

Intel® Intelligent Storage Acceleration Library (Intel® ISA-L)

API Reference Manual - Version 2.28.0

September 17, 2019

1 Intel(R) Intelligent Storage Acceleration Library	1
2 Contributing to ISA-L	3
3 v2.28 Intel Intelligent Storage Acceleration Library Release Notes	5
4 Instruction Set Requirements for arch-specific functions (non-multibinary)	11
5 Data Structure Index	13
5.1 Data Structures	13
6 File Index	15
6.1 File List	15
7 Data Structure Documentation	17
7.1 BitBuf2 Struct Reference	17
7.1.1 Detailed Description	17
7.2 inflate_huff_code_large Struct Reference	18
7.3 inflate_huff_code_small Struct Reference	18
7.4 inflate_state Struct Reference	18
7.4.1 Detailed Description	19
7.5 isal_gzip_header Struct Reference	19
7.6 isal_huff_histogram Struct Reference	20
7.6.1 Detailed Description	21
7.7 isal_hufftables Struct Reference	21
7.7.1 Detailed Description	21
7.8 isal_mod_hist Struct Reference	22
7.9 isal_zlib_header Struct Reference	22
7.10 isal_zstate Struct Reference	22
7.10.1 Detailed Description	23
7.11 isal_zstream Struct Reference	23
7.11.1 Detailed Description	24
8 File Documentation	25
8.1 crc.h File Reference	25
8.1.1 Detailed Description	26
8.1.2 Function Documentation	26
8.1.2.1 crc16_t10dif()	26
8.1.2.2 crc16_t10dif_base()	26
8.1.2.3 crc16_t10dif_copy()	27
8.1.2.4 crc16_t10dif_copy_base()	27
8.1.2.5 crc32_gzip_refl()	28

8.1.2.6	crc32_gzip_refl_base()	28
8.1.2.7	crc32_ieee()	29
8.1.2.8	crc32_ieee_base()	29
8.1.2.9	crc32_iscsi()	30
8.1.2.10	crc32_iscsi_base()	30
8.2	crc64.h File Reference	31
8.2.1	Detailed Description	32
8.2.2	Function Documentation	32
8.2.2.1	crc64_ecma_norm()	32
8.2.2.2	crc64_ecma_norm_base()	33
8.2.2.3	crc64_ecma_norm_by8()	33
8.2.2.4	crc64_ecma_refl()	34
8.2.2.5	crc64_ecma_refl_base()	34
8.2.2.6	crc64_ecma_refl_by8()	35
8.2.2.7	crc64_iso_norm()	35
8.2.2.8	crc64_iso_norm_base()	36
8.2.2.9	crc64_iso_norm_by8()	36
8.2.2.10	crc64_iso_refl()	37
8.2.2.11	crc64_iso_refl_base()	37
8.2.2.12	crc64_iso_refl_by8()	38
8.2.2.13	crc64_jones_norm()	38
8.2.2.14	crc64_jones_norm_base()	39
8.2.2.15	crc64_jones_norm_by8()	39
8.2.2.16	crc64_jones_refl()	40
8.2.2.17	crc64_jones_refl_base()	40
8.2.2.18	crc64_jones_refl_by8()	41
8.3	erasure_code.h File Reference	41
8.3.1	Detailed Description	42
8.3.2	Function Documentation	42
8.3.2.1	ec_encode_data()	42
8.3.2.2	ec_encode_data_base()	43
8.3.2.3	ec_encode_data_update()	43
8.3.2.4	ec_encode_data_update_base()	44
8.3.2.5	ec_init_tables()	44
8.3.2.6	gf_gen_cauchy1_matrix()	45
8.3.2.7	gf_gen_rs_matrix()	45
8.3.2.8	gf_inv()	46
8.3.2.9	gf_invert_matrix()	47
8.3.2.10	gf_mul()	47

8.3.2.11 gf_vect_dot_prod()	47
8.3.2.12 gf_vect_dot_prod_base()	48
8.3.2.13 gf_vect_mad()	49
8.3.2.14 gf_vect_mad_base()	49
8.4 gf_vect_mul.h File Reference	50
8.4.1 Detailed Description	50
8.4.2 Function Documentation	50
8.4.2.1 gf_vect_mul()	50
8.4.2.2 gf_vect_mul_base()	51
8.4.2.3 gf_vect_mul_init()	51
8.5 igzip_lib.h File Reference	52
8.5.1 Detailed Description	54
8.5.2 Enumeration Type Documentation	54
8.5.2.1 isal_zstate_state	54
8.5.3 Function Documentation	55
8.5.3.1 isal_adler32()	55
8.5.3.2 isal_create_hufftables()	56
8.5.3.3 isal_create_hufftables_subset()	56
8.5.3.4 isal_deflate()	57
8.5.3.5 isal_deflate_init()	58
8.5.3.6 isal_deflate_reset()	58
8.5.3.7 isal_deflate_set_dict()	58
8.5.3.8 isal_deflate_set_hufftables()	59
8.5.3.9 isal_deflate_stateless()	59
8.5.3.10 isal_deflate_stateless_init()	60
8.5.3.11 isal_gzip_header_init()	60
8.5.3.12 isal_inflate()	62
8.5.3.13 isal_inflate_init()	63
8.5.3.14 isal_inflate_reset()	63
8.5.3.15 isal_inflate_set_dict()	63
8.5.3.16 isal_inflate_stateless()	64
8.5.3.17 isal_read_gzip_header()	64
8.5.3.18 isal_read_zlib_header()	65
8.5.3.19 isal_update_histogram()	65
8.5.3.20 isal_write_gzip_header()	66
8.5.3.21 isal_write_zlib_header()	66
8.6 isa-l.h File Reference	67
8.6.1 Detailed Description	67
8.7 mem_routines.h File Reference	67

8.7.1 Detailed Description	67
8.7.2 Function Documentation	68
8.7.2.1 isal_zero_detect()	68
8.8 raid.h File Reference	68
8.8.1 Detailed Description	69
8.8.2 Function Documentation	69
8.8.2.1 pq_check()	69
8.8.2.2 pq_check_base()	69
8.8.2.3 pq_gen()	70
8.8.2.4 pq_gen_base()	70
8.8.2.5 xor_check()	71
8.8.2.6 xor_check_base()	71
8.8.2.7 xor_gen()	72
8.8.2.8 xor_gen_base()	72
Index	75

Chapter 1

Intel(R) Intelligent Storage Acceleration Library

ISA-L is a collection of optimized low-level functions targeting storage applications. ISA-L includes:

- Erasure codes - Fast block Reed-Solomon type erasure codes for any encode/decode matrix in $GF(2^8)$.
- CRC - Fast implementations of cyclic redundancy check. Six different polynomials supported.
 - iscsi32, ieee32, t10dif, ecma64, iso64, jones64.
- Raid - calculate and operate on XOR and P+Q parity found in common RAID implementations.
- Compression - Fast deflate-compatible data compression.
- De-compression - Fast inflate-compatible data compression.

Also see:

- [ISA-L for updates](#).
- For crypto functions see [isa-l_crypto on github](#).
- The [github wiki](#) including a list of [distros/ports](#) offering binary packages.
- ISA-L [mailing list](#).
- [Contributing](#).

Building ISA-L

Prerequisites

- Assembler: nasm v2.11.01 or later (nasm v2.13 or better suggested for building in AVX512 support) or yasm version 1.2.0 or later.
- Compiler: gcc, clang, icc or VC compiler.
- Make: GNU 'make' or 'nmake' (Windows).
- Optional: Building with autotools requires autoconf/automake packages.

Autotools

To build and install the library with autotools it is usually sufficient to run:

```
./autogen.sh
./configure
make
sudo make install
```

Makefile

To use a standard makefile run:

```
make -f Makefile.unx
```

Windows

On Windows use nmake to build dll and static lib:

```
nmake -f Makefile.nmake
```

Other make targets

Other targets include:

- `make check` : create and run tests
- `make tests` : create additional unit tests
- `make perfs` : create included performance tests
- `make ex` : build examples
- `make other` : build other utilities such as compression file tests
- `make doc` : build API manual

Chapter 2

Contributing to ISA-L

Everyone is welcome to contribute. Patches may be submitted using GitHub pull requests (PRs). All commits must be signed off by the developer (`--signoff`) which indicates that you agree to the Developer Certificate of Origin. Patch discussion will happen directly on the GitHub PR. Design pre-work and general discussion occurs on the [mailing list](#). Anyone can provide feedback in either location and all discussion is welcome. Decisions on whether to merge patches will be handled by the maintainer.

License

ISA-L is licensed using a BSD 3-clause [license]. All code submitted to the project is required to carry that license.

Certificate of Origin

In order to get a clear contribution chain of trust we use the [signed-off-by language](#) used by the Linux kernel project.

Mailing List

Contributors and users are welcome to submit new request on our roadmap, submit patches, file issues, and ask questions on our [mailing list](#).

Coding Style

The coding style for ISA-L C code roughly follows linux kernel guidelines. Use the included indent script to format C code.

```
./tools/iindent your_files.c
```

And use check format script before submitting.

```
./tools/check_format.sh
```


Chapter 3

v2.28 Intel Intelligent Storage Acceleration Library Release Notes

RELEASE NOTE CONTENTS

1. KNOWN ISSUES
2. FIXED ISSUES
3. CHANGE LOG & FEATURES ADDED

1. KNOWN ISSUES

- Perf tests do not run in Windows environment.
- 32-bit lib is not supported in Windows.

2. FIXED ISSUES

v2.28

- Fix documentation on [gf_vect_mad\(\)](#). Min length listed as 32 instead of required min 64 bytes.

v2.27

- Fix lack of install for pkg-config files

v2.26

- Fixes for sanitizer warnings.

v2.25

- Fix for nasm on Mac OS X/darwin.

v2.24

- Fix for `crc32_iscsi()`. Potential read-over for small buffer. For an input buffer length of less than 8 bytes and aligned to an 8 byte boundary, function could read past length. Previously had the possibility to cause a seg fault only for length 0 and invalid buffer passed. Calculated CRC is unchanged.
- Fix for compression/decompression of > 4GB files. For streaming compression of extremely large files, the `total_out` parameter would wrap and could potentially flag an otherwise valid lookback distance as being invalid. `Total_out` is still 32bit for zlib compatibility. No inconsistent compressed buffers were generated by the issue.

v2.23

- Fix for histogram generation base function.
- Fix library build warnings on macOS.
- Fix `igzip` to use `bsf` instruction when `tzcnt` is not available.

v2.22

- Fix ISA-L builds for other architectures. Base function and examples sanitized for non-IA builds.
- Fix fuzz test script to work with llvm 6.0 builtin `libFuzz`.

v2.20

- Inflate `total_out` behavior corrected for in-progress decompression. Previously `total_out` represented the total bytes decompressed into the output buffer or temp internal buffer. This is changed to be only the bytes put into the output buffer.
- Fixed issue with `isal_create_hufftables_subset`. Affects semi-dynamic compression use case when explicitly creating hufftables from histogram. The `_hufftables_subset` function could fail to generate length symbols for any length that were never seen.

v2.19

- Fix erasure code test that violates rs matrix bounds.
- Fix 0 length file and looping errors in `igzip_inflate_test`.

v2.18

-
- Mac OS X/darwin systems no longer require the `-target=darwin` config option. The autoconf canonical build should detect.

v2.17

- Fix igzip using 32K window and a shared object
- Fix igzip undefined instruction error on Nehalem.
- Fixed issue in crc performance tests where OS optimizations turned cold cache tests into warm tests.

v2.15

- Fix for windows register save in `gf_6vect_mad_avx2.asm`. Only affects windows versions of `ec_encode_data_update()` running with AVX2. A GP register was not properly restored resulting in corruption on return.

v2.14

- Building in unit directories is no longer supported removing the issue of leftover object files causing the top-level make build to fail.

v2.10

- Fix for windows register save overlap in `gf_{3-6}vect_dot_prod_sse.asm`. Only affects windows versions of erasure code. GP register saves/restore were pushed to same stack area as XMM.

3. CHANGE LOG & FEATURES ADDED

v2.28

- New next-arch versions of 64-bit CRC. All norm and reflected 64-bit polynomials are expanded to utilize `vp_cmulqdq`.

v2.27

- New multi-threaded compression option for igzip cli tool

v2.26

- Adler32 added to external API.
- Multi-arch improvements.
- Performance test improvements.

v2.25

- Igzip performance improvements and features.
 - Performance improvements for uncompressable files. Random or uncompressable files can be up to 3x faster in level 1 or 2 compression.
 - Additional small file performance improvements.
 - New options in igzip cli: use name from header or not, test compressed file.
- Multi-arch autoconf script.
 - Autoconf should detect architecture and run base functions at minimum.

v2.24

- Igzip small file performance improvements and new features.
 - Better performance on small files.
 - New gzip/zlib header and trailer handling.
 - New gzip/zlib header parsing helper functions.
 - New user-space compression/decompression tool igzip.
- New mem unit added with first function [isal_zero_detect\(\)](#).

v2.23

- Igzip inflate (decompression) performance improvements.
 - Implemented multi-byte decode for inflate. Decode can pack up to three symbols into the decode table making some compressed streams decompress much faster depending on the prevalence of short codes.

v2.22

- Igzip: AVX2 version of level 3 compression added.
- Erasure code examples
 - New examples for standard EC encode and decode.
 - Example of piggyback EC encode and decode.

v2.21

- Igzip improvements
 - New compression levels added. ISA-L fast deflate now has more levels to balance speed vs. target compression level. Level 0, 1 are as in previous generations. New levels 2 & 3 target higher compression roughly comparable to zlib levels 2-3. Level 3 is currently only optimized for processors with AVX512 instructions.
- New T10dif & copy function - [crc16_t10dif_copy\(\)](#)

-
- CRC and copy was added to emulate T10dif operations such as DIF insert and strip. This function stitches together CRC and memcpy operations eliminating an extra data read.
 - CRC32 iscsi performance improvements
 - Fixes issue under some distributions where warm cache performance was reduced.

v2.20

- lgzip improvements
 - Optimized deflate_hash in compression functions. Improves performance of using preset dictionary.
 - Removed alignment restrictions on input structure.

v2.19

- lgzip improvements
 - Add optimized Adler-32 checksum.
 - Implement zlib compression format.
 - Add stateful dictionary support.
 - Add struct reset functions for both deflate and inflate.
- Reflected IEEE format CRC32 is released out. Function interface is named crc32_gzip_refl.
- Exact work condition of Erasure Code Reed-Solomon Matrix is determined by new added program gen_rs_↔ matrix_limits.

v2.18

- New 2-pass fully-dynamic deflate compression (level -1). ISA-L fast deflate now has two levels. Level 0 (default) is the same as previous generations. Setting to level 1 will switch to the fully-dynamic compression that will typically reach higher compression ratios.
- RAID AVX512 functions.

v2.17

- New fast decompression (inflate)
- Compression improvements (deflate)
 - Speed and compression ratio improvements.
 - Fast custom Huffman code generation.
 - New features:
 - * Run-time option of gzip crc calculation and headers/trailer.
 - * Choice of static header (BTYP 01) blocks.
 - * LARGE_WINDOW, 32K history, now default.
 - * Stateless full flush mode.

- CRC64
 - Six new 64-bit polynomials supported. Normal and reflected versions of ECMA, ISO and Jones polynomials.

v2.16

- Units added: crc, raid, igzip (deflate compression).

v2.15

- Erasure code updates. New AVX512 versions.
- Nasm support. ISA-L ported to build with nasm or yasm assembler.
- Windows DLL support. Windows builds DLL by default.

v2.14

- Autoconf and autotools build allows easier porting to additional systems. Previous make system still available to embedded users with Makefile.unx.
- Includes update for building on Mac OS X/darwin systems. Add `–target=darwin` to `./configure` step.

v2.13

- Erasure code improvements
 - 32-bit port of optimized `gf_vect_dot_prod()` functions. This makes `ec_encode_data()` functions much faster on 32-bit processors.
 - Avoton performance improvements. Performance on Avoton for `gf_vect_dot_prod()` and `ec_encode_data()` can improve by as much as 20%.

v2.11

- Incremental erasure code. New functions added to erasure code to handle single source update of code blocks. The function `ec_encode_data_update()` works with parameters similar to `ec_encode_data()` but are called incrementally with each source block. These versions are useful when source blocks are not all available at once.

v2.10

- Erasure code updates
 - New AVX and AVX2 support functions.
 - Changes min len requirement on `gf_vect_dot_prod()` to 32 from 16.
 - Tests include both source and parity recovery with `ec_encode_data()`.
 - New encoding examples with Vandermonde or Cauchy matrix.

v2.8

- First open release of erasure code unit that is part of ISA-L.

Chapter 4

Instruction Set Requirements for arch-specific functions (non-multibinary)

Global [crc64_ecma_norm_by8](#) (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

Global [crc64_ecma_refl_by8](#) (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

Global [crc64_iso_norm_by8](#) (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

Global [crc64_iso_refl_by8](#) (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

Global [crc64_jones_norm_by8](#) (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

Global [crc64_jones_refl_by8](#) (uint64_t init_crc, const unsigned char *buf, uint64_t len)
SSE3, CLMUL

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

BitBuf2		
	Holds Bit Buffer information	17
inflate_huff_code_large	18
inflate_huff_code_small	18
inflate_state		
	Holds decompression state information	18
isal_gzip_header	19
isal_huff_histogram		
	Holds histogram of deflate symbols	20
isal_hufftables		
	Holds the huffman tree used to huffman encode the input stream	21
isal_mod_hist	22
isal_zlib_header	22
isal_zstate		
	Holds the internal state information for input and output compression streams	22
isal_zstream		
	Holds stream information	23

Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

aarch64_multibinary.h	??
crc.h	CRC functions	25
crc64.h	CRC64 functions	31
erasure_code.h	Interface to functions supporting erasure code encode and decode	41
gf_vect_mul.h	Interface to functions for vector (block) multiplication in GF(2 ⁸)	50
igzip_lib.h	This file defines the igzip compression and decompression interface, a high performance deflate compression interface for storage applications	52
isa-l.h	Include for ISA-L library	67
mem_routines.h	Interface to storage mem operations	67
raid.h	Interface to RAID functions - XOR and P+Q calculation	68
unaligned.h	??

Chapter 7

Data Structure Documentation

7.1 BitBuf2 Struct Reference

Holds Bit Buffer information.

```
#include <igzip_lib.h>
```

Data Fields

- [uint64_t m_bits](#)
bits in the bit buffer
- [uint32_t m_bit_count](#)
number of valid bits in the bit buffer
- [uint8_t * m_out_buf](#)
current index of buffer to write to
- [uint8_t * m_out_end](#)
end of buffer to write to
- [uint8_t * m_out_start](#)
start of buffer to write to

7.1.1 Detailed Description

Holds Bit Buffer information.

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

- [igzip_lib.h](#)

7.2 inflate_huff_code_large Struct Reference

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

- [igzip_lib.h](#)

7.3 inflate_huff_code_small Struct Reference

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

- [igzip_lib.h](#)

7.4 inflate_state Struct Reference

Holds decompression state information.

```
#include <igzip_lib.h>
```

Data Fields

- `uint8_t * next_out`
Next output Byte.
- `uint32_t avail_out`
Number of bytes available at next_out.
- `uint32_t total_out`
Total bytes written out so far.
- `uint8_t * next_in`
Next input byte.
- `uint64_t read_in`
Bits buffered to handle unaligned streams.
- `uint32_t avail_in`
Number of bytes available at next_in.
- `int32_t read_in_length`
Bits in read_in.
- `struct inflate_huff_code_large lit_huff_code`
Structure for decoding lit/len symbols.
- `struct inflate_huff_code_small dist_huff_code`
Structure for decoding dist symbols.
- `enum isal_block_state block_state`
Current decompression state.
- `uint32_t dict_length`
Length of dictionary used.

- `uint32_t bfinal`
Flag identifying final block.
- `uint32_t crc_flag`
Flag identifying whether to track of crc.
- `uint32_t crc`
Contains crc or Adler32 of output if `crc_flag` is set.
- `uint32_t hist_bits`
Log base 2 of maximum lookback distance.
- `int32_t copy_overflow_length`
Length left to copy when outbuffer overflow occurred.
- `int32_t copy_overflow_distance`
Lookback distance when outbuffer overflow occurred.
- `int16_t tmp_in_size`
Number of bytes in `tmp_in_buffer`.
- `int32_t tmp_out_valid`
Number of bytes in `tmp_out_buffer`.
- `int32_t tmp_out_processed`
Number of bytes processed in `tmp_out_buffer`.
- `uint8_t tmp_in_buffer` [ISAL_DEF_MAX_HDR_SIZE]
Temporary buffer containing data from the input stream.
- `uint8_t tmp_out_buffer` [2 * ISAL_DEF_HIST_SIZE + ISAL_LOOK_AHEAD]
Temporary buffer containing data from the output stream.
- `int32_t type0_block_len`
Length left to read of type 0 block when outbuffer overflow occurred.
- `int32_t count`
Count of bytes remaining to be parsed.

7.4.1 Detailed Description

Holds decompression state information.

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

- [igzip_lib.h](#)

7.5 isal_gzip_header Struct Reference

Data Fields

- `uint32_t text`
Optional Text hint.
- `uint32_t time`
Unix modification time in gzip header.
- `uint32_t xflags`

- xflags in gzip header*
- `uint32_t os`
 - OS in gzip header.*
- `uint8_t * extra`
 - Extra field in gzip header.*
- `uint32_t extra_buf_len`
 - Length of extra buffer.*
- `uint32_t extra_len`
 - Actual length of gzip header extra field.*
- `char * name`
 - Name in gzip header.*
- `uint32_t name_buf_len`
 - Length of name buffer.*
- `char * comment`
 - Comments in gzip header.*
- `uint32_t comment_buf_len`
 - Length of comment buffer.*
- `uint32_t hcrc`
 - Header crc or header crc flag.*
- `uint32_t flags`
 - Internal data.*

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

- [igzip_lib.h](#)

7.6 isal_huff_histogram Struct Reference

Holds histogram of deflate symbols.

```
#include <igzip_lib.h>
```

Data Fields

- `uint64_t lit_len_histogram` [ISAL_DEF_LIT_LEN_SYMBOLS]
 - Histogram of Literal/Len symbols seen.*
- `uint64_t dist_histogram` [ISAL_DEF_DIST_SYMBOLS]
 - Histogram of Distance Symbols seen.*
- `uint16_t hash_table` [IGZIP_LVL0_HASH_SIZE]
 - Tmp space used as a hash table.*

7.6.1 Detailed Description

Holds histogram of deflate symbols.

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

- [igzip_lib.h](#)

7.7 isal_hufftables Struct Reference

Holds the huffman tree used to huffman encode the input stream.

```
#include <igzip_lib.h>
```

Data Fields

- `uint8_t deflate_hdr` [ISAL_DEF_MAX_HDR_SIZE]
deflate huffman tree header
- `uint32_t deflate_hdr_count`
Number of whole bytes in deflate_huff_hdr.
- `uint32_t deflate_hdr_extra_bits`
Number of bits in the partial byte in header.
- `uint32_t dist_table` [IGZIP_DIST_TABLE_SIZE]
bits 4:0 are the code length, bits 31:5 are the code
- `uint32_t len_table` [IGZIP_LEN_TABLE_SIZE]
bits 4:0 are the code length, bits 31:5 are the code
- `uint16_t lit_table` [IGZIP_LIT_TABLE_SIZE]
literal code
- `uint8_t lit_table_sizes` [IGZIP_LIT_TABLE_SIZE]
literal code length
- `uint16_t dcodes` [30 - IGZIP_DECODE_OFFSET]
distance code
- `uint8_t dcodes_sizes` [30 - IGZIP_DECODE_OFFSET]
distance code length

7.7.1 Detailed Description

Holds the huffman tree used to huffman encode the input stream.

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

- [igzip_lib.h](#)

7.8 isal_mod_hist Struct Reference

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

- [igzip_lib.h](#)

7.9 isal_zlib_header Struct Reference

Data Fields

- [uint32_t info](#)
base-2 logarithm of the LZ77 window size minus 8
- [uint32_t level](#)
Compression level (fastest, fast, default, maximum)
- [uint32_t dict_id](#)
Dictionary id.
- [uint32_t dict_flag](#)
Whether to use a dictionary.

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

- [igzip_lib.h](#)

7.10 isal_zstate Struct Reference

Holds the internal state information for input and output compression streams.

```
#include <igzip_lib.h>
```

Data Fields

- [uint32_t total_in_start](#)
Not used, may be replaced with something else.
- [uint32_t block_next](#)
Start of current deflate block in the input.
- [uint32_t block_end](#)
End of current deflate block in the input.
- [uint32_t dist_mask](#)
Distance mask used.
- [enum isal_zstate_state state](#)
Current state in processing the data stream.
- [struct BitBuf2 bitbuf](#)

- Bit Buffer.*
- `uint32_t crc`
 - Current checksum without finalize step if any (adler)*
- `uint8_t has_wrap_hdr`
 - keeps track of wrapper header*
- `uint8_t has_eob_hdr`
 - keeps track of eob_hdr (with BFINAL set)*
- `uint8_t has_eob`
 - keeps track of eob on the last deflate block*
- `uint8_t has_hist`
 - flag to track if there is match history*
- `uint16_t has_level_buf_init`
 - flag to track if user supplied memory has been initialized.*
- `uint32_t count`
 - used for partial header/trailer writes*
- `uint8_t tmp_out_buff [16]`
 - temporary array*
- `uint32_t tmp_out_start`
 - temporary variable*
- `uint32_t tmp_out_end`
 - temporary variable*
- `uint32_t b_bytes_valid`
 - number of valid bytes in buffer*
- `uint32_t b_bytes_processed`
 - number of bytes processed in buffer*
- `uint8_t buffer [2 *IGZIP_HIST_SIZE+ISAL_LOOK_AHEAD]`
 - Internal buffer.*
- `uint16_t head [IGZIP_LVL0_HASH_SIZE]`
 - Hash array.*

7.10.1 Detailed Description

Holds the internal state information for input and output compression streams.

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

- [igzip_lib.h](#)

7.11 isal_zstream Struct Reference

Holds stream information.

```
#include <igzip_lib.h>
```

Data Fields

- `uint8_t * next_in`
Next input byte.
- `uint32_t avail_in`
number of bytes available at next_in
- `uint32_t total_in`
total number of bytes read so far
- `uint8_t * next_out`
Next output byte.
- `uint32_t avail_out`
number of bytes available at next_out
- `uint32_t total_out`
total number of bytes written so far
- `struct isal_hufftables * hufftables`
Huffman encoding used when compressing.
- `uint32_t level`
Compression level to use.
- `uint32_t level_buf_size`
Size of level_buf.
- `uint8_t * level_buf`
User allocated buffer required for different compression levels.
- `uint16_t end_of_stream`
non-zero if this is the last input buffer
- `uint16_t flush`
Flush type can be NO_FLUSH, SYNC_FLUSH or FULL_FLUSH.
- `uint16_t gzip_flag`
Indicate if gzip compression is to be performed.
- `uint16_t hist_bits`
Log base 2 of maximum lookback distance, 0 is use default.
- `struct isal_zstate internal_state`
Internal state for this stream.

7.11.1 Detailed Description

Holds stream information.

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

- [igzip_lib.h](#)

Chapter 8

File Documentation

8.1 crc.h File Reference

CRC functions.

```
#include <stdint.h>
```

Functions

- `uint16_t crc16_t10dif` (`uint16_t init_crc`, `const unsigned char *buf`, `uint64_t len`)
Generate CRC from the T10 standard, runs appropriate version.
- `uint16_t crc16_t10dif_copy` (`uint16_t init_crc`, `uint8_t *dst`, `uint8_t *src`, `uint64_t len`)
Generate CRC and copy T10 standard, runs appropriate version.
- `uint32_t crc32_ieee` (`uint32_t init_crc`, `const unsigned char *buf`, `uint64_t len`)
Generate CRC from the IEEE standard, runs appropriate version.
- `uint32_t crc32_gzip_refl` (`uint32_t init_crc`, `const unsigned char *buf`, `uint64_t len`)
Generate the customized CRC based on RFC 1952 CRC (<http://www.ietf.org/rfc/rfc1952.txt>) standard, runs appropriate version.
- `unsigned int crc32_iscsi` (`unsigned char *buffer`, `int len`, `unsigned int init_crc`)
ISCSI CRC function, runs appropriate version.
- `unsigned int crc32_iscsi_base` (`unsigned char *buffer`, `int len`, `unsigned int crc_init`)
ISCSI CRC function, baseline version.
- `uint16_t crc16_t10dif_base` (`uint16_t seed`, `uint8_t *buf`, `uint64_t len`)
Generate CRC from the T10 standard, runs baseline version.
- `uint16_t crc16_t10dif_copy_base` (`uint16_t init_crc`, `uint8_t *dst`, `uint8_t *src`, `uint64_t len`)
Generate CRC and copy T10 standard, runs baseline version.
- `uint32_t crc32_ieee_base` (`uint32_t seed`, `uint8_t *buf`, `uint64_t len`)
Generate CRC from the IEEE standard, runs baseline version.
- `uint32_t crc32_gzip_refl_base` (`uint32_t seed`, `uint8_t *buf`, `uint64_t len`)
Generate the customized CRC based on RFC 1952 CRC (<http://www.ietf.org/rfc/rfc1952.txt>) standard, runs baseline version.

8.1.1 Detailed Description

CRC functions.

8.1.2 Function Documentation

8.1.2.1 `crc16_t10dif()`

```
uint16_t crc16_t10dif (
    uint16_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from the T10 standard, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

16 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 16 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.1.2.2 `crc16_t10dif_base()`

```
uint16_t crc16_t10dif_base (
    uint16_t seed,
    uint8_t * buf,
    uint64_t len )
```

Generate CRC from the T10 standard, runs baseline version.

Returns

16 bit CRC

Parameters

<i>seed</i>	initial CRC value, 16 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.1.2.3 `crc16_t10dif_copy()`

```
uint16_t crc16_t10dif_copy (
    uint16_t init_crc,
    uint8_t * dst,
    uint8_t * src,
    uint64_t len )
```

Generate CRC and copy T10 standard, runs appropriate version.

Stitched CRC + copy function.

Returns

16 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 16 bits
<i>dst</i>	buffer destination for copy
<i>src</i>	buffer source to crc + copy
<i>len</i>	buffer length in bytes (64-bit data)

8.1.2.4 `crc16_t10dif_copy_base()`

```
uint16_t crc16_t10dif_copy_base (
    uint16_t init_crc,
    uint8_t * dst,
    uint8_t * src,
    uint64_t len )
```

Generate CRC and copy T10 standard, runs baseline version.

Returns

16 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 16 bits
<i>dst</i>	buffer destination for copy
<i>src</i>	buffer source to crc + copy
<i>len</i>	buffer length in bytes (64-bit data)

8.1.2.5 crc32_gzip_refl()

```
uint32_t crc32_gzip_refl (
    uint32_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate the customized CRC based on RFC 1952 CRC (<http://www.ietf.org/rfc/rfc1952.txt>) standard, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Note: CRC32 IEEE standard is widely used in HDLC, Ethernet, Gzip and many others. Its polynomial is 0x04C11DB7 in normal and 0xEDB88320 in reflection (or reverse). In ISA-L CRC, function `crc32_ieee` is actually designed for normal CRC32 IEEE version. And function `crc32_gzip_refl` is actually designed for reflected CRC32 IEEE. These two versions of CRC32 IEEE are not compatible with each other. Users who want to replace their not optimized `crc32_ieee` with ISA-L's `crc32` function should be careful of that. Since many applications use CRC32 IEEE reflected version, Please have a check whether `crc32_gzip_refl` is right one for you instead of `crc32_ieee`.

Returns

32 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 32 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.1.2.6 crc32_gzip_refl_base()

```
uint32_t crc32_gzip_refl_base (
    uint32_t seed,
    uint8_t * buf,
    uint64_t len )
```

Generate the customized CRC based on RFC 1952 CRC (<http://www.ietf.org/rfc/rfc1952.txt>) standard, runs baseline version.

Returns

32 bit CRC

Parameters

<i>seed</i>	initial CRC value, 32 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.1.2.7 `crc32_ieee()`

```
uint32_t crc32_ieee (
    uint32_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from the IEEE standard, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime. Note: CRC32 IEEE standard is widely used in HDLC, Ethernet, Gzip and many others. Its polynomial is 0x04C11DB7 in normal and 0xEDB88320 in reflection (or reverse). In ISA-L CRC, function `crc32_ieee` is actually designed for normal CRC32 IEEE version. And function `crc32_gzip_refl` is actually designed for reflected CRC32 IEEE. These two versions of CRC32 IEEE are not compatible with each other. Users who want to replace their not optimized `crc32_ieee` with ISA-L's `crc32` function should be careful of that. Since many applications use CRC32 IEEE reflected version, Please have a check whether `crc32_gzip_refl` is right one for you instead of `crc32_ieee`.

Returns

32 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 32 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.1.2.8 `crc32_ieee_base()`

```
uint32_t crc32_ieee_base (
```

```

uint32_t seed,
uint8_t * buf,
uint64_t len )

```

Generate CRC from the IEEE standard, runs baseline version.

Returns

32 bit CRC

Parameters

<i>seed</i>	initial CRC value, 32 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.1.2.9 crc32_iscsi()

```

unsigned int crc32_iscsi (
    unsigned char * buffer,
    int len,
    unsigned int init_crc )

```

ISCSI CRC function, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

32 bit CRC

Parameters

<i>buffer</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes
<i>init_crc</i>	initial CRC value

8.1.2.10 crc32_iscsi_base()

```

unsigned int crc32_iscsi_base (
    unsigned char * buffer,

```

```
int len,
unsigned int crc_init )
```

ISCSI CRC function, baseline version.

Returns

32 bit CRC

Parameters

<i>buffer</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes
<i>crc_init</i>	initial CRC value

8.2 crc64.h File Reference

CRC64 functions.

```
#include <stdint.h>
```

Functions

- `uint64_t crc64_ecma_refl` (`uint64_t init_crc, const unsigned char *buf, uint64_t len`)
Generate CRC from ECMA-182 standard in reflected format, runs appropriate version.
- `uint64_t crc64_ecma_norm` (`uint64_t init_crc, const unsigned char *buf, uint64_t len`)
Generate CRC from ECMA-182 standard in normal format, runs appropriate version.
- `uint64_t crc64_iso_refl` (`uint64_t init_crc, const unsigned char *buf, uint64_t len`)
Generate CRC from ISO standard in reflected format, runs appropriate version.
- `uint64_t crc64_iso_norm` (`uint64_t init_crc, const unsigned char *buf, uint64_t len`)
Generate CRC from ISO standard in normal format, runs appropriate version.
- `uint64_t crc64_jones_refl` (`uint64_t init_crc, const unsigned char *buf, uint64_t len`)
Generate CRC from "Jones" coefficients in reflected format, runs appropriate version.
- `uint64_t crc64_jones_norm` (`uint64_t init_crc, const unsigned char *buf, uint64_t len`)
Generate CRC from "Jones" coefficients in normal format, runs appropriate version.
- `uint64_t crc64_ecma_refl_by8` (`uint64_t init_crc, const unsigned char *buf, uint64_t len`)
Generate CRC from ECMA-182 standard in reflected format.
- `uint64_t crc64_ecma_norm_by8` (`uint64_t init_crc, const unsigned char *buf, uint64_t len`)
Generate CRC from ECMA-182 standard in normal format.
- `uint64_t crc64_ecma_refl_base` (`uint64_t init_crc, const unsigned char *buf, uint64_t len`)
Generate CRC from ECMA-182 standard in reflected format, runs baseline version.
- `uint64_t crc64_ecma_norm_base` (`uint64_t init_crc, const unsigned char *buf, uint64_t len`)
Generate CRC from ECMA-182 standard in normal format, runs baseline version.
- `uint64_t crc64_iso_refl_by8` (`uint64_t init_crc, const unsigned char *buf, uint64_t len`)

- Generate CRC from ISO standard in reflected format.*

 - `uint64_t crc64_iso_norm_by8` (`uint64_t init_crc`, `const unsigned char *buf`, `uint64_t len`)

Generate CRC from ISO standard in normal format.

 - `uint64_t crc64_iso_refl_base` (`uint64_t init_crc`, `const unsigned char *buf`, `uint64_t len`)

Generate CRC from ISO standard in reflected format, runs baseline version.

 - `uint64_t crc64_iso_norm_base` (`uint64_t init_crc`, `const unsigned char *buf`, `uint64_t len`)

Generate CRC from ISO standard in normal format, runs baseline version.

 - `uint64_t crc64_jones_refl_by8` (`uint64_t init_crc`, `const unsigned char *buf`, `uint64_t len`)

Generate CRC from "Jones" coefficients in reflected format.

 - `uint64_t crc64_jones_norm_by8` (`uint64_t init_crc`, `const unsigned char *buf`, `uint64_t len`)

Generate CRC from "Jones" coefficients in normal format.

 - `uint64_t crc64_jones_refl_base` (`uint64_t init_crc`, `const unsigned char *buf`, `uint64_t len`)

Generate CRC from "Jones" coefficients in reflected format, runs baseline version.

 - `uint64_t crc64_jones_norm_base` (`uint64_t init_crc`, `const unsigned char *buf`, `uint64_t len`)

Generate CRC from "Jones" coefficients in normal format, runs baseline version.

8.2.1 Detailed Description

CRC64 functions.

8.2.2 Function Documentation

8.2.2.1 `crc64_ecma_norm()`

```
uint64_t crc64_ecma_norm (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ECMA-182 standard in normal format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.2 `crc64_ecma_norm_base()`

```
uint64_t crc64_ecma_norm_base (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ECMA-182 standard in normal format, runs baseline version.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.3 `crc64_ecma_norm_by8()`

```
uint64_t crc64_ecma_norm_by8 (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ECMA-182 standard in normal format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.4 crc64_ecma_refl()

```
uint64_t crc64_ecma_refl (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ECMA-182 standard in reflected format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.5 crc64_ecma_refl_base()

```
uint64_t crc64_ecma_refl_base (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ECMA-182 standard in reflected format, runs baseline version.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.6 crc64_ecma_refl_by8()

```
uint64_t crc64_ecma_refl_by8 (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ECMA-182 standard in reflected format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.7 crc64_iso_norm()

```
uint64_t crc64_iso_norm (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ISO standard in normal format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.8 `crc64_iso_norm_base()`

```
uint64_t crc64_iso_norm_base (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ISO standard in normal format, runs baseline version.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.9 `crc64_iso_norm_by8()`

```
uint64_t crc64_iso_norm_by8 (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ISO standard in normal format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.10 `crc64_iso_refl()`

```
uint64_t crc64_iso_refl (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ISO standard in reflected format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.11 `crc64_iso_refl_base()`

```
uint64_t crc64_iso_refl_base (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ISO standard in reflected format, runs baseline version.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.12 `crc64_iso_refl_by8()`

```
uint64_t crc64_iso_refl_by8 (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from ISO standard in reflected format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.13 `crc64_jones_norm()`

```
uint64_t crc64_jones_norm (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from "Jones" coefficients in normal format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.14 `crc64_jones_norm_base()`

```
uint64_t crc64_jones_norm_base (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from "Jones" coefficients in normal format, runs baseline version.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.15 `crc64_jones_norm_by8()`

```
uint64_t crc64_jones_norm_by8 (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from "Jones" coefficients in normal format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.16 `crc64_jones_refl()`

```
uint64_t crc64_jones_refl (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from "Jones" coefficients in reflected format, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.17 `crc64_jones_refl_base()`

```
uint64_t crc64_jones_refl_base (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from "Jones" coefficients in reflected format, runs baseline version.

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.2.2.18 crc64_jones_refl_by8()

```
uint64_t crc64_jones_refl_by8 (
    uint64_t init_crc,
    const unsigned char * buf,
    uint64_t len )
```

Generate CRC from "Jones" coefficients in reflected format.

Requires SSE3, CLMUL

Returns

64 bit CRC

Parameters

<i>init_crc</i>	initial CRC value, 64 bits
<i>buf</i>	buffer to calculate CRC on
<i>len</i>	buffer length in bytes (64-bit data)

8.3 erasure_code.h File Reference

Interface to functions supporting erasure code encode and decode.

```
#include "gf_vect_mul.h"
```

Functions

- void [ec_init_tables](#) (int k, int rows, unsigned char *a, unsigned char *gftbls)
Initialize tables for fast Erasure Code encode and decode.
- void [ec_encode_data](#) (int len, int k, int rows, unsigned char *gftbls, unsigned char **data, unsigned char **coding)
Generate or decode erasure codes on blocks of data, runs appropriate version.
- void [ec_encode_data_base](#) (int len, int srcs, int dests, unsigned char *v, unsigned char **src, unsigned char **dest)
Generate or decode erasure codes on blocks of data, runs baseline version.
- void [ec_encode_data_update](#) (int len, int k, int rows, int vec_i, unsigned char *g_tbls, unsigned char *data, unsigned char **coding)
Generate update for encode or decode of erasure codes from single source, runs appropriate version.
- void [ec_encode_data_update_base](#) (int len, int k, int rows, int vec_i, unsigned char *v, unsigned char *data, unsigned char **dest)
Generate update for encode or decode of erasure codes from single source.

- void `gf_vect_dot_prod_base` (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char *dest)
GF(2⁸) vector dot product, runs baseline version.
- void `gf_vect_dot_prod` (int len, int vlen, unsigned char *gftbls, unsigned char **src, unsigned char *dest)
GF(2⁸) vector dot product, runs appropriate version.
- void `gf_vect_mad` (int len, int vec, int vec_i, unsigned char *gftbls, unsigned char *src, unsigned char *dest)
GF(2⁸) vector multiply accumulate, runs appropriate version.
- void `gf_vect_mad_base` (int len, int vec, int vec_i, unsigned char *v, unsigned char *src, unsigned char *dest)
GF(2⁸) vector multiply accumulate, baseline version.
- unsigned char `gf_mul` (unsigned char a, unsigned char b)
Single element GF(2⁸) multiply.
- unsigned char `gf_inv` (unsigned char a)
Single element GF(2⁸) inverse.
- void `gf_gen_rs_matrix` (unsigned char *a, int m, int k)
Generate a matrix of coefficients to be used for encoding.
- void `gf_gen_cauchy1_matrix` (unsigned char *a, int m, int k)
Generate a Cauchy matrix of coefficients to be used for encoding.
- int `gf_invert_matrix` (unsigned char *in, unsigned char *out, const int n)
Invert a matrix in GF(2⁸)

8.3.1 Detailed Description

Interface to functions supporting erasure code encode and decode.

This file defines the interface to optimized functions used in erasure codes. Encode and decode of erasures in $G \leftarrow F(2^8)$ are made by calculating the dot product of the symbols (bytes in $GF(2^8)$) across a set of buffers and a set of coefficients. Values for the coefficients are determined by the type of erasure code. Using a general dot product means that any sequence of coefficients may be used including erasure codes based on random coefficients. Multiple versions of dot product are supplied to calculate 1-6 output vectors in one pass. Base GF multiply and divide functions can be sped up by defining `GF_LARGE_TABLES` at the expense of memory size.

8.3.2 Function Documentation

8.3.2.1 `ec_encode_data()`

```
void ec_encode_data (
    int len,
    int k,
    int rows,
    unsigned char * gftbls,
    unsigned char ** data,
    unsigned char ** coding )
```

Generate or decode erasure codes on blocks of data, runs appropriate version.

Given a list of source data blocks, generate one or multiple blocks of encoded data as specified by a matrix of $G \leftarrow F(2^8)$ coefficients. When given a suitable set of coefficients, this function will perform the fast generation or decoding of Reed-Solomon type erasure codes.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

<i>len</i>	Length of each block of data (vector) of source or dest data.
<i>k</i>	The number of vector sources or rows in the generator matrix for coding.
<i>rows</i>	The number of output vectors to concurrently encode/decode.
<i>gftbls</i>	Pointer to array of input tables generated from coding coefficients in ec_init_tables() . Must be of size 32*k*rows
<i>data</i>	Array of pointers to source input buffers.
<i>coding</i>	Array of pointers to coded output buffers.

Returns

none

8.3.2.2 `ec_encode_data_base()`

```
void ec_encode_data_base (
    int len,
    int srcs,
    int dests,
    unsigned char * v,
    unsigned char ** src,
    unsigned char ** dest )
```

Generate or decode erasure codes on blocks of data, runs baseline version.

Baseline version of [ec_encode_data\(\)](#) with same parameters.

8.3.2.3 `ec_encode_data_update()`

```
void ec_encode_data_update (
    int len,
    int k,
    int rows,
    int vec_i,
    unsigned char * g_tbls,
    unsigned char * data,
    unsigned char ** coding )
```

Generate update for encode or decode of erasure codes from single source, runs appropriate version.

Given one source data block, update one or multiple blocks of encoded data as specified by a matrix of GF(2⁸) coefficients. When given a suitable set of coefficients, this function will perform the fast generation or decoding of Reed-Solomon type erasure codes from one input source at a time.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

<i>len</i>	Length of each block of data (vector) of source or dest data.
<i>k</i>	The number of vector sources or rows in the generator matrix for coding.
<i>rows</i>	The number of output vectors to concurrently encode/decode.
<i>vec_i</i>	The vector index corresponding to the single input source.
<i>g_tbls</i>	Pointer to array of input tables generated from coding coefficients in ec_init_tables() . Must be of size 32*k*rows
<i>data</i>	Pointer to single input source used to update output parity.
<i>coding</i>	Array of pointers to coded output buffers.

Returns

none

8.3.2.4 ec_encode_data_update_base()

```
void ec_encode_data_update_base (
    int len,
    int k,
    int rows,
    int vec_i,
    unsigned char * v,
    unsigned char * data,
    unsigned char ** dest )
```

Generate update for encode or decode of erasure codes from single source.

Baseline version of [ec_encode_data_update\(\)](#).

8.3.2.5 ec_init_tables()

```
void ec_init_tables (
    int k,
    int rows,
    unsigned char * a,
    unsigned char * gftbls )
```

Initialize tables for fast Erasure Code encode and decode.

Generates the expanded tables needed for fast encode or decode for erasure codes on blocks of data. 32bytes is generated for each input coefficient.

Parameters

<i>k</i>	The number of vector sources or rows in the generator matrix for coding.
<i>rows</i>	The number of output vectors to concurrently encode/decode.
<i>a</i>	Pointer to sets of arrays of input coefficients used to encode or decode data.
<i>gftbls</i>	Pointer to start of space for concatenated output tables generated from input coefficients. Must be of size $32*k*rows$.

Returns

none

8.3.2.6 gf_gen_cauchy1_matrix()

```
void gf_gen_cauchy1_matrix (
    unsigned char * a,
    int m,
    int k )
```

Generate a Cauchy matrix of coefficients to be used for encoding.

Cauchy matrix example of encoding coefficients where high portion of matrix is identity matrix I and lower portion is constructed as $1/(i + j) \mid i \neq j, i:\{0,k-1\} j:\{k,m-1\}$. Any sub-matrix of a Cauchy matrix should be invertable.

Parameters

<i>a</i>	[$m \times k$] array to hold coefficients
<i>m</i>	number of rows in matrix corresponding to srcs + parity.
<i>k</i>	number of columns in matrix corresponding to srcs.

Returns

none

8.3.2.7 gf_gen_rs_matrix()

```
void gf_gen_rs_matrix (
    unsigned char * a,
    int m,
    int k )
```

Generate a matrix of coefficients to be used for encoding.

Vandermonde matrix example of encoding coefficients where high portion of matrix is identity matrix I and lower portion is constructed as $2^{i*(j-k+1)}$ $i:\{0,k-1\}$ $j:\{k,m-1\}$. Commonly used method for choosing coefficients in erasure encoding but does not guarantee invertable for every sub matrix. For large pairs of m and k it is possible to find cases where the decode matrix chosen from sources and parity is not invertable. Users may want to adjust for certain pairs m and k . If m and k satisfy one of the following inequalities, no adjustment is required:

- $k \leq 3$
- $k = 4, m \leq 25$
- $k = 5, m \leq 10$
- $k \leq 21, m - k = 4$
- $m - k \leq 3$.

Parameters

<i>a</i>	[m x k] array to hold coefficients
<i>m</i>	number of rows in matrix corresponding to srcs + parity.
<i>k</i>	number of columns in matrix corresponding to srcs.

Returns

none

8.3.2.8 gf_inv()

```
unsigned char gf_inv (
    unsigned char a )
```

Single element $GF(2^8)$ inverse.

Parameters

<i>a</i>	Input element
----------	---------------

Returns

Field element b such that $a \times b = \{1\}$

8.3.2.9 gf_invert_matrix()

```
int gf_invert_matrix (
    unsigned char * in,
    unsigned char * out,
    const int n )
```

Invert a matrix in GF(2⁸)

Parameters

<i>in</i>	input matrix
<i>out</i>	output matrix such that [in] x [out] = [I] - identity matrix
<i>n</i>	size of matrix [nxn]

Returns

0 successful, other fail on singular input matrix

8.3.2.10 gf_mul()

```
unsigned char gf_mul (
    unsigned char a,
    unsigned char b )
```

Single element GF(2⁸) multiply.

Parameters

<i>a</i>	Multiplicand a
<i>b</i>	Multiplicand b

Returns

Product of a and b in GF(2⁸)

8.3.2.11 gf_vect_dot_prod()

```
void gf_vect_dot_prod (
    int len,
    int vlen,
```

```

unsigned char * gftbbs,
unsigned char ** src,
unsigned char * dest )

```

GF(2⁸) vector dot product, runs appropriate version.

Does a GF(2⁸) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32*vlen byte constant array based on the input coefficients.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

<i>len</i>	Length of each vector in bytes. Must be ≥ 32 .
<i>vlen</i>	Number of vector sources.
<i>gftbbs</i>	Pointer to 32*vlen byte array of pre-calculated constants based on the array of input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Pointer to destination data array.

Returns

none

8.3.2.12 gf_vect_dot_prod_base()

```

void gf_vect_dot_prod_base (
    int len,
    int vlen,
    unsigned char * gftbbs,
    unsigned char ** src,
    unsigned char * dest )

```

GF(2⁸) vector dot product, runs baseline version.

Does a GF(2⁸) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32*vlen byte constant array based on the input coefficients.

Parameters

<i>len</i>	Length of each vector in bytes. Must be ≥ 16 .
<i>vlen</i>	Number of vector sources.
<i>gftbbs</i>	Pointer to 32*vlen byte array of pre-calculated constants based on the array of input coefficients. Only elements 32*CONST*j + 1 of this array are used, where j = (0, 1, 2...) and CONST is the number of elements in the array of input coefficients. The elements used correspond to the original input coefficients.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Pointer to destination data array.

Returns

none

8.3.2.13 gf_vect_mad()

```
void gf_vect_mad (
    int len,
    int vec,
    int vec_i,
    unsigned char * gftbls,
    unsigned char * src,
    unsigned char * dest )
```

GF(2⁸) vector multiply accumulate, runs appropriate version.

Does a GF(2⁸) multiply across each byte of input source with expanded constant and add to destination array. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32*vec byte constant array based on the input coefficients.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

<i>len</i>	Length of each vector in bytes. Must be ≥ 64 .
<i>vec</i>	The number of vector sources or rows in the generator matrix for coding.
<i>vec_i</i>	The vector index corresponding to the single input source.
<i>gftbls</i>	Pointer to array of input tables generated from coding coefficients in ec_init_tables() . Must be of size 32*vec.
<i>src</i>	Array of pointers to source inputs.
<i>dest</i>	Pointer to destination data array.

Returns

none

8.3.2.14 gf_vect_mad_base()

```
void gf_vect_mad_base (
    int len,
    int vec,
    int vec_i,
    unsigned char * v,
```

```

    unsigned char * src,
    unsigned char * dest )

```

GF(2⁸) vector multiply accumulate, baseline version.

Baseline version of [gf_vect_mad\(\)](#) with same parameters.

8.4 gf_vect_mul.h File Reference

Interface to functions for vector (block) multiplication in GF(2⁸).

Functions

- int [gf_vect_mul](#) (int len, unsigned char *gftbl, void *src, void *dest)
GF(2⁸) vector multiply by constant, runs appropriate version.
- void [gf_vect_mul_init](#) (unsigned char c, unsigned char *gftbl)
Initialize 32-byte constant array for GF(2⁸) vector multiply.
- void [gf_vect_mul_base](#) (int len, unsigned char *a, unsigned char *src, unsigned char *dest)
GF(2⁸) vector multiply by constant, runs baseline version.

8.4.1 Detailed Description

Interface to functions for vector (block) multiplication in GF(2⁸).

This file defines the interface to routines used in fast RAID rebuild and erasure codes.

8.4.2 Function Documentation

8.4.2.1 gf_vect_mul()

```

int gf_vect_mul (
    int len,
    unsigned char * gftbl,
    void * src,
    void * dest )

```

GF(2⁸) vector multiply by constant, runs appropriate version.

Does a GF(2⁸) vector multiply $b = Ca$ where a and b are arrays and C is a single field element in GF(2⁸). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant C . $gftbl(C) = \{C\{00\}, C\{01\}, C\{02\}, \dots, C\{0f\}\}, \{C\{00\}, C\{10\}, C\{20\}, \dots, C\{f0\}\}$. len and src must be aligned to 32B.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

<i>len</i>	Length of vector in bytes. Must be aligned to 32B.
<i>gftbl</i>	Pointer to 32-byte array of pre-calculated constants based on C.
<i>src</i>	Pointer to src data array. Must be aligned to 32B.
<i>dest</i>	Pointer to destination data array. Must be aligned to 32B.

Returns

0 pass, other fail

8.4.2.2 gf_vect_mul_base()

```
void gf_vect_mul_base (
    int len,
    unsigned char * a,
    unsigned char * src,
    unsigned char * dest )
```

GF(2⁸) vector multiply by constant, runs baseline version.

Does a GF(2⁸) vector multiply $b = Ca$ where a and b are arrays and C is a single field element in GF(2⁸). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant C . $gftbl(C) = \{C\{00\}, C\{01\}, C\{02\}, \dots, C\{0f\}\}, \{C\{00\}, C\{10\}, C\{20\}, \dots, C\{f0\}\}$. len and src must be aligned to 32B.

Parameters

<i>len</i>	Length of vector in bytes. Must be aligned to 32B.
<i>a</i>	Pointer to 32-byte array of pre-calculated constants based on C. only use 2nd element is used.
<i>src</i>	Pointer to src data array. Must be aligned to 32B.
<i>dest</i>	Pointer to destination data array. Must be aligned to 32B.

8.4.2.3 gf_vect_mul_init()

```
void gf_vect_mul_init (
    unsigned char c,
    unsigned char * gftbl )
```

Initialize 32-byte constant array for GF(2⁸) vector multiply.

Calculates array $\{C\{00\}, C\{01\}, C\{02\}, \dots, C\{0f\}\}, \{C\{00\}, C\{10\}, C\{20\}, \dots, C\{f0\}\}$ as required by other fast vector multiply functions.

Parameters

<i>c</i>	Constant input.
<i>gftbl</i>	Table output.

8.5 igzip_lib.h File Reference

This file defines the igzip compression and decompression interface, a high performance deflate compression interface for storage applications.

```
#include <stdint.h>
```

Data Structures

- struct [isal_huff_histogram](#)
Holds histogram of deflate symbols.
- struct [isal_mod_hist](#)
- struct [BitBuf2](#)
Holds Bit Buffer information.
- struct [isal_zlib_header](#)
- struct [isal_gzip_header](#)
- struct [isal_zstate](#)
Holds the internal state information for input and output compression streams.
- struct [isal_hufftables](#)
Holds the huffman tree used to huffman encode the input stream.
- struct [isal_zstream](#)
Holds stream information.
- struct [inflate_huff_code_large](#)
- struct [inflate_huff_code_small](#)
- struct [inflate_state](#)
Holds decompression state information.

Enumerations

- enum [isal_zstate_state](#) {
[ZSTATE_NEW_HDR](#), [ZSTATE_HDR](#), [ZSTATE_CREATE_HDR](#), [ZSTATE_BODY](#),
[ZSTATE_FLUSH_READ_BUFFER](#), [ZSTATE_TYPE0_BODY](#), [ZSTATE_SYNC_FLUSH](#), [ZSTATE_FLUSH_WRITE_BUFFER](#),
[ZSTATE_TRL](#), [ZSTATE_END](#), [ZSTATE_TMP_NEW_HDR](#), [ZSTATE_TMP_HDR](#),
[ZSTATE_TMP_CREATE_HDR](#), [ZSTATE_TMP_BODY](#), [ZSTATE_TMP_FLUSH_READ_BUFFER](#), [ZSTATE_TMP_TYPE0_BODY](#),
[ZSTATE_TMP_SYNC_FLUSH](#), [ZSTATE_TMP_FLUSH_WRITE_BUFFER](#), [ZSTATE_TMP_TRL](#), [ZSTATE_TMP_END](#)
 }
Compression State please note ZSTATE_TRL only applies for GZIP compression.

Functions

- void [isal_update_histogram](#) (uint8_t *in_stream, int length, struct [isal_huff_histogram](#) *histogram)

Updates histograms to include the symbols found in the input stream. Since this function only updates the histograms, it can be called on multiple streams to get a histogram better representing the desired data set. When first using histogram it must be initialized by zeroing the structure.
- int [isal_create_hufftables](#) (struct [isal_hufftables](#) *hufftables, struct [isal_huff_histogram](#) *histogram)

Creates a custom huffman code for the given histograms in which every literal and repeat length is assigned a code and all possible lookback distances are assigned a code.
- int [isal_create_hufftables_subset](#) (struct [isal_hufftables](#) *hufftables, struct [isal_huff_histogram](#) *histogram)

Creates a custom huffman code for the given histograms like [isal_create_hufftables\(\)](#) except literals with 0 frequency in the histogram are not assigned a code.
- void [isal_deflate_init](#) (struct [isal_zstream](#) *stream)

Initialize compression stream data structure.
- void [isal_deflate_reset](#) (struct [isal_zstream](#) *stream)

Reinitialize compression stream data structure. Performs the same action as [isal_deflate_init](#), but does not change user supplied input such as the level, flush type, compression wrapper (like gzip), hufftables, and end_of_stream_flag.
- void [isal_gzip_header_init](#) (struct [isal_gzip_header](#) *gz_hdr)

Set gzip header default values.
- uint32_t [isal_write_gzip_header](#) (struct [isal_zstream](#) *stream, struct [isal_gzip_header](#) *gz_hdr)

Write gzip header to output stream.
- uint32_t [isal_write_zlib_header](#) (struct [isal_zstream](#) *stream, struct [isal_zlib_header](#) *z_hdr)

Write zlib header to output stream.
- int [isal_deflate_set_hufftables](#) (struct [isal_zstream](#) *stream, struct [isal_hufftables](#) *hufftables, int type)

Set stream to use a new Huffman code.
- void [isal_deflate_stateless_init](#) (struct [isal_zstream](#) *stream)

Initialize compression stream data structure.
- int [isal_deflate_set_dict](#) (struct [isal_zstream](#) *stream, uint8_t *dict, uint32_t dict_len)

Set compression dictionary to use.
- int [isal_deflate](#) (struct [isal_zstream](#) *stream)

Fast data (deflate) compression for storage applications.
- int [isal_deflate_stateless](#) (struct [isal_zstream](#) *stream)

Fast data (deflate) stateless compression for storage applications.
- void [isal_inflate_init](#) (struct [inflate_state](#) *state)

Initialize decompression state data structure.
- void [isal_inflate_reset](#) (struct [inflate_state](#) *state)

Reinitialize decompression state data structure.
- int [isal_inflate_set_dict](#) (struct [inflate_state](#) *state, uint8_t *dict, uint32_t dict_len)

Set decompression dictionary to use.
- int [isal_read_gzip_header](#) (struct [inflate_state](#) *state, struct [isal_gzip_header](#) *gz_hdr)

Read and return gzip header information.
- int [isal_read_zlib_header](#) (struct [inflate_state](#) *state, struct [isal_zlib_header](#) *zlib_hdr)

Read and return zlib header information.
- int [isal_inflate](#) (struct [inflate_state](#) *state)

Fast data (deflate) decompression for storage applications.
- int [isal_inflate_stateless](#) (struct [inflate_state](#) *state)

Fast data (deflate) stateless decompression for storage applications.
- uint32_t [isal_adler32](#) (uint32_t init, const unsigned char *buf, uint64_t len)

Calculate Adler-32 checksum, runs appropriate version.

8.5.1 Detailed Description

This file defines the igzip compression and decompression interface, a high performance deflate compression interface for storage applications.

Deflate is a widely used compression standard that can be used standalone, it also forms the basis of gzip and zlib compression formats. Igzip supports the following flush features:

- No Flush: The default method where no special flush is performed.
- Sync flush: whereby `isal_deflate()` finishes the current deflate block at the end of each input buffer. The deflate block is byte aligned by appending an empty stored block.
- Full flush: whereby `isal_deflate()` finishes and aligns the deflate block as in sync flush but also ensures that subsequent block's history does not look back beyond this point and new blocks are fully independent.

Igzip also supports compression levels from `ISAL_DEF_MIN_LEVEL` to `ISAL_DEF_MAX_LEVEL`.

Igzip contains some behavior configurable at compile time. These configurable options are:

- `IGZIP_HIST_SIZE` - Defines the window size. The default value is 32K (note K represents 1024), but 8K is also supported. Powers of 2 which are at most 32K may also work.
- `LONGER_HUFFTABLES` - Defines whether to use a larger hufftables structure which may increase performance with smaller `IGZIP_HIST_SIZE` values. By default this option is not defined. This define sets `IGZIP_HIST_SIZE` to be 8 if `IGZIP_HIST_SIZE > 8K`.

As an example, to compile gzip with an 8K window size, in a terminal run

```
gmake D="-D IGZIP_HIST_SIZE=8*1024"
```

on Linux and FreeBSD, or with

```
nmake -f Makefile.nmake D="-D
* IGZIP_HIST_SIZE=8*1024"
```

on Windows.

8.5.2 Enumeration Type Documentation

8.5.2.1 `isal_zstate_state`

```
enum isal_zstate_state
```

Compression State please note `ZSTATE_TRL` only applies for GZIP compression.

Enumerator

ZSTATE_NEW_HDR	Header to be written.
ZSTATE_HDR	Header state.
ZSTATE_CREATE_HDR	Header to be created.
ZSTATE_BODY	Body state.
ZSTATE_FLUSH_READ_BUFFER	Flush buffer.
ZSTATE_TYPE0_BODY	Type0 block header to be written. Type0 block body to be written
ZSTATE_SYNC_FLUSH	Write sync flush block.
ZSTATE_FLUSH_WRITE_BUFFER	Flush bitbuf.
ZSTATE_TRL	Trailer state.
ZSTATE_END	End state.
ZSTATE_TMP_NEW_HDR	Temporary Header to be written.
ZSTATE_TMP_HDR	Temporary Header state.
ZSTATE_TMP_CREATE_HDR	Temporary Header to be created state.
ZSTATE_TMP_BODY	Temporary Body state.
ZSTATE_TMP_FLUSH_READ_BUFFER	Flush buffer.
ZSTATE_TMP_TYPE0_BODY	Temporary Type0 block header to be written. Temporary Type0 block body to be written
ZSTATE_TMP_SYNC_FLUSH	Write sync flush block.
ZSTATE_TMP_FLUSH_WRITE_BUFFER	Flush bitbuf.
ZSTATE_TMP_TRL	Temporary Trailer state.
ZSTATE_TMP_END	Temporary End state.

8.5.3 Function Documentation

8.5.3.1 isal_adler32()

```
uint32_t isal_adler32 (
    uint32_t init,
    const unsigned char * buf,
    uint64_t len )
```

Calculate Adler-32 checksum, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

<i>init</i>	initial Adler-32 value
<i>buf</i>	buffer to calculate checksum on
<i>len</i>	buffer length in bytes

Returns

32-bit Adler-32 checksum

8.5.3.2 isal_create_hufftables()

```
int isal_create_hufftables (
    struct isal_hufftables * hufftables,
    struct isal_huff_histogram * histogram )
```

Creates a custom huffman code for the given histograms in which every literal and repeat length is assigned a code and all possible lookback distances are assigned a code.

Parameters

<i>hufftables</i>	the output structure containing the huffman code
<i>histogram</i>	histogram containing frequency of literal symbols, repeat lengths and lookback distances

Returns

Returns a non zero value if an invalid huffman code was created.

8.5.3.3 isal_create_hufftables_subset()

```
int isal_create_hufftables_subset (
    struct isal_hufftables * hufftables,
    struct isal_huff_histogram * histogram )
```

Creates a custom huffman code for the given histograms like [isal_create_hufftables\(\)](#) except literals with 0 frequency in the histogram are not assigned a code.

Parameters

<i>hufftables</i>	the output structure containing the huffman code
<i>histogram</i>	histogram containing frequency of literal symbols, repeat lengths and lookback distances

Returns

Returns a non zero value if an invalid huffman code was created.

8.5.3.4 isal_deflate()

```
int isal_deflate (
    struct isal_zstream * stream )
```

Fast data (deflate) compression for storage applications.

The call to `isal_deflate()` will take data from the input buffer (updating `next_in`, `avail_in` and write a compressed stream to the output buffer (updating `next_out` and `avail_out`). The function returns when either the input buffer is empty or the output buffer is full.

On entry to `isal_deflate()`, `next_in` points to an input buffer and `avail_in` indicates the length of that buffer. Similarly `next_out` points to an empty output buffer and `avail_out` indicates the size of that buffer.

The fields `total_in` and `total_out` start at 0 and are updated by `isal_deflate()`. These reflect the total number of bytes read or written so far.

When the last input buffer is passed in, signaled by setting the `end_of_stream`, the routine will complete compression at the end of the input buffer, as long as the output buffer is big enough.

The compression level can be set by setting `level` to any value between `ISAL_DEF_MIN_LEVEL` and `ISAL_DEF_MAX_LEVEL`. When the compression level is `ISAL_DEF_MIN_LEVEL`, hufftables can be set to a table trained for the the specific data type being compressed to achieve better compression. When a higher compression level is desired, a larger generic memory buffer needs to be supplied by setting `level_buf` and `level_buf_size` to represent the chunk of memory. For level `x`, the suggest size for this buffer this buffer is `ISAL_DEFL_LVLx_DEFAULT`. The defines `ISAL_DEFL_LVLx_MIN`, `ISAL_DEFL_LVLx_SMALL`, `ISAL_DEFL_LVLx_MEDIUM`, `ISAL_DEFL_LVLx_LARGE`, and `ISAL_DEFL_LVLx_EXTRA_LARGE` are also provided as other suggested sizes.

The equivalent of the zlib `FLUSH_SYNC` operation is currently supported. Flush types can be `NO_FLUSH`, `SYNC_FLUSH` or `FULL_FLUSH`. Default flush type is `NO_FLUSH`. A `SYNC_` or `FULL_` flush will byte align the deflate block by appending an empty stored block once all input has been compressed, including the buffered input. Checking that the `out_buffer` is not empty or that `internal_state.state = ZSTATE_NEW_HDR` is sufficient to guarantee all input has been flushed. Additionally `FULL_FLUSH` will ensure look back history does not include previous blocks so new blocks are fully independent. Switching between flush types is supported.

If a compression dictionary is required, the dictionary can be set calling `isal_deflate_set_dictionary` before calling `isal_deflate`.

If the `gzip_flag` is set to `IGZIP_GZIP`, a generic gzip header and the gzip trailer are written around the deflate compressed data. If `gzip_flag` is set to `IGZIP_GZIP_NO_HDR`, then only the gzip trailer is written. A full-featured header is supported by the `isal_write_{gzip,zlib}_header()` functions.

Parameters

<i>stream</i>	Structure holding state information on the compression streams.
---------------	---

Returns

`COMP_OK` (if everything is ok), `INVALID_FLUSH` (if an invalid `FLUSH` is selected), `ISAL_INVALID_LEVEL` (if an invalid compression level is selected), `ISAL_INVALID_LEVEL_BUF` (if the level buffer is not large enough).

8.5.3.5 isal_deflate_init()

```
void isal_deflate_init (
    struct isal_zstream * stream )
```

Initialize compression stream data structure.

Parameters

<i>stream</i>	Structure holding state information on the compression streams.
---------------	---

Returns

none

8.5.3.6 isal_deflate_reset()

```
void isal_deflate_reset (
    struct isal_zstream * stream )
```

Reinitialize compression stream data structure. Performs the same action as `isal_deflate_init`, but does not change user supplied input such as the level, flush type, compression wrapper (like `gzip`), hufftables, and `end_of_stream_flag`.

Parameters

<i>stream</i>	Structure holding state information on the compression streams.
---------------	---

Returns

none

8.5.3.7 isal_deflate_set_dict()

```
int isal_deflate_set_dict (
    struct isal_zstream * stream,
    uint8_t * dict,
    uint32_t dict_len )
```

Set compression dictionary to use.

This function is to be called after `isal_deflate_init`, or after completing a `SYNC_FLUSH` or `FULL_FLUSH` and before the next call do `isal_deflate`. If the dictionary is longer than `IGZIP_HIST_SIZE`, only the last `IGZIP_HIST_SIZE` bytes will be used.

Parameters

<i>stream</i>	Structure holding state information on the compression streams.
<i>dict</i>	Array containing dictionary to use.
<i>dict_len</i>	Length of dict.

Returns

COMP_OK, ISAL_INVALID_STATE (dictionary could not be set)

8.5.3.8 isal_deflate_set_hufftables()

```
int isal_deflate_set_hufftables (
    struct isal_zstream * stream,
    struct isal_hufftables * hufftables,
    int type )
```

Set stream to use a new Huffman code.

Sets the Huffman code to be used in compression before compression start or after the successful completion of a SYNC_FLUSH or FULL_FLUSH. If type has value IGZIP_HUFFTABLE_DEFAULT, the stream is set to use the default Huffman code. If type has value IGZIP_HUFFTABLE_STATIC, the stream is set to use the deflate standard static Huffman code, or if type has value IGZIP_HUFFTABLE_CUSTOM, the stream is set to use the [isal_hufftables](#) structure input to isal_deflate_set_hufftables.

Parameters

<i>stream</i>	Structure holding state information on the compression stream.
<i>hufftables</i>	new huffman code to use if type is set to IGZIP_HUFFTABLE_CUSTOM.
<i>type</i>	Flag specifying what hufftable to use.

Returns

Returns INVALID_OPERATION if the stream was unmodified. This may be due to the stream being in a state where changing the huffman code is not allowed or an invalid input is provided.

8.5.3.9 isal_deflate_stateless()

```
int isal_deflate_stateless (
    struct isal_zstream * stream )
```

Fast data (deflate) stateless compression for storage applications.

Stateless (one shot) compression routine with a similar interface to [isal_deflate\(\)](#) but operates on entire input buffer at one time. Parameter `avail_out` must be large enough to fit the entire compressed output. Max expansion is limited to the input size plus the header size of a stored/raw block.

When the compression level is set to 1, unlike in [isal_deflate\(\)](#), `level_buf` may be optionally set depending on what performance is desired.

For stateless the flush types `NO_FLUSH` and `FULL_FLUSH` are supported. `FULL_FLUSH` will byte align the output deflate block so additional blocks can be easily appended.

If the `gzip_flag` is set to `IGZIP_GZIP`, a generic gzip header and the gzip trailer are written around the deflate compressed data. If `gzip_flag` is set to `IGZIP_GZIP_NO_HDR`, then only the gzip trailer is written.

Parameters

<i>stream</i>	Structure holding state information on the compression streams.
---------------	---

Returns

`COMP_OK` (if everything is ok), `INVALID_FLUSH` (if an invalid FLUSH is selected), `ISAL_INVALID_LEVEL` (if an invalid compression level is selected), `ISAL_INVALID_LEVEL_BUF` (if the level buffer is not large enough), `STATELESS_OVERFLOW` (if output buffer will not fit output).

8.5.3.10 isal_deflate_stateless_init()

```
void isal_deflate_stateless_init (
    struct isal_zstream * stream )
```

Initialize compression stream data structure.

Parameters

<i>stream</i>	Structure holding state information on the compression streams.
---------------	---

Returns

none

8.5.3.11 isal_gzip_header_init()

```
void isal_gzip_header_init (
    struct isal_gzip_header * gz_hdr )
```

Set gzip header default values.

Parameters

<i>gz_hdr</i>	Gzip header to initialize.
---------------	----------------------------

8.5.3.12 isal_inflate()

```
int isal_inflate (
    struct inflate_state * state )
```

Fast data (deflate) decompression for storage applications.

On entry to `isal_inflate()`, `next_in` points to an input buffer and `avail_in` indicates the length of that buffer. Similarly `next_out` points to an empty output buffer and `avail_out` indicates the size of that buffer.

The field `total_out` starts at 0 and is updated by `isal_inflate()`. This reflects the total number of bytes written so far.

The call to `isal_inflate()` will take data from the input buffer (updating `next_in`, `avail_in` and write a decompressed stream to the output buffer (updating `next_out` and `avail_out`). The function returns when the input buffer is empty, the output buffer is full, invalid data is found, or in the case of zlib formatted data if a dictionary is specified. The current state of the decompression on exit can be read from `state->block-state`.

If the `crc_flag` is set to `ISAL_GZIP_NO_HDR` the gzip crc of the output is stored in `state->crc`. Alternatively, if the `crc_flag` is set to `ISAL_ZLIB_NO_HDR` the Adler32 of the output is stored in `state->crc` (checksum may not be updated until decompression is complete). When the `crc_flag` is set to `ISAL_GZIP_NO_HDR_VER` or `ISAL_ZLIB_NO_HDR_VER`, the behavior is the same, except the checksum is verified with the checksum after immediately following the deflate data. If the `crc_flag` is set to `ISAL_GZIP` or `ISAL_ZLIB`, the gzip/zlib header is parsed, `state->crc` is set to the appropriate checksum, and the checksum is verified. If the `crc_flag` is set to `ISAL_DEFLATE` (default), then the data is treated as a raw deflate block.

The element `state->hist_bits` has values from 0 to 15, where values of 1 to 15 are the log base 2 size of the matching window and 0 is the default with maximum history size.

If a dictionary is required, a call to `isal_inflate_set_dict` will set the dictionary.

Parameters

<i>state</i>	Structure holding state information on the compression streams.
--------------	---

Returns

`ISAL_DECOMP_OK` (if everything is ok), `ISAL_INVALID_BLOCK`, `ISAL_NEED_DICT`, `ISAL_INVALID_SYMBOL`, `ISAL_INVALID_LOOKBACK`, `ISAL_INVALID_WRAPPER`, `ISAL_UNSUPPORTED_METHOD`, `ISAL_INCORRECT_CHECKSUM`.

8.5.3.13 isal_inflate_init()

```
void isal_inflate_init (
    struct inflate_state * state )
```

Initialize decompression state data structure.

Parameters

<i>state</i>	Structure holding state information on the compression streams.
--------------	---

Returns

none

8.5.3.14 isal_inflate_reset()

```
void isal_inflate_reset (
    struct inflate_state * state )
```

Reinitialize decompression state data structure.

Parameters

<i>state</i>	Structure holding state information on the compression streams.
--------------	---

Returns

none

8.5.3.15 isal_inflate_set_dict()

```
int isal_inflate_set_dict (
    struct inflate_state * state,
    uint8_t * dict,
    uint32_t dict_len )
```

Set decompression dictionary to use.

This function is to be called after `isal_inflate_init`. If the dictionary is longer than `IGZIP_HIST_SIZE`, only the last `IGZIP_HIST_SIZE` bytes will be used.

Parameters

<i>state</i>	Structure holding state information on the decompression stream.
<i>dict</i>	Array containing dictionary to use.
<i>dict_len</i>	Length of dict.

Returns

COMP_OK, ISAL_INVALID_STATE (dictionary could not be set)

8.5.3.16 isal_inflate_stateless()

```
int isal_inflate_stateless (
    struct inflate_state * state )
```

Fast data (deflate) stateless decompression for storage applications.

Stateless (one shot) decompression routine with a similar interface to [isal_inflate\(\)](#) but operates on entire input buffer at one time. Parameter `avail_out` must be large enough to fit the entire decompressed output. Dictionaries are not supported.

Parameters

<i>state</i>	Structure holding state information on the compression streams.
--------------	---

Returns

ISAL_DECOMP_OK (if everything is ok), ISAL_END_INPUT (if all input was decompressed), ISAL_NEED_DICT, ISAL_OUT_OVERFLOW (if output buffer ran out of space), ISAL_INVALID_BLOCK, ISAL_INVALID_SYMBOL, ISAL_INVALID_LOOKBACK, ISAL_INVALID_WRAPPER, ISAL_UNSUPPORTED_METHOD, ISAL_INCORRECT_CHECKSUM.

8.5.3.17 isal_read_gzip_header()

```
int isal_read_gzip_header (
    struct inflate_state * state,
    struct isal_gzip_header * gz_hdr )
```

Read and return gzip header information.

On entry `state` must be initialized and `next_in` pointing to a gzip compressed buffer. The buffers `gz_hdr->extra`, `gz_hdr->name`, `gz_hdr->comments` and the buffer lengths must be set to record the corresponding field, or set to NULL to disregard that gzip header information. If one of these buffers overflows, the user can reallocate a larger buffer and call this function again to continue reading the header information.

Parameters

<i>state</i>	Structure holding state information on the decompression stream.
<i>gz_hdr</i>	Structure to return data encoded in the gzip header

Returns

ISAL_DECOMP_OK (header was successfully parsed) ISAL_END_INPUT (all input was parsed), ISAL_NAME_OVERFLOW (gz_hdr->name overflowed while parsing), ISAL_COMMENT_OVERFLOW (gz_hdr->comment overflowed while parsing), ISAL_EXTRA_OVERFLOW (gz_hdr->extra overflowed while parsing), ISAL_INVALID_WRAPPER (invalid gzip header found), ISAL_UNSUPPORTED_METHOD (deflate is not the compression method), ISAL_INCORRECT_CHECKSUM (gzip header checksum was incorrect)

8.5.3.18 isal_read_zlib_header()

```
int isal_read_zlib_header (
    struct inflate_state * state,
    struct isal_zlib_header * zlib_hdr )
```

Read and return zlib header information.

On entry state must be initialized and next_in pointing to a zlib compressed buffer.

Parameters

<i>state</i>	Structure holding state information on the decompression stream.
<i>zlib_hdr</i>	Structure to return data encoded in the zlib header

Returns

ISAL_DECOMP_OK (header was successfully parsed), ISAL_END_INPUT (all input was parsed), ISAL_UNSUPPORTED_METHOD (deflate is not the compression method), ISAL_INCORRECT_CHECKSUM (zlib header checksum was incorrect)

8.5.3.19 isal_update_histogram()

```
void isal_update_histogram (
    uint8_t * in_stream,
    int length,
    struct isal_huff_histogram * histogram )
```

Updates histograms to include the symbols found in the input stream. Since this function only updates the histograms, it can be called on multiple streams to get a histogram better representing the desired data set. When first using histogram it must be initialized by zeroing the structure.

Parameters

<i>in_stream</i>	Input stream of data.
<i>length</i>	The length of start_stream.
<i>histogram</i>	The returned histogram of lit/len/dist symbols.

8.5.3.20 isal_write_gzip_header()

```
uint32_t isal_write_gzip_header (
    struct isal_zstream * stream,
    struct isal_gzip_header * gz_hdr )
```

Write gzip header to output stream.

Writes the gzip header to the output stream. On entry this function assumes that the output buffer has been initialized, so stream->next_out, stream->avail_out and stream->total_out have been set. If the output buffer contains insufficient space, stream is not modified.

Parameters

<i>stream</i>	Structure holding state information on the compression stream.
<i>gz_hdr</i>	Structure holding the gzip header information to encode.

Returns

Returns 0 if the header is successfully written, otherwise returns the minimum size required to successfully write the gzip header to the output buffer.

8.5.3.21 isal_write_zlib_header()

```
uint32_t isal_write_zlib_header (
    struct isal_zstream * stream,
    struct isal_zlib_header * z_hdr )
```

Write zlib header to output stream.

Writes the zlib header to the output stream. On entry this function assumes that the output buffer has been initialized, so stream->next_out, stream->avail_out and stream->total_out have been set. If the output buffer contains insufficient space, stream is not modified.

Parameters

<i>stream</i>	Structure holding state information on the compression stream.
<i>z_hdr</i>	Structure holding the zlib header information to encode.

Returns

Returns 0 if the header is successfully written, otherwise returns the minimum size required to successfully write the zlib header to the output buffer.

8.6 isa-l.h File Reference

Include for ISA-L library.

```
#include <isa-l/crc.h>
#include <isa-l/crc64.h>
#include <isa-l/erasure_code.h>
#include <isa-l/gf_vect_mul.h>
#include <isa-l/igzip_lib.h>
#include <isa-l/mem_routines.h>
#include <isa-l/raid.h>
```

8.6.1 Detailed Description

Include for ISA-L library.

8.7 mem_routines.h File Reference

Interface to storage mem operations.

```
#include <stddef.h>
```

Functions

- int [isal_zero_detect](#) (void *mem, size_t len)
Detect if a memory region is all zero.

8.7.1 Detailed Description

Interface to storage mem operations.

Defines the interface for vector versions of common memory functions.

8.7.2 Function Documentation

8.7.2.1 isal_zero_detect()

```
int isal_zero_detect (
    void * mem,
    size_t len )
```

Detect if a memory region is all zero.

Zero detect function with optimizations for large blocks > 128 bytes

Parameters

<i>mem</i>	Pointer to memory region to test
<i>len</i>	Length of region in bytes

Returns

0 - region is all zeros other - region has non zero bytes

8.8 raid.h File Reference

Interface to RAID functions - XOR and P+Q calculation.

Functions

- int [xor_gen](#) (int vects, int len, void **array)
Generate XOR parity vector from N sources, runs appropriate version.
- int [xor_check](#) (int vects, int len, void **array)
Checks that array has XOR parity sum of 0 across all vectors, runs appropriate version.
- int [pq_gen](#) (int vects, int len, void **array)
Generate P+Q parity vectors from N sources, runs appropriate version.
- int [pq_check](#) (int vects, int len, void **array)
Checks that array of N sources, P and Q are consistent across all vectors, runs appropriate version.
- int [pq_gen_base](#) (int vects, int len, void **array)
Generate P+Q parity vectors from N sources, runs baseline version.
- int [xor_gen_base](#) (int vects, int len, void **array)
Generate XOR parity vector from N sources, runs baseline version.
- int [xor_check_base](#) (int vects, int len, void **array)
Checks that array has XOR parity sum of 0 across all vectors, runs baseline version.
- int [pq_check_base](#) (int vects, int len, void **array)
Checks that array of N sources, P and Q are consistent across all vectors, runs baseline version.

8.8.1 Detailed Description

Interface to RAID functions - XOR and P+Q calculation.

This file defines the interface to optimized XOR calculation (RAID5) or P+Q dual parity (RAID6). Operations are carried out on an array of pointers to sources and output arrays.

8.8.2 Function Documentation

8.8.2.1 pq_check()

```
int pq_check (
    int vects,
    int len,
    void ** array )
```

Checks that array of N sources, P and Q are consistent across all vectors, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

<i>vects</i>	Number of vectors in array including P&Q.
<i>len</i>	Length of each vector in bytes. Must be 16B aligned.
<i>array</i>	Array of pointers to source and P, Q. P and Q parity are assumed to be the last two pointers in the array. All pointers must be aligned to 16B.

Returns

0 pass, other fail

8.8.2.2 pq_check_base()

```
int pq_check_base (
    int vects,
    int len,
    void ** array )
```

Checks that array of N sources, P and Q are consistent across all vectors, runs baseline version.

Parameters

<i>vects</i>	Number of vectors in array including P&Q.
<i>len</i>	Length of each vector in bytes. Must be 16B aligned.
<i>array</i>	Array of pointers to source and P, Q. P and Q parity are assumed to be the last two pointers in the array. All pointers must be aligned to 16B.

Returns

0 pass, other fail

8.8.2.3 pq_gen()

```
int pq_gen (
    int vects,
    int len,
    void ** array )
```

Generate P+Q parity vectors from N sources, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

<i>vects</i>	Number of source+dest vectors in array.
<i>len</i>	Length of each vector in bytes. Must be 32B aligned.
<i>array</i>	Array of pointers to source and dest. For P+Q the dest is the last two pointers. ie array[vects-2], array[vects-1]. P and Q parity vectors are written to these last two pointers. Src and dest pointers must be aligned to 32B.

Returns

0 pass, other fail

8.8.2.4 pq_gen_base()

```
int pq_gen_base (
    int vects,
    int len,
    void ** array )
```

Generate P+Q parity vectors from N sources, runs baseline version.

Parameters

<i>vects</i>	Number of source+dest vectors in array.
<i>len</i>	Length of each vector in bytes. Must be 16B aligned.
<i>array</i>	Array of pointers to source and dest. For P+Q the dest is the last two pointers. ie array[vects-2], array[vects-1]. P and Q parity vectors are written to these last two pointers. Src and dest pointers must be aligned to 16B.

Returns

0 pass, other fail

8.8.2.5 xor_check()

```
int xor_check (
    int vects,
    int len,
    void ** array )
```

Checks that array has XOR parity sum of 0 across all vectors, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

<i>vects</i>	Number of vectors in array.
<i>len</i>	Length of each vector in bytes.
<i>array</i>	Array of pointers to vectors. Src and dest pointers must be aligned to 16B.

Returns

0 pass, other fail

8.8.2.6 xor_check_base()

```
int xor_check_base (
    int vects,
    int len,
    void ** array )
```

Checks that array has XOR parity sum of 0 across all vectors, runs baseline version.

Parameters

<i>vects</i>	Number of vectors in array.
<i>len</i>	Length of each vector in bytes.
<i>array</i>	Array of pointers to vectors. Src and dest pointers must be aligned to 16B.

Returns

0 pass, other fail

8.8.2.7 xor_gen()

```
int xor_gen (
    int vects,
    int len,
    void ** array )
```

Generate XOR parity vector from N sources, runs appropriate version.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

<i>vects</i>	Number of source+dest vectors in array.
<i>len</i>	Length of each vector in bytes.
<i>array</i>	Array of pointers to source and dest. For XOR the dest is the last pointer. ie array[vects-1]. Src and dest pointers must be aligned to 32B.

Returns

0 pass, other fail

8.8.2.8 xor_gen_base()

```
int xor_gen_base (
    int vects,
    int len,
    void ** array )
```

Generate XOR parity vector from N sources, runs baseline version.

Parameters

<i>vects</i>	Number of source+dest vectors in array.
<i>len</i>	Length of each vector in bytes.
<i>array</i>	Array of pointers to source and dest. For XOR the dest is the last pointer. ie array[vects-1]. Src and dest pointers must be aligned to 32B.

Returns

0 pass, other fail

Index

BitBuf2, 17

crc.h, 25

 crc16_t10dif, 26

 crc16_t10dif_base, 26

 crc16_t10dif_copy, 27

 crc16_t10dif_copy_base, 27

 crc32_gzip_refl, 28

 crc32_gzip_refl_base, 28

 crc32_ieee, 29

 crc32_ieee_base, 29

 crc32_iscsi, 30

 crc32_iscsi_base, 30

 crc16_t10dif

 crc.h, 26

 crc16_t10dif_base

 crc.h, 26

 crc16_t10dif_copy

 crc.h, 27

 crc16_t10dif_copy_base

 crc.h, 27

 crc32_gzip_refl

 crc.h, 28

 crc32_gzip_refl_base

 crc.h, 28

 crc32_ieee

 crc.h, 29

 crc32_ieee_base

 crc.h, 29

 crc32_iscsi

 crc.h, 30

 crc32_iscsi_base

 crc.h, 30

 crc64.h, 31

 crc64_ecma_norm, 32

 crc64_ecma_norm_base, 33

 crc64_ecma_norm_by8, 33

 crc64_ecma_refl, 34

 crc64_ecma_refl_base, 34

 crc64_ecma_refl_by8, 34

 crc64_iso_norm, 35

 crc64_iso_norm_base, 35

 crc64_iso_norm_by8, 36

 crc64_iso_refl, 36

 crc64_iso_refl_base, 37

 crc64_iso_refl_by8, 37

 crc64_jones_norm, 38

 crc64_jones_norm_base, 38

 crc64_jones_norm_by8, 39

 crc64_jones_refl, 39

 crc64_jones_refl_base, 40

 crc64_jones_refl_by8, 40

 crc64_ecma_norm

 crc64.h, 32

 crc64_ecma_norm_base

 crc64.h, 33

 crc64_ecma_norm_by8

 crc64.h, 33

 crc64_ecma_refl

 crc64.h, 34

 crc64_ecma_refl_base

 crc64.h, 34

 crc64_ecma_refl_by8

 crc64.h, 34

 crc64_iso_norm

 crc64.h, 35

 crc64_iso_norm_base

 crc64.h, 35

 crc64_iso_norm_by8

 crc64.h, 36

 crc64_iso_refl

 crc64.h, 36

 crc64_iso_refl_base

 crc64.h, 37

 crc64_iso_refl_by8

 crc64.h, 37

 crc64_jones_norm

 crc64.h, 38

 crc64_jones_norm_base

 crc64.h, 38

 crc64_jones_norm_by8

 crc64.h, 39

 crc64_jones_refl

 crc64.h, 39

 crc64_jones_refl_base

 crc64.h, 40

 crc64_jones_refl_by8

 crc64.h, 40

 ec_encode_data

 erasure_code.h, 42

 ec_encode_data_base

- erasure_code.h, 43
- ec_encode_data_update
 - erasure_code.h, 43
- ec_encode_data_update_base
 - erasure_code.h, 44
- ec_init_tables
 - erasure_code.h, 44
- erasure_code.h, 41
 - ec_encode_data, 42
 - ec_encode_data_base, 43
 - ec_encode_data_update, 43
 - ec_encode_data_update_base, 44
 - ec_init_tables, 44
 - gf_gen_cauchy1_matrix, 45
 - gf_gen_rs_matrix, 45
 - gf_inv, 46
 - gf_invert_matrix, 46
 - gf_mul, 47
 - gf_vect_dot_prod, 47
 - gf_vect_dot_prod_base, 48
 - gf_vect_mad, 49
 - gf_vect_mad_base, 49
- gf_gen_cauchy1_matrix
 - erasure_code.h, 45
- gf_gen_rs_matrix
 - erasure_code.h, 45
- gf_inv
 - erasure_code.h, 46
- gf_invert_matrix
 - erasure_code.h, 46
- gf_mul
 - erasure_code.h, 47
- gf_vect_dot_prod
 - erasure_code.h, 47
- gf_vect_dot_prod_base
 - erasure_code.h, 48
- gf_vect_mad
 - erasure_code.h, 49
- gf_vect_mad_base
 - erasure_code.h, 49
- gf_vect_mul
 - gf_vect_mul.h, 50
- gf_vect_mul.h, 50
 - gf_vect_mul, 50
 - gf_vect_mul_base, 51
 - gf_vect_mul_init, 51
- gf_vect_mul_base
 - gf_vect_mul.h, 51
- gf_vect_mul_init
 - gf_vect_mul.h, 51
- igzip_lib.h, 52
 - isal_adler32, 55
 - isal_create_hufftables, 56
- isal_create_hufftables_subset, 56
- isal_deflate, 56
 - isal_deflate_init, 57
 - isal_deflate_reset, 58
 - isal_deflate_set_dict, 58
 - isal_deflate_set_hufftables, 59
 - isal_deflate_stateless, 59
 - isal_deflate_stateless_init, 60
 - isal_gzip_header_init, 60
 - isal_inflate, 62
 - isal_inflate_init, 62
 - isal_inflate_reset, 63
 - isal_inflate_set_dict, 63
 - isal_inflate_stateless, 64
 - isal_read_gzip_header, 64
 - isal_read_zlib_header, 65
 - isal_update_histogram, 65
 - isal_write_gzip_header, 66
 - isal_write_zlib_header, 66
 - isal_zstate_state, 54
 - ZSTATE_BODY, 55
 - ZSTATE_CREATE_HDR, 55
 - ZSTATE_END, 55
 - ZSTATE_FLUSH_READ_BUFFER, 55
 - ZSTATE_FLUSH_WRITE_BUFFER, 55
 - ZSTATE_HDR, 55
 - ZSTATE_NEW_HDR, 55
 - ZSTATE_SYNC_FLUSH, 55
 - ZSTATE_TMP_BODY, 55
 - ZSTATE_TMP_CREATE_HDR, 55
 - ZSTATE_TMP_END, 55
 - ZSTATE_TMP_FLUSH_READ_BUFFER, 55
 - ZSTATE_TMP_FLUSH_WRITE_BUFFER, 55
 - ZSTATE_TMP_HDR, 55
 - ZSTATE_TMP_NEW_HDR, 55
 - ZSTATE_TMP_SYNC_FLUSH, 55
 - ZSTATE_TMP_TRL, 55
 - ZSTATE_TMP_TYPE0_BODY, 55
 - ZSTATE_TRL, 55
 - ZSTATE_TYPE0_BODY, 55
- inflate_huff_code_large, 18
- inflate_huff_code_small, 18
- inflate_state, 18
- isa-l.h, 67
- isal_adler32
 - igzip_lib.h, 55
- isal_create_hufftables
 - igzip_lib.h, 56
- isal_create_hufftables_subset
 - igzip_lib.h, 56
- isal_deflate
 - igzip_lib.h, 56
- isal_deflate_init
 - igzip_lib.h, 57

- isal_deflate_reset
 - igzip_lib.h, 58
- isal_deflate_set_dict
 - igzip_lib.h, 58
- isal_deflate_set_hufftables
 - igzip_lib.h, 59
- isal_deflate_stateless
 - igzip_lib.h, 59
- isal_deflate_stateless_init
 - igzip_lib.h, 60
- isal_gzip_header, 19
- isal_gzip_header_init
 - igzip_lib.h, 60
- isal_huff_histogram, 20
- isal_hufftables, 21
- isal_inflate
 - igzip_lib.h, 62
- isal_inflate_init
 - igzip_lib.h, 62
- isal_inflate_reset
 - igzip_lib.h, 63
- isal_inflate_set_dict
 - igzip_lib.h, 63
- isal_inflate_stateless
 - igzip_lib.h, 64
- isal_mod_hist, 22
- isal_read_gzip_header
 - igzip_lib.h, 64
- isal_read_zlib_header
 - igzip_lib.h, 65
- isal_update_histogram
 - igzip_lib.h, 65
- isal_write_gzip_header
 - igzip_lib.h, 66
- isal_write_zlib_header
 - igzip_lib.h, 66
- isal_zero_detect
 - mem_routines.h, 68
- isal_zlib_header, 22
- isal_zstate, 22
- isal_zstate_state
 - igzip_lib.h, 54
- isal_zstream, 23

- mem_routines.h, 67
 - isal_zero_detect, 68

- pq_check
 - raid.h, 69
- pq_check_base
 - raid.h, 69
- pq_gen
 - raid.h, 70
- pq_gen_base
 - raid.h, 70

- raid.h, 68
 - pq_check, 69
 - pq_check_base, 69
 - pq_gen, 70
 - pq_gen_base, 70
 - xor_check, 71
 - xor_check_base, 71
 - xor_gen, 72
 - xor_gen_base, 72

- xor_check
 - raid.h, 71
- xor_check_base
 - raid.h, 71
- xor_gen
 - raid.h, 72
- xor_gen_base
 - raid.h, 72

- ZSTATE_BODY
 - igzip_lib.h, 55
- ZSTATE_CREATE_HDR
 - igzip_lib.h, 55
- ZSTATE_END
 - igzip_lib.h, 55
- ZSTATE_FLUSH_READ_BUFFER
 - igzip_lib.h, 55
- ZSTATE_FLUSH_WRITE_BUFFER
 - igzip_lib.h, 55
- ZSTATE_HDR
 - igzip_lib.h, 55
- ZSTATE_NEW_HDR
 - igzip_lib.h, 55
- ZSTATE_SYNC_FLUSH
 - igzip_lib.h, 55
- ZSTATE_TMP_BODY
 - igzip_lib.h, 55
- ZSTATE_TMP_CREATE_HDR
 - igzip_lib.h, 55
- ZSTATE_TMP_END
 - igzip_lib.h, 55
- ZSTATE_TMP_FLUSH_READ_BUFFER
 - igzip_lib.h, 55
- ZSTATE_TMP_FLUSH_WRITE_BUFFER
 - igzip_lib.h, 55
- ZSTATE_TMP_HDR
 - igzip_lib.h, 55
- ZSTATE_TMP_NEW_HDR
 - igzip_lib.h, 55
- ZSTATE_TMP_SYNC_FLUSH
 - igzip_lib.h, 55
- ZSTATE_TMP_TRL
 - igzip_lib.h, 55
- ZSTATE_TMP_TYPE0_BODY
 - igzip_lib.h, 55

ZSTATE_TRL
 igzip_lib.h, [55](#)
ZSTATE_TYPE0_BODY
 igzip_lib.h, [55](#)