNGColorType colortype,
            unsigned bitdepth )
```

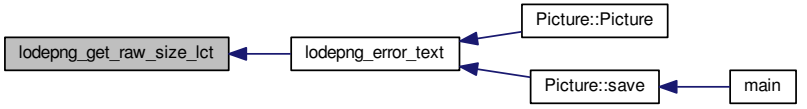
Definition at line 2716 of file lodepng.cpp.

```
2717 {
2718   /*will not overflow for any color type if roughly w * h < 268435455*/
2719   size_t bpp = lodepng_get_bpp_lct(colortype, bitdepth);
2720   size_t n = w * h;
2721   return ((n / 8) * bpp) + ((n & 7) * bpp + 7) / 8;
2722 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



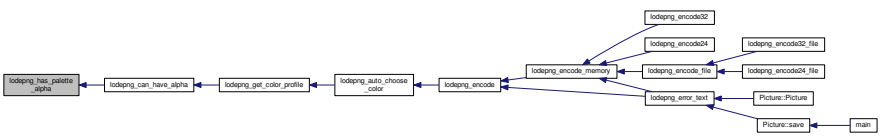
4.1.3.127 lodepng_has_palette_alpha()

```
unsigned lodepng_has_palette_alpha (
            const LodePNGColorMode * info )
```

Definition at line 2691 of file lodepng.cpp.

```
2692 {
2693   size_t i;
2694   for(i = 0; i != info->palettesize; ++i)
2695   {
2696     if(info->palette[i * 4 + 3] < 255) return 1;
2697   }
2698   return 0;
2699 }
```

Here is the caller graph for this function:



4.1.3.128 lodepng_huffman_code_lengths()

```
unsigned lodepng_huffman_code_lengths (
            unsigned * lengths,
            const unsigned * frequencies,
            size_t numcodes,
            unsigned maxbitlen )
```

Definition at line 789 of file lodepng.cpp.

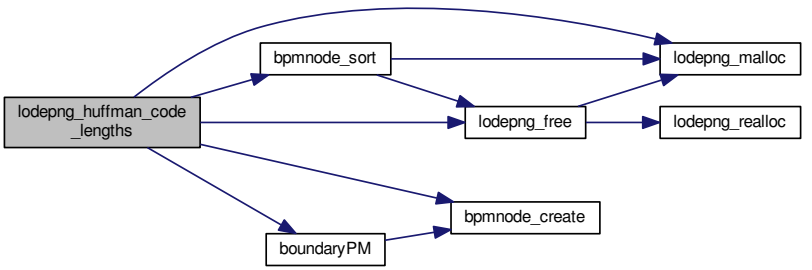
```
791 {
792   unsigned error = 0;
793   unsigned i;
794   size_t numpresent = 0; /*number of symbols with non-zero frequency*/
795   BPMNode* leaves; /*the symbols, only those with > 0 frequency*/
796
797   if(numcodes == 0) return 80; /*error: a tree of 0 symbols is not supposed to b
798   if((1u << maxbitlen) < numcodes) return 80; /*error: represent all symbols*/
799
800   leaves = (BPMNode*)lodepng_malloc(numcodes * sizeof(*leaves));
801   if(!leaves) return 83; /*alloc fail*/
802
803   for(i = 0; i != numcodes; ++i)
804   {
805     if(frequencies[i] > 0)
806     {
807       leaves[numpresent].weight = (int)frequencies[i];
808       leaves[numpresent].index = i;
809       ++numpresent;
810     }
811   }
812
```

```
813   for(i = 0; i != numcodes; ++i) lengths[i] = 0;
814
815   /*ensure at least two present symbols. There should be at least one symbol
816   according to RFC 1951 section 3.2.7. Some decoders incorrectly require two. To
817   make these work as well ensure there are at least two symbols. The
818   Package-Merge code below also doesn't work correctly if there's only one
819   symbol, it'd give it the theoritical 0 bits but in practice zlib wants 1 bit*/
820   if(numpresent == 0)
821   {
822     lengths[0] = lengths[1] = 1; /*note that for RFC 1951 section 3.2.7, only le
823   }
824   else if(numpresent == 1)
825   {
826     lengths[leaves[0].index] = 1;
827     lengths[leaves[0].index == 0 ? 1 : 0] = 1;
828   }
829   else
830   {
831     BPMLists lists;
832     BPMNode* node;
833
834     bpmnode_sort(leaves, numpresent);
835
836     lists.listsize = maxbitlen;
837     lists.memsize = 2 * maxbitlen * (maxbitlen + 1);
838     lists.nextfree = 0;
839     lists.numfree = lists.memsize;
840     lists.memory = (BPMNode*)lodepng_malloc(lists.
      memsize * sizeof(*lists.memory));
841     lists.freelist = (BPMNode**)lodepng_malloc(lists.
      memsize * sizeof(BPMNode*));
```

```
842     lists.chains0 = (BPMNode**)lodepng_malloc(lists.
      listsize * sizeof(BPMNode*));
843     lists.chains1 = (BPMNode**)lodepng_malloc(lists.
      listsize * sizeof(BPMNode*));
844     if(!lists.memory || !lists.freelist || !lists.chains0 || !lists.
      chains1) error = 83; /*alloc fail*/
845
846     if(!error)
847     {
848       for(i = 0; i != lists.memsize; ++i) lists.freelist[i] = &lists.
      memory[i];
849
850       bpmnode_create(&lists, leaves[0].weight, 1, 0);
851       bpmnode_create(&lists, leaves[1].weight, 2, 0);
852
853       for(i = 0; i != lists.listsize; ++i)
854       {
855         lists.chains0[i] = &lists.memory[0];
856         lists.chains1[i] = &lists.memory[1];
857       }
858
859       /*each boundaryPM call adds one chain to the last list, and we need 2 * nu
860       for(i = 2; i != 2 * numpresent - 2; ++i) boundaryPM(&lists, leaves, numpre
      maxbitlen - 1, (int)i);
861
862       for(node = lists.chains1[maxbitlen - 1]; node; node = node->tail)
863       {
864         for(i = 0; i != node->index; ++i) ++lengths[leaves[i].index];
865       }
866     }
867
```

```
868     lodepng_free(lists.memory);
869     lodepng_free(lists.freelist);
870     lodepng_free(lists.chains0);
871     lodepng_free(lists.chains1);
872   }
873
874   lodepng_free(leaves);
875   return error;
876 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

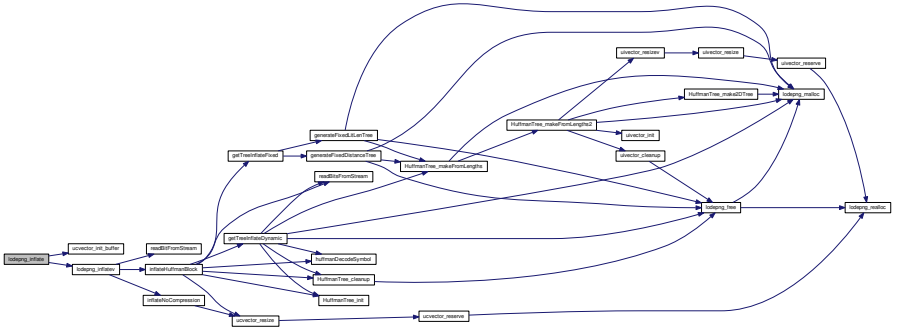
4.1.3.129 lodepng_inflate()

```
unsigned lodepng_inflate (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGDecompressSettings * settings )
```

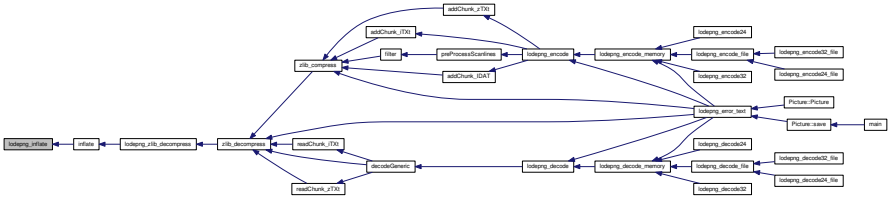
Definition at line 1283 of file lodepng.cpp.

```
1286 {
1287   unsigned error;
1288   ucvector v;
1289   ucvector_init_buffer(&v, *out, *outsize);
1290   error = lodepng_inflatev(&v, in, insize, settings);
1291   *out = v.data;
1292   *outsize = v.size;
1293   return error;
1294 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.130 lodepng_inflatev()

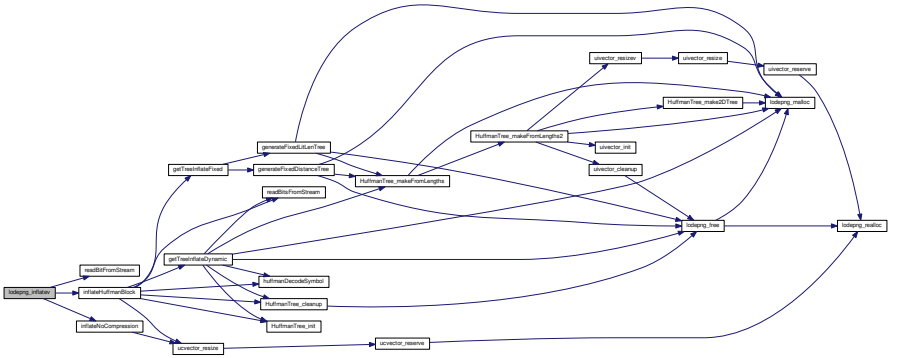
```
static unsigned lodepng_inflatev (
            ucvector * out,
            const unsigned char * in,
            size_t insize,
            const LodePNGDecompressSettings * settings )  [static]
```

Definition at line 1253 of file lodepng.cpp.

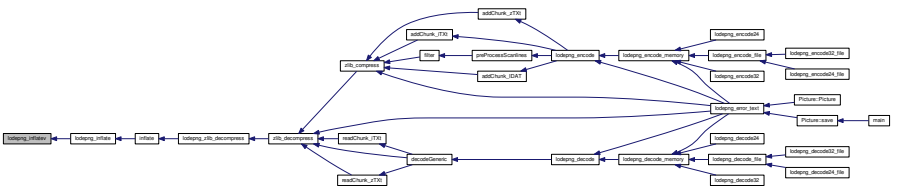
```
1256 {
1257   /*bit pointer in the "in" data, current byte is bp >> 3, current bit is bp &
       byte)*/
1258   size_t bp = 0;
1259   unsigned BFINAL = 0;
1260   size_t pos = 0; /*byte position in the out buffer*/
1261   unsigned error = 0;
1262
1263   (void)settings;
```

```
1264
1265   while(!BFINAL)
1266   {
1267     unsigned BTYPE;
1268     if(bp + 2 >= insize * 8) return 52; /*error, bit pointer will jump past mem
1269     BFINAL = readBitFromStream(&bp, in);
1270     BTYPE = 1u * readBitFromStream(&bp, in);
1271     BTYPE += 2u * readBitFromStream(&bp, in);
1272
1273     if(BTYPE == 3) return 20; /*error: invalid BTYPE*/
1274     else if(BTYPE == 0) error = inflateNoCompression(out, in, &bp, &pos, insize
       compression*/
1275     else error = inflateHuffmanBlock(out, in, &bp, &pos, insize, BTYPE); /*comp
       BTYPE 01 or 10*/
1276
1277     if(error) return error;
1278   }
1279
1280   return error;
1281 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



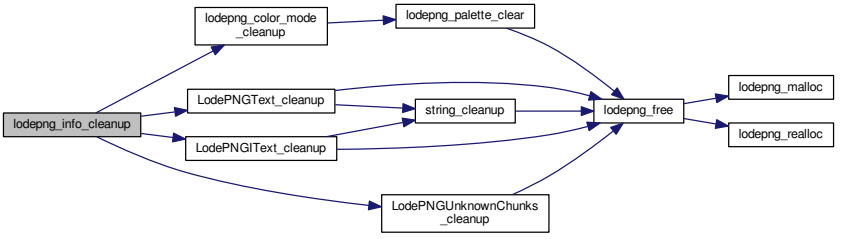
4.1.3.131 lodepng_info_cleanup()

```
void lodepng_info_cleanup (
            LodePNGInfo * info )
```

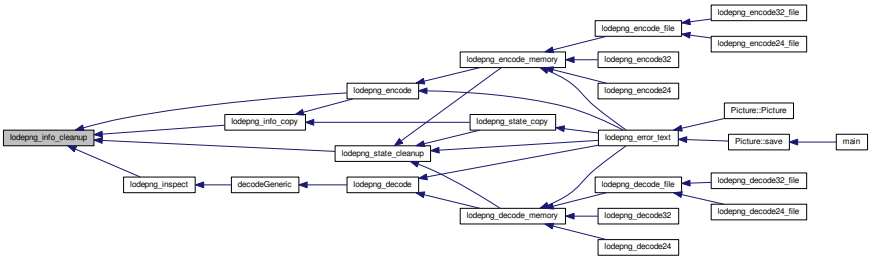
Definition at line 2943 of file lodepng.cpp.

```
2944 {
2945   lodepng_color_mode_cleanup(&info->color);
2946 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
2947   LodePNGText_cleanup(info);
2948   LodePNGIText_cleanup(info);
2949 
2950   LodePNGUnknownChunks_cleanup(info);
2951 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
2952 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.132 lodepng_info_copy()

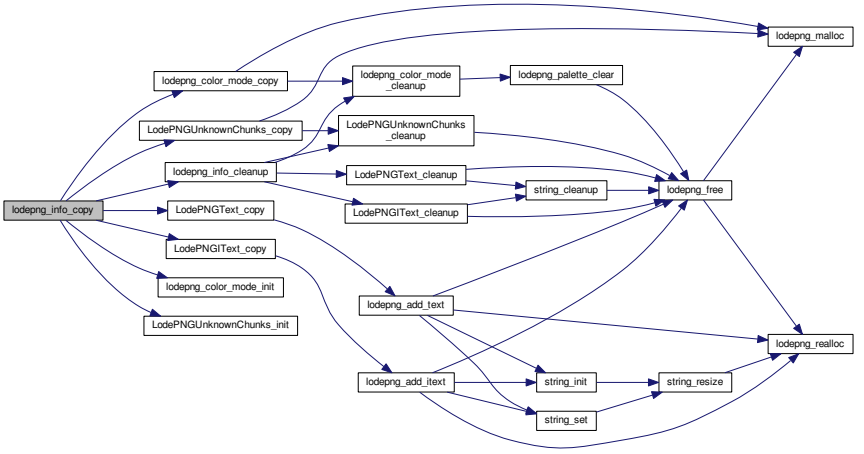
```
unsigned lodepng_info_copy (
            LodePNGInfo * dest,
            const LodePNGInfo * source )
```

Definition at line 2954 of file lodepng.cpp.

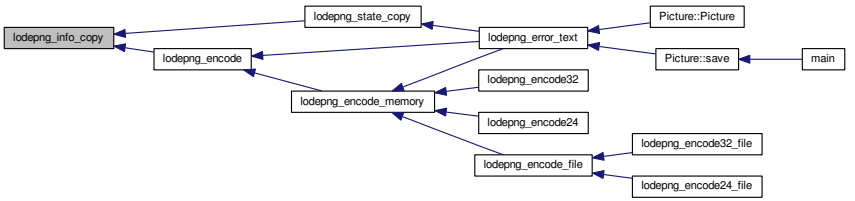
```
2955 {
2956   lodepng_info_cleanup(dest);
2957   *dest = *source;
2958   lodepng_color_mode_init(&dest->color);
2959   CERROR_TRY_RETURN(lodepng_color_mode_copy(&dest->
      color, &source->color));
2960
2961 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
2962   CERROR_TRY_RETURN(LodePNGText_copy(dest, source));
```

```
2963   CERROR_TRY_RETURN(LodePNGIText_copy(dest, source));
2964
2965   LodePNGUnknownChunks_init(dest);
2966   CERROR_TRY_RETURN(LodePNGUnknownChunks_copy(dest, source));
2967 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
2968   return 0;
2969 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.133 lodepng_info_init()

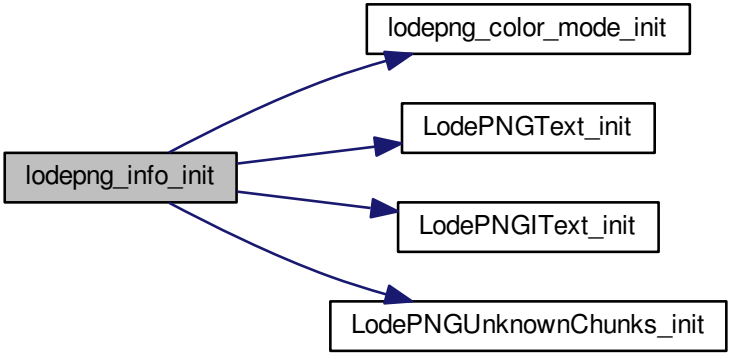
```
void lodepng_info_init (
            LodePNGInfo * info )
```

Definition at line 2923 of file lodepng.cpp.

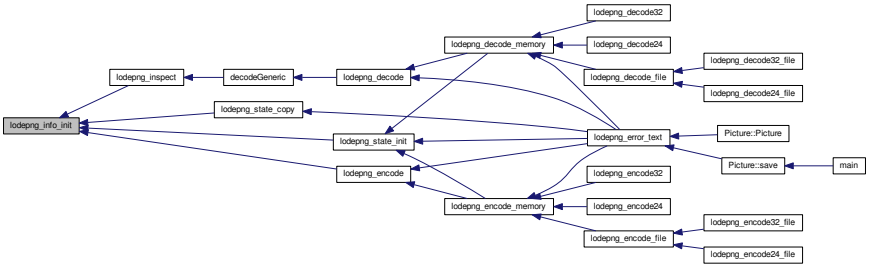
```
2924 {
2925   lodepng_color_mode_init(&info->color);
2926   info->interlace_method = 0;
2927   info->compression_method = 0;
2928   info->filter_method = 0;
2929 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
2930   info->background_defined = 0;
2931   info->background_r = info->background_g = info->
      background_b = 0;
2932
```

```
2933   LodePNGText_init(info);
2934   LodePNGIText_init(info);
2935
2936   info->time_defined = 0;
2937   info->phys_defined = 0;
2938
2939   LodePNGUnknownChunks_init(info);
2940 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
2941 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.134 lodepng_info_swap()

```
void lodepng_info_swap (
            LodePNGInfo * a,
            LodePNGInfo * b )
```

Definition at line 2971 of file lodepng.cpp.

```
2972 {
2973   LodePNGInfo temp = *a;
2974   *a = *b;
2975   *b = temp;
2976 }
```

4.1.3.135 lodepng_inspect()

```
unsigned lodepng_inspect (
            unsigned * w,
            unsigned * h,
            LodePNGState * state,
            const unsigned char * in,
            size_t insize )
```

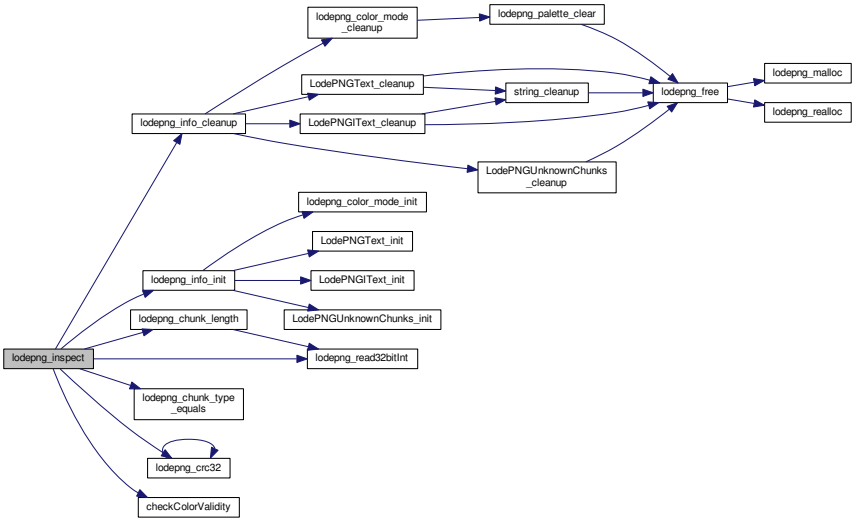
Definition at line 3903 of file lodepng.cpp.

```
3905 {
3906   LodePNGInfo* info = &state->info_png;
3907   if(insize == 0 || in == 0)
3908   {
3909     CERROR_RETURN_ERROR(state->error, 48); /*error: the given data is empty*/
3910   }
3911   if(insize < 33)
3912   {
3913     CERROR_RETURN_ERROR(state->error, 27); /*error: the data length is smaller
      the length of a PNG header*/
3914   }
3915
3916   /*when decoding a new PNG image, make sure all parameters created after previ
3917   lodepng_info_cleanup(info);
3918   lodepng_info_init(info);
3919
3920   if(in[0] != 137 || in[1] != 80 || in[2] != 78 || in[3] != 71
3921      || in[4] != 13 || in[5] != 10 || in[6] != 26 || in[7] != 10)
3922   {
3923     CERROR_RETURN_ERROR(state->error, 28); /*error: the first 8 bytes are not t
      correct PNG signature*/
```

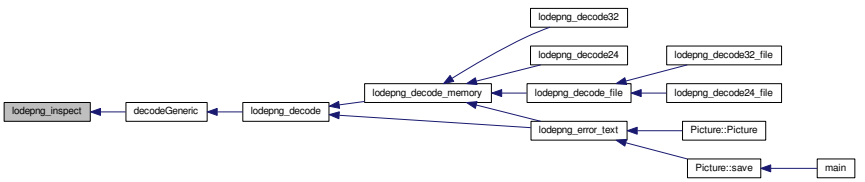
```
3924   }
3925   if(lodepng_chunk_length(in + 8) != 13)
3926   {
3927     CERROR_RETURN_ERROR(state->error, 94); /*error: header size must be 13 byte
3928   }
3929   if(!lodepng_chunk_type_equals(in + 8, "IHDR"))
3930   {
3931     CERROR_RETURN_ERROR(state->error, 29); /*error: it doesn't start with a IHD
       chunk!*/
3932   }
3933
3934   /*read the values given in the header*/
3935   *w = lodepng_read32bitInt(&in[16]);
3936   *h = lodepng_read32bitInt(&in[20]);
3937   info->color.bitdepth = in[24];
3938   info->color.colortype = (LodePNGColorType)in[25];
3939   info->compression_method = in[26];
3940   info->filter_method = in[27];
3941   info->interlace_method = in[28];
3942
3943   if(*w == 0 || *h == 0)
3944   {
3945     CERROR_RETURN_ERROR(state->error, 93);
3946   }
3947
3948   if(!state->decoder.ignore_crc)
3949   {
3950     unsigned CRC = lodepng_read32bitInt(&in[29]);
3951     unsigned checksum = lodepng_crc32(&in[12], 17);
3952     if(CRC != checksum)
3953     {
```

```
3954       CERROR_RETURN_ERROR(state->error, 57); /*invalid CRC*/
3955     }
3956   }
3957
3958   /*error: only compression method 0 is allowed in the specification*/
3959   if(info->compression_method != 0) CERROR_RETURN_ERROR(state->
      error, 32);
3960   /*error: only filter method 0 is allowed in the specification*/
3961   if(info->filter_method != 0) CERROR_RETURN_ERROR(state->
      error, 33);
3962   /*error: only interlace methods 0 and 1 exist in the specification*/
3963   if(info->interlace_method > 1) CERROR_RETURN_ERROR(state->
      error, 34);
3964
3965   state->error = checkColorValidity(info->color.
      colortype, info->color.bitdepth);
3966   return state->error;
3967 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



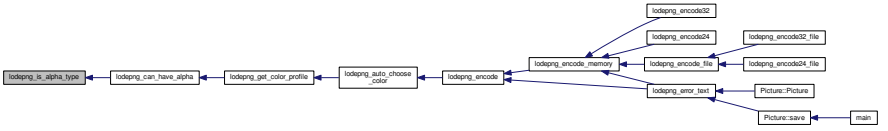
4.1.3.136 lodepng_is_alpha_type()

```
unsigned lodepng_is_alpha_type (
            const LodePNGColorMode * info )
```

Definition at line 2681 of file lodepng.cpp.

```
2682 {
2683   return (info->colortype & 4) != 0; /*4 or 6*/
2684 }
```

Here is the caller graph for this function:



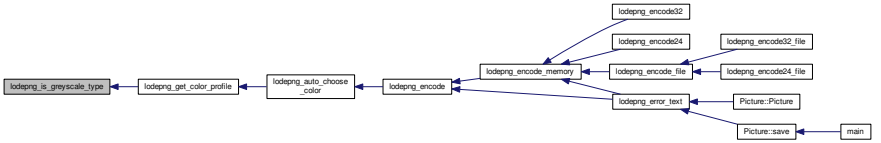
4.1.3.137 lodepng_is_greyscale_type()

```
unsigned lodepng_is_greyscale_type (
            const LodePNGColorMode * info )
```

Definition at line 2676 of file lodepng.cpp.

```
2677 {
2678   return info->colortype == LCT_GREY || info->colortype ==
      LCT_GREY_ALPHA;
2679 }
```

Here is the caller graph for this function:



4.1.3.138 lodepng_is_palette_type()

```
unsigned lodepng_is_palette_type (
            const LodePNGColorMode * info )
```

Definition at line 2686 of file lodepng.cpp.

```
2687 {
2688   return info->colortype == LCT_PALETTE;
2689 }
```

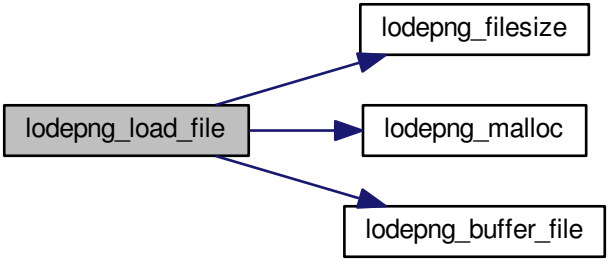
4.1.3.139 lodepng_load_file()

```
unsigned lodepng_load_file (
            unsigned char ** out,
            size_t * outsize,
            const char * filename )
```

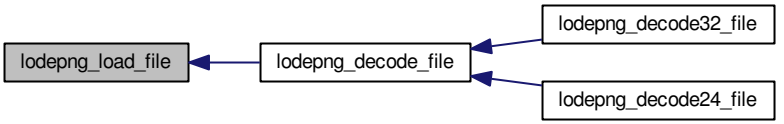
Definition at line 387 of file lodepng.cpp.

```
388 {
389   long size = lodepng_filesize(filename);
390   if (size < 0) return 78;
391   *outsize = (size_t)size;
392
393   *out = (unsigned char*)lodepng_malloc((size_t)size);
394   if(!(*out) && size > 0) return 83; /*the above malloc failed*/
395
396   return lodepng_buffer_file(*out, (size_t)size, filename);
397 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



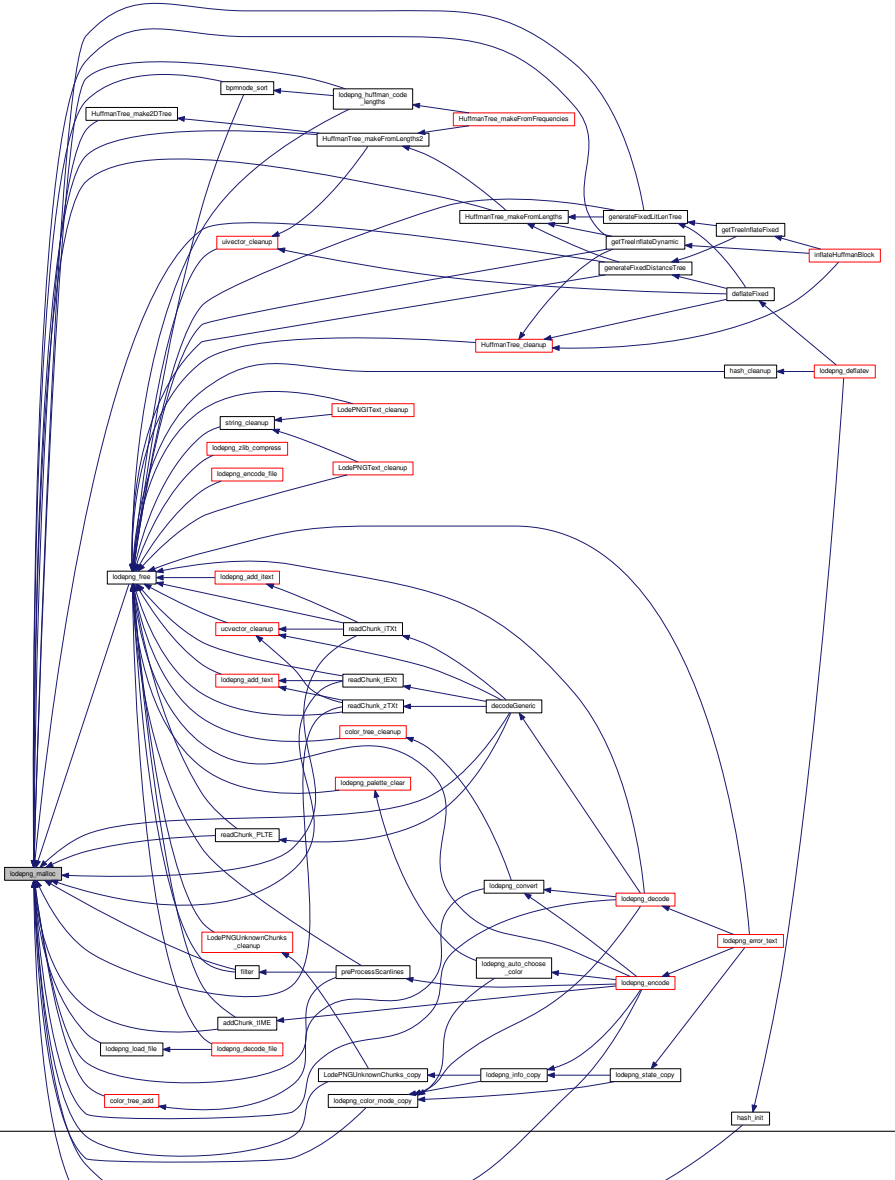
4.1.3.140 lodepng_malloc()

```
static void* lodepng_malloc (
            size_t size )  [static]
```

Definition at line 63 of file lodepng.cpp.

```
64 {
65   return malloc(size);
66 }
```

Here is the caller graph for this function:



4.1.3.141 lodepng_palette_add()

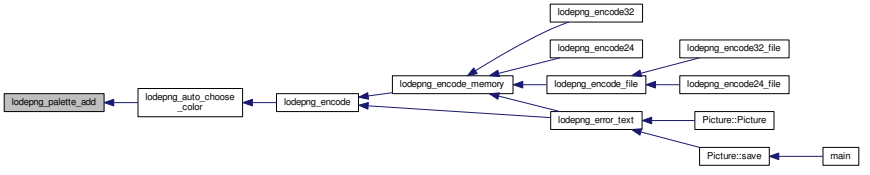
```
unsigned lodepng_palette_add (
            LodePNGColorMode * info,
            unsigned char r,
            unsigned char g,
            unsigned char b,
            unsigned char a )
```

Definition at line 2644 of file lodepng.cpp.

```
2646 {
2647   unsigned char* data;
2648   /*the same resize technique as C++ std::vectors is used, and here it's made s
2649   the max of 256 colors, it'll have the exact alloc size*/
2650   if(!info->palette) /*allocate palette if empty*/
2651   {
2652     /*room for 256 colors with 4 bytes each*/
2653     data = (unsigned char*)lodepng_realloc(info->palette, 1024);
2654     if(!data) return 83; /*alloc fail*/
2655     else info->palette = data;
2656   }
2657   info->palette[4 * info->palettesize + 0] = r;
2658   info->palette[4 * info->palettesize + 1] = g;
2659   info->palette[4 * info->palettesize + 2] = b;
2660   info->palette[4 * info->palettesize + 3] = a;
2661   ++info->palettesize;
2662   return 0;
2663 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



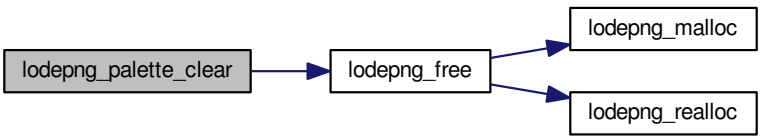
4.1.3.142 lodepng_palette_clear()

```
void lodepng_palette_clear (
            LodePNGColorMode * info )
```

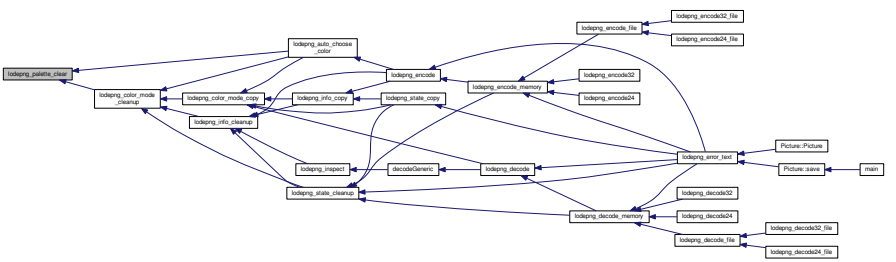
Definition at line 2637 of file lodepng.cpp.

```
2638 {
2639   if(info->palette) lodepng_free(info->palette);
2640   info->palette = 0;
2641   info->palettesize = 0;
2642 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



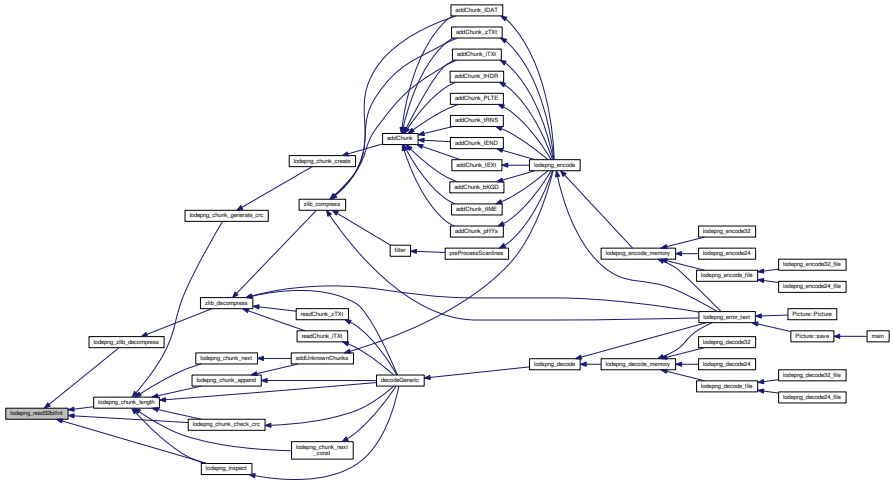
4.1.3.143 lodepng_read32bitInt()

```
unsigned lodepng_read32bitInt (
            const unsigned char * buffer )
```

Definition at line 320 of file lodepng.cpp.

```
321 {
322   return (unsigned)((buffer[0] << 24) | (buffer[1] << 16) | (buffer[2] << 8) | b
323 }
```

Here is the caller graph for this function:



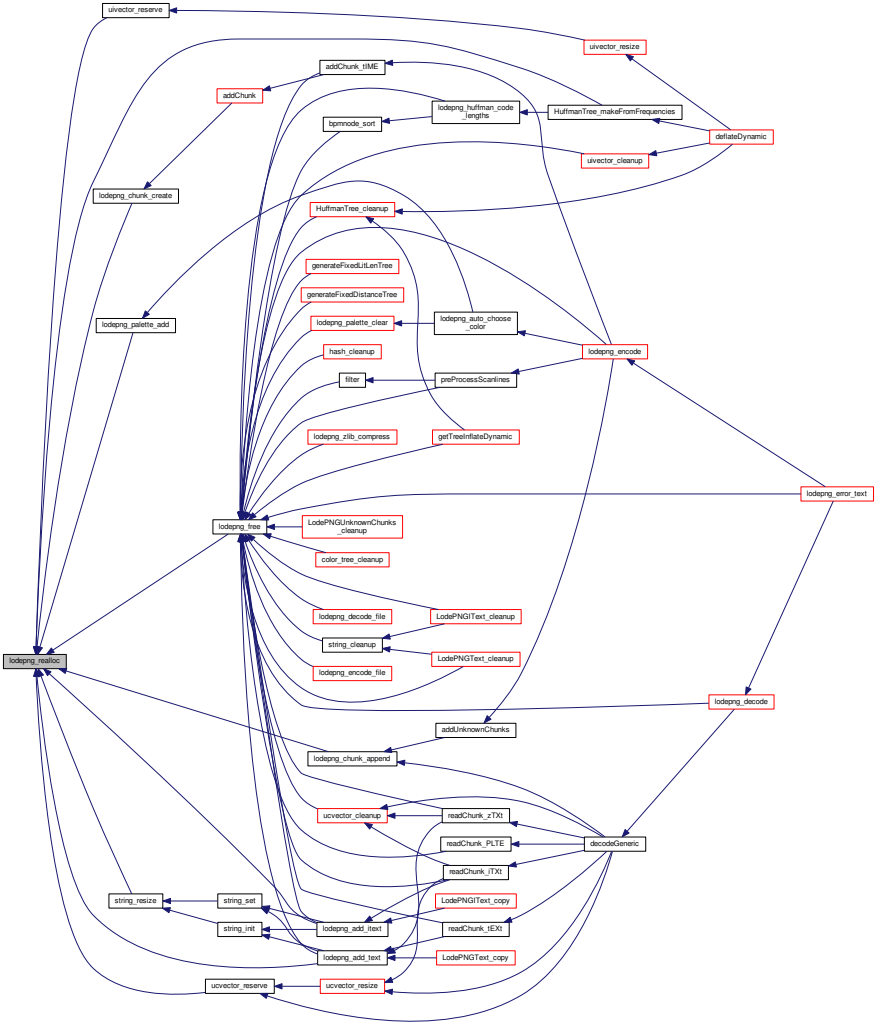
4.1.3.144 lodepng_realloc()

```
static void* lodepng_realloc (
            void * ptr,
            size_t new_size )  [static]
```

Definition at line 68 of file lodepng.cpp.

```
69 {
70   return realloc(ptr, new_size);
71 }
```

Here is the caller graph for this function:



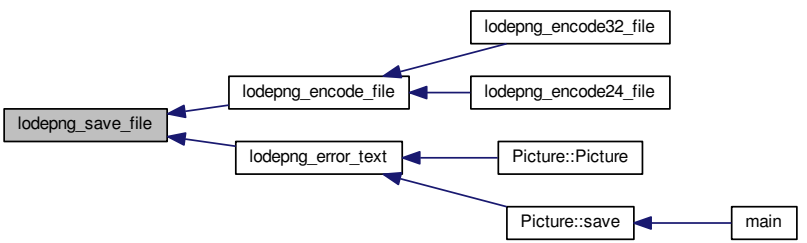
4.1.3.145 lodepng_save_file()

```
unsigned lodepng_save_file (
            const unsigned char * buffer,
            size_t buffersize,
            const char * filename )
```

Definition at line 400 of file lodepng.cpp.

```
401 {
402   FILE* file;
403   file = fopen(filename, "wb" );
404   if(!file) return 79;
405   fwrite((char*)buffer , 1 , buffersize, file);
406   fclose(file);
407   return 0;
408 }
```

Here is the caller graph for this function:



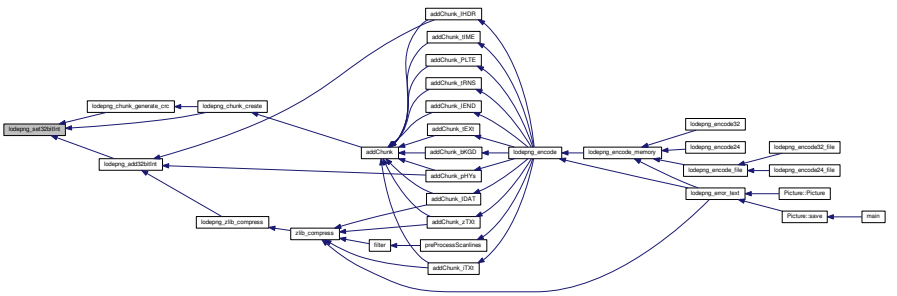
4.1.3.146 lodepng_set32bitInt()

```
static void lodepng_set32bitInt (
            unsigned char * buffer,
            unsigned value )  [static]
```

Definition at line 327 of file lodepng.cpp.

```
328 {
329   buffer[0] = (unsigned char)((value >> 24) & 0xff);
330   buffer[1] = (unsigned char)((value >> 16) & 0xff);
331   buffer[2] = (unsigned char)((value >>  8) & 0xff);
332   buffer[3] = (unsigned char)((value      ) & 0xff);
333 }
```

Here is the caller graph for this function:



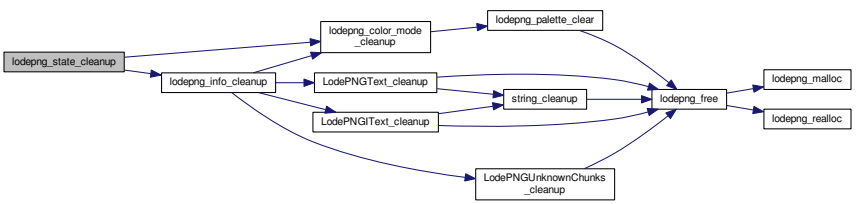
4.1.3.147 lodepng_state_cleanup()

```
void lodepng_state_cleanup (
            LodePNGState * state )
```

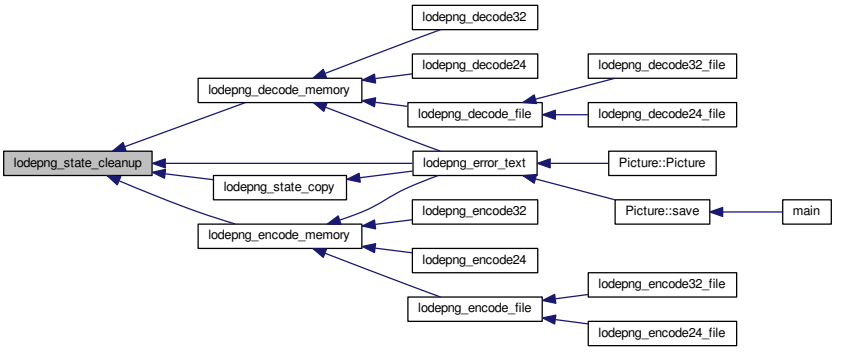
Definition at line 4843 of file lodepng.cpp.

```
4844 {
4845   lodepng_color_mode_cleanup(&state->info_raw);
4846   lodepng_info_cleanup(&state->info_png);
4847 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.148 lodepng_state_copy()

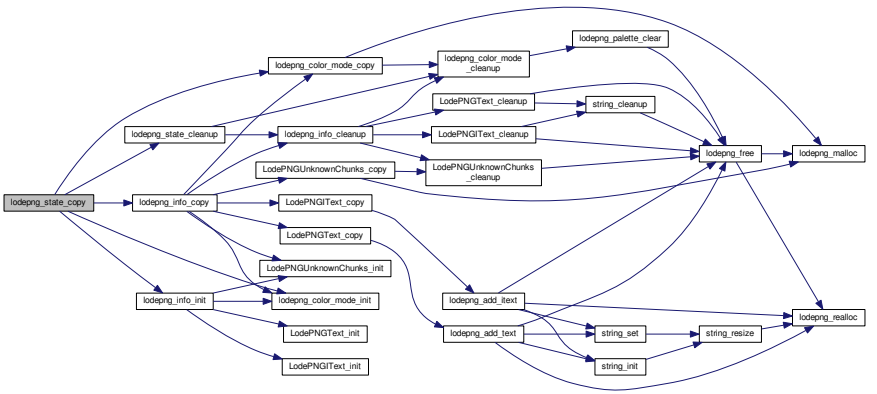
```
void lodepng_state_copy (
            LodePNGState * dest,
            const LodePNGState * source )
```

Definition at line 4849 of file lodepng.cpp.

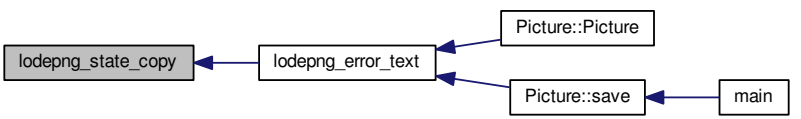
```
4850 {
4851   lodepng_state_cleanup(dest);
4852   *dest = *source;
4853   lodepng_color_mode_init(&dest->info_raw);
4854   lodepng_info_init(&dest->info_png);
```

```
4855   dest->error = lodepng_color_mode_copy(&dest->
       info_raw, &source->info_raw); if(dest->error) return;
4856   dest->error = lodepng_info_copy(&dest->info_png, &source->
       info_png); if(dest->error) return;
4857 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



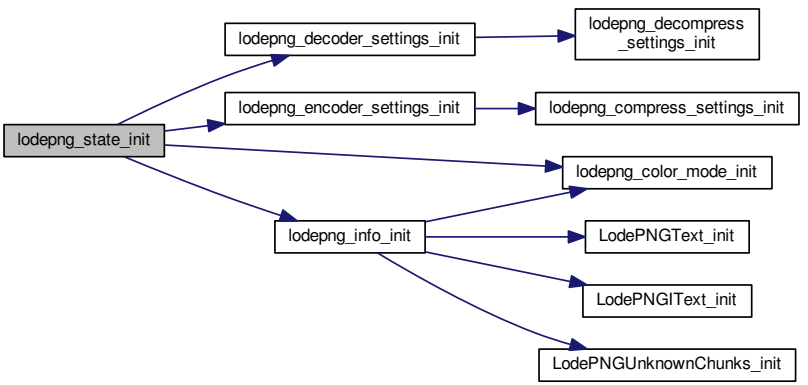
4.1.3.149 lodepng_state_init()

```
void lodepng_state_init (
            LodePNGState * state )
```

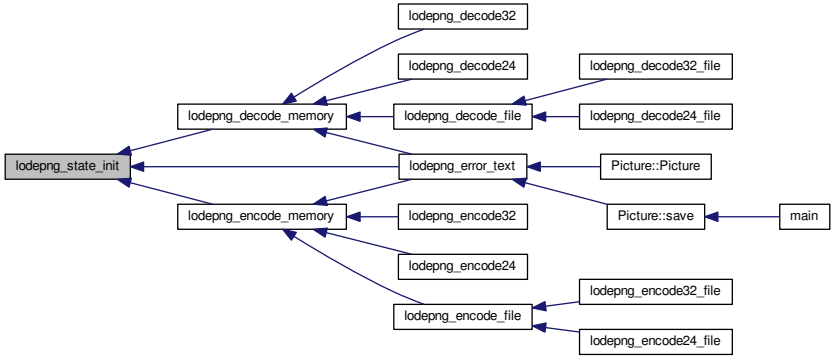
Definition at line 4830 of file lodepng.cpp.

```
4831 {
4832 #ifdef LODEPNG_COMPILE_DECODER
4833   lodepng_decoder_settings_init(&state->decoder);
4834 #endif /*LODEPNG_COMPILE_DECODER*/
4835 #ifdef LODEPNG_COMPILE_ENCODER
4836   lodepng_encoder_settings_init(&state->encoder);
4837 #endif /*LODEPNG_COMPILE_ENCODER*/
4838   lodepng_color_mode_init(&state->info_raw);
4839   lodepng_info_init(&state->info_png);
4840   state->error = 1;
4841 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.150 lodepng_zlib_compress()

```
unsigned lodepng_zlib_compress (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGCompressSettings * settings )
```

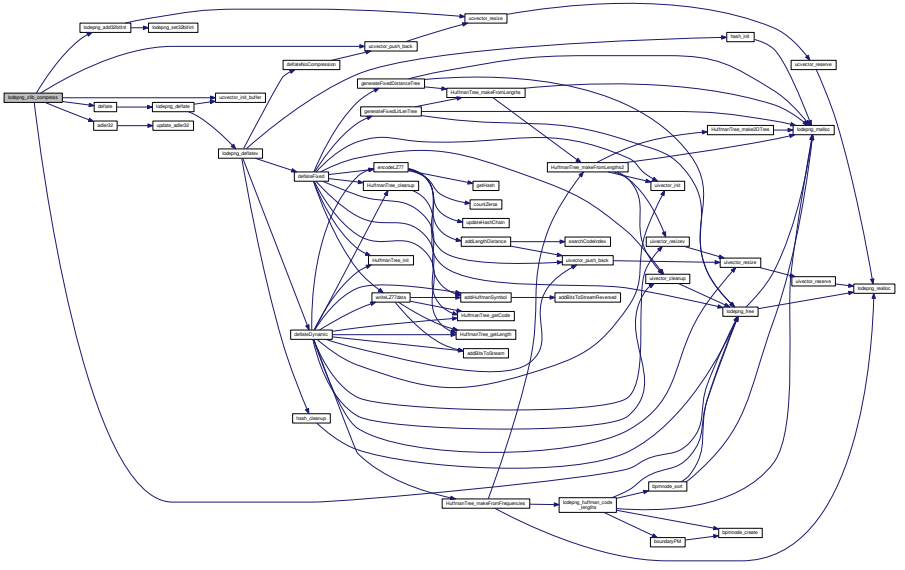
Definition at line 2188 of file lodepng.cpp.

```
2190 {
2191   /*initially, *out must be NULL and outsize 0, if you just give some random *o
```

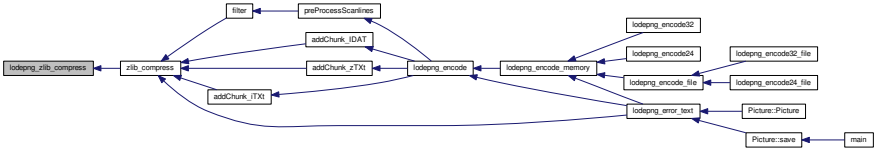
```
2192   that's pointing to a non allocated buffer, this'll crash*/
2193   ucvector outv;
2194   size_t i;
2195   unsigned error;
2196   unsigned char* deflatedata = 0;
2197   size_t deflatesize = 0;
2198
2199   /*zlib data: 1 byte CMF (CM+CINFO), 1 byte FLG, deflate data, 4 byte ADLER32
       data*/
2200   unsigned CMF = 120; /*0b01111000: CM 8, CINFO 7. With CINFO 7, any window siz
2201   unsigned FLEVEL = 0;
2202   unsigned FDICT = 0;
2203   unsigned CMFFLG = 256 * CMF + FDICT * 32 + FLEVEL * 64;
2204   unsigned FCHECK = 31 - CMFFLG % 31;
2205   CMFFLG += FCHECK;
2206
2207   /*ucvector-controlled version of the output buffer, for dynamic array*/
2208   ucvector_init_buffer(&outv, *out, *outsize);
2209
2210   ucvector_push_back(&outv, (unsigned char)(CMFFLG >> 8));
2211   ucvector_push_back(&outv, (unsigned char)(CMFFLG & 255));
2212
2213   error = deflate(&deflatedata, &deflatesize, in, insize, settings);
2214
2215   if(!error)
2216   {
2217     unsigned ADLER32 = adler32(in, (unsigned)insize);
2218     for(i = 0; i != deflatesize; ++i) ucvector_push_back(&outv, deflatedata[i])
2219     lodepng_free(deflatedata);
2220     lodepng_add32bitInt(&outv, ADLER32);
2221   }
```

```
2222
2223   *out = outv.data;
2224   *outsize = outv.size;
2225
2226   return error;
2227 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.151 lodepng_zlib_decompress()

```
unsigned lodepng_zlib_decompress (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGDecompressSettings * settings )
```

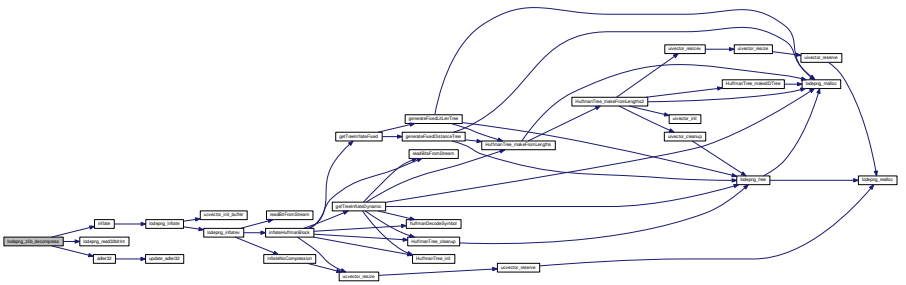
Definition at line 2126 of file lodepng.cpp.

```
2128 {
2129   unsigned error = 0;
2130   unsigned CM, CINFO, FDICT;
2131
2132   if(insize < 2) return 53; /*error, size of zlib data too small*/
2133   /*read information from zlib header*/
2134   if((in[0] * 256 + in[1]) % 31 != 0)
2135   {
2136     /*error: 256 * in[0] + in[1] must be a multiple of 31, the FCHECK value is
```

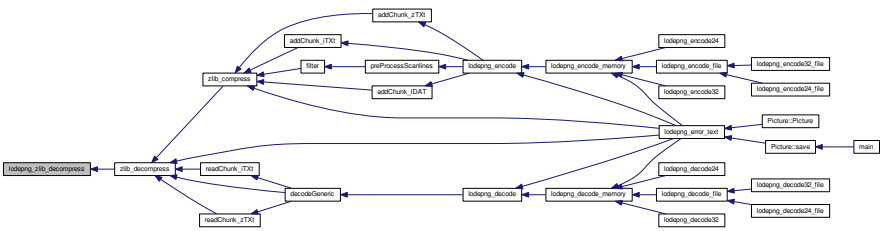
```
    */
2137      return 24;
2138    }
2139
2140    CM = in[0] & 15;
2141    CINFO = (in[0] >> 4) & 15;
2142    /*FCHECK = in[1] & 31;*/ /*FCHECK is already tested above*/
2143    FDICT = (in[1] >> 5) & 1;
2144    /*FLEVEL = (in[1] >> 6) & 3;*/ /*FLEVEL is not used here*/
2145
2146    if(CM != 8 || CINFO > 7)
2147    {
2148      /*error: only compression method 8: inflate with sliding window of 32k is s
2149      return 25;
2150    }
2151    if(FDICT != 0)
2152    {
2153      /*error: the specification of PNG says about the zlib stream:
2154        "The additional flags shall not specify a preset dictionary."*/
2155      return 26;
2156    }
2157
2158    error = inflate(out, outsize, in + 2, insize - 2, settings);
2159    if(error) return error;
2160
2161    if(!settings->ignore_adler32)
2162    {
2163      unsigned ADLER32 = lodepng_read32bitInt(&in[insize - 4]);
2164      unsigned checksum = adler32(*out, (unsigned)(*outsize));
2165      if(checksum != ADLER32) return 58; /*error, adler checksum not correct, dat
2166    }
```

```
2167
2168   return 0; /*no error*/
2169 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



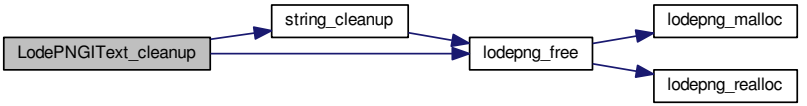
4.1.3.152 LodePNGIText_cleanup()

```
static void LodePNGIText_cleanup (
            LodePNGInfo * info )  [static]
```

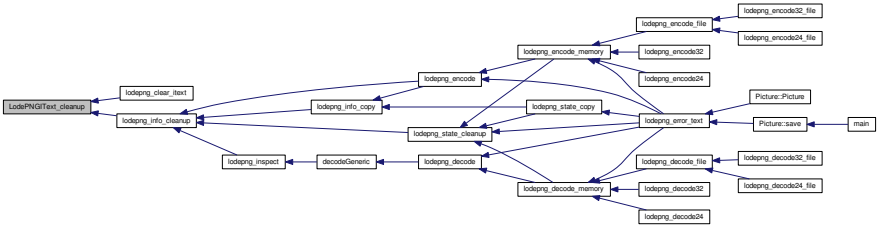
Definition at line 2848 of file lodepng.cpp.

```
2849 {
2850   size_t i;
2851   for(i = 0; i != info->itext_num; ++i)
2852   {
2853     string_cleanup(&info->itext_keys[i]);
2854     string_cleanup(&info->itext_langtags[i]);
2855     string_cleanup(&info->itext_transkeys[i]);
2856     string_cleanup(&info->itext_strings[i]);
2857   }
2858   lodepng_free(info->itext_keys);
2859   lodepng_free(info->itext_langtags);
2860   lodepng_free(info->itext_transkeys);
2861   lodepng_free(info->itext_strings);
2862 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.153 LodePNGIText_copy()

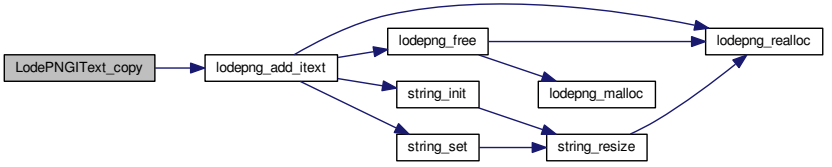
```
static unsigned LodePNGIText_copy (
            LodePNGInfo * dest,
            const LodePNGInfo * source )  [static]
```

Definition at line 2864 of file lodepng.cpp.

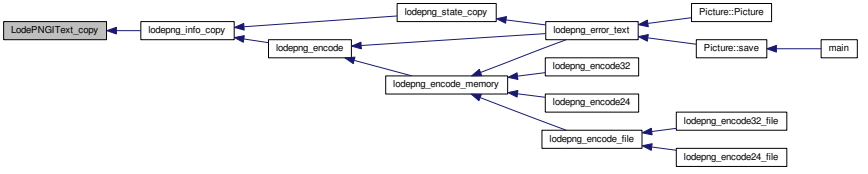
```
2865 {
2866   size_t i = 0;
2867   dest->itext_keys = 0;
2868   dest->itext_langtags = 0;
2869   dest->itext_transkeys = 0;
2870   dest->itext_strings = 0;
2871   dest->itext_num = 0;
2872   for(i = 0; i != source->itext_num; ++i)
2873   {
2874     CERROR_TRY_RETURN(lodepng_add_itext(dest, source->
```

```
      itext_keys[i], source->itext_langtags[i],
2875                                          source->itext_transkeys[i], source->
      itext_strings[i]));
2876   }
2877   return 0;
2878 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



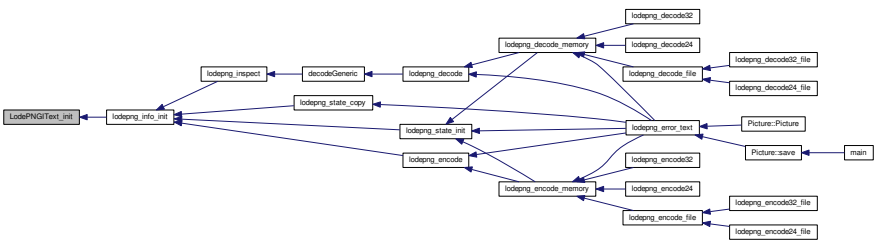
4.1.3.154 LodePNGIText_init()

```
static void LodePNGIText_init (
            LodePNGInfo * info )  [static]
```

Definition at line 2839 of file lodepng.cpp.

```
2840 {
2841   info->itext_num = 0;
2842   info->itext_keys = NULL;
2843   info->itext_langtags = NULL;
2844   info->itext_transkeys = NULL;
2845   info->itext_strings = NULL;
2846 }
```

Here is the caller graph for this function:



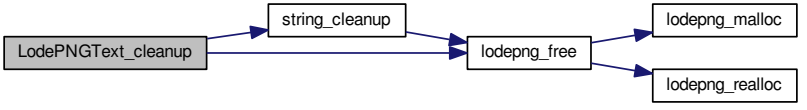
4.1.3.155 LodePNGText_cleanup()

```
static void LodePNGText_cleanup (
            LodePNGInfo * info )  [static]
```

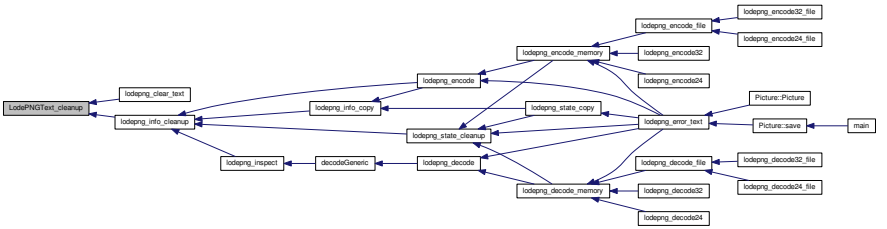
Definition at line 2783 of file lodepng.cpp.

```
2784 {
2785   size_t i;
2786   for(i = 0; i != info->text_num; ++i)
2787   {
2788     string_cleanup(&info->text_keys[i]);
2789     string_cleanup(&info->text_strings[i]);
2790   }
2791   lodepng_free(info->text_keys);
2792   lodepng_free(info->text_strings);
2793 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



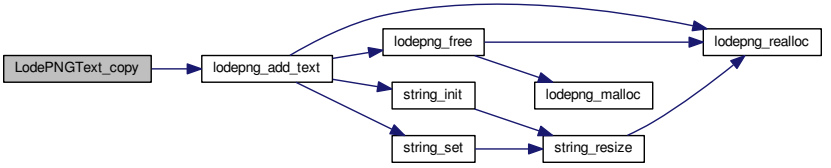
4.1.3.156 LodePNGText_copy()

```
static unsigned LodePNGText_copy (
            LodePNGInfo * dest,
            const LodePNGInfo * source )  [static]
```

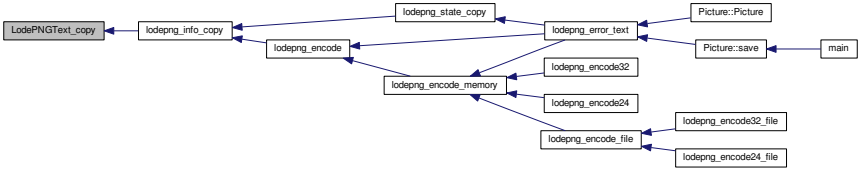
Definition at line 2795 of file lodepng.cpp.

```
2796 {
2797   size_t i = 0;
2798   dest->text_keys = 0;
2799   dest->text_strings = 0;
2800   dest->text_num = 0;
2801   for(i = 0; i != source->text_num; ++i)
2802   {
2803     CERROR_TRY_RETURN(lodepng_add_text(dest, source->
      text_keys[i], source->text_strings[i]));
2804   }
2805   return 0;
2806 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



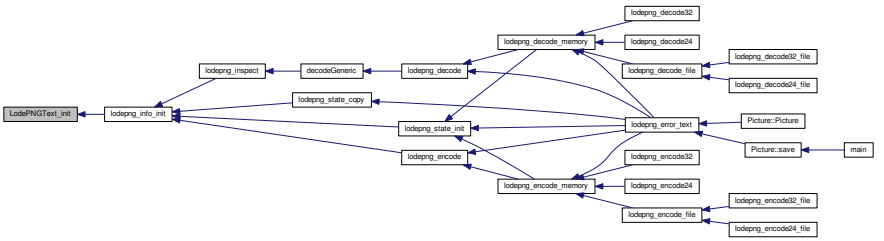
4.1.3.157 LodePNGText_init()

```
static void LodePNGText_init (
            LodePNGInfo * info )  [static]
```

Definition at line 2776 of file lodepng.cpp.

```
2777 {
2778   info->text_num = 0;
2779   info->text_keys = NULL;
2780   info->text_strings = NULL;
2781 }
```

Here is the caller graph for this function:



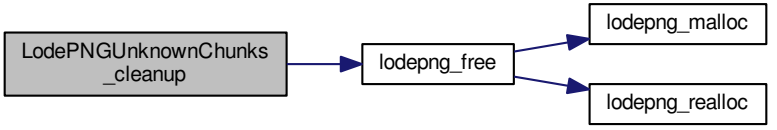
4.1.3.158 LodePNGUnknownChunks_cleanup()

```
static void LodePNGUnknownChunks_cleanup (
            LodePNGInfo * info )  [static]
```

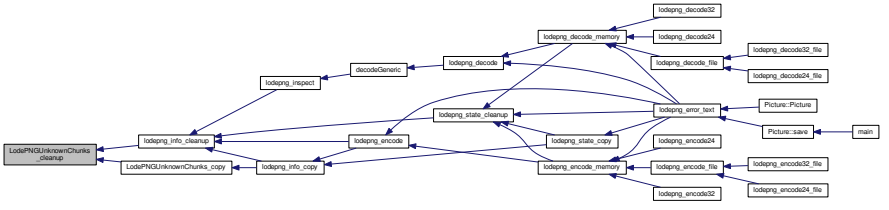
Definition at line 2747 of file lodepng.cpp.

```
2748 {
2749   unsigned i;
2750   for(i = 0; i != 3; ++i) lodepng_free(info->unknown_chunks_data[i]);
2751 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



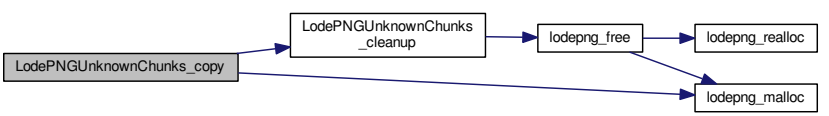
4.1.3.159 LodePNGUnknownChunks_copy()

```
static unsigned LodePNGUnknownChunks_copy (
            LodePNGInfo * dest,
            const LodePNGInfo * src )  [static]
```

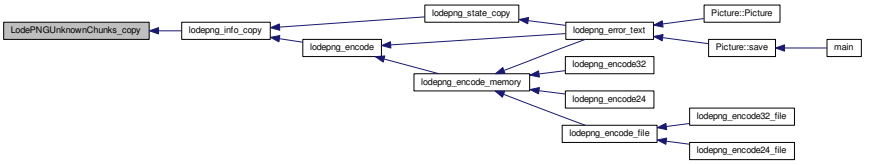
Definition at line 2753 of file lodepng.cpp.

```
2754 {
2755   unsigned i;
2756
2757   LodePNGUnknownChunks_cleanup(dest);
2758
2759   for(i = 0; i != 3; ++i)
2760   {
2761     size_t j;
2762     dest->unknown_chunks_size[i] = src->unknown_chunks_size[i];
2763     dest->unknown_chunks_data[i] = (unsigned char*)
      lodepng_malloc(src->unknown_chunks_size[i]);
2764     if(!dest->unknown_chunks_data[i] && dest->
      unknown_chunks_size[i]) return 83; /*alloc fail*/
2765     for(j = 0; j < src->unknown_chunks_size[i]; ++j)
2766     {
2767       dest->unknown_chunks_data[i][j] = src->
      unknown_chunks_data[i][j];
2768     }
2769   }
2770
2771   return 0;
2772 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



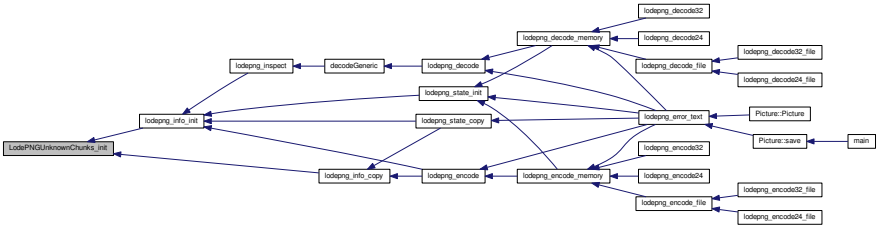
4.1.3.160 LodePNGUnknownChunks_init()

```
static void LodePNGUnknownChunks_init (
            LodePNGInfo * info )  [static]
```

Definition at line 2740 of file lodepng.cpp.

```
2741 {
2742   unsigned i;
2743   for(i = 0; i != 3; ++i) info->unknown_chunks_data[i] = 0;
2744   for(i = 0; i != 3; ++i) info->unknown_chunks_size[i] = 0;
2745 }
```

Here is the caller graph for this function:



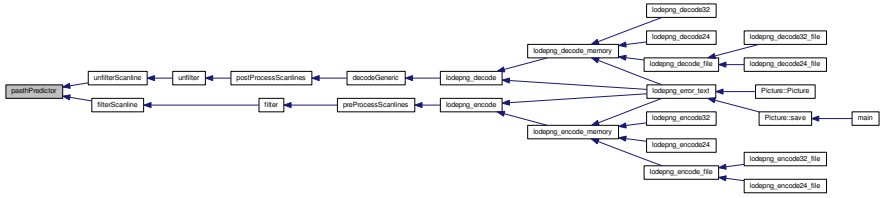
4.1.3.161 paethPredictor()

```
static unsigned char paethPredictor (
            short a,
            short b,
            short c )  [static]
```

Definition at line 3835 of file lodepng.cpp.

```
3836 {
3837   short pa = abs(b - c);
3838   short pb = abs(a - c);
3839   short pc = abs(a + b - c - c);
3840
3841   if(pc < pa && pc < pb) return (unsigned char)c;
3842   else if(pb < pa) return (unsigned char)b;
3843   else return (unsigned char)a;
3844 }
```

Here is the caller graph for this function:



4.1.3.162 postProcessScanlines()

```
static unsigned postProcessScanlines (
            unsigned char * out,
            unsigned char * in,
            unsigned w,
            unsigned h,
            const LodePNGInfo * info_png )  [static]
```

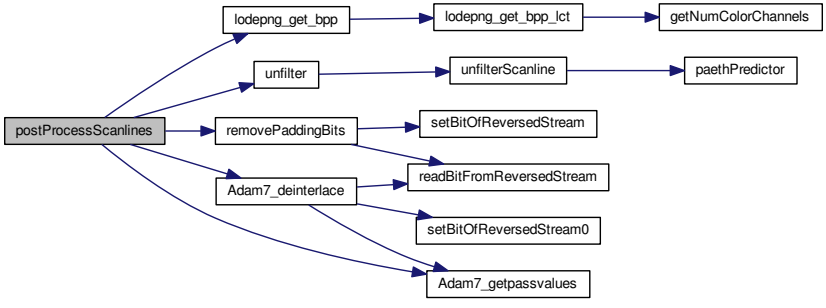
Definition at line 4165 of file lodepng.cpp.

```
4167 {
4168   /*
4169   This function converts the filtered-padded-interlaced data into pure 2D image
       colortype.
4170   Steps:
4171   *) if no Adam7: 1) unfilter 2) remove padding bits (= posible extra bits per
4172   *) if adam7: 1) 7x unfilter 2) 7x remove padding bits 3) Adam7_deinterlace
4173   NOTE: the in buffer will be overwritten with intermediate data!
```

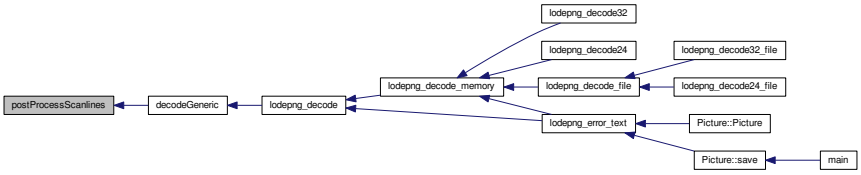
```
4174   */
4175   unsigned bpp = lodepng_get_bpp(&info_png->color);
4176   if(bpp == 0) return 31; /*error: invalid colortype*/
4177
4178   if(info_png->interlace_method == 0)
4179   {
4180     if(bpp < 8 && w * bpp != ((w * bpp + 7) / 8) * 8)
4181     {
4182       CERROR_TRY_RETURN(unfilter(in, in, w, h, bpp));
4183       removePaddingBits(out, in, w * bpp, ((w * bpp + 7) / 8) * 8, h);
4184     }
4185     /*we can immediately filter into the out buffer, no other steps needed*/
4186     else CERROR_TRY_RETURN(unfilter(out, in, w, h, bpp));
4187   }
4188   else /*interlace_method is 1 (Adam7)*/
4189   {
4190     unsigned passw[7], passh[7]; size_t filter_passstart[8], padded_passstart[8
4191     unsigned i;
4192
4193     Adam7_getpassvalues(passw, passh, filter_passstart, padded_passstart, passs
     h, bpp);
4194
4195     for(i = 0; i != 7; ++i)
4196     {
4197       CERROR_TRY_RETURN(unfilter(&in[padded_passstart[i]], &in[filter_passstart
     ], passw[i], passh[i], bpp));
4198       /*TODO: possible efficiency improvement: if in this reduced image the bit
4199       move bytes instead of bits or move not at all*/
4200       if(bpp < 8)
4201       {
4202         /*remove padding bits in scanlines; after this there still may be paddi
```

```
4203         bits between the different reduced images: each reduced image still sta
4204         removePaddingBits(&in[passstart[i]], &in[padded_passstart[i]], passw[i]
4205                           ((passw[i] * bpp + 7) / 8) * 8, passh[i]);
4206       }
4207     }
4208
4209     Adam7_deinterlace(out, in, w, h, bpp);
4210   }
4211
4212   return 0;
4213 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.163 preProcessScanlines()

```
static unsigned preProcessScanlines (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            unsigned w,
            unsigned h,
            const LodePNGInfo * info_png,
            const LodePNGEncoderSettings * settings )  [static]
```

Definition at line 5513 of file lodepng.cpp.

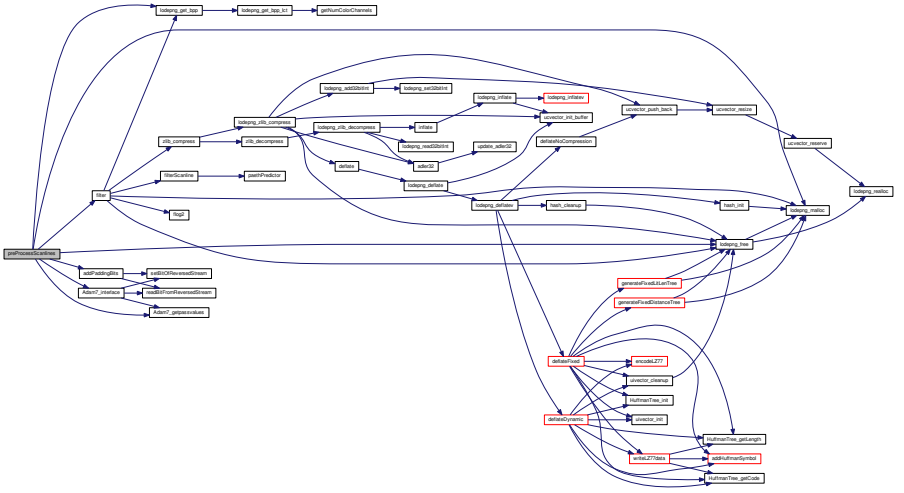
```
5516 {
5517   /*
5518   This function converts the pure 2D image with the PNG's colortype, into filte
       Steps:
5519   *) if no Adam7: 1) add padding bits (= posible extra bits per scanline if bpp
5520   *) if adam7: 1) Adam7_interlace 2) 7x add padding bits 3) 7x filter
```

```
5521   */
5522   unsigned bpp = lodepng_get_bpp(&info_png->color);
5523   unsigned error = 0;
5524
5525   if(info_png->interlace_method == 0)
5526   {
5527     *outsize = h + (h * ((w * bpp + 7) / 8)); /*image size plus an extra byte p
       padding bits*/
5528     *out = (unsigned char*)lodepng_malloc(*outsize);
5529     if(!(*out) && (*outsize)) error = 83; /*alloc fail*/
5530
5531     if(!error)
5532     {
5533       /*non multiple of 8 bits per scanline, padding bits needed per scanline*/
5534       if(bpp < 8 && w * bpp != ((w * bpp + 7) / 8) * 8)
5535       {
5536         unsigned char* padded = (unsigned char*)lodepng_malloc(h * ((w * bpp +
5537         if(!padded) error = 83; /*alloc fail*/
5538         if(!error)
5539         {
5540           addPaddingBits(padded, in, ((w * bpp + 7) / 8) * 8, w * bpp, h);
5541           error = filter(*out, padded, w, h, &info_png->color, settings);
5542         }
5543         lodepng_free(padded);
5544       }
5545       else
5546       {
5547         /*we can immediately filter into the out buffer, no other steps needed*
5548         error = filter(*out, in, w, h, &info_png->color, settings);
5549       }
5550     }
```

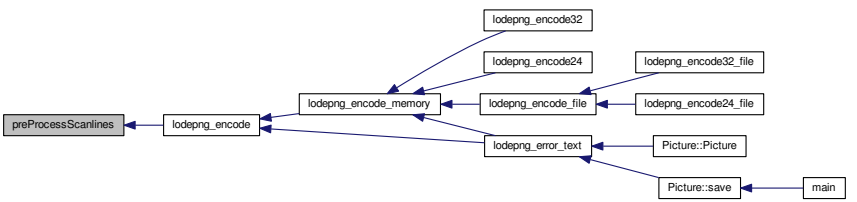
```
5551   }
5552   else /*interlace_method is 1 (Adam7)*/
5553   {
5554     unsigned passw[7], passh[7];
5555     size_t filter_passstart[8], padded_passstart[8], passstart[8];
5556     unsigned char* adam7;
5557
5558     Adam7_getpassvalues(passw, passh, filter_passstart, padded_passstart, passs
      h, bpp);
5559
5560     *outsize = filter_passstart[7]; /*image size plus an extra byte per scanlin
5561     *out = (unsigned char*)lodepng_malloc(*outsize);
5562     if(!(*out)) error = 83; /*alloc fail*/
5563
5564     adam7 = (unsigned char*)lodepng_malloc(passstart[7]);
5565     if(!adam7 && passstart[7]) error = 83; /*alloc fail*/
5566
5567     if(!error)
5568     {
5569       unsigned i;
5570
5571       Adam7_interlace(adam7, in, w, h, bpp);
5572       for(i = 0; i != 7; ++i)
5573       {
5574         if(bpp < 8)
5575         {
5576           unsigned char* padded = (unsigned char*)lodepng_malloc(padded_passsta
      padded_passstart[i]);
5577           if(!padded) ERROR_BREAK(83); /*alloc fail*/
5578           addPaddingBits(padded, &adam7[passstart[i]],
5579                          ((passw[i] * bpp + 7) / 8) * 8, passw[i] * bpp, passh[
```

```
5580           error = filter(&(*out)[filter_passstart[i]], padded,
5581                          passw[i], passh[i], &info_png->color, settings);
5582           lodepng_free(padded);
5583         }
5584         else
5585         {
5586           error = filter(&(*out)[filter_passstart[i]], &adam7[padded_passstart[
5587                          passw[i], passh[i], &info_png->color, settings);
5588         }
5589
5590         if(error) break;
5591       }
5592     }
5593
5594     lodepng_free(adam7);
5595   }
5596
5597   return error;
5598 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



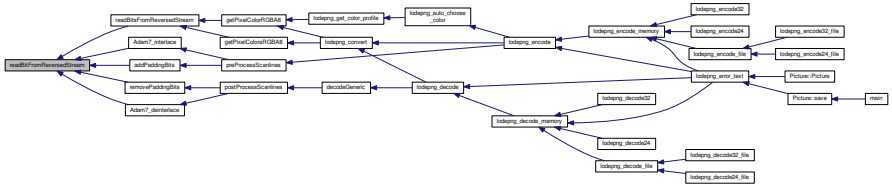
4.1.3.164 readBitFromReversedStream()

```
static unsigned char readBitFromReversedStream (
            size_t * bitpointer,
            const unsigned char * bitstream )  [static]
```

Definition at line 2377 of file lodepng.cpp.

```
2378 {
2379   unsigned char result = (unsigned char)((bitstream[(*bitpointer) >> 3] >> (7 -
      ));
2380   ++(*bitpointer);
2381   return result;
2382 }
```

Here is the caller graph for this function:



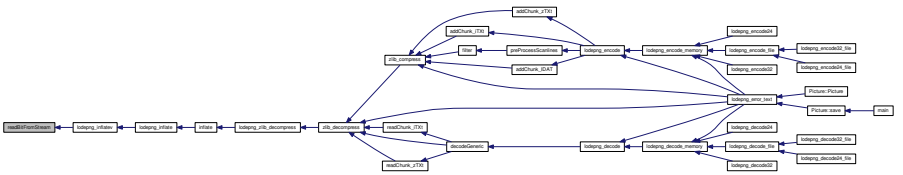
4.1.3.165 readBitFromStream()

```
static unsigned char readBitFromStream (
            size_t * bitpointer,
            const unsigned char * bitstream )  [static]
```

Definition at line 447 of file lodepng.cpp.

```
448 {
449   unsigned char result = (unsigned char)(READBIT(*bitpointer, bitstream));
450   ++(*bitpointer);
451   return result;
452 }
```

Here is the caller graph for this function:



4.1.3.166 readBitsFromReversedStream()

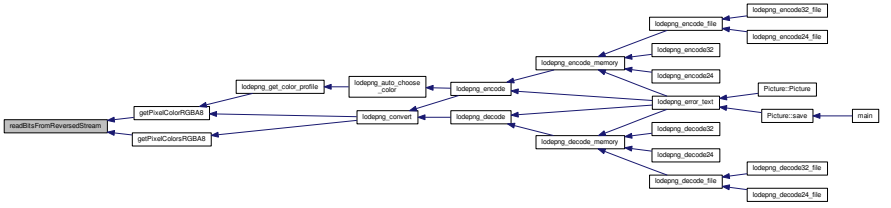
```
static unsigned readBitsFromReversedStream (
            size_t * bitpointer,
            const unsigned char * bitstream,
            size_t nbits )  [static]
```

Definition at line 2384 of file lodepng.cpp.

```
2385 {
2386   unsigned result = 0;
2387   size_t i;
2388   for(i = 0 ; i < nbits; ++i)
2389   {
2390     result <<= 1;
2391     result |= (unsigned)readBitFromReversedStream(bitpointer, bitstream);
2392   }
2393   return result;
2394 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



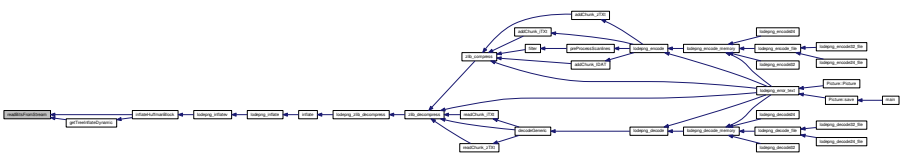
4.1.3.167 readBitsFromStream()

```
static unsigned readBitsFromStream (
            size_t * bitpointer,
            const unsigned char * bitstream,
            size_t nbits )  [static]
```

Definition at line 454 of file lodepng.cpp.

```
455 {
456   unsigned result = 0, i;
457   for(i = 0; i != nbits; ++i)
458   {
459     result += ((unsigned)READBIT(*bitpointer, bitstream)) << i;
460     ++(*bitpointer);
461   }
462   return result;
463 }
```

Here is the caller graph for this function:



4.1.3.168 readChunk_bKGD()

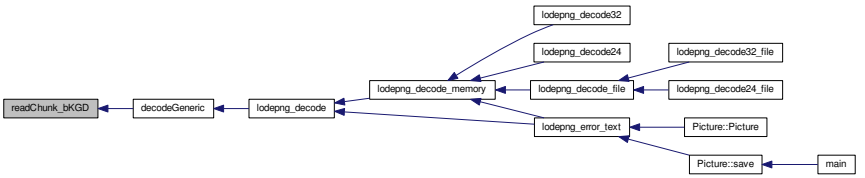
```
static unsigned readChunk_bKGD (
            LodePNGInfo * info,
            const unsigned char * data,
            size_t chunkLength )  [static]
```

Definition at line 4275 of file lodepng.cpp.

```
4276 {
4277   if(info->color.colortype == LCT_PALETTE)
4278   {
4279     /*error: this chunk must be 1 byte for indexed color image*/
4280     if(chunkLength != 1) return 43;
4281
4282     info->background_defined = 1;
4283     info->background_r = info->background_g = info->
      background_b = data[0];
4284   }
4285   else if(info->color.colortype == LCT_GREY || info->color.
      colortype == LCT_GREY_ALPHA)
4286   {
4287     /*error: this chunk must be 2 bytes for greyscale image*/
4288     if(chunkLength != 2) return 44;
4289
4290     info->background_defined = 1;
4291     info->background_r = info->background_g = info->
      background_b = 256u * data[0] + data[1];
4292   }
4293   else if(info->color.colortype == LCT_RGB || info->color.
      colortype == LCT_RGBA)
4294   {
```

```
4295     /*error: this chunk must be 6 bytes for greyscale image*/
4296     if(chunkLength != 6) return 45;
4297
4298     info->background_defined = 1;
4299     info->background_r = 256u * data[0] + data[1];
4300     info->background_g = 256u * data[2] + data[3];
4301     info->background_b = 256u * data[4] + data[5];
4302   }
4303
4304   return 0; /* OK */
4305 }
```

Here is the caller graph for this function:



4.1.3.169 readChunk_iTXt()

```
static unsigned readChunk_iTXt (
            LodePNGInfo * info,
            const LodePNGDecompressSettings * zlibsettings,
            const unsigned char * data,
            size_t chunkLength )  [static]
```

Definition at line 4400 of file lodepng.cpp.

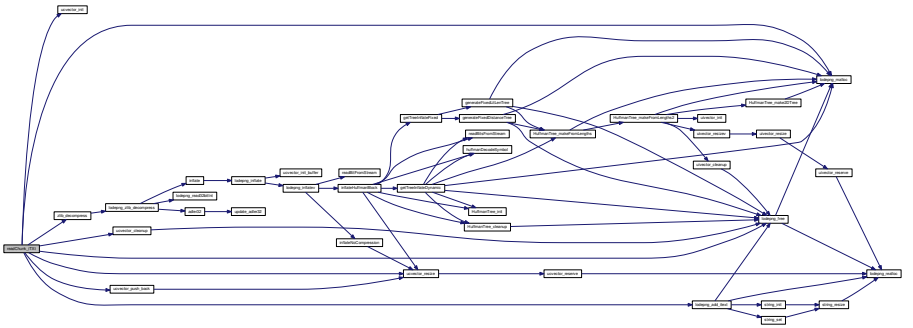
```
4402 {
4403   unsigned error = 0;
4404   unsigned i;
4405
4406   unsigned length, begin, compressed;
4407   char *key = 0, *langtag = 0, *transkey = 0;
4408   ucvector decoded;
4409   ucvector_init(&decoded);
4410
4411   while(!error) /*not really a while loop, only used to break on error*/
4412   {
4413     /*Quick check if the chunk length isn't too small. Even without check
4414     it'd still fail with other error checks below if it's too short. This just
       code.*/
4415     if(chunkLength < 5) CERROR_BREAK(error, 30); /*iTXt chunk too short*/
4416
4417     /*read the key*/
4418     for(length = 0; length < chunkLength && data[length] != 0; ++length) ;
4419     if(length + 3 >= chunkLength) CERROR_BREAK(error, 75); /*no null terminatio
      */
4420     if(length < 1 || length > 79) CERROR_BREAK(error, 89); /*keyword too short
4421
4422     key = (char*)lodepng_malloc(length + 1);
4423     if(!key) CERROR_BREAK(error, 83); /*alloc fail*/
4424
4425     key[length] = 0;
4426     for(i = 0; i != length; ++i) key[i] = (char)data[i];
4427
4428     /*read the compression method*/
4429     compressed = data[length + 1];
4430     if(data[length + 2] != 0) CERROR_BREAK(error, 72); /*the 0 byte indicating
```

```
      be 0*/
4431
4432     /*even though it's not allowed by the standard, no error is thrown if
4433     there's no null termination char, if the text is empty for the next 3 texts
4434
4435     /*read the langtag*/
4436     begin = length + 3;
4437     length = 0;
4438     for(i = begin; i < chunkLength && data[i] != 0; ++i) ++length;
4439
4440     langtag = (char*)lodepng_malloc(length + 1);
4441     if(!langtag) CERROR_BREAK(error, 83); /*alloc fail*/
4442
4443     langtag[length] = 0;
4444     for(i = 0; i != length; ++i) langtag[i] = (char)data[begin + i];
4445
4446     /*read the transkey*/
4447     begin += length + 1;
4448     length = 0;
4449     for(i = begin; i < chunkLength && data[i] != 0; ++i) ++length;
4450
4451     transkey = (char*)lodepng_malloc(length + 1);
4452     if(!transkey) CERROR_BREAK(error, 83); /*alloc fail*/
4453
4454     transkey[length] = 0;
4455     for(i = 0; i != length; ++i) transkey[i] = (char)data[begin + i];
4456
4457     /*read the actual text*/
4458     begin += length + 1;
4459
4460     length = chunkLength < begin ? 0 : chunkLength - begin;
```

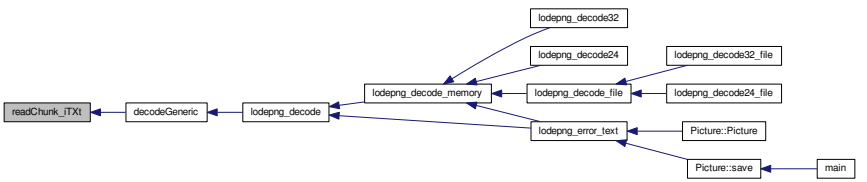
```
4461
4462     if(compressed)
4463     {
4464       /*will fail if zlib error, e.g. if length is too small*/
4465       error = zlib_decompress(&decoded.data, &decoded.size,
4466                               (unsigned char*)(&data[begin]),
4467                               length, zlibsettings);
4468       if(error) break;
4469       if(decoded.allocsize < decoded.size) decoded.allocsize = decoded.
     size;
4470       ucvector_push_back(&decoded, 0);
4471     }
4472     else
4473     {
4474       if(!ucvector_resize(&decoded, length + 1)) CERROR_BREAK(error, 83 /*alloc
      fail*/);
4475
4476       decoded.data[length] = 0;
4477       for(i = 0; i != length; ++i) decoded.data[i] = data[begin + i];
4478     }
4479
4480     error = lodepng_add_itext(info, key, langtag, transkey, (char*)decoded.
     data);
4481
4482     break;
4483   }
4484
4485   lodepng_free(key);
4486   lodepng_free(langtag);
4487   lodepng_free(transkey);
4488   ucvector_cleanup(&decoded);
```

```
4489
4490   return error;
4491 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



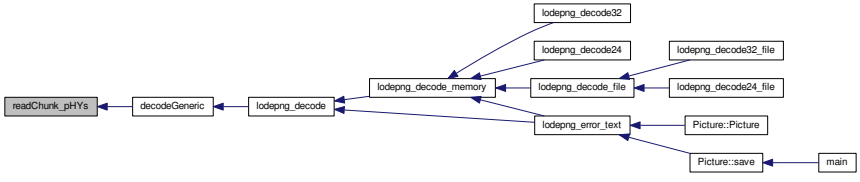
4.1.3.170 readChunk_pHYs()

```
static unsigned readChunk_pHYs (
            LodePNGInfo * info,
            const unsigned char * data,
            size_t chunkLength )  [static]
```

Definition at line 4508 of file lodepng.cpp.

```
4509 {
4510   if(chunkLength != 9) return 74; /*invalid pHYs chunk size*/
4511
4512   info->phys_defined = 1;
4513   info->phys_x = 16777216u * data[0] + 65536u * data[1] + 256u * data[2] + data
4514   info->phys_y = 16777216u * data[4] + 65536u * data[5] + 256u * data[6] + data
4515   info->phys_unit = data[8];
4516
4517   return 0; /* OK */
4518 }
```

Here is the caller graph for this function:



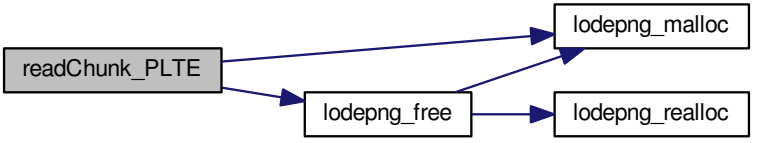
4.1.3.171 readChunk_PLTE()

```
static unsigned readChunk_PLTE (
            LodePNGColorMode * color,
            const unsigned char * data,
            size_t chunkLength )  [static]
```

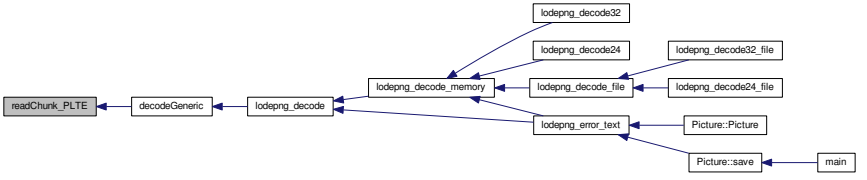
Definition at line 4215 of file lodepng.cpp.

```
4216 {
4217   unsigned pos = 0, i;
4218   if(color->palette) lodepng_free(color->palette);
4219   color->palettesize = chunkLength / 3;
4220   color->palette = (unsigned char*)lodepng_malloc(4 * color->
      palettesize);
4221   if(!color->palette && color->palettesize)
4222   {
4223     color->palettesize = 0;
4224     return 83; /*alloc fail*/
4225   }
4226   if(color->palettesize > 256) return 38; /*error: palette too big*/
4227
4228   for(i = 0; i != color->palettesize; ++i)
4229   {
4230     color->palette[4 * i + 0] = data[pos++]; /*R*/
4231     color->palette[4 * i + 1] = data[pos++]; /*G*/
4232     color->palette[4 * i + 2] = data[pos++]; /*B*/
4233     color->palette[4 * i + 3] = 255; /*alpha*/
4234   }
4235
4236   return 0; /* OK */
4237 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.172 readChunk_tEXt()

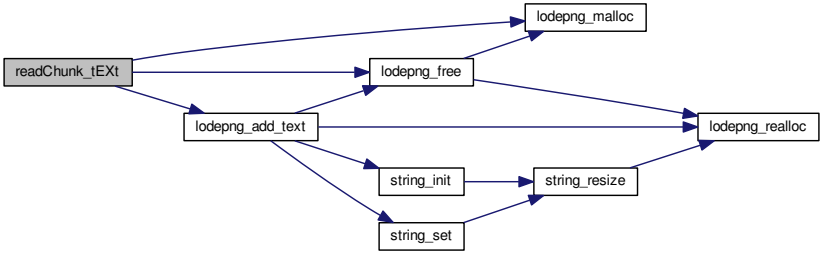
```
static unsigned readChunk_tEXt (
            LodePNGInfo * info,
            const unsigned char * data,
            size_t chunkLength )  [static]
```

Definition at line 4308 of file lodepng.cpp.

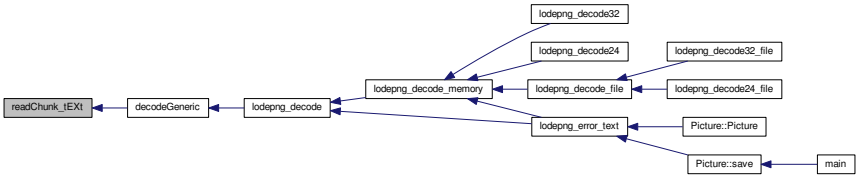
```
4309 {
4310   unsigned error = 0;
4311   char *key = 0, *str = 0;
4312   unsigned i;
4313
4314   while(!error) /*not really a while loop, only used to break on error*/
4315   {
4316     unsigned length, string2_begin;
4317
4318     length = 0;
4319     while(length < chunkLength && data[length] != 0) ++length;
4320     /*even though it's not allowed by the standard, no error is thrown if
4321     there's no null termination char, if the text is empty*/
4322     if(length < 1 || length > 79) CERROR_BREAK(error, 89); /*keyword too short
4323
4324     key = (char*)lodepng_malloc(length + 1);
4325     if(!key) CERROR_BREAK(error, 83); /*alloc fail*/
4326
4327     key[length] = 0;
4328     for(i = 0; i != length; ++i) key[i] = (char)data[i];
4329
4330     string2_begin = length + 1; /*skip keyword null terminator*/
4331
4332     length = chunkLength < string2_begin ? 0 : chunkLength - string2_begin;
4333     str = (char*)lodepng_malloc(length + 1);
4334     if(!str) CERROR_BREAK(error, 83); /*alloc fail*/
4335
4336     str[length] = 0;
4337     for(i = 0; i != length; ++i) str[i] = (char)data[string2_begin + i];
4338
4339     error = lodepng_add_text(info, key, str);
```

```
4340
4341     break;
4342   }
4343
4344   lodepng_free(key);
4345   lodepng_free(str);
4346
4347   return error;
4348 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.173 readChunk_tIME()

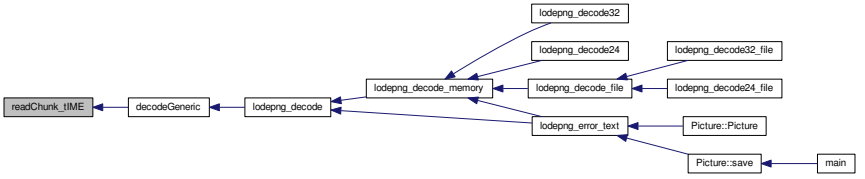
```
static unsigned readChunk_tIME (
            LodePNGInfo * info,
            const unsigned char * data,
            size_t chunkLength )  [static]
```

Definition at line 4493 of file lodepng.cpp.

```
4494 {
4495   if(chunkLength != 7) return 73; /*invalid tIME chunk size*/
4496
4497   info->time_defined = 1;
4498   info->time.year = 256u * data[0] + data[1];
4499   info->time.month = data[2];
4500   info->time.day = data[3];
4501   info->time.hour = data[4];
4502   info->time.minute = data[5];
4503   info->time.second = data[6];
```

```
4504
4505   return 0; /* OK */
4506 }
```

Here is the caller graph for this function:



4.1.3.174 readChunk_tRNS()

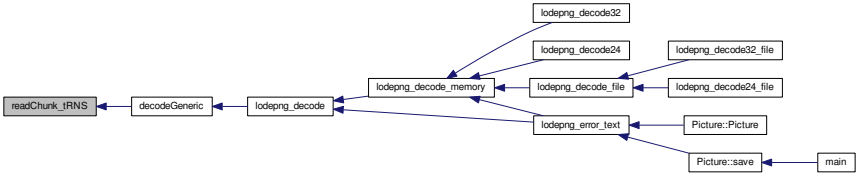
```
static unsigned readChunk_tRNS (
            LodePNGColorMode * color,
            const unsigned char * data,
            size_t chunkLength )  [static]
```

Definition at line 4239 of file lodepng.cpp.

```
4240 {
4241   unsigned i;
4242   if(color->colortype == LCT_PALETTE)
4243   {
4244     /*error: more alpha values given than there are palette entries*/
```

```
4245     if(chunkLength > color->palettesize) return 38;
4246
4247     for(i = 0; i != chunkLength; ++i) color->palette[4 * i + 3] = data[i];
4248   }
4249   else if(color->colortype == LCT_GREY)
4250   {
4251     /*error: this chunk must be 2 bytes for greyscale image*/
4252     if(chunkLength != 2) return 30;
4253
4254     color->key_defined = 1;
4255     color->key_r = color->key_g = color->key_b = 256u * data[0] + data[1];
4256   }
4257   else if(color->colortype == LCT_RGB)
4258   {
4259     /*error: this chunk must be 6 bytes for RGB image*/
4260     if(chunkLength != 6) return 41;
4261
4262     color->key_defined = 1;
4263     color->key_r = 256u * data[0] + data[1];
4264     color->key_g = 256u * data[2] + data[3];
4265     color->key_b = 256u * data[4] + data[5];
4266   }
4267   else return 42; /*error: tRNS chunk not allowed for other color models*/
4268
4269   return 0; /* OK */
4270 }
```

Here is the caller graph for this function:



4.1.3.175 readChunk_zTXt()

```
static unsigned readChunk_zTXt (
            LodePNGInfo * info,
            const LodePNGDecompressSettings * zlibsettings,
            const unsigned char * data,
            size_t chunkLength )  [static]
```

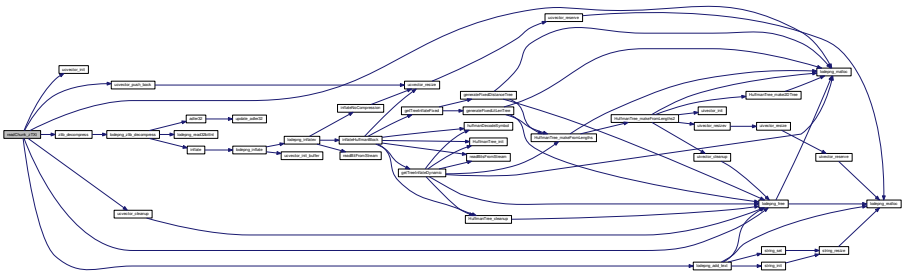
Definition at line 4351 of file lodepng.cpp.

```
4353 {
4354   unsigned error = 0;
4355   unsigned i;
4356
4357   unsigned length, string2_begin;
4358   char *key = 0;
4359   ucvector decoded;
4360
4361   ucvector_init(&decoded);
```

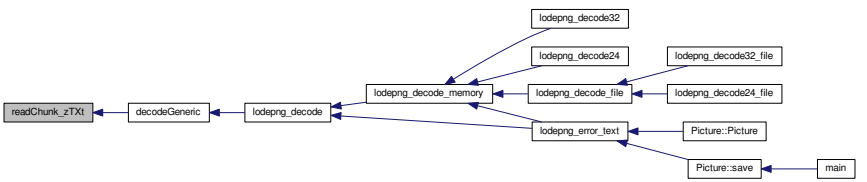
```
4362
4363   while(!error) /*not really a while loop, only used to break on error*/
4364   {
4365     for(length = 0; length < chunkLength && data[length] != 0; ++length) ;
4366     if(length + 2 >= chunkLength) CERROR_BREAK(error, 75); /*no null terminatio
4367     if(length < 1 || length > 79) CERROR_BREAK(error, 89); /*keyword too short
4368
4369     key = (char*)lodepng_malloc(length + 1);
4370     if(!key) CERROR_BREAK(error, 83); /*alloc fail*/
4371
4372     key[length] = 0;
4373     for(i = 0; i != length; ++i) key[i] = (char)data[i];
4374
4375     if(data[length + 1] != 0) CERROR_BREAK(error, 72); /*the 0 byte indicating
       be 0*/
4376
4377     string2_begin = length + 2;
4378     if(string2_begin > chunkLength) CERROR_BREAK(error, 75); /*no null terminat
4379
4380     length = chunkLength - string2_begin;
4381     /*will fail if zlib error, e.g. if length is too small*/
4382     error = zlib_decompress(&decoded.data, &decoded.size,
4383                             (unsigned char*)(&data[string2_begin]),
4384                             length, zlibsettings);
4385     if(error) break;
4386     ucvector_push_back(&decoded, 0);
4387
4388     error = lodepng_add_text(info, key, (char*)decoded.data);
4389
4390     break;
4391   }
```

```
4392
4393   lodepng_free(key);
4394   ucvector_cleanup(&decoded);
4395
4396   return error;
4397 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.176 removePaddingBits()

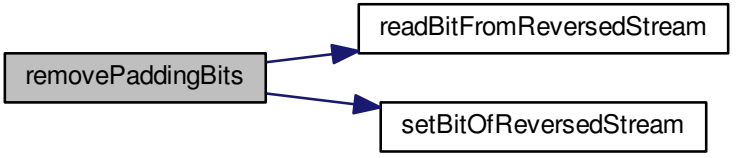
```
static void removePaddingBits (
            unsigned char * out,
            const unsigned char * in,
            size_t olinebits,
            size_t ilinebits,
            unsigned h )  [static]
```

Definition at line 4135 of file lodepng.cpp.

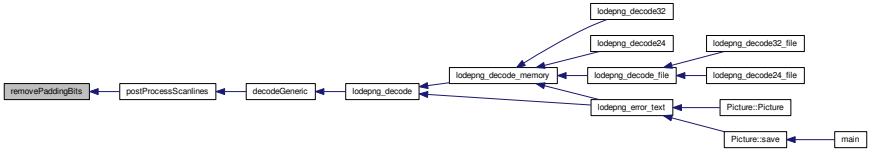
```
4137 {
4138   /*
4139   After filtering there are still padding bits if scanlines have non multiple o
4140   to be removed (except at last scanline of (Adam7-reduced) image) before worki
4141   for the Adam7 code, the color convert code and the output to the user.
4142   in and out are allowed to be the same buffer, in may also be higher but still
4143   have >= ilinebits*h bits, out must have >= olinebits*h bits, olinebits must b
4144   also used to move bits after earlier such operations happened, e.g. in a sequ
       Adam7
4145   only useful if (ilinebits - olinebits) is a value in the range 1..7
4146   */
4147   unsigned y;
4148   size_t diff = ilinebits - olinebits;
4149   size_t ibp = 0, obp = 0; /*input and output bit pointers*/
4150   for(y = 0; y < h; ++y)
4151   {
4152     size_t x;
4153     for(x = 0; x < olinebits; ++x)
4154     {
4155       unsigned char bit = readBitFromReversedStream(&ibp, in);
4156       setBitOfReversedStream(&obp, out, bit);
```

```
4157       }
4158       ibp += diff;
4159     }
4160 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.177 rgba16ToPixel()

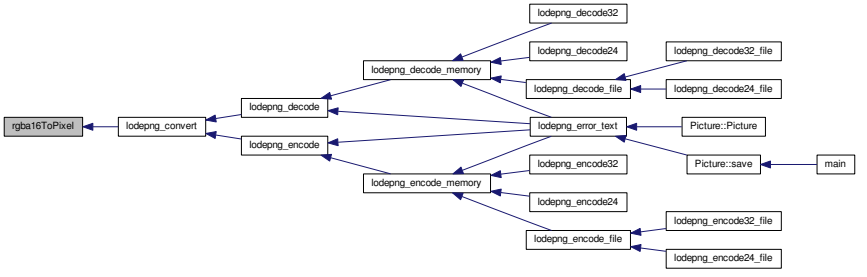
```
static void rgba16ToPixel (
            unsigned char * out,
            size_t i,
            const LodePNGColorMode * mode,
            unsigned short r,
            unsigned short g,
            unsigned short b,
            unsigned short a )  [static]
```

Definition at line 3140 of file lodepng.cpp.

```
3143 {
3144   if(mode->colortype == LCT_GREY)
3145   {
3146     unsigned short grey = r; /*((unsigned)r + g + b) / 3*/;
3147     out[i * 2 + 0] = (grey >> 8) & 255;
3148     out[i * 2 + 1] = grey & 255;
3149   }
3150   else if(mode->colortype == LCT_RGB)
3151   {
3152     out[i * 6 + 0] = (r >> 8) & 255;
3153     out[i * 6 + 1] = r & 255;
3154     out[i * 6 + 2] = (g >> 8) & 255;
3155     out[i * 6 + 3] = g & 255;
3156     out[i * 6 + 4] = (b >> 8) & 255;
3157     out[i * 6 + 5] = b & 255;
3158   }
3159   else if(mode->colortype == LCT_GREY_ALPHA)
3160   {
3161     unsigned short grey = r; /*((unsigned)r + g + b) / 3*/;
```

```
3162       out[i * 4 + 0] = (grey >> 8) & 255;
3163       out[i * 4 + 1] = grey & 255;
3164       out[i * 4 + 2] = (a >> 8) & 255;
3165       out[i * 4 + 3] = a & 255;
3166     }
3167     else if(mode->colortype == LCT_RGBA)
3168     {
3169       out[i * 8 + 0] = (r >> 8) & 255;
3170       out[i * 8 + 1] = r & 255;
3171       out[i * 8 + 2] = (g >> 8) & 255;
3172       out[i * 8 + 3] = g & 255;
3173       out[i * 8 + 4] = (b >> 8) & 255;
3174       out[i * 8 + 5] = b & 255;
3175       out[i * 8 + 6] = (a >> 8) & 255;
3176       out[i * 8 + 7] = a & 255;
3177     }
3178 }
```

Here is the caller graph for this function:



4.1.3.178 rgba8ToPixel()

```
static unsigned rgba8ToPixel (
            unsigned char * out,
            size_t i,
            const LodePNGColorMode * mode,
            ColorTree * tree,
            unsigned char r,
            unsigned char g,
            unsigned char b,
            unsigned char a )  [static]
```

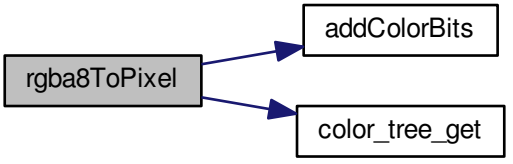
Definition at line 3066 of file lodepng.cpp.

```
3069 {
3070   if(mode->colortype == LCT_GREY)
3071   {
3072     unsigned char grey = r; /*((unsigned short)r + g + b) / 3*/;
3073     if(mode->bitdepth == 8) out[i] = grey;
3074     else if(mode->bitdepth == 16) out[i * 2 + 0] = out[i * 2 + 1] = grey;
3075     else
3076     {
3077       /*take the most significant bits of grey*/
3078       grey = (grey >> (8 - mode->bitdepth)) & ((1 << mode->bitdepth) - 1);
3079       addColorBits(out, i, mode->bitdepth, grey);
3080     }
3081   }
3082   else if(mode->colortype == LCT_RGB)
3083   {
3084     if(mode->bitdepth == 8)
3085     {
3086       out[i * 3 + 0] = r;
```

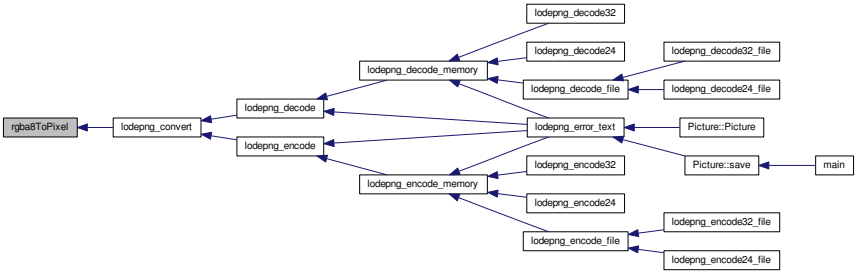
```
3087        out[i * 3 + 1] = g;
3088        out[i * 3 + 2] = b;
3089      }
3090      else
3091      {
3092        out[i * 6 + 0] = out[i * 6 + 1] = r;
3093        out[i * 6 + 2] = out[i * 6 + 3] = g;
3094        out[i * 6 + 4] = out[i * 6 + 5] = b;
3095      }
3096    }
3097    else if(mode->colortype == LCT_PALETTE)
3098    {
3099      int index = color_tree_get(tree, r, g, b, a);
3100      if(index < 0) return 82; /*color not in palette*/
3101      if(mode->bitdepth == 8) out[i] = index;
3102      else addColorBits(out, i, mode->bitdepth, (unsigned)index);
3103    }
3104    else if(mode->colortype == LCT_GREY_ALPHA)
3105    {
3106      unsigned char grey = r; /*((unsigned short)r + g + b) / 3*/;
3107      if(mode->bitdepth == 8)
3108      {
3109        out[i * 2 + 0] = grey;
3110        out[i * 2 + 1] = a;
3111      }
3112      else if(mode->bitdepth == 16)
3113      {
3114        out[i * 4 + 0] = out[i * 4 + 1] = grey;
3115        out[i * 4 + 2] = out[i * 4 + 3] = a;
3116      }
3117    }
```

```
3118   else if(mode->colortype == LCT_RGBA)
3119   {
3120     if(mode->bitdepth == 8)
3121     {
3122       out[i * 4 + 0] = r;
3123       out[i * 4 + 1] = g;
3124       out[i * 4 + 2] = b;
3125       out[i * 4 + 3] = a;
3126     }
3127     else
3128     {
3129       out[i * 8 + 0] = out[i * 8 + 1] = r;
3130       out[i * 8 + 2] = out[i * 8 + 3] = g;
3131       out[i * 8 + 4] = out[i * 8 + 5] = b;
3132       out[i * 8 + 6] = out[i * 8 + 7] = a;
3133     }
3134   }
3135
3136   return 0; /*no error*/
3137 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.179 searchCodeIndex()

```
static size_t searchCodeIndex (
            const unsigned * array,
            size_t array_size,
            size_t value )  [static]
```

Definition at line 1328 of file lodepng.cpp.

```
1329 {
1330   /*binary search (only small gain over linear). TODO: use CPU log2 instruction
      */
1331   size_t left = 1;
1332   size_t right = array_size - 1;
1333
1334   while(left <= right) {
1335     size_t mid = (left + right) >> 1;
1336     if (array[mid] >= value) right = mid - 1;
1337     else left = mid + 1;
1338   }
1339   if(left >= array_size || array[left] > value) left--;
1340   return left;
1341 }
```

Here is the caller graph for this function:

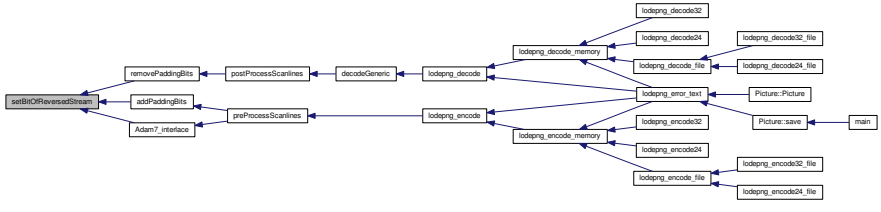
4.1.3.180 setBitOfReversedStream()

```
static void setBitOfReversedStream (
            size_t * bitpointer,
            unsigned char * bitstream,
            unsigned char bit )  [static]
```

Definition at line 2409 of file lodepng.cpp.

```
2410 {
2411   /*the current bit in bitstream may be 0 or 1 for this to work*/
2412   if(bit == 0) bitstream[(*bitpointer) >> 3] &=  (unsigned char)(~(1 << (7 - ((
2413   else         bitstream[(*bitpointer) >> 3] |=  (1 << (7 - ((*bitpointer) & 0x
2414   ++(*bitpointer);
2415 }
```

Here is the caller graph for this function:



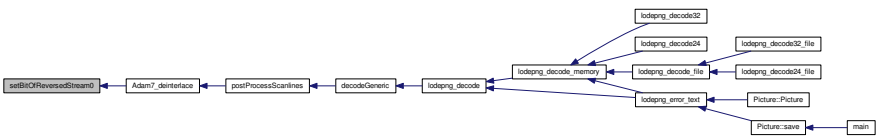
4.1.3.181 setBitOfReversedStream0()

```
static void setBitOfReversedStream0 (
            size_t * bitpointer,
            unsigned char * bitstream,
            unsigned char bit )  [static]
```

Definition at line 2397 of file lodepng.cpp.

```
2398 {
2399   /*the current bit in bitstream must be 0 for this to work*/
2400   if(bit)
2401   {
2402     /*earlier bit of huffman code is in a lesser significant bit of an earlier
2403     bitstream[(*bitpointer) >> 3] |= (bit << (7 - ((*bitpointer) & 0x7)));
2404   }
2405   ++(*bitpointer);
2406 }
```

Here is the caller graph for this function:



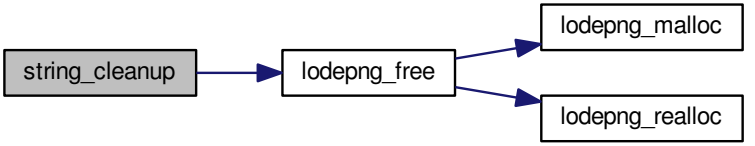
4.1.3.182 string_cleanup()

```
static void string_cleanup (
            char ** out )  [static]
```

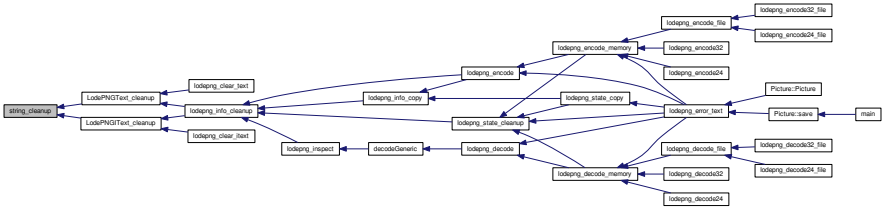
Definition at line 298 of file lodepng.cpp.

```
299 {
300   lodepng_free(*out);
301   *out = NULL;
302 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



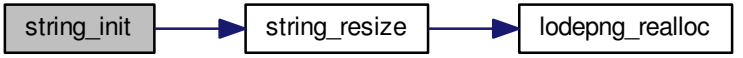
4.1.3.183 string_init()

```
static void string_init (
            char ** out )  [static]
```

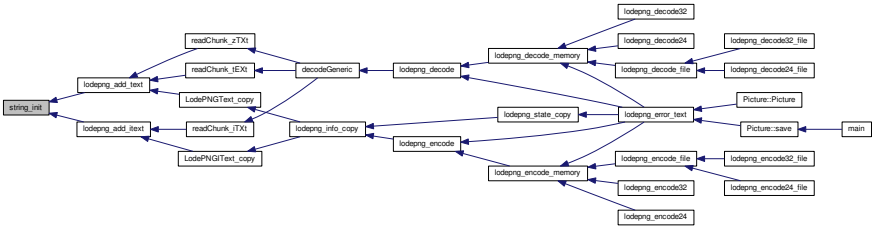
Definition at line 291 of file lodepng.cpp.

```
292 {
293   *out = NULL;
294   string_resize(out, 0);
295 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



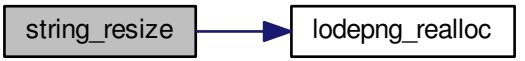
4.1.3.184 string_resize()

```
static unsigned string_resize (
            char ** out,
            size_t size )  [static]
```

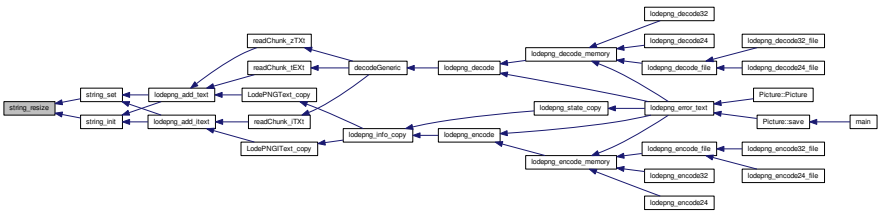
Definition at line 279 of file lodepng.cpp.

```
280 {
281   char* data = (char*)lodepng_realloc(*out, size + 1);
282   if(data)
283   {
284     data[size] = 0; /*null termination char*/
285     *out = data;
286   }
287   return data != 0;
288 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



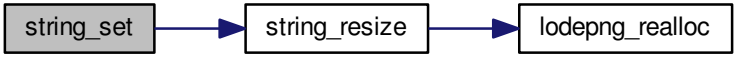
4.1.3.185 string_set()

```
static void string_set (
            char ** out,
            const char * in )  [static]
```

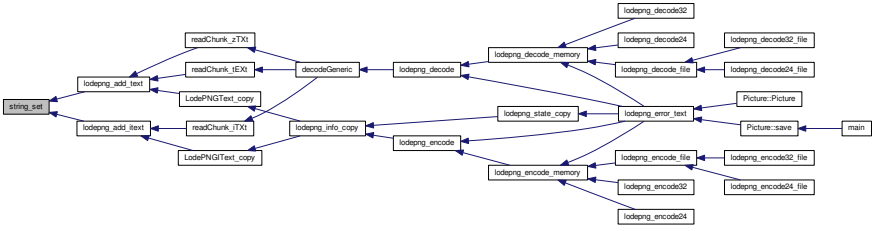
Definition at line 304 of file lodepng.cpp.

```
305 {
306   size_t insize = strlen(in), i;
307   if(string_resize(out, insize))
308   {
309     for(i = 0; i != insize; ++i)
310     {
311       (*out)[i] = in[i];
312     }
313   }
314 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.186 ucvector_cleanup()

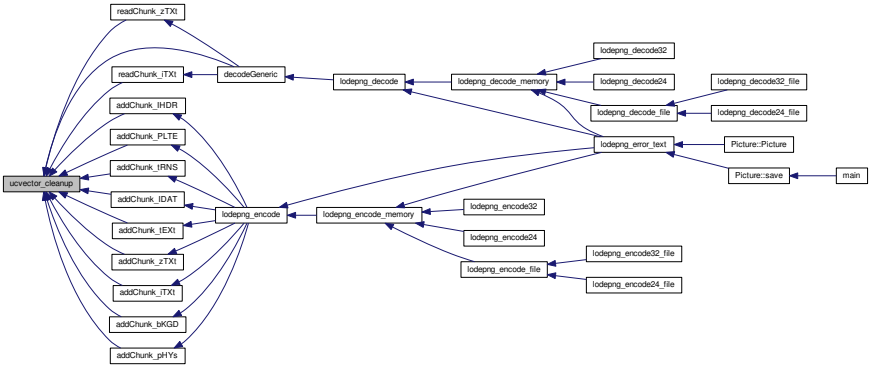
```
static void ucvector_cleanup (
            void * p )  [static]
```

Definition at line 239 of file lodepng.cpp.

```
240 {
241   ((ucvector*)p)->size = ((ucvector*)p)->allocsize = 0;
242   lodepng_free(((ucvector*)p)->data);
243   ((ucvector*)p)->data = NULL;
244 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



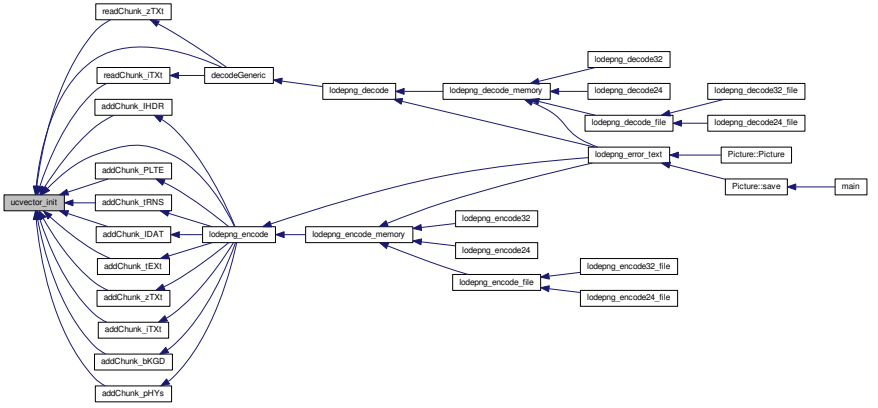
4.1.3.187 ucvector_init()

```
static void ucvector_init (
            ucvector * p )  [static]
```

Definition at line 246 of file lodepng.cpp.

```
247 {
248   p->data = NULL;
249   p->size = p->allocsize = 0;
250 }
```

Here is the caller graph for this function:



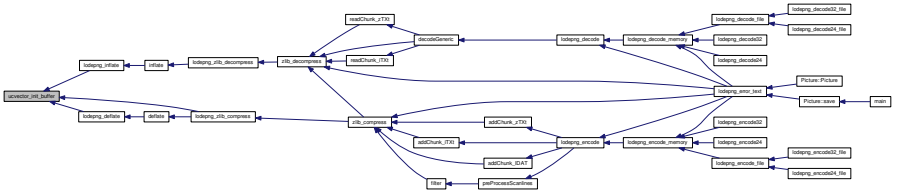
4.1.3.188 ucvector_init_buffer()

```
static void ucvector_init_buffer (
            ucvector * p,
            unsigned char * buffer,
            size_t size )  [static]
```

Definition at line 256 of file lodepng.cpp.

```
257 {
258   p->data = buffer;
259   p->allocsize = p->size = size;
260 }
```

Here is the caller graph for this function:



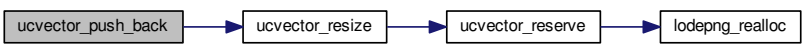
4.1.3.189 ucvector_push_back()

```
static unsigned ucvector_push_back (
            ucvector * p,
            unsigned char c )  [static]
```

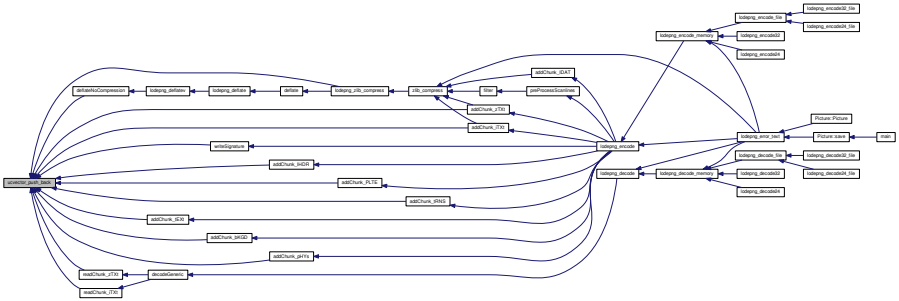
Definition at line 265 of file lodepng.cpp.

```
266 {
267   if(!ucvector_resize(p, p->size + 1)) return 0;
268   p->data[p->size - 1] = c;
269   return 1;
270 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.190 ucvector_reserve()

```
static unsigned ucvector_reserve (
            ucvector * p,
            size_t allocsize )  [static]
```

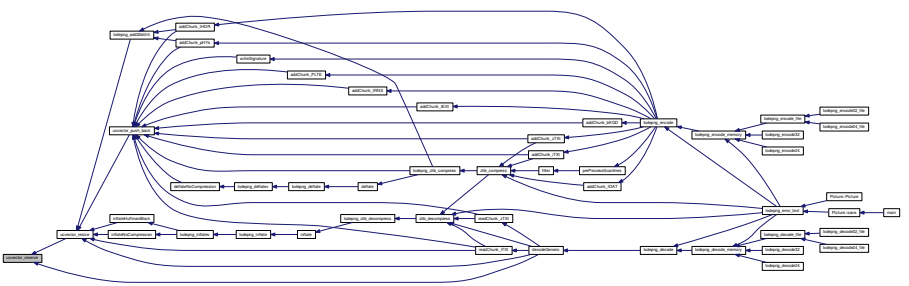
Definition at line 213 of file lodepng.cpp.

```
214 {
215   if(allocsize > p->allocsize)
216   {
217     size_t newsize = (allocsize > p->allocsize * 2) ? allocsize : (allocsize * 3
218     void* data = lodepng_realloc(p->data, newsize);
219     if(data)
220     {
221       p->allocsize = newsize;
```

```
222         p->data = (unsigned char*)data;
223       }
224       else return 0; /*error: not enough memory*/
225     }
226     return 1;
227 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



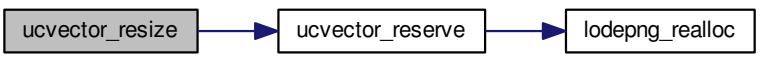
4.1.3.191 ucvector_resize()

```
static unsigned ucvector_resize (
            ucvector * p,
            size_t size )  [static]
```

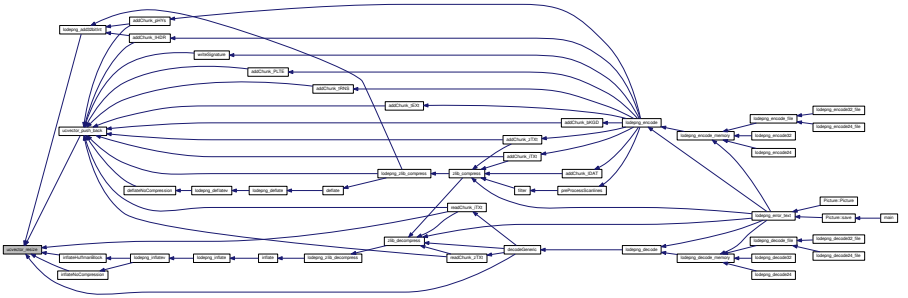
Definition at line 230 of file lodepng.cpp.

```
231 {
232   if(!ucvector_reserve(p, size * sizeof(unsigned char))) return 0;
233   p->size = size;
234   return 1; /*success*/
235 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.192 uivector_cleanup()

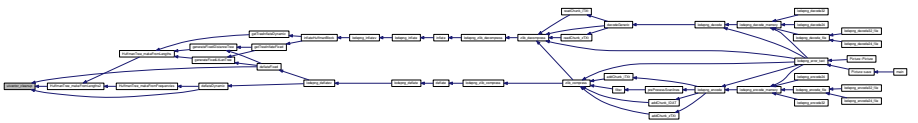
```
static void uivector_cleanup (
            void * p )  [static]
```

Definition at line 144 of file lodepng.cpp.

```
145 {
146   ((uivector*)p)->size = ((uivector*)p)->allocsize = 0;
147   lodepng_free(((uivector*)p)->data);
148   ((uivector*)p)->data = NULL;
149 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



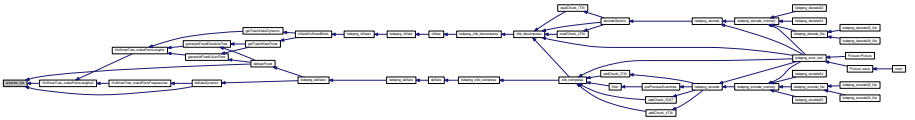
4.1.3.193 uivector_init()

```
static void uivector_init (
            uivector * p )  [static]
```

Definition at line 185 of file lodepng.cpp.

```
186 {
187   p->data = NULL;
188   p->size = p->allocsize = 0;
189 }
```

Here is the caller graph for this function:



4.1.3.194 uivector_push_back()

```
static unsigned uivector_push_back (
            uivector * p,
            unsigned c )  [static]
```

Definition at line 193 of file lodepng.cpp.

```
194 {
195   if(!uivector_resize(p, p->size + 1)) return 0;
196   p->data[p->size - 1] = c;
197   return 1;
198 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.1.3.195 uivector_reserve()

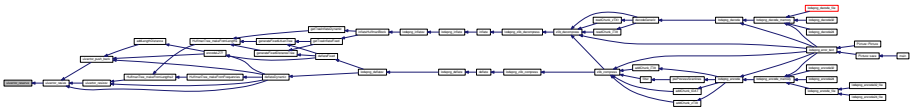
```
static unsigned uivector_reserve (
            uivector * p,
            size_t allocsize )  [static]
```

Definition at line 152 of file lodepng.cpp.

```
153 {
154   if(allocsize > p->allocsize)
155   {
156     size_t newsize = (allocsize > p->allocsize * 2) ? allocsize : (allocsize * 3
157     void* data = lodepng_realloc(p->data, newsize);
158     if(data)
159     {
160       p->allocsize = newsize;
161       p->data = (unsigned*)data;
162     }
163     else return 0; /*error: not enough memory*/
164   }
165   return 1;
166 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



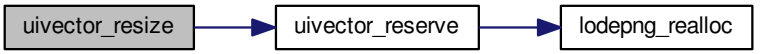
4.1.3.196 uivector_resize()

```
static unsigned uivector_resize (
            uivector * p,
            size_t size )  [static]
```

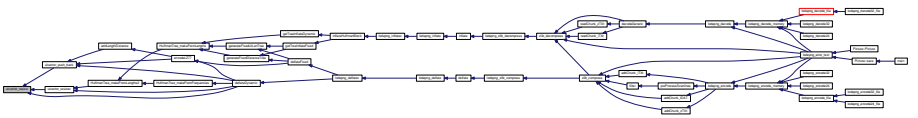
Definition at line 169 of file lodepng.cpp.

```
170 {
171   if(!uivector_reserve(p, size * sizeof(unsigned))) return 0;
172   p->size = size;
173   return 1; /*success*/
174 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.197 uivector_resizev()

```
static unsigned uivector_resizev (
            uivector * p,
            size_t size,
            unsigned value )  [static]
```

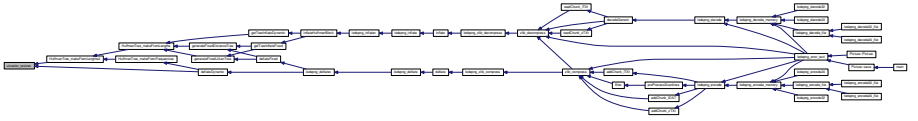
Definition at line 177 of file lodepng.cpp.

```
178 {
179   size_t oldsize = p->size, i;
```

```
180   if(!uivector_resize(p, size)) return 0;
181   for(i = oldsize; i < size; ++i) p->data[i] = value;
182   return 1;
183 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.1.3.198 unfilter()

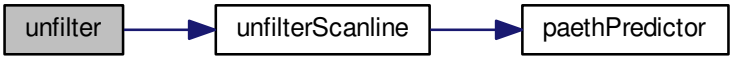
```
static unsigned unfilter (
            unsigned char * out,
            const unsigned char * in,
            unsigned w,
            unsigned h,
            unsigned bpp )  [static]
```

Definition at line 4043 of file lodepng.cpp.

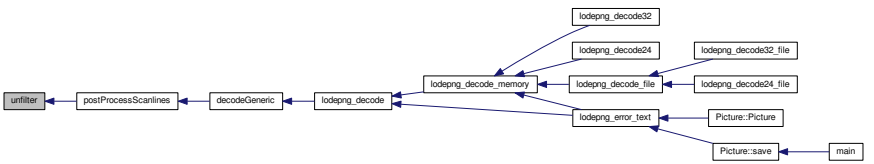
```
4044 {
4045   /*
4046   For PNG filter method 0
4047   this function unfilters a single image (e.g. without interlacing this is call
       times)
4048   out must have enough bytes allocated already, in must have the scanlines + 1
4049   w and h are image dimensions or dimensions of reduced image, bpp is bits per
4050   in and out are allowed to be the same memory address (but aren't the same siz
       filter bytes)
4051   */
4052
4053   unsigned y;
4054   unsigned char* prevline = 0;
4055
4056   /*bytewidth is used for filtering, is 1 when bpp < 8, number of bytes per pix
4057   size_t bytewidth = (bpp + 7) / 8;
4058   size_t linebytes = (w * bpp + 7) / 8;
4059
4060   for(y = 0; y < h; ++y)
4061   {
4062     size_t outindex = linebytes * y;
4063     size_t inindex = (1 + linebytes) * y; /*the extra filterbyte added to each
4064     unsigned char filterType = in[inindex];
4065
4066     CERROR_TRY_RETURN(unfilterScanline(&out[outindex], &in[inindex + 1],
     prevline, bytewidth, filterType, linebytes));
4067
4068     prevline = &out[outindex];
4069   }
4070
4071   return 0;
```

```
4072 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.199 unfilterScanline()

```
static unsigned unfilterScanline (
            unsigned char * recon,
            const unsigned char * scanline,
            const unsigned char * precon,
```

```
        size_t bytewidth,
        unsigned char filterType,
        size_t length )  [static]
```

Definition at line 3969 of file lodepng.cpp.

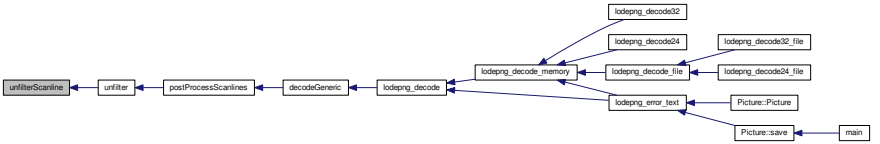
```
3971 {
3972   /*
3973   For PNG filter method 0
3974   unfilter a PNG image scanline by scanline. when the pixels are smaller than 1
3975   the filter works byte per byte (bytewidth = 1)
3976   precon is the previous unfiltered scanline, recon the result, scanline the cu
3977   the incoming scanlines do NOT include the filtertype byte, that one is given
       instead
3978   recon and scanline MAY be the same memory address! precon must be disjoint.
3979   */
3980
3981   size_t i;
3982   switch(filterType)
3983   {
3984     case 0:
3985       for(i = 0; i != length; ++i) recon[i] = scanline[i];
3986       break;
3987     case 1:
3988       for(i = 0; i != bytewidth; ++i) recon[i] = scanline[i];
3989       for(i = bytewidth; i < length; ++i) recon[i] = scanline[i] + recon[i - by
3990       break;
3991     case 2:
3992       if(precon)
3993       {
3994         for(i = 0; i != length; ++i) recon[i] = scanline[i] + precon[i];
3995       }
```

```
3996       else
3997       {
3998         for(i = 0; i != length; ++i) recon[i] = scanline[i];
3999       }
4000       break;
4001     case 3:
4002       if(precon)
4003       {
4004         for(i = 0; i != bytewidth; ++i) recon[i] = scanline[i] + (precon[i] >>
4005         for(i = bytewidth; i < length; ++i) recon[i] = scanline[i] + ((recon[i
      1);
4006       }
4007       else
4008       {
4009         for(i = 0; i != bytewidth; ++i) recon[i] = scanline[i];
4010         for(i = bytewidth; i < length; ++i) recon[i] = scanline[i] + (recon[i -
4011       }
4012       break;
4013     case 4:
4014       if(precon)
4015       {
4016         for(i = 0; i != bytewidth; ++i)
4017         {
4018           recon[i] = (scanline[i] + precon[i]); /*paethPredictor(0, precon[i],
4019         }
4020         for(i = bytewidth; i < length; ++i)
4021         {
4022           recon[i] = (scanline[i] + paethPredictor(recon[i - bytewidth], precon
      - bytewidth]));
4023         }
4024       }
```

```
4025       else
4026       {
4027         for(i = 0; i != bytewidth; ++i)
4028         {
4029           recon[i] = scanline[i];
4030         }
4031         for(i = bytewidth; i < length; ++i)
4032         {
4033           /*paethPredictor(recon[i - bytewidth], 0, 0) is always recon[i - byte
4034           recon[i] = (scanline[i] + recon[i - bytewidth]);
4035         }
4036       }
4037       break;
4038     default: return 36; /*error: unexisting filter type given*/
4039   }
4040   return 0;
4041 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.1.3.200 update_adler32()

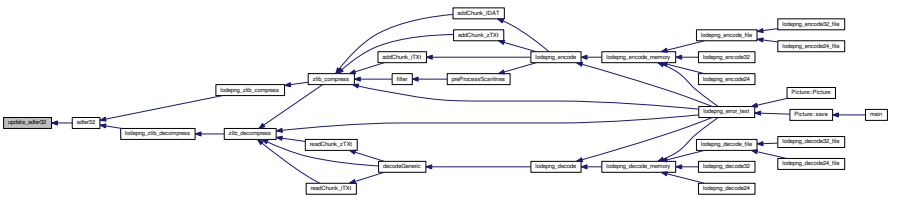
```
static unsigned update_adler32 (
            unsigned adler,
            const unsigned char * data,
            unsigned len )  [static]
```

Definition at line 2091 of file lodepng.cpp.

```
2092 {
2093    unsigned s1 = adler & 0xffff;
2094    unsigned s2 = (adler >> 16) & 0xffff;
2095
2096   while(len > 0)
2097   {
2098     /*at least 5550 sums can be done before the sums overflow, saving a lot of
2099     unsigned amount = len > 5550 ? 5550 : len;
2100     len -= amount;
2101     while(amount > 0)
2102     {
```

```
2103         s1 += (*data++);
2104         s2 += s1;
2105         --amount;
2106       }
2107       s1 %= 65521;
2108       s2 %= 65521;
2109     }
2110
2111     return (s2 << 16) | s1;
2112 }
```

Here is the caller graph for this function:



4.1.3.201 updateHashChain()

```
static void updateHashChain (
            Hash * hash,
            size_t wpos,
            unsigned hashval,
            unsigned short numzeros )  [static]
```

Definition at line 1454 of file lodepng.cpp.

```
1455 {
1456   hash->val[wpos] = (int)hashval;
1457   if(hash->head[hashval] != -1) hash->chain[wpos] = hash->head[hashval];
1458   hash->head[hashval] = wpos;
1459
1460   hash->zeros[wpos] = numzeros;
1461   if(hash->headz[numzeros] != -1) hash->chainz[wpos] = hash->headz[numzeros];
1462   hash->headz[numzeros] = wpos;
1463 }
```

Here is the caller graph for this function:

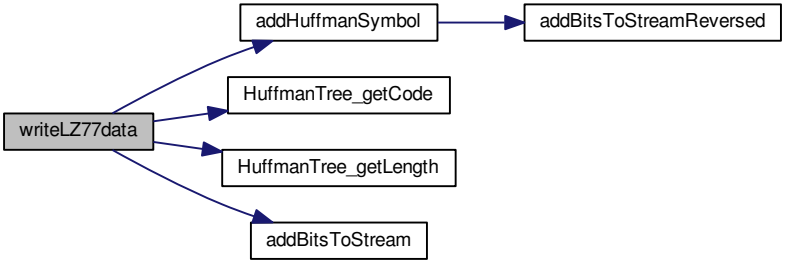
4.1.3.202 writeLZ77data()

```
static void writeLZ77data (
            size_t * bp,
            ucvector * out,
            const uivector * lz77_encoded,
            const HuffmanTree * tree_ll,
            const HuffmanTree * tree_d )  [static]
```

Definition at line 1695 of file lodepng.cpp.

```
1697 {
1698   size_t i = 0;
1699   for(i = 0; i != lz77_encoded->size; ++i)
1700   {
1701     unsigned val = lz77_encoded->data[i];
1702     addHuffmanSymbol(bp, out, HuffmanTree_getCode(tree_ll, val),
     HuffmanTree_getLength(tree_ll, val));
1703     if(val > 256) /*for a length code, 3 more things have to be added*/
1704     {
1705       unsigned length_index = val - FIRST_LENGTH_CODE_INDEX;
1706       unsigned n_length_extra_bits = LENGTHEXTRA[length_index];
1707       unsigned length_extra_bits = lz77_encoded->data[++i];
1708
1709       unsigned distance_code = lz77_encoded->data[++i];
1710
1711       unsigned distance_index = distance_code;
1712       unsigned n_distance_extra_bits = DISTANCEEXTRA[distance_index];
1713       unsigned distance_extra_bits = lz77_encoded->data[++i];
1714
1715       addBitsToStream(bp, out, length_extra_bits, n_length_extra_bits);
1716       addHuffmanSymbol(bp, out, HuffmanTree_getCode(tree_d,
     distance_code),
1717                        HuffmanTree_getLength(tree_d, distance_code));
1718       addBitsToStream(bp, out, distance_extra_bits, n_distance_extra_bits);
1719     }
1720   }
1721 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

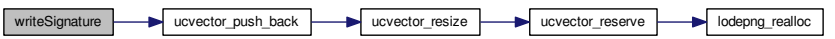
4.1.3.203 writeSignature()

```
static void writeSignature (
            ucvector * out )  [static]
```

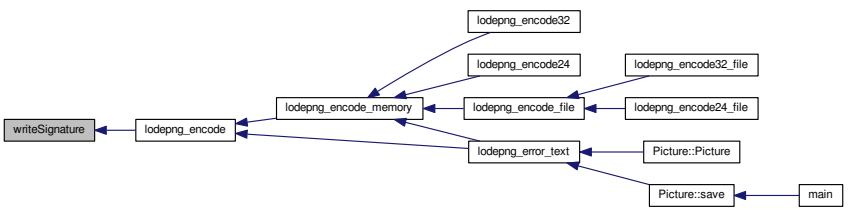
Definition at line 4875 of file lodepng.cpp.

```
4876 {
4877   /*8 bytes PNG signature, aka the magic bytes*/
4878   ucvector_push_back(out, 137);
4879   ucvector_push_back(out, 80);
4880   ucvector_push_back(out, 78);
4881   ucvector_push_back(out, 71);
4882   ucvector_push_back(out, 13);
4883   ucvector_push_back(out, 10);
4884   ucvector_push_back(out, 26);
4885   ucvector_push_back(out, 10);
4886 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



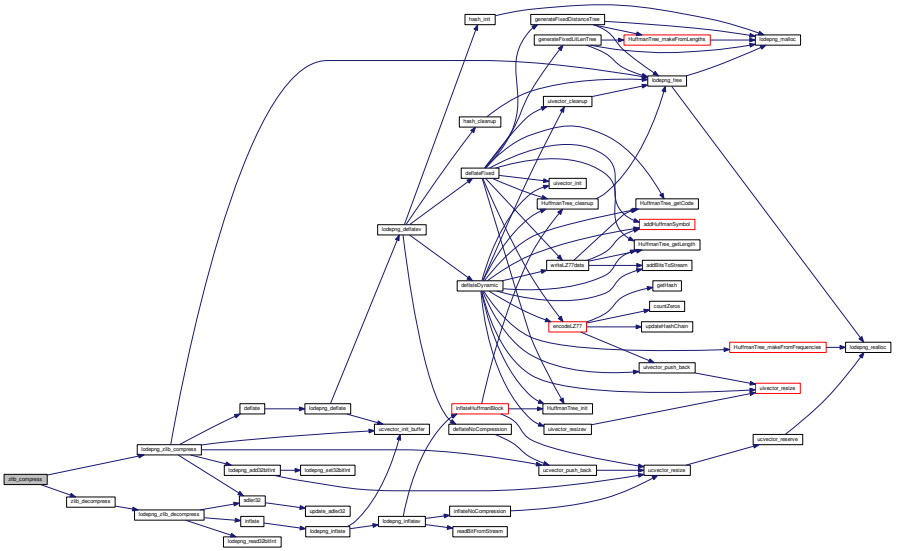
4.1.3.204 zlib_compress()

```
static unsigned zlib_compress (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGCompressSettings * settings )  [static]
```

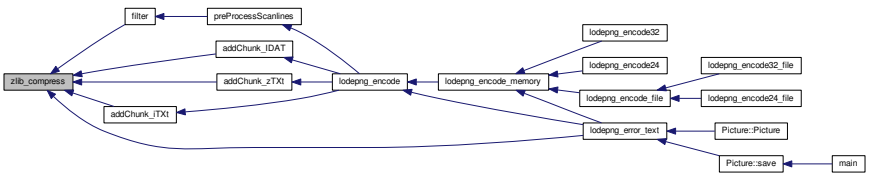
Definition at line 2230 of file lodepng.cpp.

```
2232 {
2233   if(settings->custom_zlib)
2234   {
2235     return settings->custom_zlib(out, outsize, in, insize, settings);
2236   }
2237   else
2238   {
2239     return lodepng_zlib_compress(out, outsize, in, insize, settings);
2240   }
2241 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



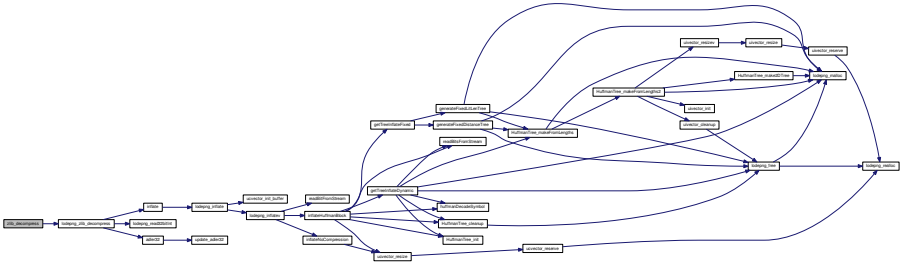
4.1.3.205 zlib_decompress()

```
static unsigned zlib_decompress (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGDecompressSettings * settings )  [static]
```

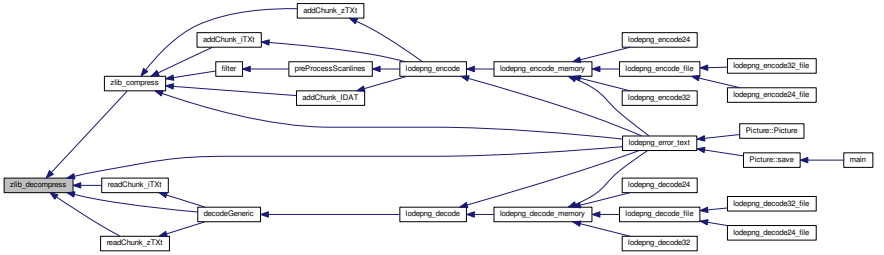
Definition at line 2171 of file lodepng.cpp.

```
2173 {
2174   if(settings->custom_zlib)
2175   {
2176     return settings->custom_zlib(out, outsize, in, insize, settings);
2177   }
2178   else
2179   {
2180     return lodepng_zlib_decompress(out, outsize, in, insize, settings);
2181   }
2182 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.4 Variable Documentation

4.1.4.1 ADAM7_DX

```
const unsigned ADAM7_DX[7] = { 8, 8, 4, 4, 2, 2, 1 }  [static]
```

Definition at line 3850 of file lodepng.cpp.

4.1.4.2 ADAM7_DY

```
const unsigned ADAM7_DY[7] = { 8, 8, 8, 4, 4, 2, 2 }  [static]
```

Definition at line 3851 of file lodepng.cpp.

4.1.4.3 ADAM7_IX

```
const unsigned ADAM7_IX[7] = { 0, 4, 0, 2, 0, 1, 0 }  [static]
```

Definition at line 3848 of file lodepng.cpp.

4.1.4.4 ADAM7_IY

```
const unsigned ADAM7_IY[7] = { 0, 0, 4, 0, 2, 0, 1 }  [static]
```

Definition at line 3849 of file lodepng.cpp.

4.1.4.5 CLCL_ORDER

```
const unsigned CLCL_ORDER[NUM_CODE_LENGTH_CODES] = {16, 17, 18, 0, 8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 1, 15}
[static]
```

Definition at line 502 of file lodepng.cpp.

4.1.4.6 DISTANCEBASE

```
const unsigned DISTANCEBASE[30]  [static]
```

Initial value:

```
= {1, 2, 3, 4, 5, 7, 9, 13, 17, 25, 33, 49, 65, 97, 129, 193, 257, 385, 513,
     769, 1025, 1537, 2049, 3073, 4097, 6145, 8193, 12289, 16385, 24577}
```

Definition at line 491 of file lodepng.cpp.

4.1.4.7 DISTANCEEXTRA

```
const unsigned DISTANCEEXTRA[30]  [static]
```

Initial value:

```
= {0, 0, 0, 0, 1, 1, 2,  2,  3,  3,  4,  4,  5,  5,   6,   6,   7,   7,   8,
       8,    9,    9,   10,   10,   11,   11,   12,    12,    13,    13}
```

Definition at line 496 of file lodepng.cpp.

4.1.4.8 HASH_BIT_MASK

```
const unsigned HASH_BIT_MASK = 65535  [static]
```

Definition at line 1365 of file lodepng.cpp.

4.1.4.9 HASH_NUM_VALUES

```
const unsigned HASH_NUM_VALUES = 65536  [static]
```

Definition at line 1364 of file lodepng.cpp.

4.1.4.10 LENGTHBASE

```
const unsigned LENGTHBASE[29]  [static]
```

Initial value:

```
= {3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15, 17, 19, 23, 27, 31, 35, 43, 51, 59,
     67, 83, 99, 115, 131, 163, 195, 227, 258}
```

Definition at line 481 of file lodepng.cpp.

4.1.4.11 LENGTHEXTRA

```
const unsigned LENGTHEXTRA[29]  [static]
```

Initial value:

```
= {0, 0, 0, 0, 0, 0, 0,  0,  1,  1,  1,  1,  2,  2,  2,  2,  3,  3,  3,  3,
      4,  4,  4,   4,   5,   5,   5,   5,   0}
```

Definition at line 486 of file lodepng.cpp.

4.1.4.12 lodepng_crc32_table

```
unsigned lodepng_crc32_table[256]  [static]
```

Definition at line 2323 of file lodepng.cpp.

4.1.4.13 lodepng_default_compress_settings

```
const LodePNGCompressSettings lodepng_default_compress_settings = {2, 1, DEFAULT_WINDOWSIZE, 3, 128, 1, 0, 0, 0}
```

Definition at line 2288 of file lodepng.cpp.

4.1.4.14 lodepng_default_decompress_settings

```
const LodePNGDecompressSettings lodepng_default_decompress_settings = {0, 0, 0, 0}
```

Definition at line 2304 of file lodepng.cpp.

4.1.4.15 LODEPNG_VERSION_STRING

```
const char* LODEPNG_VERSION_STRING = "20161127"
```

Definition at line 42 of file lodepng.cpp.

4.1.4.16 MAX_SUPPORTED_DEFLATE_LENGTH

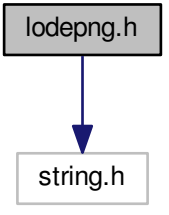
```
const size_t MAX_SUPPORTED_DEFLATE_LENGTH = 258  [static]
```

Definition at line 1318 of file lodepng.cpp.

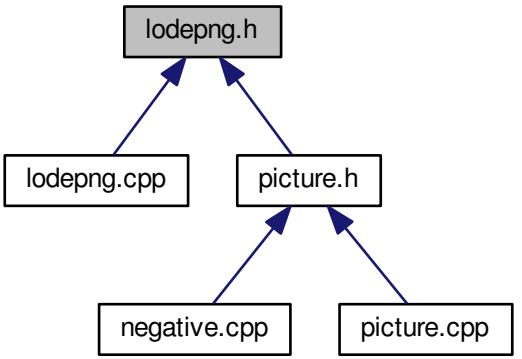
4.2 lodepng.h File Reference

```
#include <string.h>
```

Include dependency graph for lodepng.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct LodePNGDecompressSettings
- struct LodePNGCompressSettings
- struct LodePNGColorMode
- struct LodePNGTime
- struct LodePNGInfo
- struct LodePNGDecoderSettings
- struct LodePNGColorProfile
- struct LodePNGEncoderSettings
- struct LodePNGState

Macros

- #define LODEPNG_COMPILE_ZLIB
- #define LODEPNG_COMPILE_PNG
- #define LODEPNG_COMPILE_DECODER
- #define LODEPNG_COMPILE_ENCODER
- #define LODEPNG_COMPILE_DISK
- #define LODEPNG_COMPILE_ANCILLARY_CHUNKS
- #define LODEPNG_COMPILE_ERROR_TEXT
- #define LODEPNG_COMPILE_ALLOCATORS

Typedefs

- typedef enum LodePNGColorType LodePNGColorType
- typedef struct LodePNGDecompressSettings LodePNGDecompressSettings
- typedef struct LodePNGCompressSettings LodePNGCompressSettings
- typedef struct LodePNGColorMode LodePNGColorMode
- typedef struct LodePNGTime LodePNGTime
- typedef struct LodePNGInfo LodePNGInfo
- typedef struct LodePNGDecoderSettings LodePNGDecoderSettings
- typedef enum LodePNGFilterStrategy LodePNGFilterStrategy
- typedef struct LodePNGColorProfile LodePNGColorProfile
- typedef struct LodePNGEncoderSettings LodePNGEncoderSettings
- typedef struct LodePNGState LodePNGState

Enumerations

- enum LodePNGColorType {
  LCT_GREY = 0, LCT_RGB = 2, LCT_PALETTE = 3, LCT_GREY_ALPHA = 4,
  LCT_RGBA = 6 }
- enum LodePNGFilterStrategy {
  LFS_ZERO, LFS_MINSUM, LFS_ENTROPY, LFS_BRUTE_FORCE,
  LFS_PREDEFINED }

Functions

- unsigned lodepng_decode_memory (unsigned char **out, unsigned *w, unsigned *h, const unsigned char *in, size_t insize, LodePNGColorType colortype, unsigned bitdepth)
- unsigned lodepng_decode32 (unsigned char **out, unsigned *w, unsigned *h, const unsigned char *in, size_t insize)
- unsigned lodepng_decode24 (unsigned char **out, unsigned *w, unsigned *h, const unsigned char *in, size_t insize)
- unsigned lodepng_decode_file (unsigned char **out, unsigned *w, unsigned *h, const char *filename, LodePNGColorType colortype, unsigned bitdepth)
- unsigned lodepng_decode32_file (unsigned char **out, unsigned *w, unsigned *h, const char *filename)
- unsigned lodepng_decode24_file (unsigned char **out, unsigned *w, unsigned *h, const char *filename)
- unsigned lodepng_encode_memory (unsigned char **out, size_t *outsize, const unsigned char *image, unsigned w, unsigned h, LodePNGColorType colortype, unsigned bitdepth)
- unsigned lodepng_encode32 (unsigned char **out, size_t *outsize, const unsigned char *image, unsigned w, unsigned h)
- unsigned lodepng_encode24 (unsigned char **out, size_t *outsize, const unsigned char *image, unsigned w, unsigned h)
- unsigned lodepng_encode_file (const char *filename, const unsigned char *image, unsigned w, unsigned h, LodePNGColorType colortype, unsigned bitdepth)
- unsigned lodepng_encode32_file (const char *filename, const unsigned char *image, unsigned w, unsigned h)
- unsigned lodepng_encode24_file (const char *filename, const unsigned char *image, unsigned w, unsigned h)
- const char * lodepng_error_text (unsigned code)
- void lodepng_decompress_settings_init (LodePNGDecompressSettings *settings)
- void lodepng_compress_settings_init (LodePNGCompressSettings *settings)
- void lodepng_color_mode_init (LodePNGColorMode *info)
- void lodepng_color_mode_cleanup (LodePNGColorMode *info)
- unsigned lodepng_color_mode_copy (LodePNGColorMode *dest, const LodePNGColorMode *source)
- void lodepng_palette_clear (LodePNGColorMode *info)
- unsigned lodepng_palette_add (LodePNGColorMode *info, unsigned char r, unsigned char g, unsigned char b, unsigned char a)
- unsigned lodepng_get_bpp (const LodePNGColorMode *info)
- unsigned lodepng_get_channels (const LodePNGColorMode *info)
- unsigned lodepng_is_greyscale_type (const LodePNGColorMode *info)
- unsigned lodepng_is_alpha_type (const LodePNGColorMode *info)
- unsigned lodepng_is_palette_type (const LodePNGColorMode *info)
- unsigned lodepng_has_palette_alpha (const LodePNGColorMode *info)
- unsigned lodepng_can_have_alpha (const LodePNGColorMode *info)
- size_t lodepng_get_raw_size (unsigned w, unsigned h, const LodePNGColorMode *color)

- void lodepng_info_init (LodePNGInfo *info)
- void lodepng_info_cleanup (LodePNGInfo *info)
- unsigned lodepng_info_copy (LodePNGInfo *dest, const LodePNGInfo *source)
- void lodepng_clear_text (LodePNGInfo *info)
- unsigned lodepng_add_text (LodePNGInfo *info, const char *key, const char *str)
- void lodepng_clear_itext (LodePNGInfo *info)
- unsigned lodepng_add_itext (LodePNGInfo *info, const char *key, const char *langtag, const char *transkey, const char *str)
- unsigned lodepng_convert (unsigned char *out, const unsigned char *in, const LodePNGColorMode *mode_out, const LodePNGColorMode *mode_in, unsigned w, unsigned h)
- void lodepng_decoder_settings_init (LodePNGDecoderSettings *settings)
- void lodepng_color_profile_init (LodePNGColorProfile *profile)
- unsigned lodepng_get_color_profile (LodePNGColorProfile *profile, const unsigned char *image, unsigned w, unsigned h, const LodePNGColorMode *mode_in)
- unsigned lodepng_auto_choose_color (LodePNGColorMode *mode_out, const unsigned char *image, unsigned w, unsigned h, const LodePNGColorMode *mode_in)
- void lodepng_encoder_settings_init (LodePNGEncoderSettings *settings)
- void lodepng_state_init (LodePNGState *state)
- void lodepng_state_cleanup (LodePNGState *state)
- void lodepng_state_copy (LodePNGState *dest, const LodePNGState *source)
- unsigned lodepng_decode (unsigned char **out, unsigned *w, unsigned *h, LodePNGState *state, const unsigned char *in, size_t insize)
- unsigned lodepng_inspect (unsigned *w, unsigned *h, LodePNGState *state, const unsigned char *in, size_t insize)
- unsigned lodepng_encode (unsigned char **out, size_t *outsize, const unsigned char *image, unsigned w, unsigned h, LodePNGState *state)
- unsigned lodepng_chunk_length (const unsigned char *chunk)
- void lodepng_chunk_type (char type[5], const unsigned char *chunk)
- unsigned char lodepng_chunk_type_equals (const unsigned char *chunk, const char *type)
- unsigned char lodepng_chunk_ancillary (const unsigned char *chunk)
- unsigned char lodepng_chunk_private (const unsigned char *chunk)
- unsigned char lodepng_chunk_safetocopy (const unsigned char *chunk)
- unsigned char * lodepng_chunk_data (unsigned char *chunk)
- const unsigned char * lodepng_chunk_data_const (const unsigned char *chunk)
- unsigned lodepng_chunk_check_crc (const unsigned char *chunk)
- void lodepng_chunk_generate_crc (unsigned char *chunk)

- unsigned char * lodepng_chunk_next (unsigned char *chunk)
- const unsigned char * lodepng_chunk_next_const (const unsigned char *chunk)
- unsigned lodepng_chunk_append (unsigned char **out, size_t *outlength, const unsigned char *chunk)
- unsigned lodepng_chunk_create (unsigned char **out, size_t *outlength, unsigned length, const char *type, const unsigned char *data)
- unsigned lodepng_crc32 (const unsigned char *buf, size_t len)
- unsigned lodepng_inflate (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGDecompressSettings *settings)
- unsigned lodepng_zlib_decompress (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGDecompressSettings *settings)
- unsigned lodepng_zlib_compress (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGCompressSettings *settings)
- unsigned lodepng_huffman_code_lengths (unsigned *lengths, const unsigned *frequencies, size_t numcodes, unsigned maxbitlen)
- unsigned lodepng_deflate (unsigned char **out, size_t *outsize, const unsigned char *in, size_t insize, const LodePNGCompressSettings *settings)
- unsigned lodepng_load_file (unsigned char **out, size_t *outsize, const char *filename)
- unsigned lodepng_save_file (const unsigned char *buffer, size_t buffersize, const char *filename)

Variables

- const char * LODEPNG_VERSION_STRING
- const LodePNGDecompressSettings lodepng_default_decompress_settings
- const LodePNGCompressSettings lodepng_default_compress_settings

4.2.1 Macro Definition Documentation

4.2.1.1 LODEPNG_COMPILE_ALLOCATORS

```
#define LODEPNG_COMPILE_ALLOCATORS
```

Definition at line 75 of file lodepng.h.

4.2.1.2 LODEPNG_COMPILE_ANCILLARY_CHUNKS

```
#define LODEPNG_COMPILE_ANCILLARY_CHUNKS
```

Definition at line 65 of file lodepng.h.

4.2.1.3 LODEPNG_COMPILE_DECODER

```
#define LODEPNG_COMPILE_DECODER
```

Definition at line 53 of file lodepng.h.

4.2.1.4 LODEPNG_COMPILE_DISK

```
#define LODEPNG_COMPILE_DISK
```

Definition at line 61 of file lodepng.h.

4.2.1.5 LODEPNG_COMPILE_ENCODER

```
#define LODEPNG_COMPILE_ENCODER
```

Definition at line 57 of file lodepng.h.

4.2.1.6 LODEPNG_COMPILE_ERROR_TEXT

```
#define LODEPNG_COMPILE_ERROR_TEXT
```

Definition at line 69 of file lodepng.h.

4.2.1.7 LODEPNG_COMPILE_PNG

```
#define LODEPNG_COMPILE_PNG
```

Definition at line 49 of file lodepng.h.

4.2.1.8 LODEPNG_COMPILE_ZLIB

```
#define LODEPNG_COMPILE_ZLIB
```

Definition at line 45 of file lodepng.h.

4.2.2 Typedef Documentation

4.2.2.1 LodePNGColorMode

```
typedef struct LodePNGColorMode LodePNGColorMode
```

4.2.2.2 LodePNGColorProfile

```
typedef struct LodePNGColorProfile LodePNGColorProfile
```

4.2.2.3 LodePNGColorType

```
typedef enum LodePNGColorType LodePNGColorType
```

4.2.2.4 LodePNGCompressSettings

```
typedef struct LodePNGCompressSettings LodePNGCompressSettings
```

Definition at line 283 of file lodepng.h.

4.2.2.5 LodePNGDecoderSettings

```
typedef struct LodePNGDecoderSettings LodePNGDecoderSettings
```

4.2.2.6 LodePNGDecompressSettings

```
typedef struct LodePNGDecompressSettings LodePNGDecompressSettings
```

Definition at line 255 of file lodepng.h.

4.2.2.7 LodePNGEncoderSettings

```
typedef struct LodePNGEncoderSettings LodePNGEncoderSettings
```

4.2.2.8 LodePNGFilterStrategy

```
typedef enum LodePNGFilterStrategy LodePNGFilterStrategy
```

4.2.2.9 LodePNGInfo

```
typedef struct LodePNGInfo LodePNGInfo
```

4.2.2.10 LodePNGState

```
typedef struct LodePNGState LodePNGState
```

4.2.2.11 LodePNGTime

```
typedef struct LodePNGTime LodePNGTime
```

4.2.3 Enumeration Type Documentation

4.2.3.1 LodePNGColorType

```
enum LodePNGColorType
```

Enumerator

LCT_GREY	
LCT_RGB	
LCT_PALETTE	
LCT_GREY_ALPHA	
LCT_RGBA	

Definition at line 91 of file lodepng.h.

```
92 {
93   LCT_GREY = 0, /*greyscale: 1,2,4,8,16 bit*/
94   LCT_RGB = 2, /*RGB: 8,16 bit*/
95   LCT_PALETTE = 3, /*palette: 1,2,4,8 bit*/
96   LCT_GREY_ALPHA = 4, /*greyscale with alpha: 8,16 bit*/
97   LCT_RGBA = 6 /*RGB with alpha: 8,16 bit*/
98 } LodePNGColorType;
```

4.2.3.2 LodePNGFilterStrategy

```
enum LodePNGFilterStrategy
```

Enumerator

LFS_ZERO	
LFS_MINSUM	
LFS_ENTROPY	
LFS_BRUTE_FORCE	
LFS_PREDEFINED	

Definition at line 539 of file lodepng.h.

```
540 {
541   /*every filter at zero*/
542   LFS_ZERO,
543   /*Use filter that gives minimum sum, as described in the official PNG filter h
544   LFS_MINSUM,
545   /*Use the filter type that gives smallest Shannon entropy for this scanline. D
546   on the image, this is better or worse than minsum.*/
547   LFS_ENTROPY,
548   /*
549   Brute-force-search PNG filters by compressing each filter for each scanline.
550   Experimental, very slow, and only rarely gives better compression than MINSUM.
551   */
552   LFS_BRUTE_FORCE,
553   /*use predefined_filters buffer: you specify the filter type for each scanline
554   LFS_PREDEFINED
555 } LodePNGFilterStrategy;
```

4.2.4 Function Documentation

4.2.4.1 lodepng_add_itext()

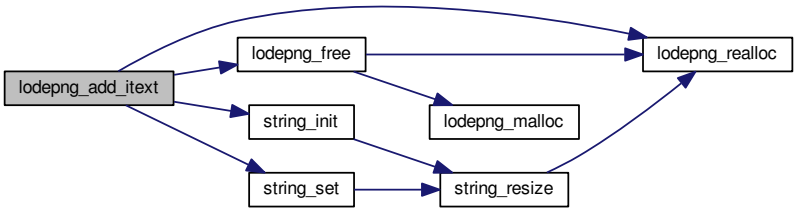
```
unsigned lodepng_add_itext (
            LodePNGInfo * info,
            const char * key,
            const char * langtag,
            const char * transkey,
            const char * str )
```

Definition at line 2885 of file lodepng.cpp.

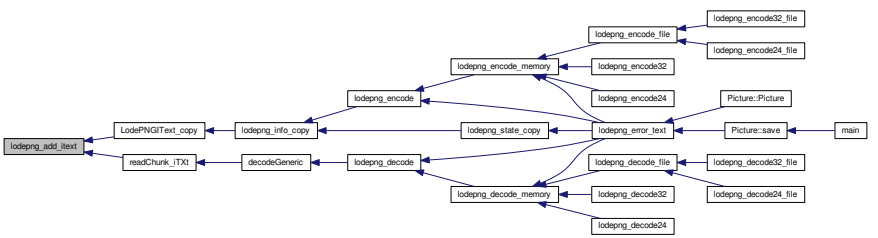
```
2887 {
2888   char** new_keys = (char**)(lodepng_realloc(info->itext_keys, sizeof(char*) *
     info->itext_num + 1)));
2889   char** new_langtags = (char**)(lodepng_realloc(info->
     itext_langtags, sizeof(char*) * (info->itext_num + 1)));
2890   char** new_transkeys = (char**)(lodepng_realloc(info->
     itext_transkeys, sizeof(char*) * (info->itext_num + 1)));
2891   char** new_strings = (char**)(lodepng_realloc(info->
     itext_strings, sizeof(char*) * (info->itext_num + 1)));
2892   if(!new_keys || !new_langtags || !new_transkeys || !new_strings)
2893   {
2894     lodepng_free(new_keys);
2895     lodepng_free(new_langtags);
2896     lodepng_free(new_transkeys);
2897     lodepng_free(new_strings);
2898     return 83; /*alloc fail*/
2899   }
2900
2901   ++info->itext_num;
2902   info->itext_keys = new_keys;
2903   info->itext_langtags = new_langtags;
2904   info->itext_transkeys = new_transkeys;
2905   info->itext_strings = new_strings;
2906
2907   string_init(&info->itext_keys[info->itext_num - 1]);
2908   string_set(&info->itext_keys[info->itext_num - 1], key);
2909
2910   string_init(&info->itext_langtags[info->itext_num - 1]);
2911   string_set(&info->itext_langtags[info->itext_num - 1], langtag);
2912
2913   string_init(&info->itext_transkeys[info->itext_num - 1]);
```

```
2914   string_set(&info->itext_transkeys[info->itext_num - 1], transkey);
2915
2916   string_init(&info->itext_strings[info->itext_num - 1]);
2917   string_set(&info->itext_strings[info->itext_num - 1], str);
2918
2919   return 0;
2920 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.2 lodepng_add_text()

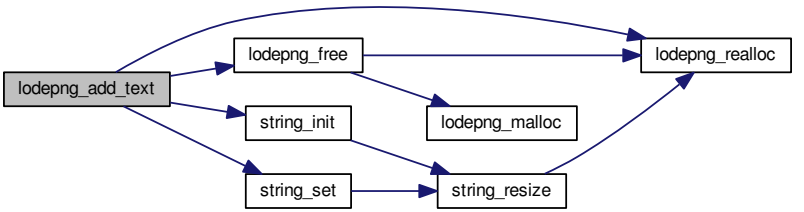
```
unsigned lodepng_add_text (
            LodePNGInfo * info,
            const char * key,
            const char * str )
```

Definition at line 2813 of file lodepng.cpp.

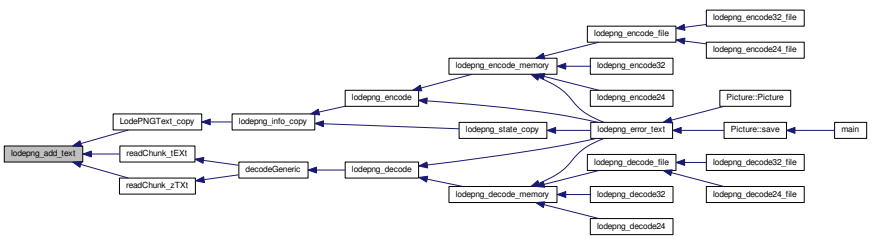
```
2814 {
2815   char** new_keys = (char**)(lodepng_realloc(info->text_keys, sizeof(char*) * (
      ->text_num + 1)));
2816   char** new_strings = (char**)(lodepng_realloc(info->
      text_strings, sizeof(char*) * (info->text_num + 1)));
2817   if(!new_keys || !new_strings)
2818   {
2819     lodepng_free(new_keys);
2820     lodepng_free(new_strings);
2821     return 83; /*alloc fail*/
2822   }
2823
2824   ++info->text_num;
2825   info->text_keys = new_keys;
2826   info->text_strings = new_strings;
2827
2828   string_init(&info->text_keys[info->text_num - 1]);
2829   string_set(&info->text_keys[info->text_num - 1], key);
2830
2831   string_init(&info->text_strings[info->text_num - 1]);
2832   string_set(&info->text_strings[info->text_num - 1], str);
```

```
2833
2834   return 0;
2835 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.3 lodepng_auto_choose_color()

```
unsigned lodepng_auto_choose_color (
            LodePNGColorMode * mode_out,
            const unsigned char * image,
            unsigned w,
            unsigned h,
            const LodePNGColorMode * mode_in )
```

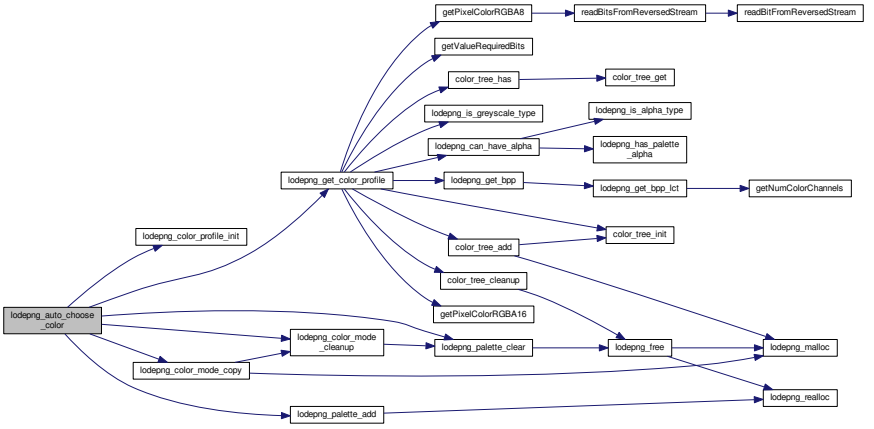
Definition at line 3763 of file lodepng.cpp.

```
3766 {
3767   LodePNGColorProfile prof;
3768   unsigned error = 0;
3769   unsigned i, n, palettebits, palette_ok;
3770
3771   lodepng_color_profile_init(&prof);
3772   error = lodepng_get_color_profile(&prof, image, w, h, mode_in);
3773   if(error) return error;
3774   mode_out->key_defined = 0;
3775
3776   if(prof.key && w * h <= 16)
3777   {
3778     prof.alpha = 1; /*too few pixels to justify tRNS chunk overhead*/
3779     prof.key = 0;
3780     if(prof.bits < 8) prof.bits = 8; /*PNG has no alphachannel modes with less
      channel*/
3781   }
3782   n = prof.numcolors;
3783   palettebits = n <= 2 ? 1 : (n <= 4 ? 2 : (n <= 16 ? 4 : 8));
3784   palette_ok = n <= 256 && prof.bits <= 8;
3785   if(w * h < n * 2) palette_ok = 0; /*don't add palette overhead if image has o
```

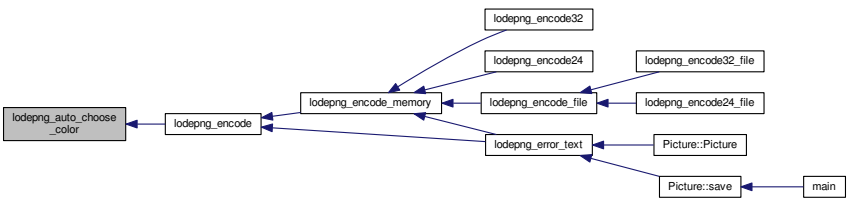
```
3786   if(!prof.colored && prof.bits <= palettebits) palette_ok = 0; /*grey is less
3787
3788   if(palette_ok)
3789   {
3790     unsigned char* p = prof.palette;
3791     lodepng_palette_clear(mode_out); /*remove potential earlier palette*/
3792     for(i = 0; i != prof.numcolors; ++i)
3793     {
3794       error = lodepng_palette_add(mode_out, p[i * 4 + 0], p[i * 4 + 1], p[i * 4
    i * 4 + 3]);
3795       if(error) break;
3796     }
3797
3798     mode_out->colortype = LCT_PALETTE;
3799     mode_out->bitdepth = palettebits;
3800
3801     if(mode_in->colortype == LCT_PALETTE && mode_in->
    palettesize >= mode_out->palettesize
3802         && mode_in->bitdepth == mode_out->bitdepth)
3803     {
3804       /*If input should have same palette colors, keep original to preserve its
     conversion*/
3805       lodepng_color_mode_cleanup(mode_out);
3806       lodepng_color_mode_copy(mode_out, mode_in);
3807     }
3808   }
3809   else /*8-bit or 16-bit per channel*/
3810   {
3811     mode_out->bitdepth = prof.bits;
3812     mode_out->colortype = prof.alpha ? (prof.colored ?
    LCT_RGBA : LCT_GREY_ALPHA)
```

```
3813                                        : (prof.colored ? LCT_RGB :
      LCT_GREY);
3814
3815     if(prof.key)
3816     {
3817       unsigned mask = (1u << mode_out->bitdepth) - 1u; /*profile always uses 16
       it*/
3818       mode_out->key_r = prof.key_r & mask;
3819       mode_out->key_g = prof.key_g & mask;
3820       mode_out->key_b = prof.key_b & mask;
3821       mode_out->key_defined = 1;
3822     }
3823   }
3824
3825   return error;
3826 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



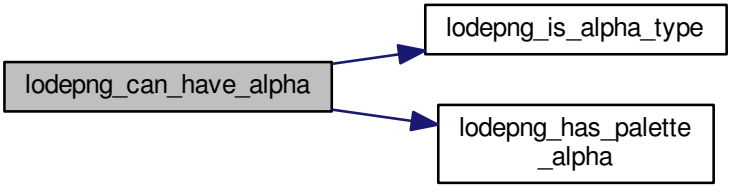
4.2.4.4 lodepng_can_have_alpha()

```
unsigned lodepng_can_have_alpha (
            const LodePNGColorMode * info )
```

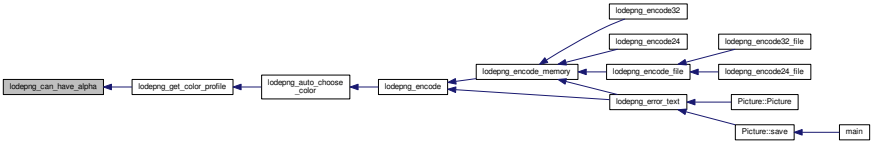
Definition at line 2701 of file lodepng.cpp.

```
2702 {
2703   return info->key_defined
2704       || lodepng_is_alpha_type(info)
2705       || lodepng_has_palette_alpha(info);
2706 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



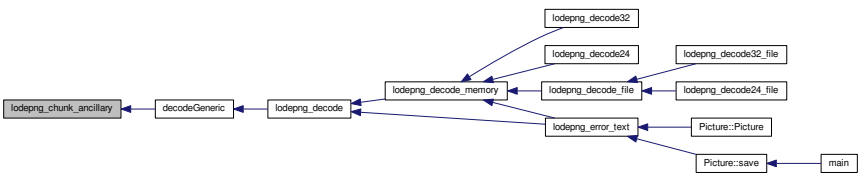
4.2.4.5 lodepng_chunk_ancillary()

```
unsigned char lodepng_chunk_ancillary (
            const unsigned char * chunk )
```

Definition at line 2439 of file lodepng.cpp.

```
2440 {
2441   return((chunk[4] & 32) != 0);
2442 }
```

Here is the caller graph for this function:



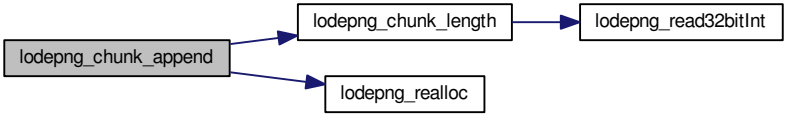
4.2.4.6 lodepng_chunk_append()

```
unsigned lodepng_chunk_append (
            unsigned char ** out,
            size_t * outlength,
            const unsigned char * chunk )
```

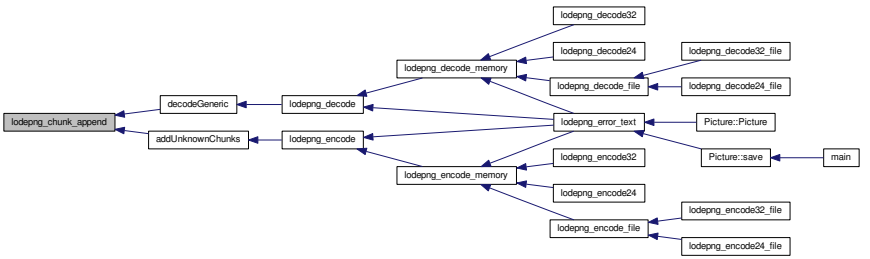
Definition at line 2493 of file lodepng.cpp.

```
2494 {
2495   unsigned i;
2496   unsigned total_chunk_length = lodepng_chunk_length(chunk) + 12;
2497   unsigned char *chunk_start, *new_buffer;
2498   size_t new_length = (*outlength) + total_chunk_length;
2499   if(new_length < total_chunk_length || new_length < (*outlength)) return 77; /
2500
2501   new_buffer = (unsigned char*)lodepng_realloc(*out, new_length);
2502   if(!new_buffer) return 83; /*alloc fail*/
2503   (*out) = new_buffer;
2504   (*outlength) = new_length;
2505   chunk_start = &(*out)[new_length - total_chunk_length];
2506
2507   for(i = 0; i != total_chunk_length; ++i) chunk_start[i] = chunk[i];
2508
2509   return 0;
2510 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



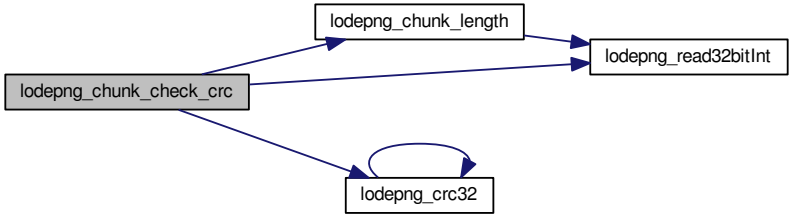
4.2.4.7 lodepng_chunk_check_crc()

```
unsigned lodepng_chunk_check_crc (
            const unsigned char * chunk )
```

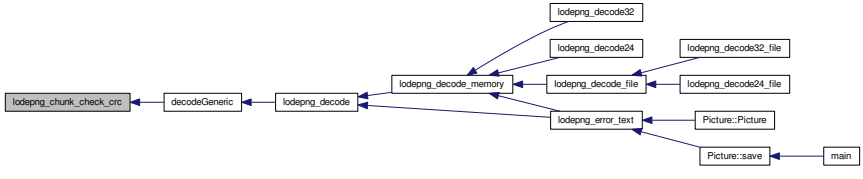
Definition at line 2464 of file lodepng.cpp.

```
2465 {
2466   unsigned length = lodepng_chunk_length(chunk);
2467   unsigned CRC = lodepng_read32bitInt(&chunk[length + 8]);
2468   /*the CRC is taken of the data and the 4 chunk type letters, not the length*/
2469   unsigned checksum = lodepng_crc32(&chunk[4], length + 4);
2470   if(CRC != checksum) return 1;
2471   else return 0;
2472 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.8 lodepng_chunk_create()

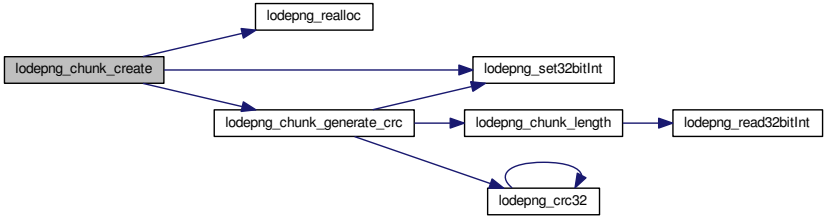
```
unsigned lodepng_chunk_create (
            unsigned char ** out,
            size_t * outlength,
            unsigned length,
            const char * type,
            const unsigned char * data )
```

Definition at line 2512 of file lodepng.cpp.

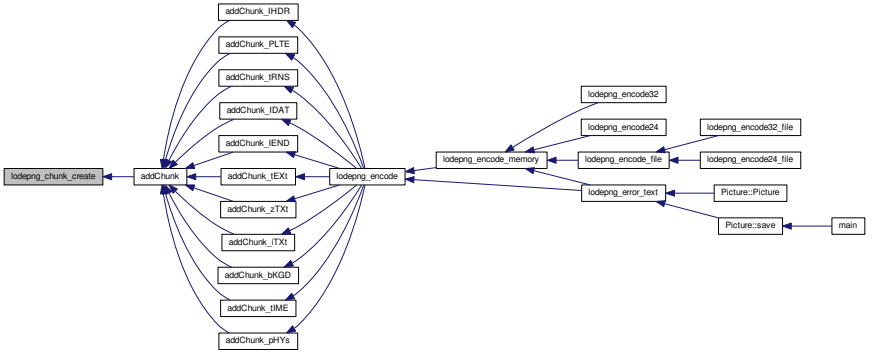
```
2514 {
2515   unsigned i;
2516   unsigned char *chunk, *new_buffer;
2517   size_t new_length = (*outlength) + length + 12;
2518   if(new_length < length + 12 || new_length < (*outlength)) return 77; /*intege
2519   new_buffer = (unsigned char*)lodepng_realloc(*out, new_length);
2520   if(!new_buffer) return 83; /*alloc fail*/
2521   (*out) = new_buffer;
2522   (*outlength) = new_length;
2523   chunk = &(*out)[(*outlength) - length - 12];
2524
2525   /*1: length*/
2526   lodepng_set32bitInt(chunk, (unsigned)length);
2527
2528   /*2: chunk name (4 letters)*/
2529   chunk[4] = (unsigned char)type[0];
2530   chunk[5] = (unsigned char)type[1];
2531   chunk[6] = (unsigned char)type[2];
2532   chunk[7] = (unsigned char)type[3];
```

```
2533
2534   /*3: the data*/
2535   for(i = 0; i != length; ++i) chunk[8 + i] = data[i];
2536
2537   /*4: CRC (of the chunkname characters and the data)*/
2538   lodepng_chunk_generate_crc(chunk);
2539
2540   return 0;
2541 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.9 lodepng_chunk_data()

```
unsigned char* lodepng_chunk_data (
            unsigned char * chunk )
```

Definition at line 2454 of file lodepng.cpp.

```
2455 {
2456   return &chunk[8];
2457 }
```

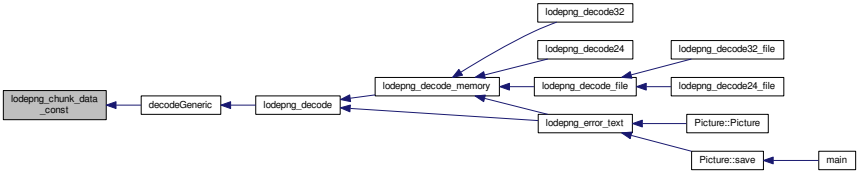
4.2.4.10 lodepng_chunk_data_const()

```
const unsigned char* lodepng_chunk_data_const (
            const unsigned char * chunk )
```

Definition at line 2459 of file lodepng.cpp.

```
2460 {
2461   return &chunk[8];
2462 }
```

Here is the caller graph for this function:



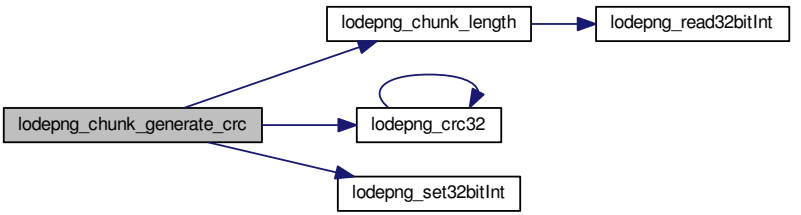
4.2.4.11 lodepng_chunk_generate_crc()

```
void lodepng_chunk_generate_crc (
            unsigned char * chunk )
```

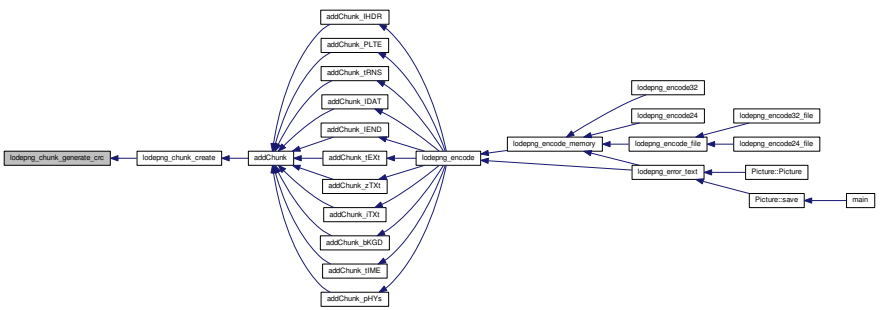
Definition at line 2474 of file lodepng.cpp.

```
2475 {
2476   unsigned length = lodepng_chunk_length(chunk);
2477   unsigned CRC = lodepng_crc32(&chunk[4], length + 4);
2478   lodepng_set32bitInt(chunk + 8 + length, CRC);
2479 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.12 lodepng_chunk_length()

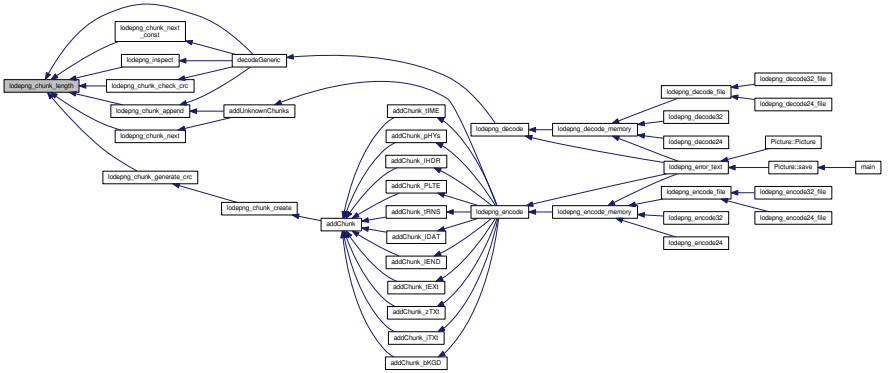
```
unsigned lodepng_chunk_length (
            const unsigned char * chunk )
```

Definition at line 2421 of file lodepng.cpp.

```
2422 {
2423   return lodepng_read32bitInt(&chunk[0]);
2424 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



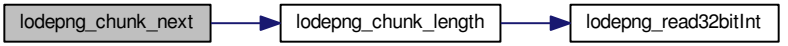
4.2.4.13 lodepng_chunk_next()

```
unsigned char* lodepng_chunk_next (
            unsigned char * chunk )
```

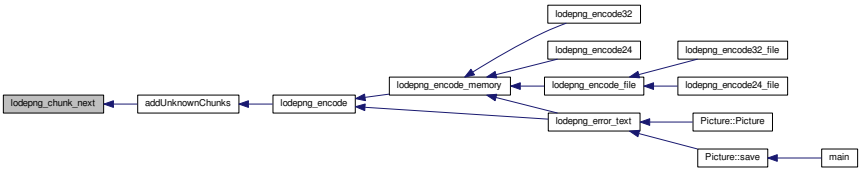
Definition at line 2481 of file lodepng.cpp.

```
2482 {
2483   unsigned total_chunk_length = lodepng_chunk_length(chunk) + 12;
2484   return &chunk[total_chunk_length];
2485 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



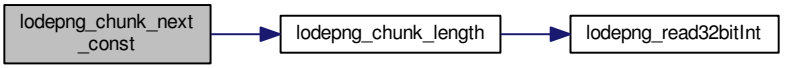
4.2.4.14 lodepng_chunk_next_const()

```
const unsigned char* lodepng_chunk_next_const (
            const unsigned char * chunk )
```

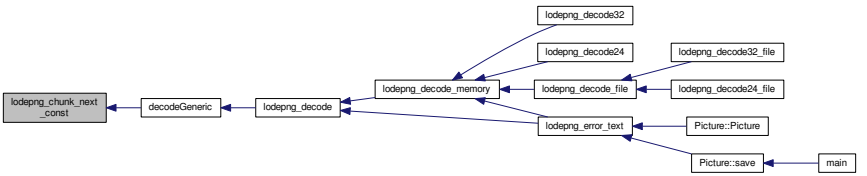
Definition at line 2487 of file lodepng.cpp.

```
2488 {
2489   unsigned total_chunk_length = lodepng_chunk_length(chunk) + 12;
2490   return &chunk[total_chunk_length];
2491 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.15 lodepng_chunk_private()

```
unsigned char lodepng_chunk_private (
            const unsigned char * chunk )
```

Definition at line 2444 of file lodepng.cpp.

```
2445 {
2446   return((chunk[6] & 32) != 0);
2447 }
```

4.2.4.16 lodepng_chunk_safetocopy()

```
unsigned char lodepng_chunk_safetocopy (
            const unsigned char * chunk )
```

Definition at line 2449 of file lodepng.cpp.

```
2450 {
2451   return((chunk[7] & 32) != 0);
2452 }
```

4.2.4.17 lodepng_chunk_type()

```
void lodepng_chunk_type (
            char type[5],
            const unsigned char * chunk )
```

Definition at line 2426 of file lodepng.cpp.

```
2427 {
2428   unsigned i;
2429   for(i = 0; i != 4; ++i) type[i] = (char)chunk[4 + i];
2430   type[4] = 0; /*null termination char*/
2431 }
```

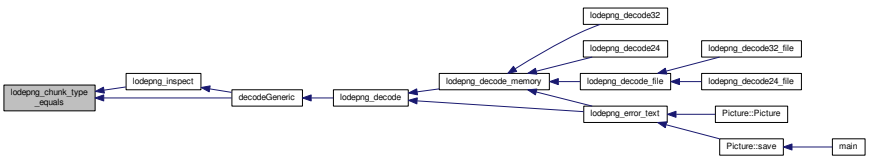
4.2.4.18 lodepng_chunk_type_equals()

```
unsigned char lodepng_chunk_type_equals (
            const unsigned char * chunk,
            const char * type )
```

Definition at line 2433 of file lodepng.cpp.

```
2434 {
2435   if(strlen(type) != 4) return 0;
2436   return (chunk[4] == type[0] && chunk[5] == type[1] && chunk[6] == type[2] &&
2437 }
```

Here is the caller graph for this function:



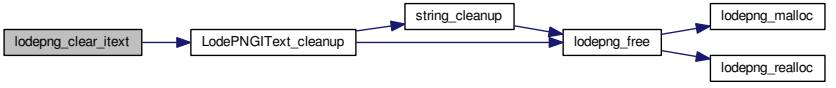
4.2.4.19 lodepng_clear_itext()

```
void lodepng_clear_itext (
            LodePNGInfo * info )
```

Definition at line 2880 of file lodepng.cpp.

```
2881 {
2882   LodePNGIText_cleanup(info);
2883 }
```

Here is the call graph for this function:



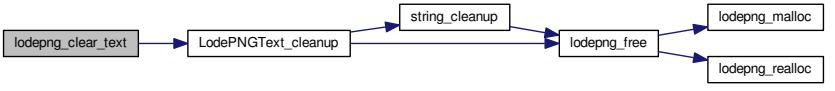
4.2.4.20 lodepng_clear_text()

```
void lodepng_clear_text (
            LodePNGInfo * info )
```

Definition at line 2808 of file lodepng.cpp.

```
2809 {
2810   LodePNGText_cleanup(info);
2811 }
```

Here is the call graph for this function:



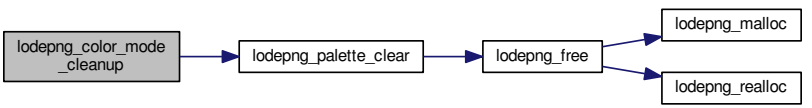
4.2.4.21 lodepng_color_mode_cleanup()

```
void lodepng_color_mode_cleanup (
            LodePNGColorMode * info )
```

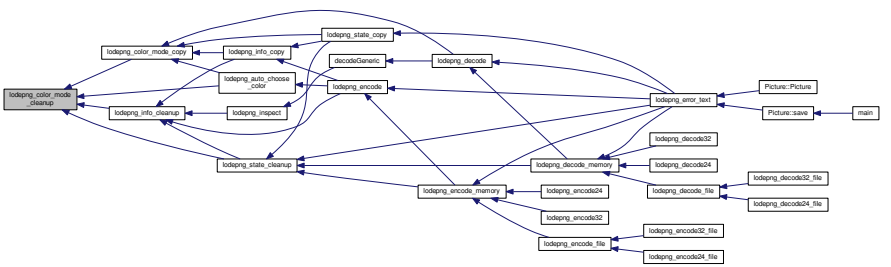
Definition at line 2593 of file lodepng.cpp.

```
2594 {
2595   lodepng_palette_clear(info);
2596 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



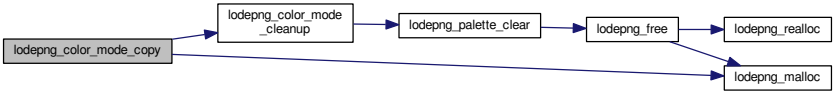
4.2.4.22 lodepng_color_mode_copy()

```
unsigned lodepng_color_mode_copy (
            LodePNGColorMode * dest,
            const LodePNGColorMode * source )
```

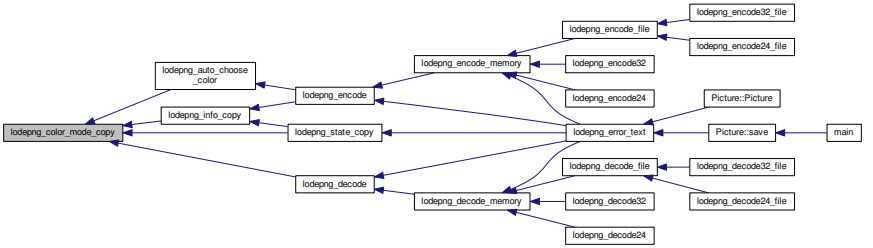
Definition at line 2598 of file lodepng.cpp.

```
2599 {
2600   size_t i;
2601   lodepng_color_mode_cleanup(dest);
2602   *dest = *source;
2603   if(source->palette)
2604   {
2605     dest->palette = (unsigned char*)lodepng_malloc(1024);
2606     if(!dest->palette && source->palettesize) return 83; /*alloc fail*/
2607     for(i = 0; i != source->palettesize * 4; ++i) dest->palette[i] = source->
     palette[i];
2608   }
2609   return 0;
2610 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



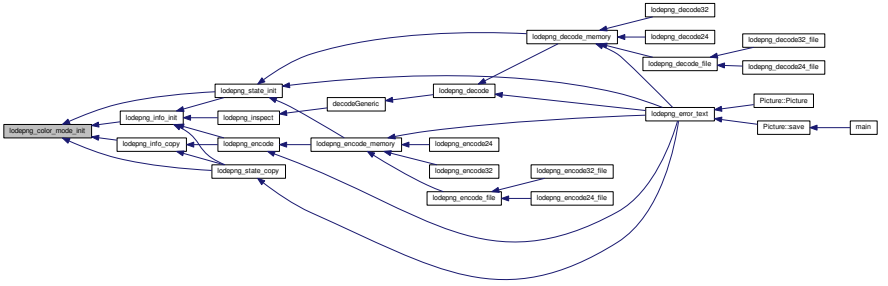
4.2.4.23 lodepng_color_mode_init()

```
void lodepng_color_mode_init (
            LodePNGColorMode * info )
```

Definition at line 2583 of file lodepng.cpp.

```
2584 {
2585   info->key_defined = 0;
2586   info->key_r = info->key_g = info->key_b = 0;
2587   info->colortype = LCT_RGBA;
2588   info->bitdepth = 8;
2589   info->palette = 0;
2590   info->palettesize = 0;
2591 }
```

Here is the caller graph for this function:



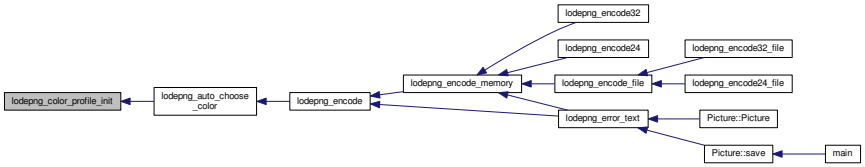
4.2.4.24 lodepng_color_profile_init()

```
void lodepng_color_profile_init (
            LodePNGColorProfile * profile )
```

Definition at line 3533 of file lodepng.cpp.

```
3534 {
3535   profile->colored = 0;
3536   profile->key = 0;
3537   profile->key_r = profile->key_g = profile->key_b = 0;
3538   profile->alpha = 0;
3539   profile->numcolors = 0;
3540   profile->bits = 1;
3541 }
```

Here is the caller graph for this function:



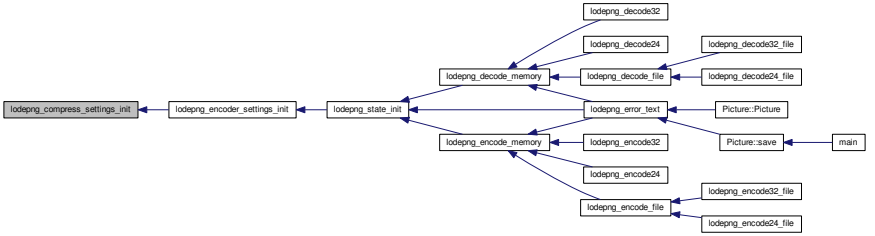
4.2.4.25 lodepng_compress_settings_init()

```
void lodepng_compress_settings_init (
            LodePNGCompressSettings * settings )
```

Definition at line 2273 of file lodepng.cpp.

```
2274 {
2275   /*compress with dynamic huffman tree (not in the mathematical sense, just not
2276   settings->btype = 2;
2277   settings->use_lz77 = 1;
2278   settings->windowsize = DEFAULT_WINDOWSIZE;
2279   settings->minmatch = 3;
2280   settings->nicematch = 128;
2281   settings->lazymatching = 1;
2282
2283   settings->custom_zlib = 0;
2284   settings->custom_deflate = 0;
2285   settings->custom_context = 0;
2286 }
```

Here is the caller graph for this function:



4.2.4.26 lodepng_convert()

```
unsigned lodepng_convert (
            unsigned char * out,
            const unsigned char * in,
            const LodePNGColorMode * mode_out,
            const LodePNGColorMode * mode_in,
            unsigned w,
            unsigned h )
```

Definition at line 3459 of file lodepng.cpp.

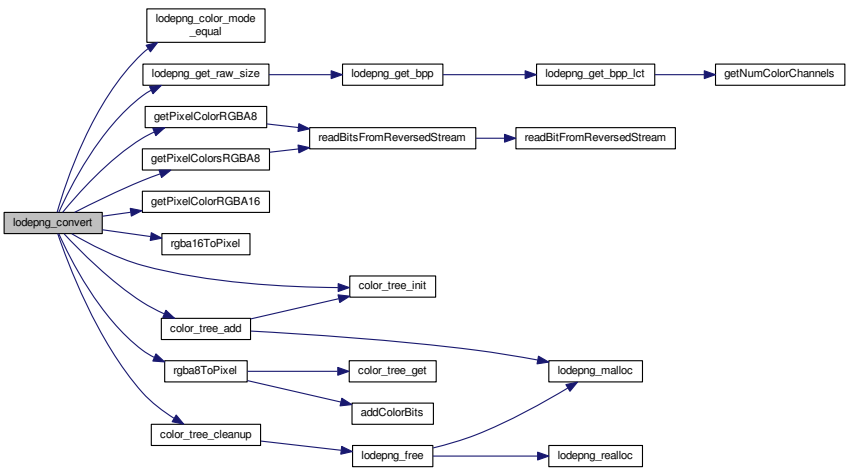
```
3462 {
3463   size_t i;
3464   ColorTree tree;
3465   size_t numpixels = w * h;
3466
3467   if(lodepng_color_mode_equal(mode_out, mode_in))
```

```
3468   {
3469     size_t numbytes = lodepng_get_raw_size(w, h, mode_in);
3470     for(i = 0; i != numbytes; ++i) out[i] = in[i];
3471     return 0;
3472   }
3473
3474   if(mode_out->colortype == LCT_PALETTE)
3475   {
3476     size_t palettesize = mode_out->palettesize;
3477     const unsigned char* palette = mode_out->palette;
3478     size_t palsize = 1u << mode_out->bitdepth;
3479     /*if the user specified output palette but did not give the values, assume
3480     they want the values of the input color type (assuming that one is palette)
3481     Note that we never create a new palette ourselves.*/
3482     if(palettesize == 0)
3483     {
3484       palettesize = mode_in->palettesize;
3485       palette = mode_in->palette;
3486     }
3487     if(palettesize < palsize) palsize = palettesize;
3488     color_tree_init(&tree);
3489     for(i = 0; i != palsize; ++i)
3490     {
3491       const unsigned char* p = &palette[i * 4];
3492       color_tree_add(&tree, p[0], p[1], p[2], p[3], i);
3493     }
3494   }
3495
3496   if(mode_in->bitdepth == 16 && mode_out->bitdepth == 16)
3497   {
3498     for(i = 0; i != numpixels; ++i)
```

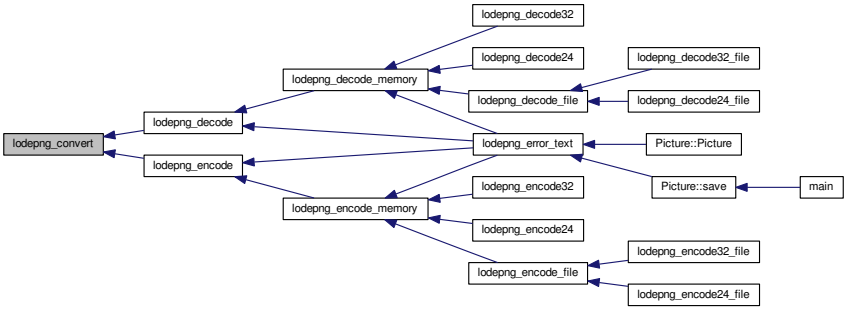
```
3499     {
3500       unsigned short r = 0, g = 0, b = 0, a = 0;
3501       getPixelColorRGBA16(&r, &g, &b, &a, in, i, mode_in);
3502       rgba16ToPixel(out, i, mode_out, r, g, b, a);
3503     }
3504   }
3505   else if(mode_out->bitdepth == 8 && mode_out->colortype ==
      LCT_RGBA)
3506   {
3507     getPixelColorsRGBA8(out, numpixels, 1, in, mode_in);
3508   }
3509   else if(mode_out->bitdepth == 8 && mode_out->colortype ==
      LCT_RGB)
3510   {
3511     getPixelColorsRGBA8(out, numpixels, 0, in, mode_in);
3512   }
3513   else
3514   {
3515     unsigned char r = 0, g = 0, b = 0, a = 0;
3516     for(i = 0; i != numpixels; ++i)
3517     {
3518       getPixelColorRGBA8(&r, &g, &b, &a, in, i, mode_in);
3519       CERROR_TRY_RETURN(rgba8ToPixel(out, i, mode_out, &tree, r, g, b, a));
3520     }
3521   }
3522
3523   if(mode_out->colortype == LCT_PALETTE)
3524   {
3525     color_tree_cleanup(&tree);
3526   }
3527
```

```
3528   return 0; /*no error*/
3529 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



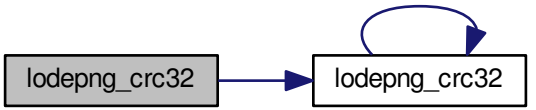
4.2.4.27 lodepng_crc32()

```
unsigned lodepng_crc32 (
            const unsigned char * buf,
            size_t len )
```

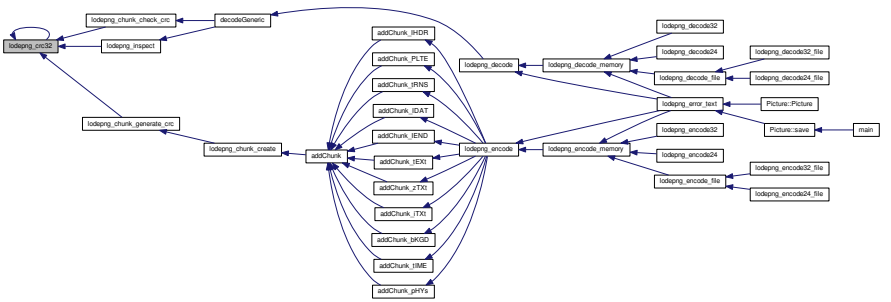
Definition at line 2359 of file lodepng.cpp.

```
2360 {
2361   unsigned r = 0xffffffffu;
2362   size_t i;
2363   for(i = 0; i < length; ++i)
2364   {
2365     r = lodepng_crc32_table[(r ^ data[i]) & 0xff] ^ (r >> 8);
2366   }
2367   return r ^ 0xffffffffu;
2368 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.28 lodepng_decode()

```
unsigned lodepng_decode (
            unsigned char ** out,
```

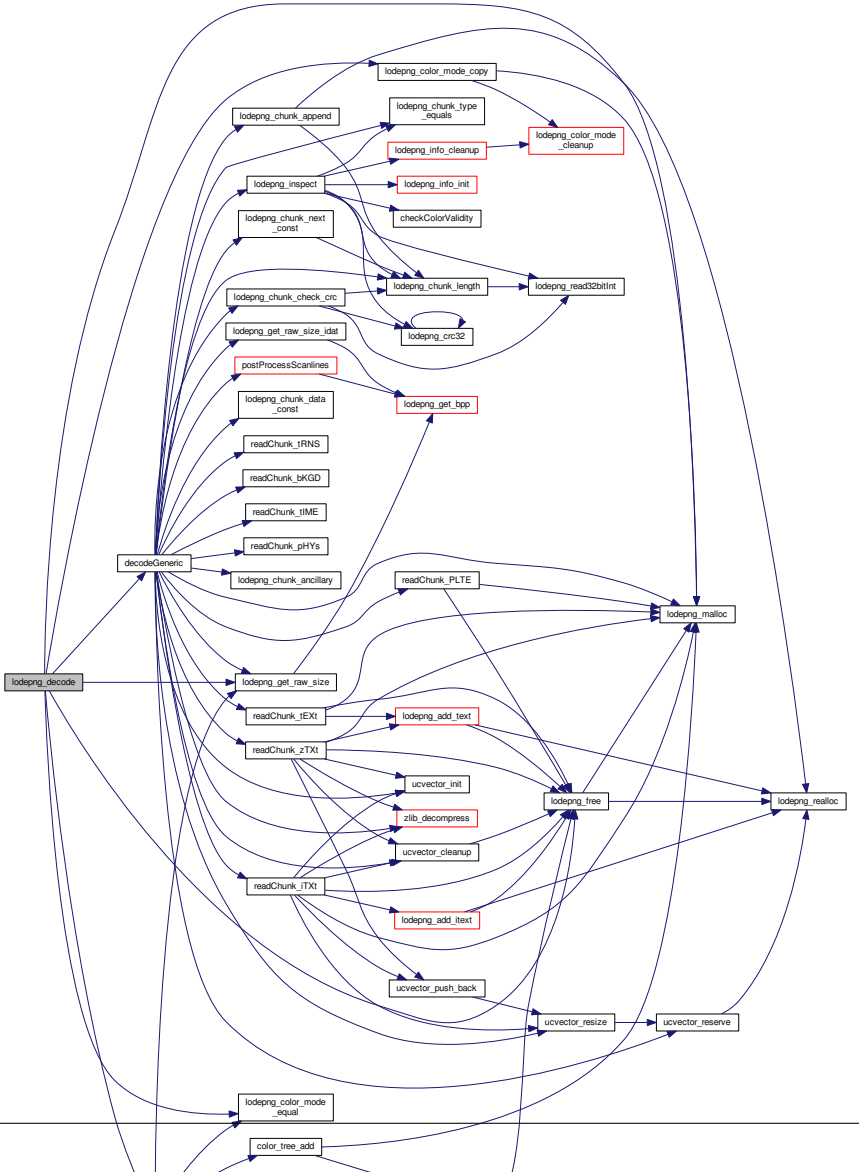
```
    unsigned * w,
    unsigned * h,
    LodePNGState * state,
    const unsigned char * in,
    size_t insize )
```

Definition at line 4723 of file lodepng.cpp.

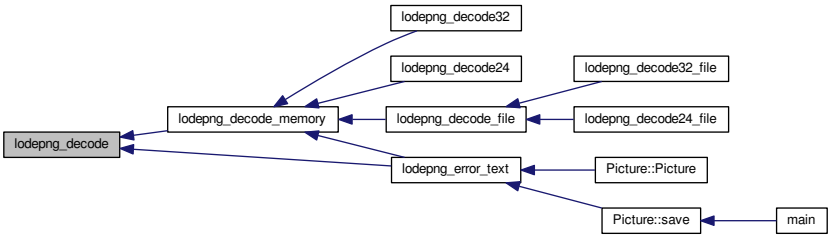
```
4726 {
4727   *out = 0;
4728   decodeGeneric(out, w, h, state, in, insize);
4729   if(state->error) return state->error;
4730   if(!state->decoder.color_convert ||
      lodepng_color_mode_equal(&state->info_raw, &state->
      info_png.color))
4731   {
4732     /*same color type, no copying or converting of data needed*/
4733     /*store the info_png color settings on the info_raw so that the info_raw st
4734     the raw image has to the end user*/
4735     if(!state->decoder.color_convert)
4736     {
4737       state->error = lodepng_color_mode_copy(&state->
      info_raw, &state->info_png.color);
4738       if(state->error) return state->error;
4739     }
4740   }
4741   else
4742   {
4743     /*color conversion needed; sort of copy of the data*/
4744     unsigned char* data = *out;
4745     size_t outsize;
4746
```

```
4747     /*TODO: check if this works according to the statement in the documentation
4748     from greyscale input color type, to 8-bit greyscale or greyscale with alpha
4749     if(!(state->info_raw.colortype == LCT_RGB || state->
     info_raw.colortype == LCT_RGBA)
4750        && !(state->info_raw.bitdepth == 8))
4751     {
4752       return 56; /*unsupported color mode conversion*/
4753     }
4754
4755     outsize = lodepng_get_raw_size(*w, *h, &state->info_raw);
4756     *out = (unsigned char*)lodepng_malloc(outsize);
4757     if(!(*out))
4758     {
4759       state->error = 83; /*alloc fail*/
4760     }
4761     else state->error = lodepng_convert(*out, data, &state->
     info_raw,
4762                                         &state->info_png.color, *w, *h);
4763     lodepng_free(data);
4764   }
4765   return state->error;
4766 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



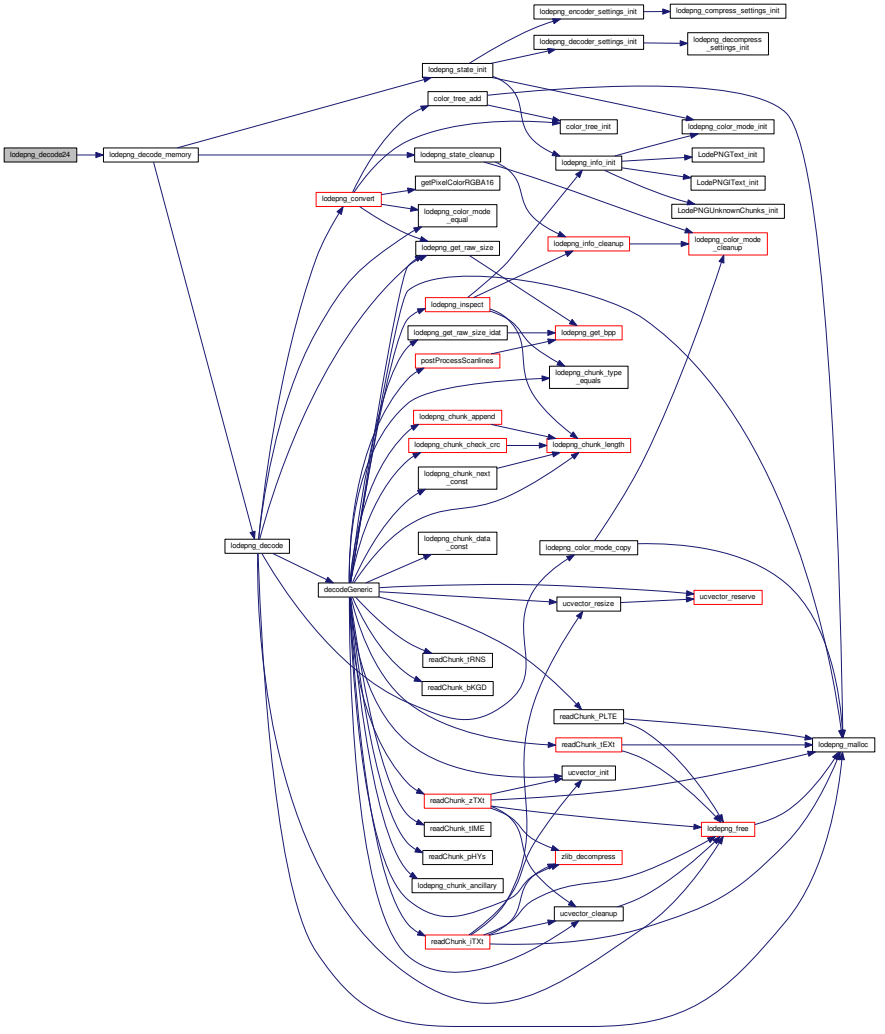
4.2.4.29 lodepng_decode24()

```
unsigned lodepng_decode24 (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const unsigned char * in,
            size_t insize )
```

Definition at line 4786 of file lodepng.cpp.

```
4787 {
4788   return lodepng_decode_memory(out, w, h, in, insize,
      LCT_RGB, 8);
4789 }
```

Here is the call graph for this function:



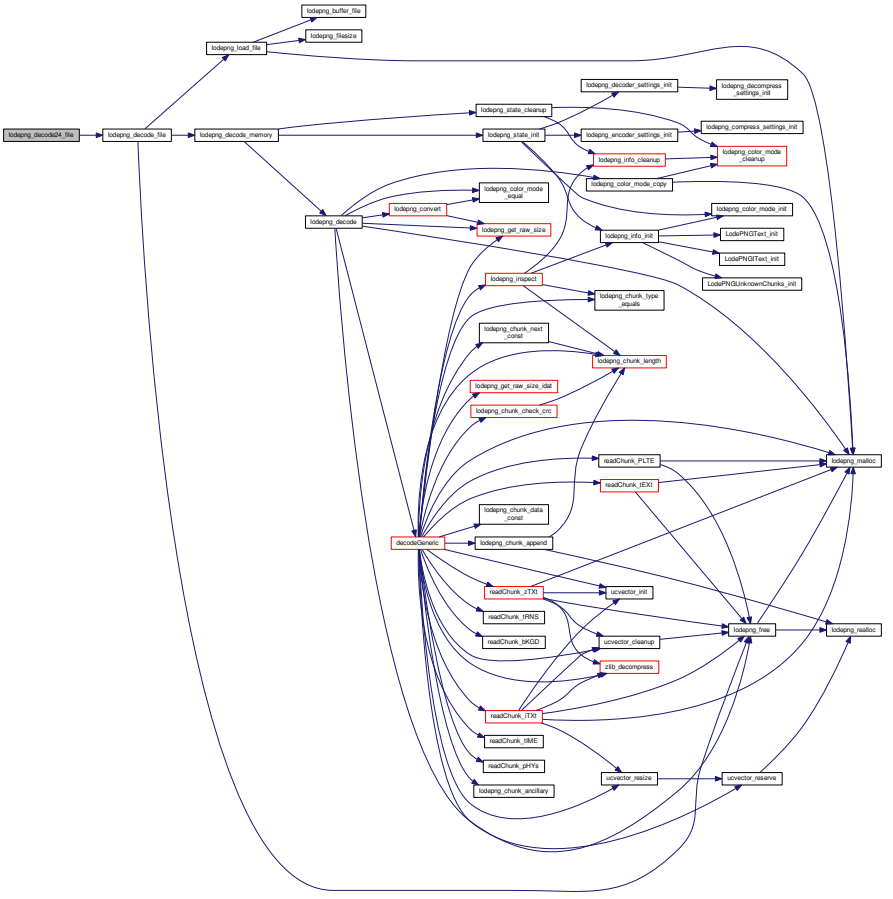
4.2.4.30 lodepng_decode24_file()

```
unsigned lodepng_decode24_file (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const char * filename )
```

Definition at line 4809 of file lodepng.cpp.

```
4810 {
4811   return lodepng_decode_file(out, w, h, filename, LCT_RGB, 8);
4812 }
```

Here is the call graph for this function:



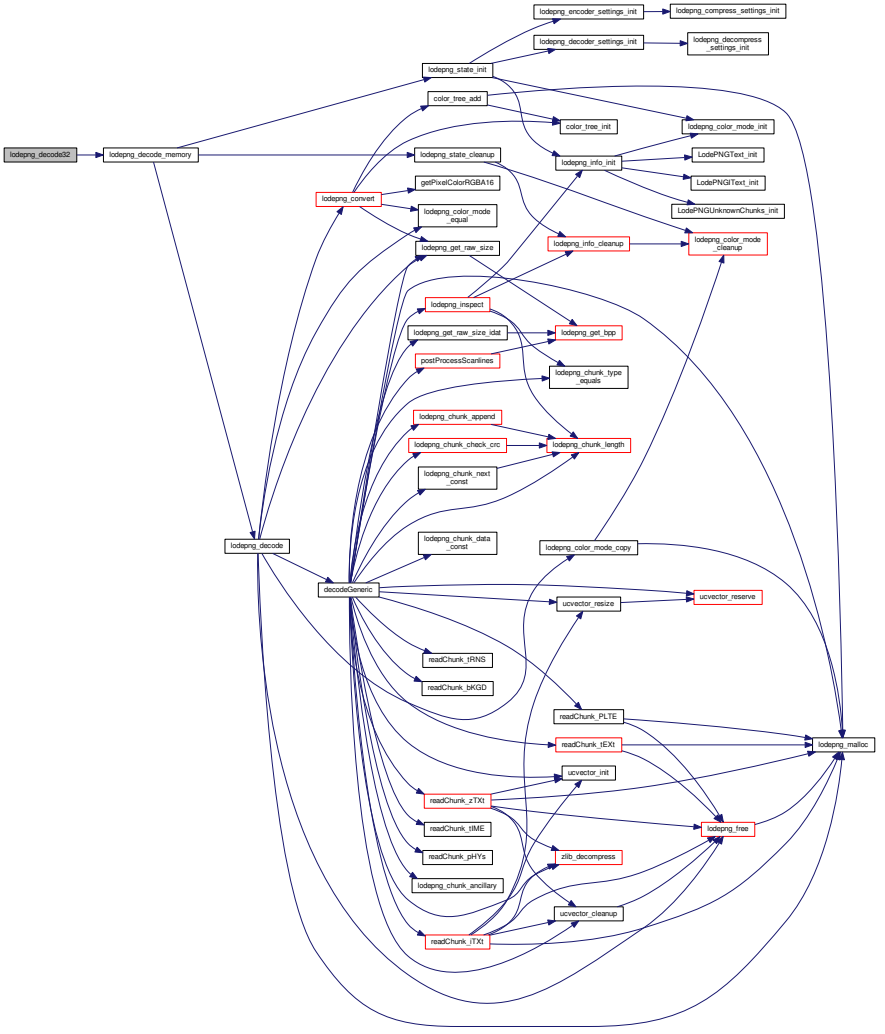
4.2.4.31 lodepng_decode32()

```
unsigned lodepng_decode32 (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const unsigned char * in,
            size_t insize )
```

Definition at line 4781 of file lodepng.cpp.

```
4782 {
4783   return lodepng_decode_memory(out, w, h, in, insize,
      LCT_RGBA, 8);
4784 }
```

Here is the call graph for this function:



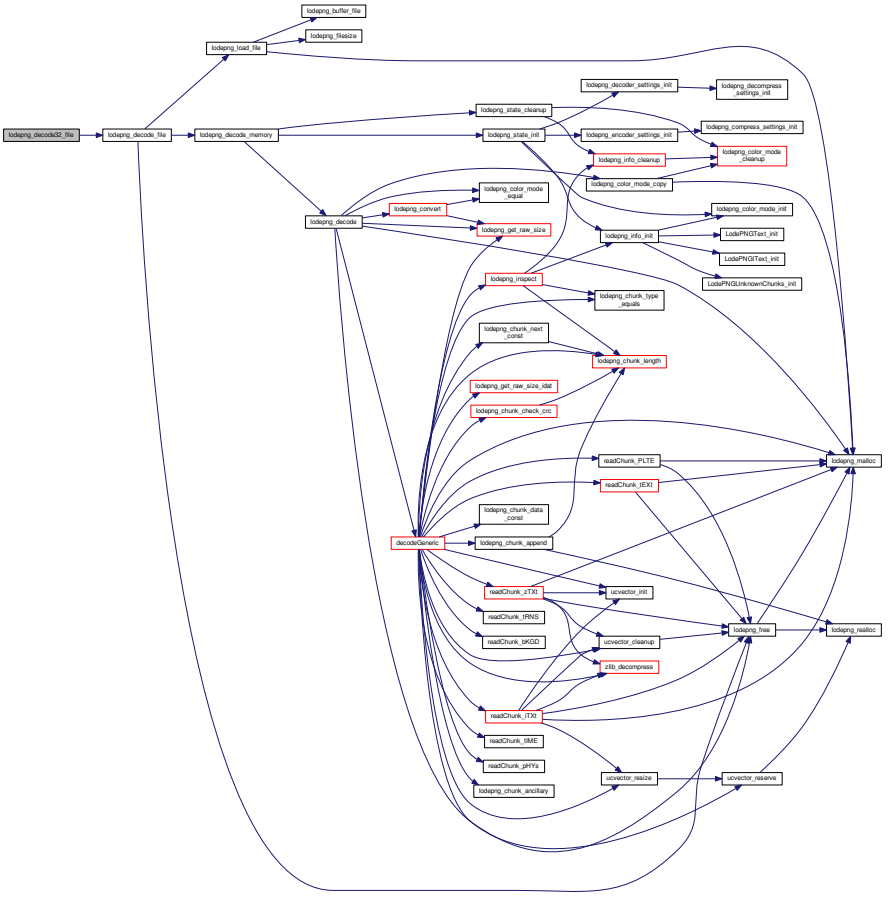
4.2.4.32 lodepng_decode32_file()

```
unsigned lodepng_decode32_file (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const char * filename )
```

Definition at line 4804 of file lodepng.cpp.

```
4805 {
4806   return lodepng_decode_file(out, w, h, filename, LCT_RGBA, 8);
4807 }
```

Here is the call graph for this function:



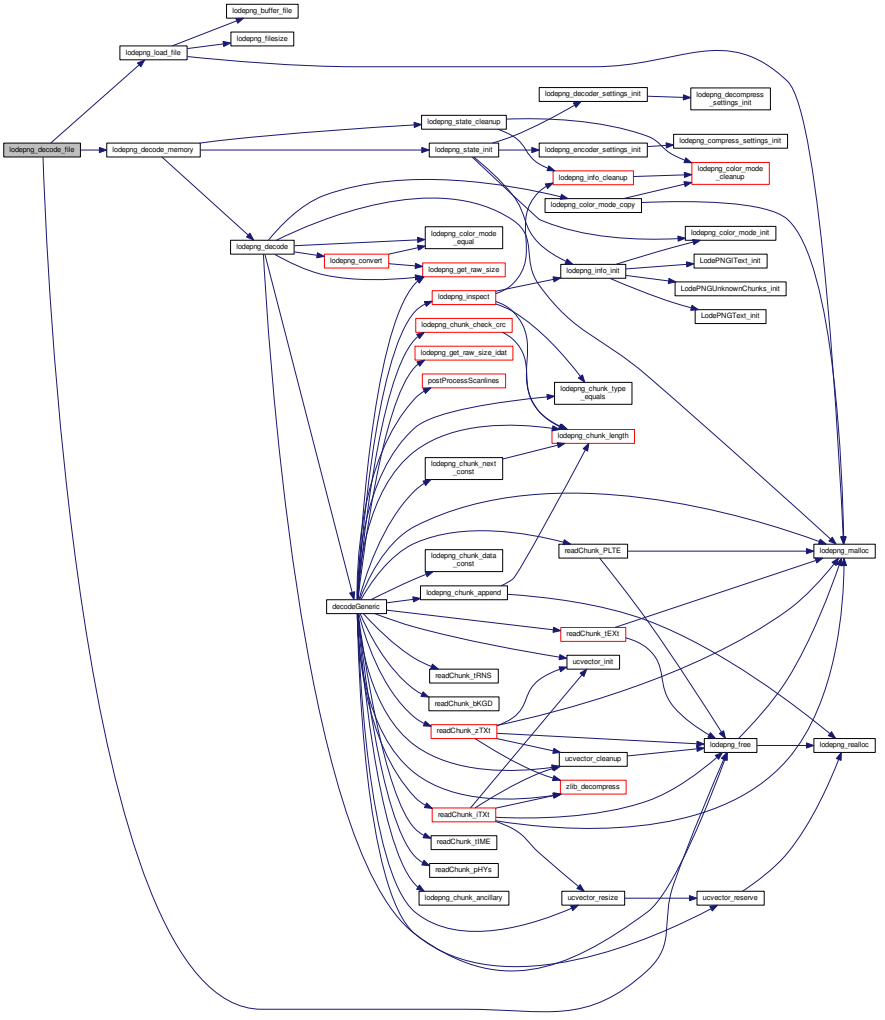
4.2.4.33 lodepng_decode_file()

```
unsigned lodepng_decode_file (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const char * filename,
            LodePNGColorType colortype,
            unsigned bitdepth )
```

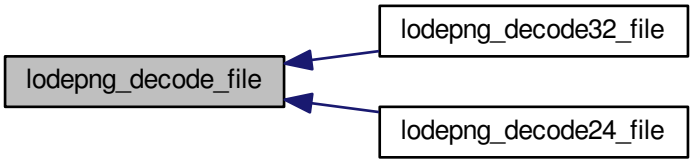
Definition at line 4792 of file lodepng.cpp.

```
4794 {
4795   unsigned char* buffer = 0;
4796   size_t buffersize;
4797   unsigned error;
4798   error = lodepng_load_file(&buffer, &buffersize, filename);
4799   if(!error) error = lodepng_decode_memory(out, w, h, buffer, buffersize, color
      bitdepth);
4800   lodepng_free(buffer);
4801   return error;
4802 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.34 lodepng_decode_memory()

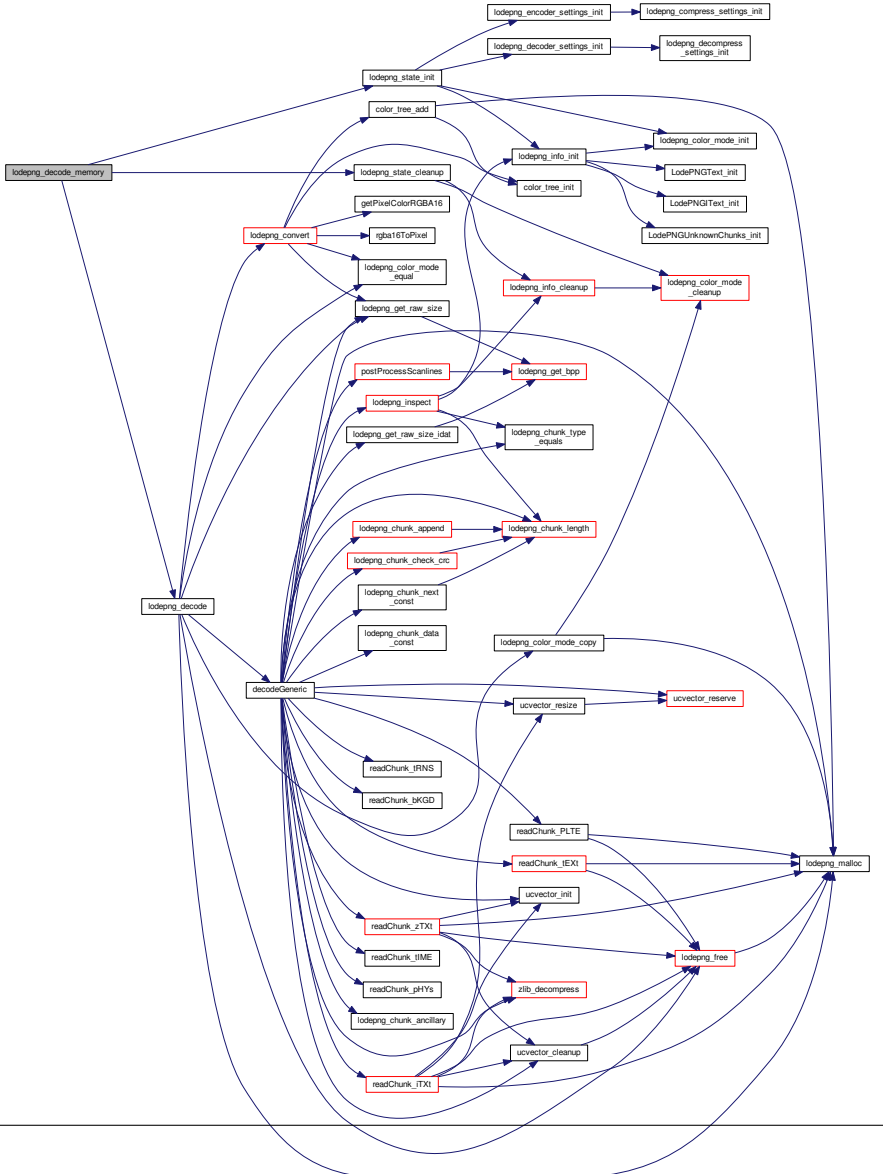
```
unsigned lodepng_decode_memory (
            unsigned char ** out,
            unsigned * w,
            unsigned * h,
            const unsigned char * in,
            size_t insize,
            LodePNGColorType colortype,
            unsigned bitdepth )
```

Definition at line 4768 of file lodepng.cpp.

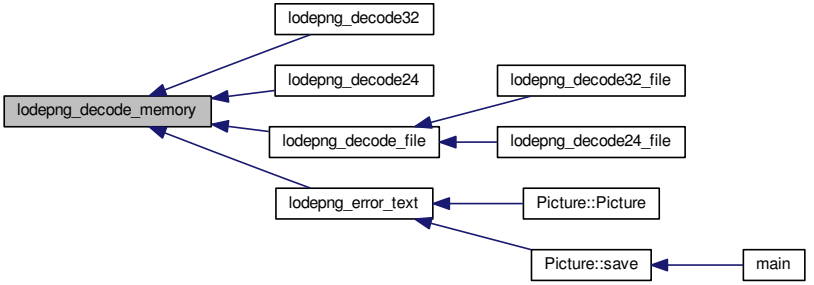
```
4770 {
4771   unsigned error;
4772   LodePNGState state;
```

```
4773   lodepng_state_init(&state);
4774   state.info_raw.colortype = colortype;
4775   state.info_raw.bitdepth = bitdepth;
4776   error = lodepng_decode(out, w, h, &state, in, insize);
4777   lodepng_state_cleanup(&state);
4778   return error;
4779 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.35 lodepng_decoder_settings_init()

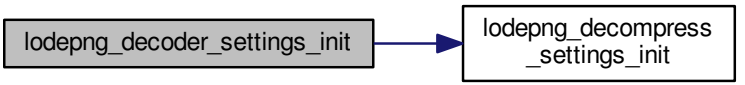
```
void lodepng_decoder_settings_init (
            LodePNGDecoderSettings * settings )
```

Definition at line 4815 of file lodepng.cpp.

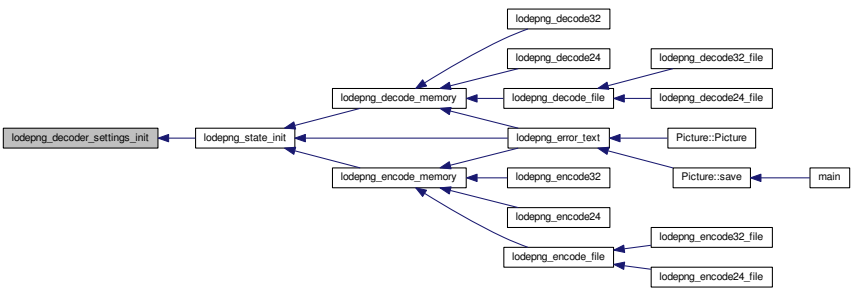
```
4816 {
4817   settings->color_convert = 1;
4818 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
4819   settings->read_text_chunks = 1;
4820   settings->remember_unknown_chunks = 0;
4821 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
4822   settings->ignore_crc = 0;
```

```
4823   lodepng_decompress_settings_init(&settings->
       zlibsettings);
4824 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



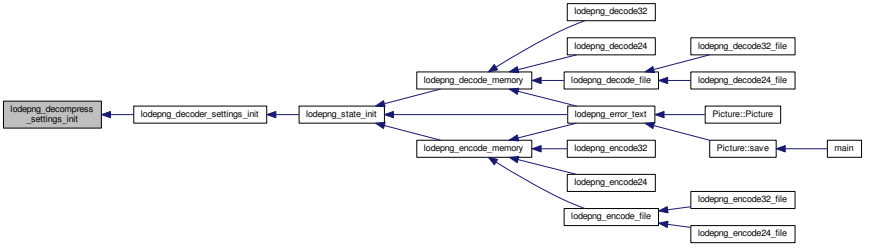
4.2.4.36 lodepng_decompress_settings_init()

```
void lodepng_decompress_settings_init (
            LodePNGDecompressSettings * settings )
```

Definition at line 2295 of file lodepng.cpp.

```
2296 {
2297   settings->ignore_adler32 = 0;
2298
2299   settings->custom_zlib = 0;
2300   settings->custom_inflate = 0;
2301   settings->custom_context = 0;
2302 }
```

Here is the caller graph for this function:



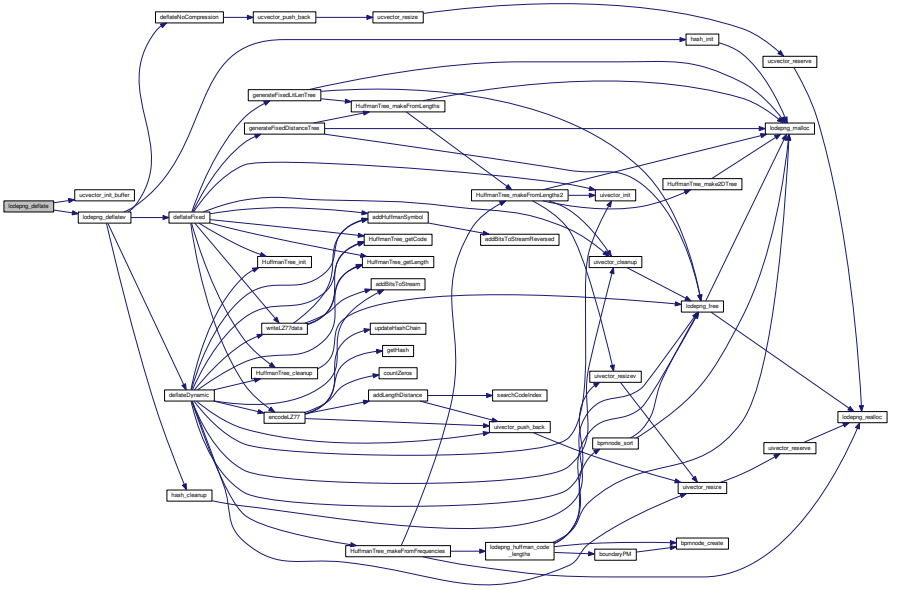
4.2.4.37 lodepng_deflate()

```
unsigned lodepng_deflate (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGCompressSettings * settings )
```

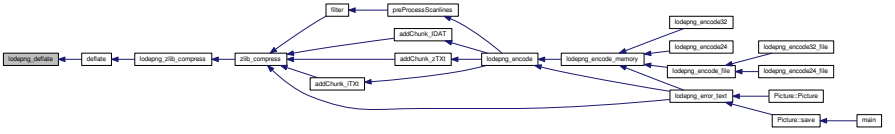
Definition at line 2058 of file lodepng.cpp.

```
2061 {
2062   unsigned error;
2063   ucvector v;
2064   ucvector_init_buffer(&v, *out, *outsize);
2065   error = lodepng_deflatev(&v, in, insize, settings);
2066   *out = v.data;
2067   *outsize = v.size;
2068   return error;
2069 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.38 lodepng_encode()

```
unsigned lodepng_encode (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * image,
            unsigned w,
            unsigned h,
            LodePNGState * state )
```

Definition at line 5640 of file lodepng.cpp.

```
5643 {
5644   LodePNGInfo info;
5645   ucvector outv;
5646   unsigned char* data = 0; /*uncompressed version of the IDAT chunk data*/
5647   size_t datasize = 0;
5648
5649   /*provide some proper output values if error will happen*/
5650   *out = 0;
5651   *outsize = 0;
5652   state->error = 0;
5653
5654   lodepng_info_init(&info);
5655   lodepng_info_copy(&info, &state->info_png);
5656
5657   if((info.color.colortype == LCT_PALETTE || state->
      encoder.force_palette)
5658       && (info.color.palettesize == 0 || info.color.
      palettesize > 256))
5659   {
5660     state->error = 68; /*invalid palette size, it is only allowed to be 1-256*/
```

```
5661     return state->error;
5662   }
5663
5664   if(state->encoder.auto_convert)
5665   {
5666     state->error = lodepng_auto_choose_color(&info.
       color, image, w, h, &state->info_raw);
5667   }
5668   if(state->error) return state->error;
5669
5670   if(state->encoder.zlibsettings.btype > 2)
5671   {
5672     CERROR_RETURN_ERROR(state->error, 61); /*error: unexisting btype*/
5673   }
5674   if(state->info_png.interlace_method > 1)
5675   {
5676     CERROR_RETURN_ERROR(state->error, 71); /*error: unexisting interlace mode*/
5677   }
5678
5679   state->error = checkColorValidity(info.color.
       colortype, info.color.bitdepth);
5680   if(state->error) return state->error; /*error: unexisting color type given*/
5681   state->error = checkColorValidity(state->info_raw.
       colortype, state->info_raw.bitdepth);
5682   if(state->error) return state->error; /*error: unexisting color type given*/
5683
5684   if(!lodepng_color_mode_equal(&state->info_raw, &info.
       color))
5685   {
5686     unsigned char* converted;
5687     size_t size = (w * h * (size_t)lodepng_get_bpp(&info.color) + 7) / 8;
```

```
5688
5689     converted = (unsigned char*)lodepng_malloc(size);
5690     if(!converted && size) state->error = 83; /*alloc fail*/
5691     if(!state->error)
5692     {
5693       state->error = lodepng_convert(converted, image, &info.
     color, &state->info_raw, w, h);
5694     }
5695     if(!state->error) preProcessScanlines(&data, &datasize, converted, w, h, &i
      &state->encoder);
5696     lodepng_free(converted);
5697   }
5698   else preProcessScanlines(&data, &datasize, image, w, h, &info, &state->
     encoder);
5699
5700   ucvector_init(&outv);
5701   while(!state->error) /*while only executed once, to break on error*/
5702   {
5703 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
5704     size_t i;
5705 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
5706     /*write signature and chunks*/
5707     writeSignature(&outv);
5708     /*IHDR*/
5709     addChunk_IHDR(&outv, w, h, info.color.colortype, info.
     color.bitdepth, info.interlace_method);
5710 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
5711     /*unknown chunks between IHDR and PLTE*/
5712     if(info.unknown_chunks_data[0])
5713     {
5714       state->error = addUnknownChunks(&outv, info.
```

```
      unknown_chunks_data[0], info.unknown_chunks_size[0]);
5715       if(state->error) break;
5716     }
5717 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
5718     /*PLTE*/
5719     if(info.color.colortype == LCT_PALETTE)
5720     {
5721       addChunk_PLTE(&outv, &info.color);
5722     }
5723     if(state->encoder.force_palette && (info.color.
      colortype == LCT_RGB || info.color.colortype ==
      LCT_RGBA))
5724     {
5725       addChunk_PLTE(&outv, &info.color);
5726     }
5727     /*tRNS*/
5728     if(info.color.colortype == LCT_PALETTE &&
      getPaletteTranslucency(info.color.palette, info.
      color.palettesize) != 0)
5729     {
5730       addChunk_tRNS(&outv, &info.color);
5731     }
5732     if((info.color.colortype == LCT_GREY || info.color.
      colortype == LCT_RGB) && info.color.key_defined)
5733     {
5734       addChunk_tRNS(&outv, &info.color);
5735     }
5736 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
5737     /*bKGD (must come between PLTE and the IDAt chunks*/
5738     if(info.background_defined) addChunk_bKGD(&outv, &info);
5739     /*pHYs (must come before the IDAT chunks)*/
```

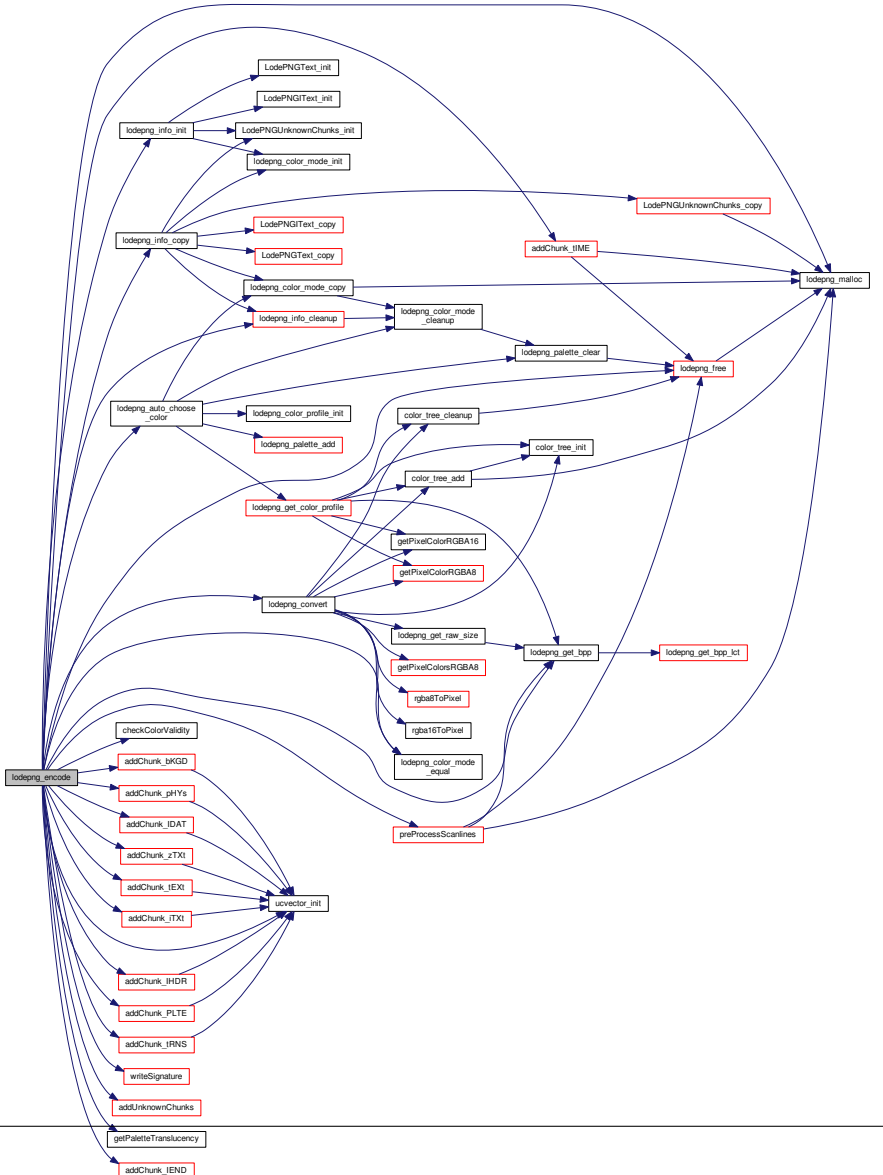
```
5740     if(info.phys_defined) addChunk_pHYs(&outv, &info);
5741
5742     /*unknown chunks between PLTE and IDAT*/
5743     if(info.unknown_chunks_data[1])
5744     {
5745       state->error = addUnknownChunks(&outv, info.
      unknown_chunks_data[1], info.unknown_chunks_size[1]);
5746       if(state->error) break;
5747     }
5748 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
5749     /*IDAT (multiple IDAT chunks must be consecutive)*/
5750     state->error = addChunk_IDAT(&outv, data, datasize, &state->
      encoder.zlibsettings);
5751     if(state->error) break;
5752 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
5753     /*tIME*/
5754     if(info.time_defined) addChunk_tIME(&outv, &info.
      time);
5755     /*tEXt and/or zTXt*/
5756     for(i = 0; i != info.text_num; ++i)
5757     {
5758       if(strlen(info.text_keys[i]) > 79)
5759       {
5760         state->error = 66; /*text chunk too large*/
5761         break;
5762       }
5763       if(strlen(info.text_keys[i]) < 1)
5764       {
5765         state->error = 67; /*text chunk too small*/
5766         break;
5767       }
```

```
5768       if(state->encoder.text_compression)
5769       {
5770         addChunk_zTXt(&outv, info.text_keys[i], info.
     text_strings[i], &state->encoder.zlibsettings);
5771       }
5772       else
5773       {
5774         addChunk_tEXt(&outv, info.text_keys[i], info.
     text_strings[i]);
5775       }
5776     }
5777     /*LodePNG version id in text chunk*/
5778     if(state->encoder.add_id)
5779     {
5780       unsigned alread_added_id_text = 0;
5781       for(i = 0; i != info.text_num; ++i)
5782       {
5783         if(!strcmp(info.text_keys[i], "LodePNG"))
5784         {
5785           alread_added_id_text = 1;
5786           break;
5787         }
5788       }
5789       if(alread_added_id_text == 0)
5790       {
5791         addChunk_tEXt(&outv, "LodePNG", LODEPNG_VERSION_STRING); /*it's
      shorter as tEXt than as zTXt chunk*/
5792       }
5793     }
5794     /*iTXt*/
5795     for(i = 0; i != info.itext_num; ++i)
```

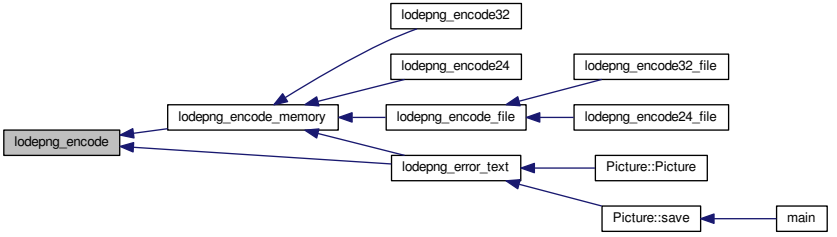
```
5796     {
5797       if(strlen(info.itext_keys[i]) > 79)
5798       {
5799         state->error = 66; /*text chunk too large*/
5800         break;
5801       }
5802       if(strlen(info.itext_keys[i]) < 1)
5803       {
5804         state->error = 67; /*text chunk too small*/
5805         break;
5806       }
5807       addChunk_iTXt(&outv, state->encoder.text_compression,
5808                     info.itext_keys[i], info.itext_langtags[i], info.
     itext_transkeys[i], info.itext_strings[i],
5809                     &state->encoder.zlibsettings);
5810     }
5811
5812     /*unknown chunks between IDAT and IEND*/
5813     if(info.unknown_chunks_data[2])
5814     {
5815       state->error = addUnknownChunks(&outv, info.
     unknown_chunks_data[2], info.unknown_chunks_size[2]);
5816       if(state->error) break;
5817     }
5818 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
5819     addChunk_IEND(&outv);
5820
5821     break; /*this isn't really a while loop; no error happened so break out now
5822   }
5823
5824   lodepng_info_cleanup(&info);
```

```
5825   lodepng_free(data);
5826   /*instead of cleaning the vector up, give it to the output*/
5827   *out = outv.data;
5828   *outsize = outv.size;
5829
5830   return state->error;
5831 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.39 lodepng_encode24()

```
unsigned lodepng_encode24 (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * image,
            unsigned w,
            unsigned h )
```

Definition at line 5854 of file lodepng.cpp.

```
5855 {
5856   return lodepng_encode_memory(out, outsize, image, w, h,
      LCT_RGB, 8);
5857 }
```

Here is the call graph for this function:

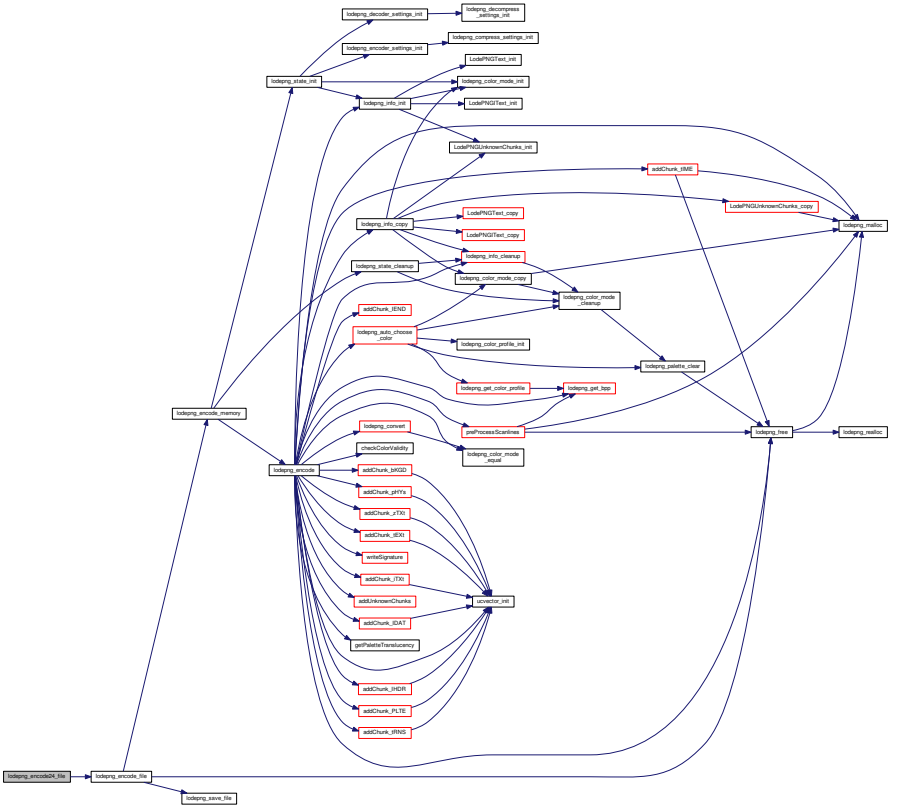
4.2.4.40 lodepng_encode24_file()

```
unsigned lodepng_encode24_file (
            const char * filename,
            const unsigned char * image,
            unsigned w,
            unsigned h )
```

Definition at line 5876 of file lodepng.cpp.

```
5877 {
5878   return lodepng_encode_file(filename, image, w, h, LCT_RGB, 8);
5879 }
```

Here is the call graph for this function:



4.2.4.41 lodepng_encode32()

```
unsigned lodepng_encode32 (
            unsigned char ** out,
```

```
        size_t * outsize,
        const unsigned char * image,
        unsigned w,
        unsigned h )
```

Definition at line 5849 of file lodepng.cpp.

```
5850 {
5851   return lodepng_encode_memory(out, outsize, image, w, h,
      LCT_RGBA, 8);
5852 }
```

Here is the call graph for this function:

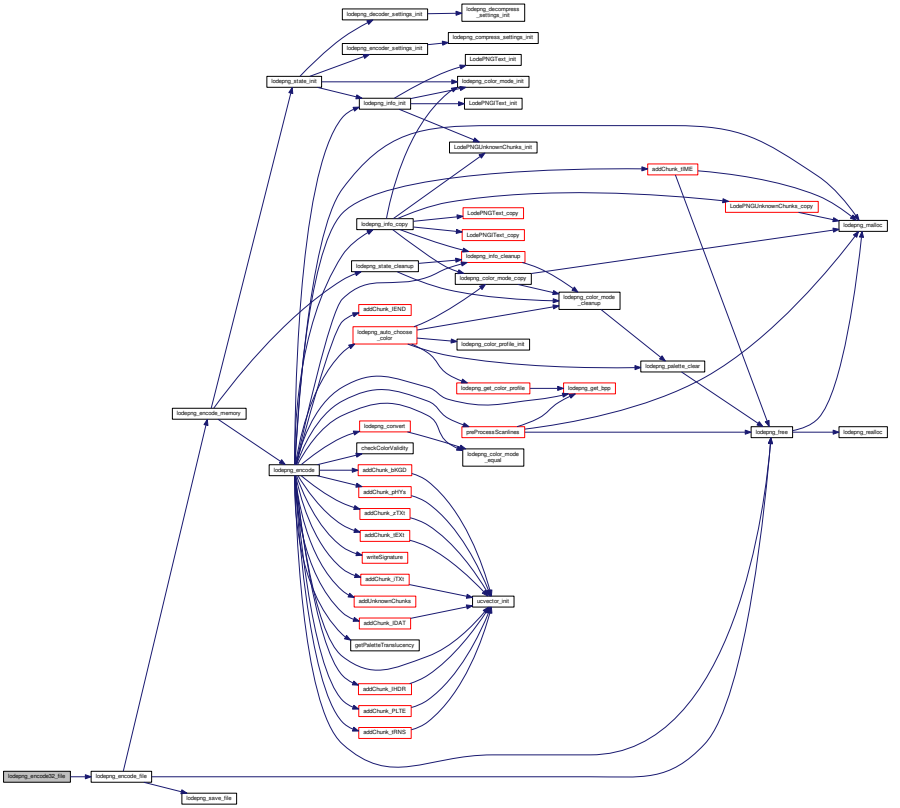
4.2.4.42 lodepng_encode32_file()

```
unsigned lodepng_encode32_file (
            const char * filename,
            const unsigned char * image,
            unsigned w,
            unsigned h )
```

Definition at line 5871 of file lodepng.cpp.

```
5872 {
5873   return lodepng_encode_file(filename, image, w, h, LCT_RGBA, 8);
5874 }
```

Here is the call graph for this function:



4.2.4.43 lodepng_encode_file()

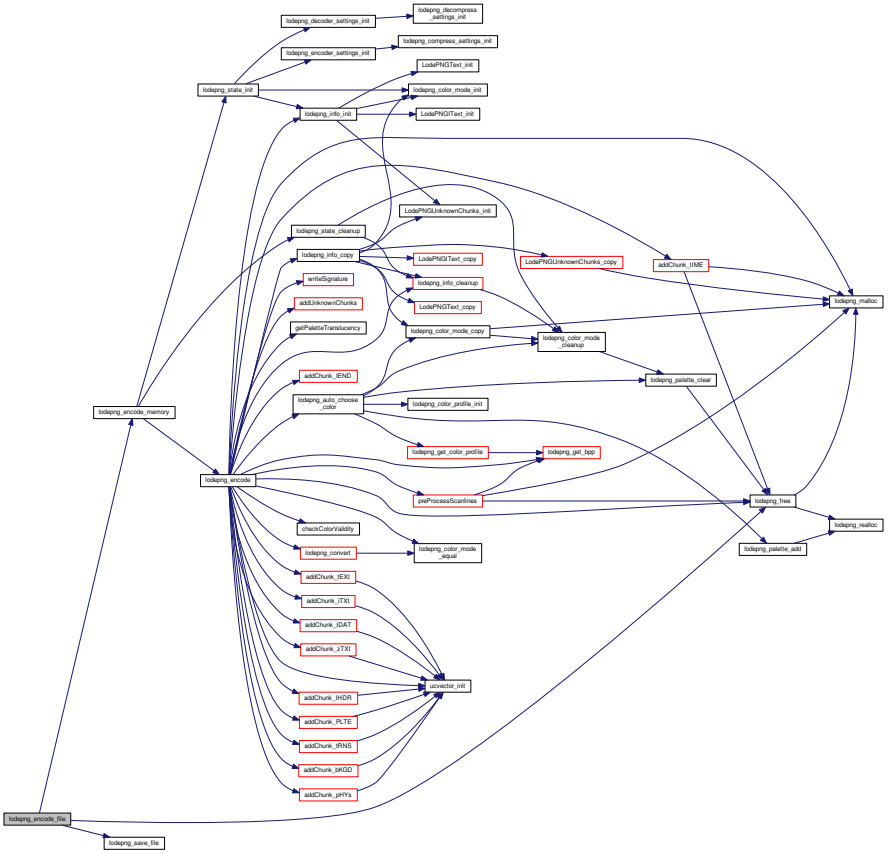
```
unsigned lodepng_encode_file (
            const char * filename,
```

```
    const unsigned char * image,
    unsigned w,
    unsigned h,
    LodePNGColorType colortype,
    unsigned bitdepth )
```

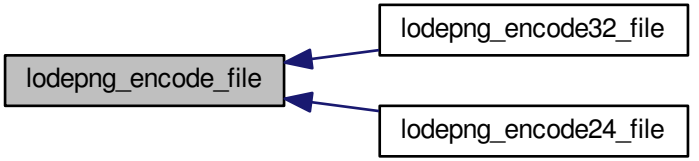
Definition at line 5860 of file lodepng.cpp.

```
5862 {
5863   unsigned char* buffer;
5864   size_t buffersize;
5865   unsigned error = lodepng_encode_memory(&buffer, &buffersize, image, w, h, col
       bitdepth);
5866   if(!error) error = lodepng_save_file(buffer, buffersize, filename);
5867   lodepng_free(buffer);
5868   return error;
5869 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.44 lodepng_encode_memory()

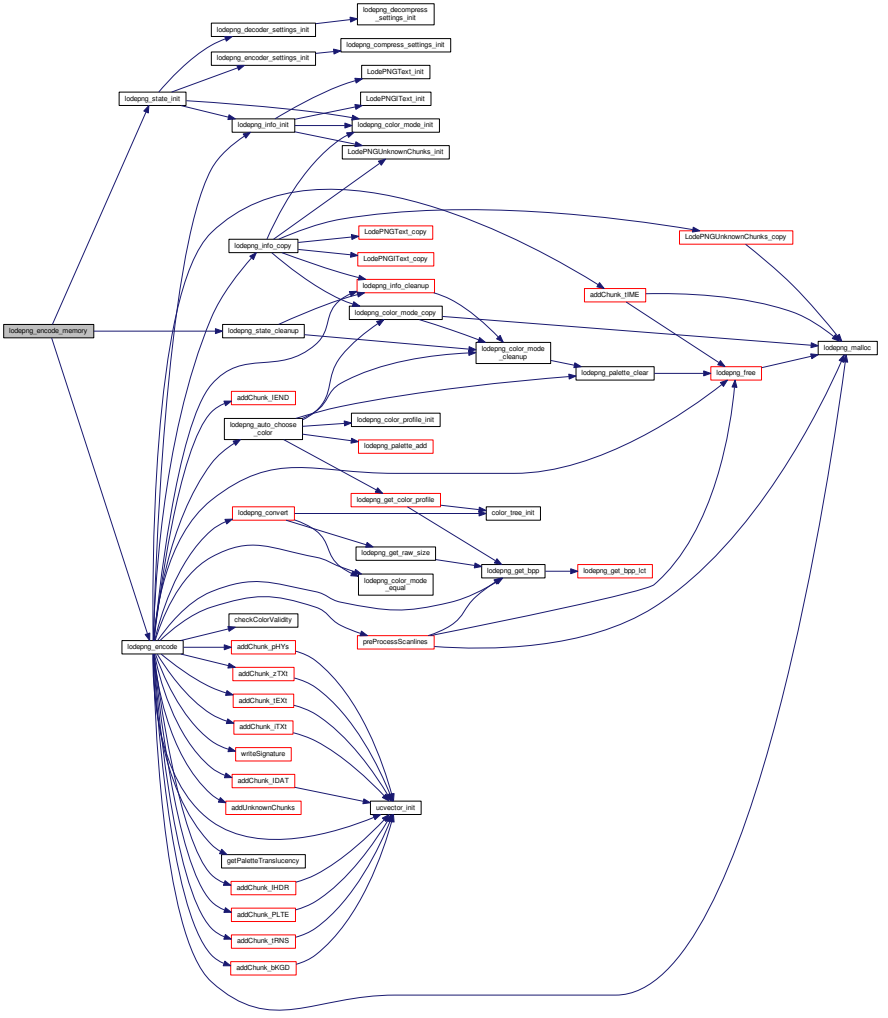
```
unsigned lodepng_encode_memory (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * image,
            unsigned w,
            unsigned h,
            LodePNGColorType colortype,
            unsigned bitdepth )
```

Definition at line 5833 of file lodepng.cpp.

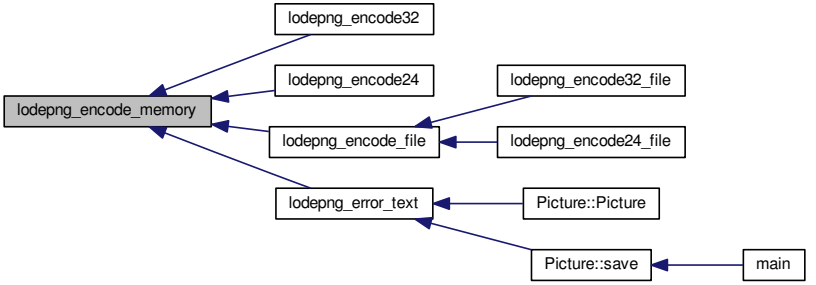
```
5835 {
5836   unsigned error;
5837   LodePNGState state;
```

```
5838   lodepng_state_init(&state);
5839   state.info_raw.colortype = colortype;
5840   state.info_raw.bitdepth = bitdepth;
5841   state.info_png.color.colortype = colortype;
5842   state.info_png.color.bitdepth = bitdepth;
5843   lodepng_encode(out, outsize, image, w, h, &state);
5844   error = state.error;
5845   lodepng_state_cleanup(&state);
5846   return error;
5847 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.45 lodepng_encoder_settings_init()

```
void lodepng_encoder_settings_init (
            LodePNGEncoderSettings * settings )
```

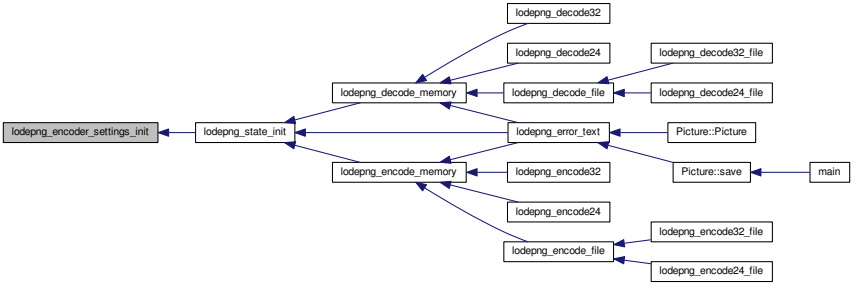
Definition at line 5882 of file lodepng.cpp.

```
5883 {
5884   lodepng_compress_settings_init(&settings->
      zlibsettings);
5885   settings->filter_palette_zero = 1;
5886   settings->filter_strategy = LFS_MINSUM;
5887   settings->auto_convert = 1;
5888   settings->force_palette = 0;
```

```
5889   settings->predefined_filters = 0;
5890 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
5891   settings->add_id = 0;
5892   settings->text_compression = 1;
5893 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
5894 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.2.4.46 lodepng_error_text()

```
const char* lodepng_error_text (
            unsigned code )
```

Definition at line 5904 of file lodepng.cpp.

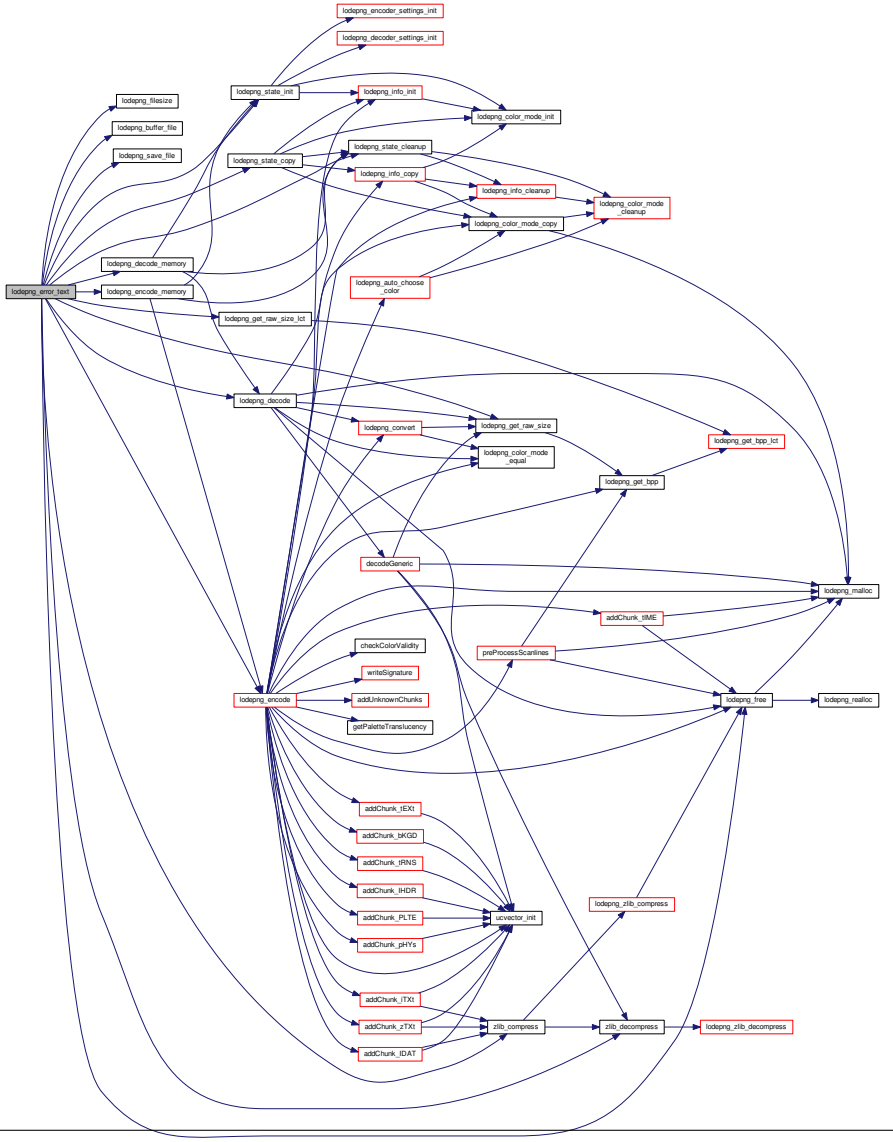
```
5905 {
5906   switch(code)
5907   {
5908     case 0: return "no error, everything went ok";
5909     case 1: return "nothing done yet"; /*the Encoder/Decoder has done nothing y
       sense yet*/
5910     case 10: return "end of input memory reached without huffman end code"; /*w
5911     case 11: return "error in code tree made it jump outside of huffman tree";
5912     case 13: return "problem while processing dynamic deflate block";
5913     case 14: return "problem while processing dynamic deflate block";
5914     case 15: return "problem while processing dynamic deflate block";
5915     case 16: return "unexisting code while processing dynamic deflate block";
5916     case 17: return "end of out buffer memory reached while inflating";
5917     case 18: return "invalid distance code while inflating";
5918     case 19: return "end of out buffer memory reached while inflating";
5919     case 20: return "invalid deflate block BTYPE encountered while decoding";
5920     case 21: return "NLEN is not ones complement of LEN in a deflate block";
5921      /*end of out buffer memory reached while inflating:
5922      This can happen if the inflated deflate data is longer than the amount of
5923      all the pixels of the image, given the color depth and image dimensions. S
5924      happen in a normal, well encoded, PNG image.*/
5925     case 22: return "end of out buffer memory reached while inflating";
5926     case 23: return "end of in buffer memory reached while inflating";
5927     case 24: return "invalid FCHECK in zlib header";
```

```
5928     case 25: return "invalid compression method in zlib header";
5929     case 26: return "FDICT encountered in zlib header while it's not used for P
5930     case 27: return "PNG file is smaller than a PNG header";
5931     /*Checks the magic file header, the first 8 bytes of the PNG file*/
5932     case 28: return "incorrect PNG signature, it's no PNG or corrupted";
5933     case 29: return "first chunk is not the header chunk";
5934     case 30: return "chunk length too large, chunk broken off at end of file";
5935     case 31: return "illegal PNG color type or bpp";
5936     case 32: return "illegal PNG compression method";
5937     case 33: return "illegal PNG filter method";
5938     case 34: return "illegal PNG interlace method";
5939     case 35: return "chunk length of a chunk is too large or the chunk too smal
5940     case 36: return "illegal PNG filter type encountered";
5941     case 37: return "illegal bit depth for this color type given";
5942     case 38: return "the palette is too big"; /*more than 256 colors*/
5943     case 39: return "more palette alpha values given in tRNS chunk than there a
5944     case 40: return "tRNS chunk has wrong size for greyscale image";
5945     case 41: return "tRNS chunk has wrong size for RGB image";
5946     case 42: return "tRNS chunk appeared while it was not allowed for this colo
5947     case 43: return "bKGD chunk has wrong size for palette image";
5948     case 44: return "bKGD chunk has wrong size for greyscale image";
5949     case 45: return "bKGD chunk has wrong size for RGB image";
5950     case 48: return "empty input buffer given to decoder. Maybe caused by non-e
5951     case 49: return "jumped past memory while generating dynamic huffman tree";
5952     case 50: return "jumped past memory while generating dynamic huffman tree";
5953     case 51: return "jumped past memory while inflating huffman block";
5954     case 52: return "jumped past memory while inflating";
5955     case 53: return "size of zlib data too small";
5956     case 54: return "repeat symbol in tree while there was no value symbol yet"
5957     /*jumped past tree while generating huffman tree, this could be when the
5958     tree will have more leaves than symbols after generating it out of the
```

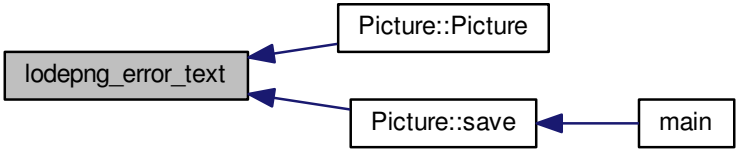
```
5959     given lenghts. They call this an oversubscribed dynamic bit lengths tree in
5960     case 55: return "jumped past tree while generating huffman tree";
5961     case 56: return "given output image colortype or bitdepth not supported for
5962     case 57: return "invalid CRC encountered (checking CRC can be disabled)";
5963     case 58: return "invalid ADLER32 encountered (checking ADLER32 can be disab
5964     case 59: return "requested color conversion not supported";
5965     case 60: return "invalid window size given in the settings of the encoder (
5966     case 61: return "invalid BTYPE given in the settings of the encoder (only 0
5967     /*LodePNG leaves the choice of RGB to greyscale conversion formula to the u
5968     case 62: return "conversion from color to greyscale not supported";
5969     case 63: return "length of a chunk too long, max allowed for PNG is 2147483
       (2^31-1)*/
5970     /*this would result in the inability of a deflated block to ever contain an
       least 1.*/
5971     case 64: return "the length of the END symbol 256 in the Huffman tree is 0"
5972     case 66: return "the length of a text chunk keyword given to the encoder is
       79 bytes";
5973     case 67: return "the length of a text chunk keyword given to the encoder is
       1 byte";
5974     case 68: return "tried to encode a PLTE chunk with a palette that has less
       colors";
5975     case 69: return "unknown chunk type with 'critical' flag encountered by the
5976     case 71: return "unexisting interlace mode given to encoder (must be 0 or 1
5977     case 72: return "while decoding, unexisting compression method encountering
       must be 0)";
5978     case 73: return "invalid tIME chunk size";
5979     case 74: return "invalid pHYs chunk size";
5980     /*length could be wrong, or data chopped off*/
5981     case 75: return "no null termination char found while decoding text chunk";
5982     case 76: return "iTXt chunk too short to contain required bytes";
5983     case 77: return "integer overflow in buffer size";
```

```
5984     case 78: return "failed to open file for reading"; /*file doesn't exist or
       reading*/
5985     case 79: return "failed to open file for writing";
5986     case 80: return "tried creating a tree of 0 symbols";
5987     case 81: return "lazy matching at pos 0 is impossible";
5988     case 82: return "color conversion to palette requested while a color isn't
5989     case 83: return "memory allocation failed";
5990     case 84: return "given image too small to contain all pixels to be encoded"
5991     case 86: return "impossible offset in lz77 encoding (internal bug)";
5992     case 87: return "must provide custom zlib function pointer if LODEPNG_COMPI
5993     case 88: return "invalid filter strategy given for LodePNGEncoderSettings.f
5994     case 89: return "text chunk keyword too short or long: must have size 1-79"
5995     /*the windowsize in the LodePNGCompressSettings. Requiring POT(==> & instea
       faster.*/
5996     case 90: return "windowsize must be a power of two";
5997     case 91: return "invalid decompressed idat size";
5998     case 92: return "too many pixels, not supported";
5999     case 93: return "zero width or height is invalid";
6000     case 94: return "header chunk must have a size of 13 bytes";
6001   }
6002   return "unknown error code";
6003 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



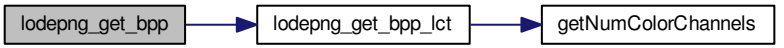
4.2.4.47 lodepng_get_bpp()

```
unsigned lodepng_get_bpp (
            const LodePNGColorMode * info )
```

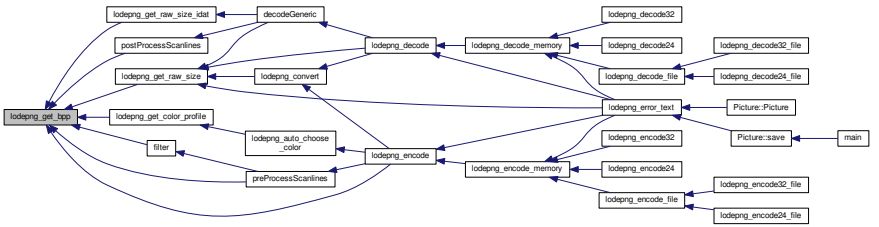
Definition at line 2665 of file lodepng.cpp.

```
2666 {
2667   /*calculate bits per pixel out of colortype and bitdepth*/
2668   return lodepng_get_bpp_lct(info->colortype, info->
      bitdepth);
2669 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.48 lodepng_get_channels()

```
unsigned lodepng_get_channels (
            const LodePNGColorMode * info )
```

Definition at line 2671 of file lodepng.cpp.

```
2672 {
2673   return getNumColorChannels(info->colortype);
2674 }
```

Here is the call graph for this function:

4.2.4.49 lodepng_get_color_profile()

```
unsigned lodepng_get_color_profile (
            LodePNGColorProfile * profile,
            const unsigned char * image,
            unsigned w,
            unsigned h,
            const LodePNGColorMode * mode_in )
```

Definition at line 3567 of file lodepng.cpp.

```
3570 {
3571   unsigned error = 0;
3572   size_t i;
3573   ColorTree tree;
3574   size_t numpixels = w * h;
3575
3576   unsigned colored_done = lodepng_is_greyscale_type(mode) ? 1 : 0;
```

```
3577   unsigned alpha_done = lodepng_can_have_alpha(mode) ? 0 : 1;
3578   unsigned numcolors_done = 0;
3579   unsigned bpp = lodepng_get_bpp(mode);
3580   unsigned bits_done = bpp == 1 ? 1 : 0;
3581   unsigned maxnumcolors = 257;
3582   unsigned sixteen = 0;
3583   if(bpp <= 8) maxnumcolors = bpp == 1 ? 2 : (bpp == 2 ? 4 : (bpp == 4 ? 16 : 2
3584
3585   color_tree_init(&tree);
3586
3587   /*Check if the 16-bit input is truly 16-bit*/
3588   if(mode->bitdepth == 16)
3589   {
3590     unsigned short r, g, b, a;
3591     for(i = 0; i != numpixels; ++i)
3592     {
3593       getPixelColorRGBA16(&r, &g, &b, &a, in, i, mode);
3594       if((r & 255) != ((r >> 8) & 255) || (g & 255) != ((g >> 8) & 255) ||
3595          (b & 255) != ((b >> 8) & 255) || (a & 255) != ((a >> 8) & 255)) /*firs
3596       {
3597         sixteen = 1;
3598         break;
3599       }
3600     }
3601   }
3602
3603   if(sixteen)
3604   {
3605     unsigned short r = 0, g = 0, b = 0, a = 0;
3606     profile->bits = 16;
3607     bits_done = numcolors_done = 1; /*counting colors no longer useful, palette
```

```
3608
3609     for(i = 0; i != numpixels; ++i)
3610     {
3611       getPixelColorRGBA16(&r, &g, &b, &a, in, i, mode);
3612
3613       if(!colored_done && (r != g || r != b))
3614       {
3615         profile->colored = 1;
3616         colored_done = 1;
3617       }
3618
3619       if(!alpha_done)
3620       {
3621         unsigned matchkey = (r == profile->key_r && g == profile->key_g && b ==
    key_b);
3622         if(a != 65535 && (a != 0 || (profile->key && !matchkey)))
3623         {
3624           profile->alpha = 1;
3625           profile->key = 0;
3626           alpha_done = 1;
3627         }
3628         else if(a == 0 && !profile->alpha && !profile->key)
3629         {
3630           profile->key = 1;
3631           profile->key_r = r;
3632           profile->key_g = g;
3633           profile->key_b = b;
3634         }
3635         else if(a == 65535 && profile->key && matchkey)
3636         {
3637           /* Color key cannot be used if an opaque pixel also has that RGB colo
```

```
3638           profile->alpha = 1;
3639           profile->key = 0;
3640           alpha_done = 1;
3641         }
3642       }
3643       if(alpha_done && numcolors_done && colored_done && bits_done) break;
3644     }
3645
3646     if(profile->key && !profile->alpha)
3647     {
3648       for(i = 0; i != numpixels; ++i)
3649       {
3650         getPixelColorRGBA16(&r, &g, &b, &a, in, i, mode);
3651         if(a != 0 && r == profile->key_r && g == profile->key_g && b == profile
    key_b)
3652         {
3653           /* Color key cannot be used if an opaque pixel also has that RGB colo
3654           profile->alpha = 1;
3655           profile->key = 0;
3656           alpha_done = 1;
3657         }
3658       }
3659     }
3660   }
3661   else /* < 16-bit */
3662   {
3663     unsigned char r = 0, g = 0, b = 0, a = 0;
3664     for(i = 0; i != numpixels; ++i)
3665     {
3666       getPixelColorRGBA8(&r, &g, &b, &a, in, i, mode);
3667
```

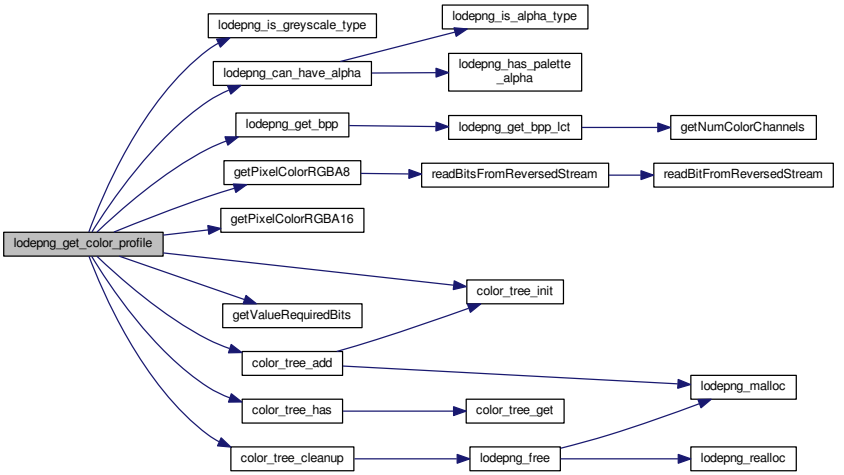
```
3668       if(!bits_done && profile->bits < 8)
3669       {
3670         /*only r is checked, < 8 bits is only relevant for greyscale*/
3671         unsigned bits = getValueRequiredBits(r);
3672         if(bits > profile->bits) profile->bits = bits;
3673       }
3674       bits_done = (profile->bits >= bpp);
3675 
3676       if(!colored_done && (r != g || r != b))
3677       {
3678         profile->colored = 1;
3679         colored_done = 1;
3680         if(profile->bits < 8) profile->bits = 8; /*PNG has no colored modes wit
      per channel*/
3681       }
3682 
3683       if(!alpha_done)
3684       {
3685         unsigned matchkey = (r == profile->key_r && g == profile->key_g && b ==
     key_b);
3686         if(a != 255 && (a != 0 || (profile->key && !matchkey)))
3687         {
3688           profile->alpha = 1;
3689           profile->key = 0;
3690           alpha_done = 1;
3691           if(profile->bits < 8) profile->bits = 8; /*PNG has no alphachannel mo
      8-bit per channel*/
3692         }
3693         else if(a == 0 && !profile->alpha && !profile->key)
3694         {
3695           profile->key = 1;
```

```
3696           profile->key_r = r;
3697           profile->key_g = g;
3698           profile->key_b = b;
3699         }
3700         else if(a == 255 && profile->key && matchkey)
3701         {
3702           /* Color key cannot be used if an opaque pixel also has that RGB colo
3703           profile->alpha = 1;
3704           profile->key = 0;
3705           alpha_done = 1;
3706           if(profile->bits < 8) profile->bits = 8; /*PNG has no alphachannel mo
      8-bit per channel*/
3707         }
3708       }
3709
3710       if(!numcolors_done)
3711       {
3712         if(!color_tree_has(&tree, r, g, b, a))
3713         {
3714           color_tree_add(&tree, r, g, b, a, profile->numcolors);
3715           if(profile->numcolors < 256)
3716           {
3717             unsigned char* p = profile->palette;
3718             unsigned n = profile->numcolors;
3719             p[n * 4 + 0] = r;
3720             p[n * 4 + 1] = g;
3721             p[n * 4 + 2] = b;
3722             p[n * 4 + 3] = a;
3723           }
3724           ++profile->numcolors;
3725           numcolors_done = profile->numcolors >= maxnumcolors;
```

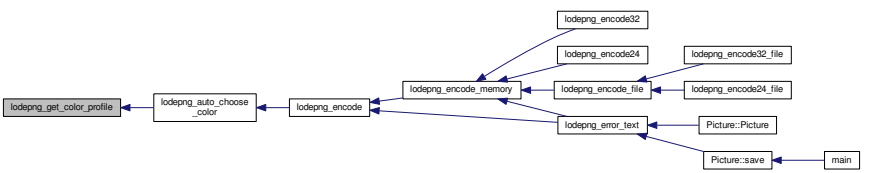
```
3726         }
3727       }
3728
3729       if(alpha_done && numcolors_done && colored_done && bits_done) break;
3730     }
3731
3732     if(profile->key && !profile->alpha)
3733     {
3734       for(i = 0; i != numpixels; ++i)
3735       {
3736         getPixelColorRGBA8(&r, &g, &b, &a, in, i, mode);
3737         if(a != 0 && r == profile->key_r && g == profile->key_g && b == profile
      key_b)
3738         {
3739           /* Color key cannot be used if an opaque pixel also has that RGB colo
3740           profile->alpha = 1;
3741           profile->key = 0;
3742           alpha_done = 1;
3743           if(profile->bits < 8) profile->bits = 8; /*PNG has no alphachannel mo
       8-bit per channel*/
3744         }
3745       }
3746     }
3747
3748     /*make the profile's key always 16-bit for consistency - repeat each byte t
3749     profile->key_r += (profile->key_r << 8);
3750     profile->key_g += (profile->key_g << 8);
3751     profile->key_b += (profile->key_b << 8);
3752   }
3753
3754   color_tree_cleanup(&tree);
```

```
3755   return error;
3756 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.50 lodepng_get_raw_size()

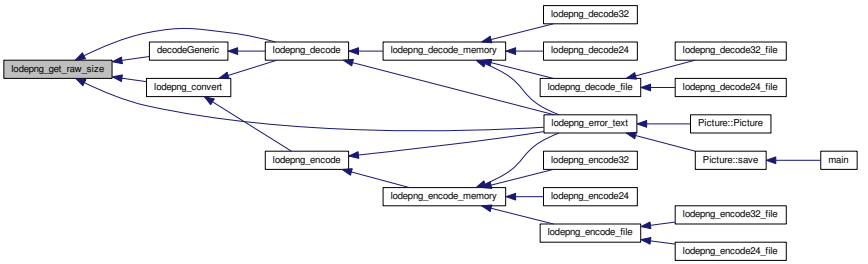
```
size_t lodepng_get_raw_size (
            unsigned w,
            unsigned h,
            const LodePNGColorMode * color )
```

Definition at line 2708 of file lodepng.cpp.

```
2709 {
2710   /*will not overflow for any color type if roughly w * h < 268435455*/
2711   size_t bpp = lodepng_get_bpp(color);
2712   size_t n = w * h;
2713   return ((n / 8) * bpp) + ((n & 7) * bpp + 7) / 8;
2714 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



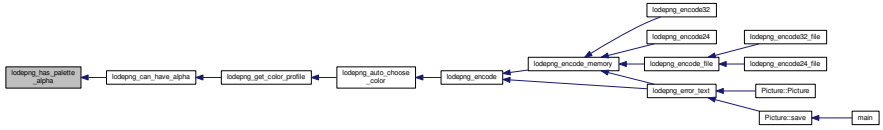
4.2.4.51 lodepng_has_palette_alpha()

```
unsigned lodepng_has_palette_alpha (
            const LodePNGColorMode * info )
```

Definition at line 2691 of file lodepng.cpp.

```
2692 {
2693   size_t i;
2694   for(i = 0; i != info->palettesize; ++i)
2695   {
2696     if(info->palette[i * 4 + 3] < 255) return 1;
2697   }
2698   return 0;
2699 }
```

Here is the caller graph for this function:



4.2.4.52 lodepng_huffman_code_lengths()

```
unsigned lodepng_huffman_code_lengths (
            unsigned * lengths,
            const unsigned * frequencies,
            size_t numcodes,
            unsigned maxbitlen )
```

Definition at line 789 of file lodepng.cpp.

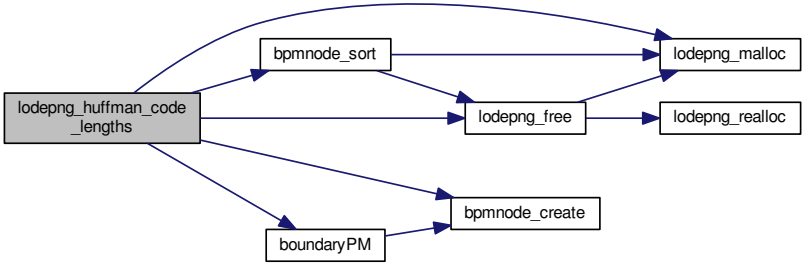
```
791 {
792   unsigned error = 0;
793   unsigned i;
794   size_t numpresent = 0; /*number of symbols with non-zero frequency*/
795   BPMNode* leaves; /*the symbols, only those with > 0 frequency*/
796
797   if(numcodes == 0) return 80; /*error: a tree of 0 symbols is not supposed to b
798   if((1u << maxbitlen) < numcodes) return 80; /*error: represent all symbols*/
799
800   leaves = (BPMNode*)lodepng_malloc(numcodes * sizeof(*leaves));
801   if(!leaves) return 83; /*alloc fail*/
```

```
802
803   for(i = 0; i != numcodes; ++i)
804   {
805     if(frequencies[i] > 0)
806     {
807       leaves[numpresent].weight = (int)frequencies[i];
808       leaves[numpresent].index = i;
809       ++numpresent;
810     }
811   }
812
813   for(i = 0; i != numcodes; ++i) lengths[i] = 0;
814
815   /*ensure at least two present symbols. There should be at least one symbol
816   according to RFC 1951 section 3.2.7. Some decoders incorrectly require two. To
817   make these work as well ensure there are at least two symbols. The
818   Package-Merge code below also doesn't work correctly if there's only one
819   symbol, it'd give it the theoritical 0 bits but in practice zlib wants 1 bit*/
820   if(numpresent == 0)
821   {
822     lengths[0] = lengths[1] = 1; /*note that for RFC 1951 section 3.2.7, only le
823   }
824   else if(numpresent == 1)
825   {
826     lengths[leaves[0].index] = 1;
827     lengths[leaves[0].index == 0 ? 1 : 0] = 1;
828   }
829   else
830   {
831     BPMLists lists;
832     BPMNode* node;
```

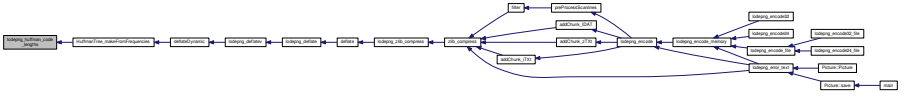
```
833
834     bpmnode_sort(leaves, numpresent);
835
836     lists.listsize = maxbitlen;
837     lists.memsize = 2 * maxbitlen * (maxbitlen + 1);
838     lists.nextfree = 0;
839     lists.numfree = lists.memsize;
840     lists.memory = (BPMNode*)lodepng_malloc(lists.
    memsize * sizeof(*lists.memory));
841     lists.freelist = (BPMNode**)lodepng_malloc(lists.
    memsize * sizeof(BPMNode*));
842     lists.chains0 = (BPMNode**)lodepng_malloc(lists.
    listsize * sizeof(BPMNode*));
843     lists.chains1 = (BPMNode**)lodepng_malloc(lists.
    listsize * sizeof(BPMNode*));
844     if(!lists.memory || !lists.freelist || !lists.chains0 || !lists.
    chains1) error = 83; /*alloc fail*/
845
846     if(!error)
847     {
848       for(i = 0; i != lists.memsize; ++i) lists.freelist[i] = &lists.
    memory[i];
849
850       bpmnode_create(&lists, leaves[0].weight, 1, 0);
851       bpmnode_create(&lists, leaves[1].weight, 2, 0);
852
853       for(i = 0; i != lists.listsize; ++i)
854       {
855         lists.chains0[i] = &lists.memory[0];
856         lists.chains1[i] = &lists.memory[1];
857       }
```

```
858
859         /*each boundaryPM call adds one chain to the last list, and we need 2 * nu
860         for(i = 2; i != 2 * numpresent - 2; ++i) boundaryPM(&lists, leaves, numpre
      maxbitlen - 1, (int)i);
861
862         for(node = lists.chains1[maxbitlen - 1]; node; node = node->tail)
863         {
864           for(i = 0; i != node->index; ++i) ++lengths[leaves[i].index];
865         }
866       }
867
868       lodepng_free(lists.memory);
869       lodepng_free(lists.freelist);
870       lodepng_free(lists.chains0);
871       lodepng_free(lists.chains1);
872     }
873
874     lodepng_free(leaves);
875     return error;
876 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



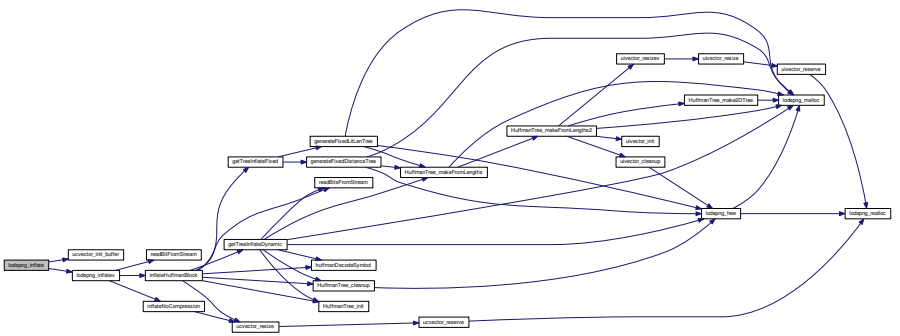
4.2.4.53 lodepng_inflate()

```
unsigned lodepng_inflate (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGDecompressSettings * settings )
```

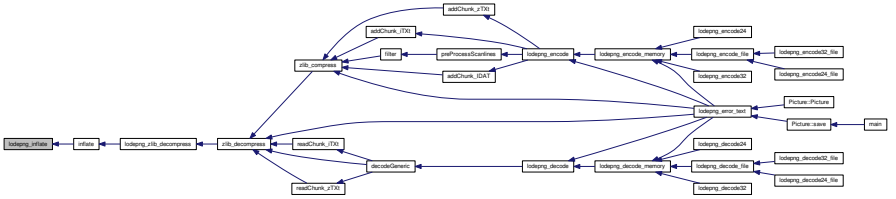
Definition at line 1283 of file lodepng.cpp.

```
1286 {
1287   unsigned error;
1288   ucvector v;
1289   ucvector_init_buffer(&v, *out, *outsize);
1290   error = lodepng_inflatev(&v, in, insize, settings);
1291   *out = v.data;
1292   *outsize = v.size;
1293   return error;
1294 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



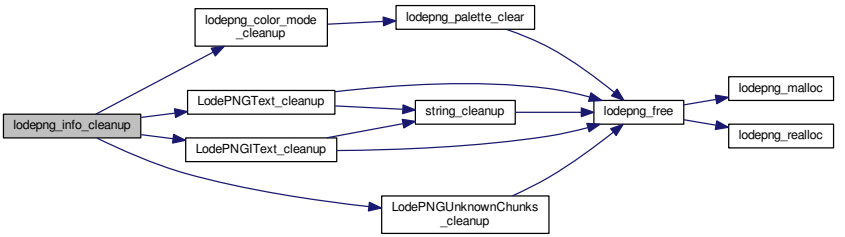
4.2.4.54 lodepng_info_cleanup()

```
void lodepng_info_cleanup (
            LodePNGInfo * info )
```

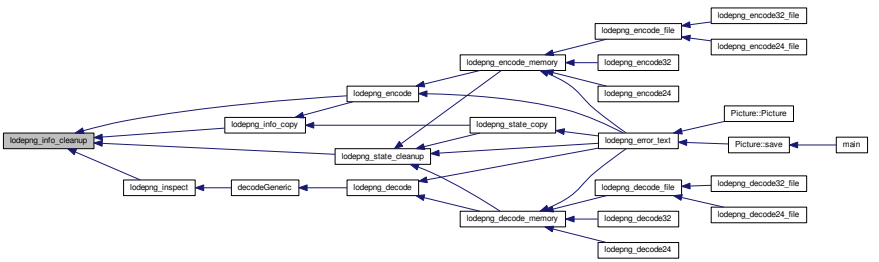
Definition at line 2943 of file lodepng.cpp.

```
2944 {
2945   lodepng_color_mode_cleanup(&info->color);
2946 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
2947   LodePNGText_cleanup(info);
2948   LodePNGIText_cleanup(info);
2949
2950   LodePNGUnknownChunks_cleanup(info);
2951 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
2952 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



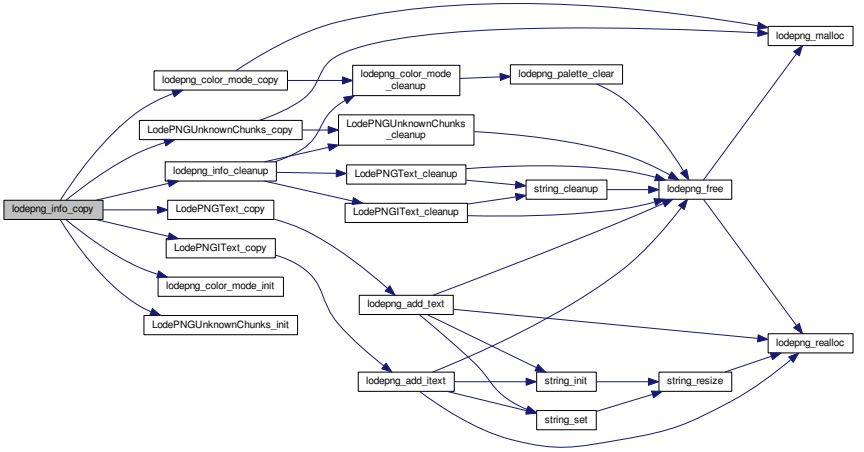
4.2.4.55 lodepng_info_copy()

```
unsigned lodepng_info_copy (
            LodePNGInfo * dest,
            const LodePNGInfo * source )
```

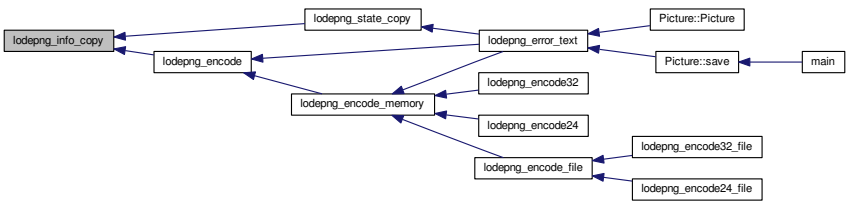
Definition at line 2954 of file lodepng.cpp.

```
2955 {
2956   lodepng_info_cleanup(dest);
2957   *dest = *source;
2958   lodepng_color_mode_init(&dest->color);
2959   CERROR_TRY_RETURN(lodepng_color_mode_copy(&dest->
      color, &source->color));
2960
2961 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
2962   CERROR_TRY_RETURN(LodePNGText_copy(dest, source));
2963   CERROR_TRY_RETURN(LodePNGIText_copy(dest, source));
2964
2965   LodePNGUnknownChunks_init(dest);
2966   CERROR_TRY_RETURN(LodePNGUnknownChunks_copy(dest, source));
2967 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
2968   return 0;
2969 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



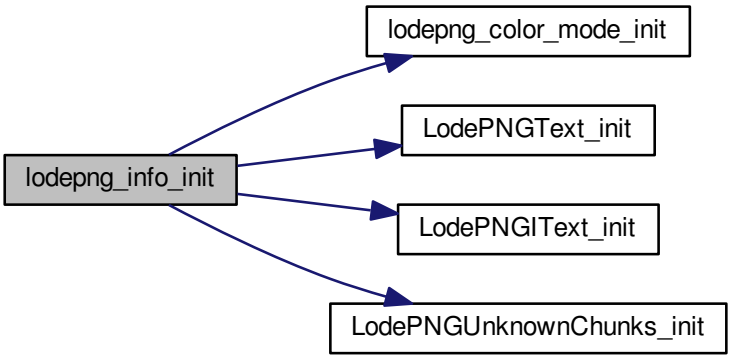
4.2.4.56 lodepng_info_init()

```
void lodepng_info_init (
            LodePNGInfo * info )
```

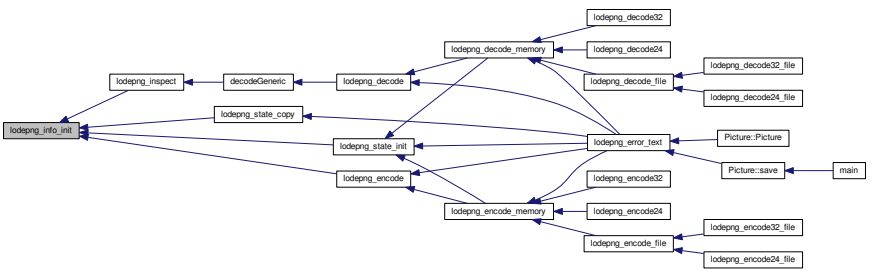
Definition at line 2923 of file lodepng.cpp.

```
2924 {
2925   lodepng_color_mode_init(&info->color);
2926   info->interlace_method = 0;
2927   info->compression_method = 0;
2928   info->filter_method = 0;
2929 #ifdef LODEPNG_COMPILE_ANCILLARY_CHUNKS
2930   info->background_defined = 0;
2931   info->background_r = info->background_g = info->
      background_b = 0;
2932
2933   LodePNGText_init(info);
2934   LodePNGIText_init(info);
2935
2936   info->time_defined = 0;
2937   info->phys_defined = 0;
2938
2939   LodePNGUnknownChunks_init(info);
2940 #endif /*LODEPNG_COMPILE_ANCILLARY_CHUNKS*/
2941 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.57 lodepng_inspect()

```
unsigned lodepng_inspect (
            unsigned * w,
            unsigned * h,
            LodePNGState * state,
            const unsigned char * in,
            size_t insize )
```

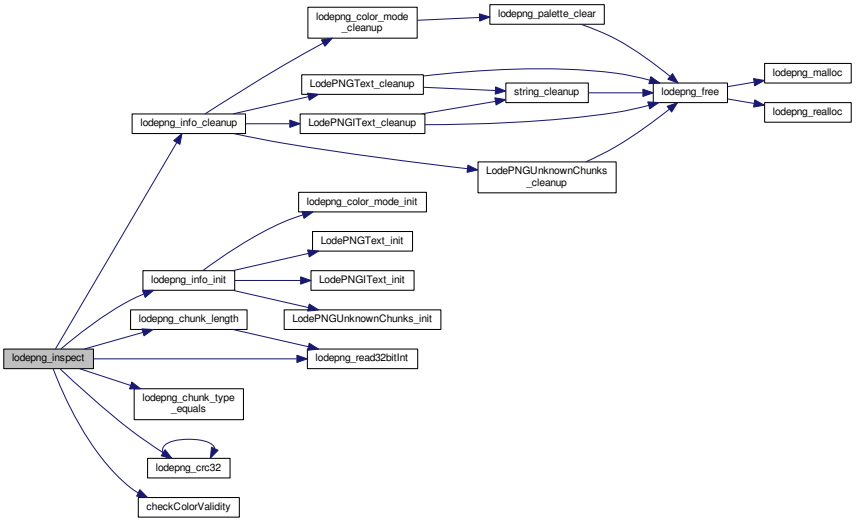
Definition at line 3903 of file lodepng.cpp.

```
3905 {
3906   LodePNGInfo* info = &state->info_png;
3907   if(insize == 0 || in == 0)
3908   {
3909     CERROR_RETURN_ERROR(state->error, 48); /*error: the given data is empty*/
3910   }
3911   if(insize < 33)
3912   {
3913     CERROR_RETURN_ERROR(state->error, 27); /*error: the data length is smaller
      than the length of a PNG header*/
3914   }
3915
3916   /*when decoding a new PNG image, make sure all parameters created after previ
3917   lodepng_info_cleanup(info);
3918   lodepng_info_init(info);
3919
3920   if(in[0] != 137 || in[1] != 80 || in[2] != 78 || in[3] != 71
3921      || in[4] != 13 || in[5] != 10 || in[6] != 26 || in[7] != 10)
3922   {
```

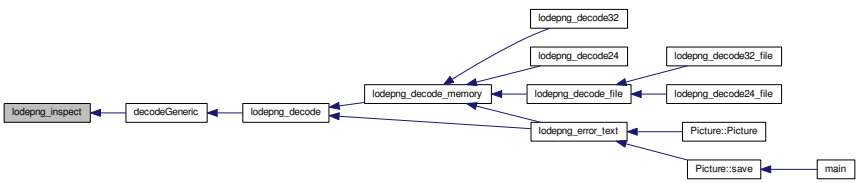
```
3923     CERROR_RETURN_ERROR(state->error, 28); /*error: the first 8 bytes are not t
       correct PNG signature*/
3924   }
3925   if(lodepng_chunk_length(in + 8) != 13)
3926   {
3927     CERROR_RETURN_ERROR(state->error, 94); /*error: header size must be 13 byte
3928   }
3929   if(!lodepng_chunk_type_equals(in + 8, "IHDR"))
3930   {
3931     CERROR_RETURN_ERROR(state->error, 29); /*error: it doesn't start with a IHD
       chunk!*/
3932   }
3933
3934   /*read the values given in the header*/
3935   *w = lodepng_read32bitInt(&in[16]);
3936   *h = lodepng_read32bitInt(&in[20]);
3937   info->color.bitdepth = in[24];
3938   info->color.colortype = (LodePNGColorType)in[25];
3939   info->compression_method = in[26];
3940   info->filter_method = in[27];
3941   info->interlace_method = in[28];
3942
3943   if(*w == 0 || *h == 0)
3944   {
3945     CERROR_RETURN_ERROR(state->error, 93);
3946   }
3947
3948   if(!state->decoder.ignore_crc)
3949   {
3950     unsigned CRC = lodepng_read32bitInt(&in[29]);
3951     unsigned checksum = lodepng_crc32(&in[12], 17);
```

```
3952     if(CRC != checksum)
3953     {
3954       CERROR_RETURN_ERROR(state->error, 57); /*invalid CRC*/
3955     }
3956   }
3957
3958   /*error: only compression method 0 is allowed in the specification*/
3959   if(info->compression_method != 0) CERROR_RETURN_ERROR(state->
      error, 32);
3960   /*error: only filter method 0 is allowed in the specification*/
3961   if(info->filter_method != 0) CERROR_RETURN_ERROR(state->
      error, 33);
3962   /*error: only interlace methods 0 and 1 exist in the specification*/
3963   if(info->interlace_method > 1) CERROR_RETURN_ERROR(state->
      error, 34);
3964
3965   state->error = checkColorValidity(info->color.
      colortype, info->color.bitdepth);
3966   return state->error;
3967 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



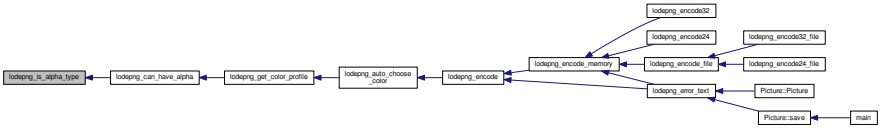
4.2.4.58 lodepng_is_alpha_type()

```
unsigned lodepng_is_alpha_type (
            const LodePNGColorMode * info )
```

Definition at line 2681 of file lodepng.cpp.

```
2682 {
2683   return (info->colortype & 4) != 0; /*4 or 6*/
2684 }
```

Here is the caller graph for this function:



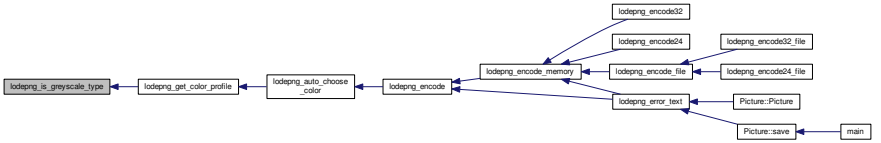
4.2.4.59 lodepng_is_greyscale_type()

```
unsigned lodepng_is_greyscale_type (
            const LodePNGColorMode * info )
```

Definition at line 2676 of file lodepng.cpp.

```
2677 {
2678   return info->colortype == LCT_GREY || info->colortype ==
      LCT_GREY_ALPHA;
2679 }
```

Here is the caller graph for this function:



4.2.4.60 lodepng_is_palette_type()

```
unsigned lodepng_is_palette_type (
            const LodePNGColorMode * info )
```

Definition at line 2686 of file lodepng.cpp.

```
2687 {
2688   return info->colortype == LCT_PALETTE;
2689 }
```

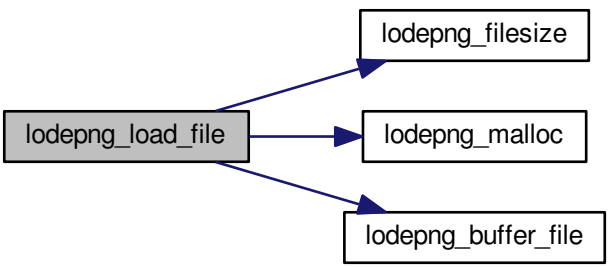
4.2.4.61 lodepng_load_file()

```
unsigned lodepng_load_file (
            unsigned char ** out,
            size_t * outsize,
            const char * filename )
```

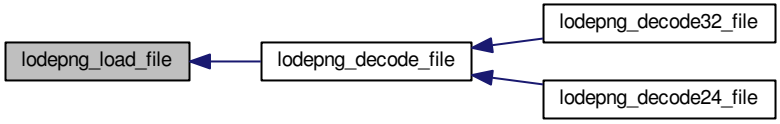
Definition at line 387 of file lodepng.cpp.

```
388 {
389   long size = lodepng_filesize(filename);
390   if (size < 0) return 78;
391   *outsize = (size_t)size;
392
393   *out = (unsigned char*)lodepng_malloc((size_t)size);
394   if(!(*out) && size > 0) return 83; /*the above malloc failed*/
395
396   return lodepng_buffer_file(*out, (size_t)size, filename);
397 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.62 lodepng_palette_add()

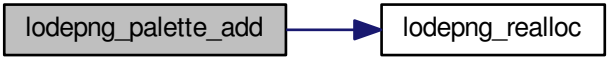
```
unsigned lodepng_palette_add (
            LodePNGColorMode * info,
            unsigned char r,
            unsigned char g,
            unsigned char b,
            unsigned char a )
```

Definition at line 2644 of file lodepng.cpp.

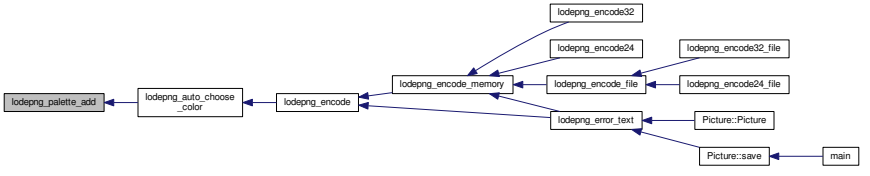
```
2646 {
2647   unsigned char* data;
2648   /*the same resize technique as C++ std::vectors is used, and here it's made s
2649   the max of 256 colors, it'll have the exact alloc size*/
2650   if(!info->palette) /*allocate palette if empty*/
2651   {
2652     /*room for 256 colors with 4 bytes each*/
```

```
2653     data = (unsigned char*)lodepng_realloc(info->palette, 1024);
2654     if(!data) return 83; /*alloc fail*/
2655     else info->palette = data;
2656   }
2657   info->palette[4 * info->palettesize + 0] = r;
2658   info->palette[4 * info->palettesize + 1] = g;
2659   info->palette[4 * info->palettesize + 2] = b;
2660   info->palette[4 * info->palettesize + 3] = a;
2661   ++info->palettesize;
2662   return 0;
2663 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



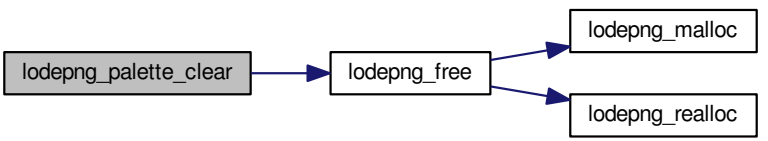
4.2.4.63 lodepng_palette_clear()

```
void lodepng_palette_clear (
            LodePNGColorMode * info )
```

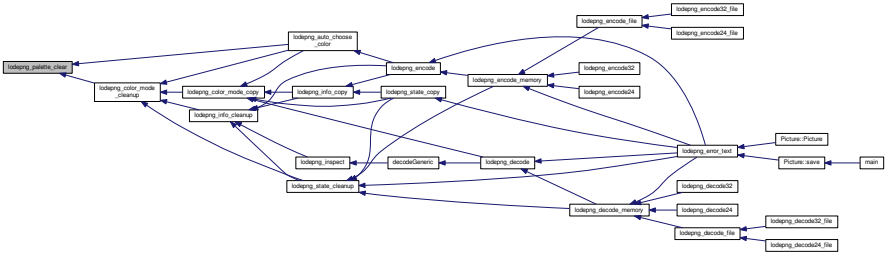
Definition at line 2637 of file lodepng.cpp.

```
2638 {
2639   if(info->palette) lodepng_free(info->palette);
2640   info->palette = 0;
2641   info->palettesize = 0;
2642 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



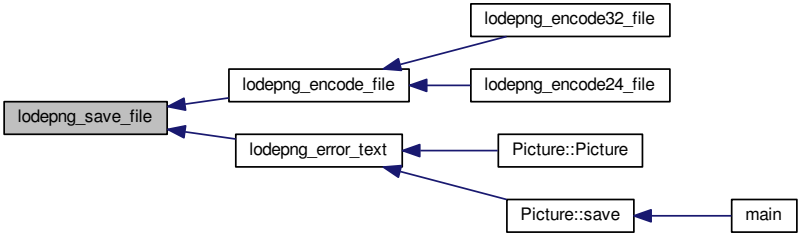
4.2.4.64 lodepng_save_file()

```
unsigned lodepng_save_file (
            const unsigned char * buffer,
            size_t buffersize,
            const char * filename )
```

Definition at line 400 of file lodepng.cpp.

```
401 {
402   FILE* file;
403   file = fopen(filename, "wb" );
404   if(!file) return 79;
405   fwrite((char*)buffer , 1 , buffersize, file);
406   fclose(file);
407   return 0;
408 }
```

Here is the caller graph for this function:



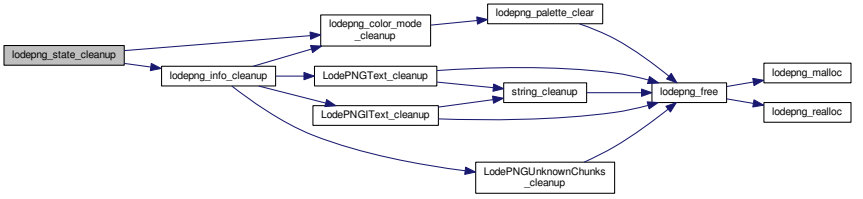
4.2.4.65 lodepng_state_cleanup()

```
void lodepng_state_cleanup (
            LodePNGState * state )
```

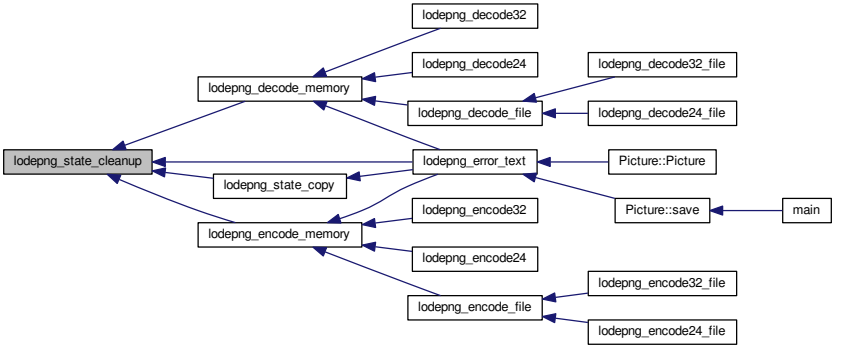
Definition at line 4843 of file lodepng.cpp.

```
4844 {
4845   lodepng_color_mode_cleanup(&state->info_raw);
4846   lodepng_info_cleanup(&state->info_png);
4847 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



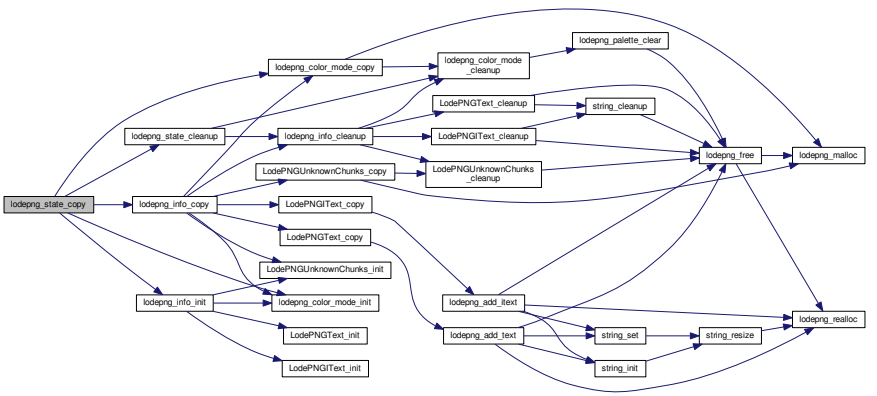
4.2.4.66 lodepng_state_copy()

```
void lodepng_state_copy (
            LodePNGState * dest,
            const LodePNGState * source )
```

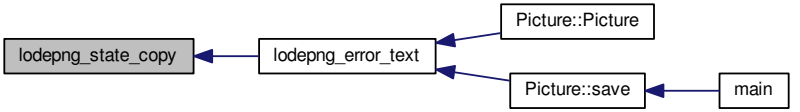
Definition at line 4849 of file lodepng.cpp.

```
4850 {
4851   lodepng_state_cleanup(dest);
4852   *dest = *source;
4853   lodepng_color_mode_init(&dest->info_raw);
4854   lodepng_info_init(&dest->info_png);
4855   dest->error = lodepng_color_mode_copy(&dest->
      info_raw, &source->info_raw); if(dest->error) return;
4856   dest->error = lodepng_info_copy(&dest->info_png, &source->
      info_png); if(dest->error) return;
4857 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



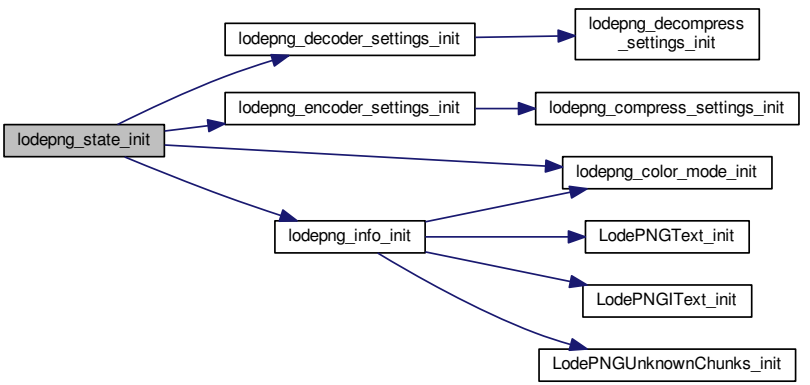
4.2.4.67 lodepng_state_init()

```
void lodepng_state_init (
            LodePNGState * state )
```

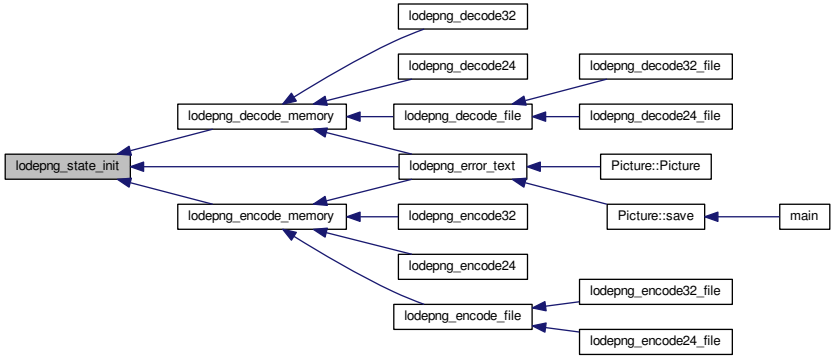
Definition at line 4830 of file lodepng.cpp.

```
4831 {
4832 #ifdef LODEPNG_COMPILE_DECODER
4833   lodepng_decoder_settings_init(&state->decoder);
4834 #endif /*LODEPNG_COMPILE_DECODER*/
4835 #ifdef LODEPNG_COMPILE_ENCODER
4836   lodepng_encoder_settings_init(&state->encoder);
4837 #endif /*LODEPNG_COMPILE_ENCODER*/
4838   lodepng_color_mode_init(&state->info_raw);
4839   lodepng_info_init(&state->info_png);
4840   state->error = 1;
4841 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.68 lodepng_zlib_compress()

```
unsigned lodepng_zlib_compress (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGCompressSettings * settings )
```

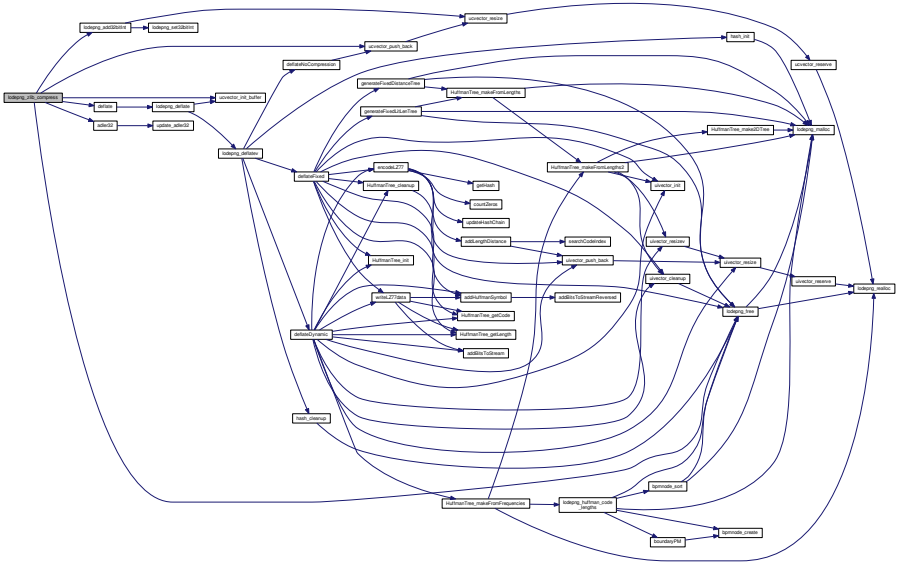
Definition at line 2188 of file lodepng.cpp.

```
2190 {
2191   /*initially, *out must be NULL and outsize 0, if you just give some random *o
```

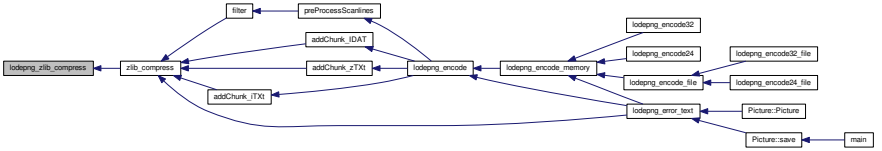
```
2192   that's pointing to a non allocated buffer, this'll crash*/
2193   ucvector outv;
2194   size_t i;
2195   unsigned error;
2196   unsigned char* deflatedata = 0;
2197   size_t deflatesize = 0;
2198
2199   /*zlib data: 1 byte CMF (CM+CINFO), 1 byte FLG, deflate data, 4 byte ADLER32
       data*/
2200   unsigned CMF = 120; /*0b01111000: CM 8, CINFO 7. With CINFO 7, any window siz
2201   unsigned FLEVEL = 0;
2202   unsigned FDICT = 0;
2203   unsigned CMFFLG = 256 * CMF + FDICT * 32 + FLEVEL * 64;
2204   unsigned FCHECK = 31 - CMFFLG % 31;
2205   CMFFLG += FCHECK;
2206
2207   /*ucvector-controlled version of the output buffer, for dynamic array*/
2208   ucvector_init_buffer(&outv, *out, *outsize);
2209
2210   ucvector_push_back(&outv, (unsigned char)(CMFFLG >> 8));
2211   ucvector_push_back(&outv, (unsigned char)(CMFFLG & 255));
2212
2213   error = deflate(&deflatedata, &deflatesize, in, insize, settings);
2214
2215   if(!error)
2216   {
2217     unsigned ADLER32 = adler32(in, (unsigned)insize);
2218     for(i = 0; i != deflatesize; ++i) ucvector_push_back(&outv, deflatedata[i])
2219     lodepng_free(deflatedata);
2220     lodepng_add32bitInt(&outv, ADLER32);
2221   }
```

```
2222
2223   *out = outv.data;
2224   *outsize = outv.size;
2225
2226   return error;
2227 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.69 lodepng_zlib_decompress()

```
unsigned lodepng_zlib_decompress (
            unsigned char ** out,
            size_t * outsize,
            const unsigned char * in,
            size_t insize,
            const LodePNGDecompressSettings * settings )
```

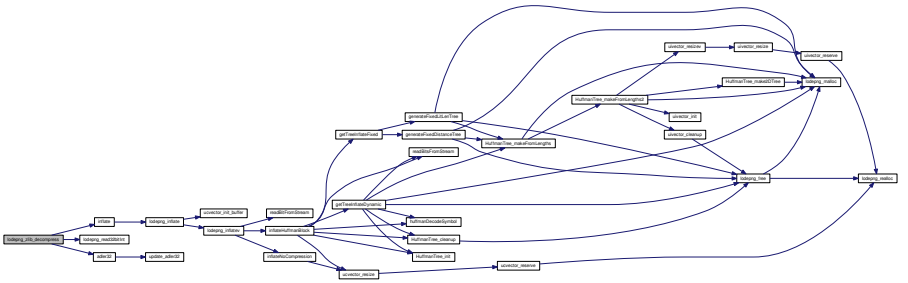
Definition at line 2126 of file lodepng.cpp.

```
2128 {
2129   unsigned error = 0;
2130   unsigned CM, CINFO, FDICT;
2131
2132   if(insize < 2) return 53; /*error, size of zlib data too small*/
2133   /*read information from zlib header*/
2134   if((in[0] * 256 + in[1]) % 31 != 0)
2135   {
2136     /*error: 256 * in[0] + in[1] must be a multiple of 31, the FCHECK value is
```

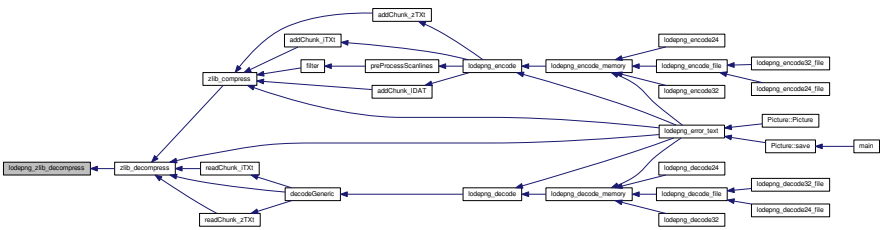
```
    */
2137    return 24;
2138  }
2139
2140  CM = in[0] & 15;
2141  CINFO = (in[0] >> 4) & 15;
2142  /*FCHECK = in[1] & 31;*/ /*FCHECK is already tested above*/
2143  FDICT = (in[1] >> 5) & 1;
2144  /*FLEVEL = (in[1] >> 6) & 3;*/ /*FLEVEL is not used here*/
2145
2146  if(CM != 8 || CINFO > 7)
2147  {
2148    /*error: only compression method 8: inflate with sliding window of 32k is s
2149    return 25;
2150  }
2151  if(FDICT != 0)
2152  {
2153    /*error: the specification of PNG says about the zlib stream:
2154      "The additional flags shall not specify a preset dictionary."*/
2155    return 26;
2156  }
2157
2158  error = inflate(out, outsize, in + 2, insize - 2, settings);
2159  if(error) return error;
2160
2161  if(!settings->ignore_adler32)
2162  {
2163    unsigned ADLER32 = lodepng_read32bitInt(&in[insize - 4]);
2164    unsigned checksum = adler32(*out, (unsigned)(*outsize));
2165    if(checksum != ADLER32) return 58; /*error, adler checksum not correct, dat
2166  }
```

```
2167
2168   return 0; /*no error*/
2169 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.5 Variable Documentation

4.2.5.1 lodepng_default_compress_settings

```
const LodePNGCompressSettings lodepng_default_compress_settings
```

Definition at line 2288 of file lodepng.cpp.

4.2.5.2 lodepng_default_decompress_settings

```
const LodePNGDecompressSettings lodepng_default_decompress_settings
```

Definition at line 2304 of file lodepng.cpp.

4.2.5.3 LODEPNG_VERSION_STRING

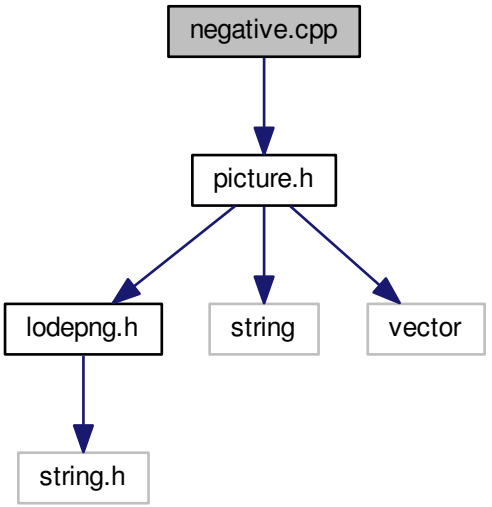
```
const char* LODEPNG_VERSION_STRING
```

Definition at line 42 of file lodepng.cpp.

4.3 negative.cpp File Reference

```
#include "picture.h"
```

Include dependency graph for negative.cpp:



Functions

- int main ()

4.3.1 Detailed Description

Convert an image into its negative: For each column For each pixel in the column Process the pixel by subtracting each of its rgb values from 255

Figure 1 Before image:

Figure 2 After image:

4.3.2 Function Documentation

4.3.2.1 main()

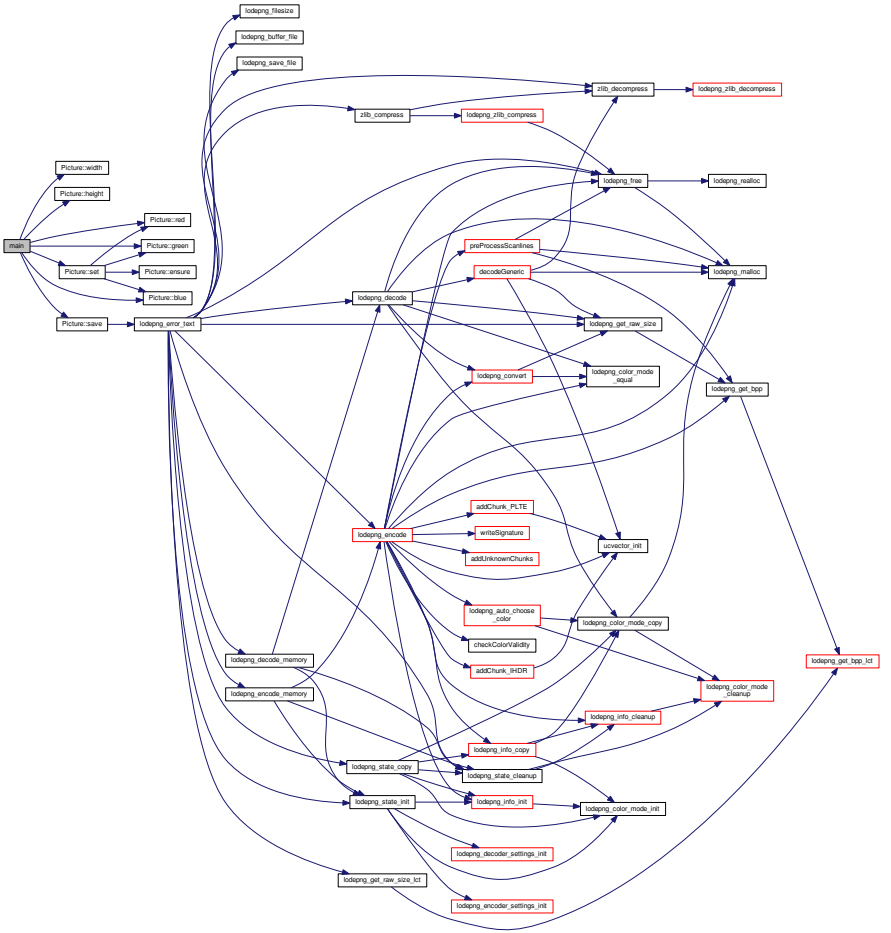
```
int main ( )
```

Definition at line 13 of file negative.cpp.

```
14 {
15     Picture pic("queen-mary.png");
16
17     for (int x = 0; x < pic.width(); x++)
18     {
```

```
19         for (int y = 0; y < pic.height(); y++)
20         {
21             int red = pic.red(x, y);
22             int green = pic.green(x, y);
23             int blue = pic.blue(x, y);
24             pic.set(x, y, 255 - red, 255 - green, 255 - blue);
25         }
26     }
27     pic.save("out.png");
28     return 0;
29 }
```

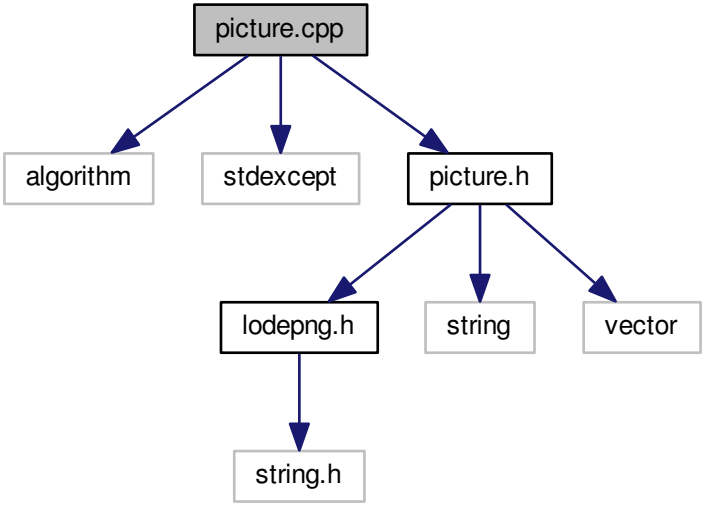
Here is the call graph for this function:



4.4 picture.cpp File Reference

```
#include <algorithm>
#include <stdexcept>
#include "picture.h"
```

Include dependency graph for picture.cpp:

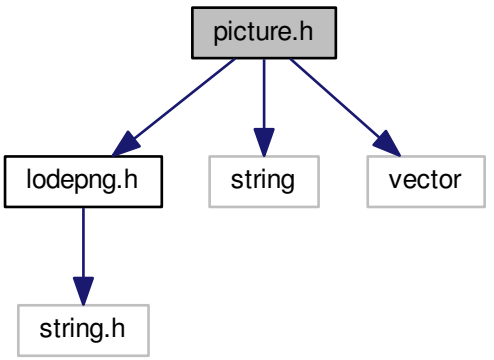


4.5 picture.h File Reference

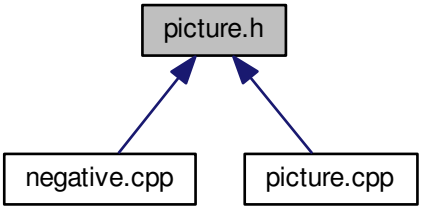
```
#include "lodepng.h"
#include <string>
```

```
#include <vector>
```

Include dependency graph for picture.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Picture

Index