Commit c75ae9c

pmqs authored and jkeenan committed last month · 34 / 35

kid51

```
cpan/Compress-Raw-Zlib - Update to version 2.221

  2.221 27 February 2026

    * Merge remote-tracking branch 'refs/remotes/origin/master'
      Fri Feb 27 12:57:24 2026 +0000
      34f0318c9687924347284704c6f3b9b2566e07fd

    * Merge pull request #39 from jkeenan/correct-changes-date-20260227
      Fri Feb 27 12:50:53 2026 +0000
      e879453ec96d85b2917ed903a3f03cf8264238f1

    * Update to 2.221
      Fri Feb 27 12:44:53 2026 +0000
      17f707ed1986acf6773d91ebf6b5c7d6ba6d32be

    * Version number wrong in Changes file. Fixes #38
      Fri Feb 27 12:38:32 2026 +0000
      2c10524538077d20779c483c667e2ead4c65a87a

  2.220 25 February 2026

    * Update to version 2.220
      Wed Feb 25 13:50:23 2026 +0000
      f9ac640e71a54ad2f8156a075127f8f99ad45586

    * zlib 1.3.2: Update references to zlib 1.3.1 to use zlib 1.3.2
      Wed Feb 25 13:45:59 2026 +0000
      16b5fc92e830af8b10c37e6a3e66806cc0db937f

    * zlib 1.3.2: Add "Perl_crz" prefix to "z_errmsg". Fixes #32
      Wed Feb 25 13:41:07 2026 +0000
      36b42f013ed5d3e6b27ce956c97b8ce5733cfe24

    * Fix spelling typo
      Tue Feb 24 19:46:13 2026 +0000
      f73b8114faad246cf5c615c7e77e36d8978393bd

  2.219 23 February 2026

      * Update to version 2.219
        Mon Feb 23 15:19:30 2026 +0000
        c257a0b73e5f925549e20602cd42f9beea9db51e
```

```
    * zlib 1.3.2:  Add a few casts to allow zlib sources to build with C++
      Mon Feb 23 14:29:02 2026 +0000
      0049fc058a9b9c1de250b00af38edf3e91cb15eb

    * Test workflows with upstream zlib 1.32
      Mon Feb 23 14:15:02 2026 +0000
      d3b865abdb11c0479d04f8b66a6350b7aad9d5b3

      * zlib 1.3.2: Add "Perl_crz_" prefix  to exported symbols
      Mon Feb 23 14:02:50 2026 +0000
      37cec4a39cc6080a29060944db89f90c614c1cd4

    * zlib-1.3.2: Force include of <stddef.h> to get definition for NULL
      Mon Feb 23 13:38:39 2026 +0000
      8907f13c905eb3a9e65514bfbbf35304a92fa76a

    * Refresh zlib-src directory with unmodified zlib-1.3.2 files
      Mon Feb 23 13:31:42 2026 +0000
      48a5180b706c1066af212656e32d73c74850c408
```

blead · v5.43.9

1 parent bf65533 commit c75ae9c

19 files changed +939 -722 lines changed

Top

Filter files…

- Porting
  - Maintainers.pl
- cpan/Compress-Raw-Zlib
  - lib/Compress/Raw
    - Zlib.pm
  - zlib-src
    - compress.c
    - crc32.c
    - deflate.c
    - deflate.h
    - infback.c
    - inffast.c

inffixed.h
inflate.c
inflate.h
inftrees.c
inftrees.h
trees.c
uncompr.c
zconf.h
zlib.h
zutil.c
zutil.h

19 files changed +939 -722 lines changed

Search within code

Porting/Maintainers.pl

```
@@ -231,8 +231,8 @@ package Maintainers;
231 231         },
232 232
233 233         'Compress::Raw::Zlib' => {
234     -           'DISTRIBUTION' => 'PMQS/Compress-Raw-Zlib-2.218.tar.gz',
235     -           'SYNCINFO'     => 'jkeenan on Wed Feb 11 11:27:39 2026',
    234 +           'DISTRIBUTION' => 'PMQS/Compress-Raw-Zlib-2.221.tar.gz',
    235 +           'SYNCINFO'     => 'jkeenan on Fri Feb 27 10:16:27 2026',
236 236             'FILES'    => q[cpan/Compress-Raw-Zlib],
237 237             'EXCLUDED' => [
238 238                 qr{^examples/},
```

.../Compress-Raw-Zlib/lib/Compress/Raw/Zlib.pm

```
@@ -10,7 +10,7 @@ use warnings ;
10 10  use bytes ;
11 11  our ($VERSION, $XS_VERSION, @ISA, @EXPORT, %EXPORT_TAGS, @EXPORT_OK, $AUTOLOAD,
       %DEFLATE_CONSTANTS, @DEFLATE_CONSTANTS);
12 12
13    - $VERSION = '2.218';
```

```
      13 + $VERSION = '2.221';
14    14   $XS_VERSION = $VERSION;
15    15   $VERSION = eval $VERSION;
16    16
```

cpan/Compress-Raw-Zlib/zlib-src/compress.c

```
      ••• @@ -1,5 +1,5 @@
1     1    /* compress.c -- compress a memory buffer
2           - * Copyright (C) 1995-2005, 2014, 2016 Jean-loup Gailly, Mark Adler
      2    + * Copyright (C) 1995-2026 Jean-loup Gailly, Mark Adler
3     3     * For conditions of distribution and use, see copyright notice in zlib.h
4     4     */
5     5
      @@ -18,13 +18,19 @@
18    18         compress2 returns Z_OK if success, Z_MEM_ERROR if there was not enough
19    19       memory, Z_BUF_ERROR if there was not enough room in the output buffer,
20    20       Z_STREAM_ERROR if the level parameter is invalid.
      21   +
      22   +     The _z versions of the functions take size_t length arguments.
21    23   */
22          - int ZEXPORT compress2(Bytef *dest, uLongf *destLen, const Bytef *source,
23          -                       uLong sourceLen, int level) {
      24   + int ZEXPORT compress2_z(Bytef *dest, z_size_t *destLen, const Bytef *source,
      25   +                         z_size_t sourceLen, int level) {
24    26       z_stream stream;
25    27       int err;
26    28       const uInt max = (uInt)-1;
27          -    uLong left;
      29   +    z_size_t left;
      30   +
      31   +    if ((sourceLen > 0 && source == NULL) ||
      32   +        destLen == NULL || (*destLen > 0 && dest == NULL))
      33   +        return Z_STREAM_ERROR;
28    34
29    35       left = *destLen;
30    36       *destLen = 0;
      @@ -43,23 +49,36 @@ int ZEXPORT compress2(Bytef *dest, uLongf *destLen, const
      Bytef *source,
43    49
```

```
44  50          do {
45  51              if (stream.avail_out == 0) {
46      -               stream.avail_out = left > (uLong)max ? max : (uInt)left;
        52  +           stream.avail_out = left > (z_size_t)max ? max : (uInt)left;
47  53                  left -= stream.avail_out;
48  54              }
49  55              if (stream.avail_in == 0) {
50      -               stream.avail_in = sourceLen > (uLong)max ? max : (uInt)sourceLen;
        56  +           stream.avail_in = sourceLen > (z_size_t)max ? max :
        57  +                                                        (uInt)sourceLen;
51  58                  sourceLen -= stream.avail_in;
52  59              }
53  60              err = deflate(&stream, sourceLen ? Z_NO_FLUSH : Z_FINISH);
54  61          } while (err == Z_OK);
55  62
56      -       *destLen = stream.total_out;
        63  +   *destLen = (z_size_t)(stream.next_out - dest);
57  64          deflateEnd(&stream);
58  65          return err == Z_STREAM_END ? Z_OK : err;
59  66      }
60      -
        67  + int ZEXPORT compress2(Bytef *dest, uLongf *destLen, const Bytef *source,
        68  +                       uLong sourceLen, int level) {
        69  +     int ret;
        70  +     z_size_t got = *destLen;
        71  +     ret = compress2_z(dest, &got, source, sourceLen, level);
        72  +     *destLen = (uLong)got;
        73  +     return ret;
        74  + }
61  75      /* ===========================================================================
62  76       */
        77  + int ZEXPORT compress_z(Bytef *dest, z_size_t *destLen, const Bytef *source,
        78  +                        z_size_t sourceLen) {
        79  +     return compress2_z(dest, destLen, source, sourceLen,
        80  +                        Z_DEFAULT_COMPRESSION);
        81  + }
63  82      int ZEXPORT compress(Bytef *dest, uLongf *destLen, const Bytef *source,
64  83                           uLong sourceLen) {
65  84          return compress2(dest, destLen, source, sourceLen, Z_DEFAULT_COMPRESSION);
```

```
@@ -69,7 +88,12 @@ int ZEXPORT compress(Bytef *dest, uLongf *destLen, const
Bytef *source,
69  88       If the default memLevel or windowBits for deflateInit() is changed, then
70  89     this function needs to be updated.
71  90    */
    91  + z_size_t ZEXPORT compressBound_z(z_size_t sourceLen) {
    92  +     z_size_t bound = sourceLen + (sourceLen >> 12) + (sourceLen >> 14) +
    93  +                      (sourceLen >> 25) + 13;
    94  +     return bound < sourceLen ? (z_size_t)-1 : bound;
    95  + }
72  96    uLong ZEXPORT compressBound(uLong sourceLen) {
73      -     return sourceLen + (sourceLen >> 12) + (sourceLen >> 14) +
74      -            (sourceLen >> 25) + 13;
    97  +     z_size_t bound = compressBound_z(sourceLen);
    98  +     return (uLong)bound != bound ? (uLong)-1 : (uLong)bound;
75  99    }
```

cpan/Compress-Raw-Zlib/zlib-src/crc32.c

```
    @@ -1,5 +1,5 @@
1   1    /* crc32.c -- compute the CRC-32 of a data stream
2       -  * Copyright (C) 1995-2022 Mark Adler
        2  +  * Copyright (C) 1995-2026 Mark Adler
3   3     * For conditions of distribution and use, see copyright notice in zlib.h
4   4     *
5   5     * This interleaved implementation of a CRC makes use of pipelined multiple
    @@ -24,11 +24,18 @@
24  24   #  include <stdio.h>
25  25   #  ifndef DYNAMIC_CRC_TABLE
26  26   #    define DYNAMIC_CRC_TABLE
27      - #  endif /* !DYNAMIC_CRC_TABLE */
28      - #endif /* MAKECRCH */
        27 + #  endif
        28 + #endif
        29 + #ifdef DYNAMIC_CRC_TABLE
        30 + #  define Z_ONCE
        31 + #endif
29  32
30  33   #include "zutil.h"      /* for Z_U4, Z_U8, z_crc_t, and FAR definitions */
31  34
        35 + #ifdef HAVE_S390X_VX
```

```
                 +  #  include "contrib/crc32vx/crc32_vx_hooks.h"
                 +  #endif
                 +
 32      39        /*
 33      40          A CRC of a message is computed on N braids of words in the message, where
 34      41          each word consists of W bytes (4 or 8). If N is 3, for example, then three
                    @@ -99,7 +106,8 @@
 99      106       #endif
 100     107
 101     108       /* If available, use the ARM processor CRC32 instruction. */
 102             - #if defined(__aarch64__) && defined(__ARM_FEATURE_CRC32) && W == 8
                 + #if defined(__aarch64__) && defined(__ARM_FEATURE_CRC32) && \
                 +     defined(W) && W == 8
 103     111       #  define ARMCRC32
 104     112       #endif
 105     113
                    @@ -152,10 +160,10 @@ local z_word_t byte_swap(z_word_t word) {
 152     160         Return a(x) multiplied by b(x) modulo p(x), where p(x) is the CRC
                    polynomial,
 153     161         reflected. For speed, this requires that a not be zero.
 154     162        */
 155             - local z_crc_t multmodp(z_crc_t a, z_crc_t b) {
 156             -     z_crc_t m, p;
                 + local uLong multmodp(uLong a, uLong b) {
                 +     uLong m, p;
 157     165
 158             -     m = (z_crc_t)1 << 31;
                 +     m = (uLong)1 << 31;
 159     167         p = 0;
 160     168         for (;;) {
 161     169             if (a & m) {
                    @@ -171,12 +179,12 @@ local z_crc_t multmodp(z_crc_t a, z_crc_t b) {
 171     179
 172     180       /*
 173     181         Return x^(n * 2^k) modulo p(x). Requires that x2n_table[] has been
 174             -   initialized.
                 +   initialized. n must not be negative.
 175     183        */
```

```
176       - local z_crc_t x2nmodp(z_off64_t n, unsigned k) {
177       -     z_crc_t p;
      184 + local uLong x2nmodp(z_off64_t n, unsigned k) {
      185 +     uLong p;
178   186
179       -     p = (z_crc_t)1 << 31;           /* x^0 == 1 */
      187 +     p = (uLong)1 << 31;             /* x^0 == 1 */
180   188       while (n) {
181   189           if (n & 1)
182   190               p = multmodp(x2n_table[k & 31], p);
          @@ -204,83 +212,8 @@ local z_crc_t FAR crc_table[256];
204   212       local void write_table64(FILE *, const z_word_t FAR *, int);
205   213   #endif /* MAKECRCH */
206   214
207       - /*
208       -   Define a once() function depending on the availability of atomics. If this
          is
209       -   compiled with DYNAMIC_CRC_TABLE defined, and if CRCs will be computed in
210       -   multiple threads, and if atomics are not available, then get_crc_table()
          must
211       -   be called to initialize the tables and must return before any threads are
212       -   allowed to compute or combine CRCs.
213       -  */
214       -
215       - /* Definition of once functionality. */
216       - typedef struct once_s once_t;
217       -
218       - /* Check for the availability of atomics. */
219       - #if defined(__STDC__) && __STDC_VERSION__ >= 201112L && \
220       -     !defined(__STDC_NO_ATOMICS__)
221       -
222       - #include <stdatomic.h>
223       -
224       - /* Structure for once(), which must be initialized with ONCE_INIT. */
225       - struct once_s {
226       -     atomic_flag begun;
227       -     atomic_int done;
228       - };
229       - #define ONCE_INIT {ATOMIC_FLAG_INIT, 0}
```

```
-
- /*
-   Run the provided init() function exactly once, even if multiple threads
-   invoke once() at the same time. The state must be a once_t initialized with
-   ONCE_INIT.
-  */
- local void once(once_t *state, void (*init)(void)) {
-     if (!atomic_load(&state->done)) {
-         if (atomic_flag_test_and_set(&state->begun))
-             while (!atomic_load(&state->done))
-                 ;
-         else {
-             init();
-             atomic_store(&state->done, 1);
-         }
-     }
- }
-
- #else   /* no atomics */
-
- /* Structure for once(), which must be initialized with ONCE_INIT. */
- struct once_s {
-     volatile int begun;
-     volatile int done;
- };
- #define ONCE_INIT {0, 0}
-
- /* Test and set. Alas, not atomic, but tries to minimize the period of
-    vulnerability. */
- local int test_and_set(int volatile *flag) {
-     int was;
-
-     was = *flag;
-     *flag = 1;
-     return was;
- }
-
- /* Run the provided init() function once. This is not thread-safe. */
- local void once(once_t *state, void (*init)(void)) {
-     if (!state->done) {
```

```
270      -          if (test_and_set(&state->begun))
271      -              while (!state->done)
272      -                  ;
273      -          else {
274      -              init();
275      -              state->done = 1;
276      -          }
277      -      }
278      - }
279      -
280      - #endif
281      -
282  215     /* State for once(). */
283      - local once_t made = ONCE_INIT;
     216 + local z_once_t made = Z_ONCE_INIT;
284  217
285  218     /*
286  219       Generate tables for a byte-wise 32-bit CRC calculation on the polynomial:
         @@ -326,7 +259,7 @@ local void make_crc_table(void) {
326  259         p = (z_crc_t)1 << 30;         /* x^1 */
327  260         x2n_table[0] = p;
328  261         for (n = 1; n < 32; n++)
329      -           x2n_table[n] = p = multmodp(p, p);
     262 +           x2n_table[n] = p = (z_crc_t)multmodp(p, p);
330  263
331  264     #ifdef W
332  265         /* initialize the braiding tables -- needs x2n_table[] */
         @@ -529,11 +462,11 @@ local void braid(z_crc_t ltl[][256], z_word_t big[]
         [256], int n, int w) {
529  462         int k;
530  463         z_crc_t i, p, q;
531  464         for (k = 0; k < w; k++) {
532      -           p = x2nmodp((n * w + 3 - k) << 3, 0);
     465 +           p = (z_crc_t)x2nmodp((n * w + 3 - k) << 3, 0);
533  466             ltl[k][0] = 0;
534  467             big[w - 1 - k][0] = 0;
535  468             for (i = 1; i < 256; i++) {
536      -               ltl[k][i] = q = multmodp(i << 24, p);
     469 +               ltl[k][i] = q = (z_crc_t)multmodp(i << 24, p);
```

```
537  470                big[w - 1 - k][i] = byte_swap(q);
538  471            }
539  472        }
     @@ -548,7 +481,7 @@ local void braid(z_crc_t ltl[][256], z_word_t big[]
     [256], int n, int w) {
548  481     */
549  482    const z_crc_t FAR * ZEXPORT get_crc_table(void) {
550  483    #ifdef DYNAMIC_CRC_TABLE
551       -    once(&made, make_crc_table);
     484  +    z_once(&made, make_crc_table);
552  485    #endif /* DYNAMIC_CRC_TABLE */
553  486        return (const z_crc_t FAR *)crc_table;
554  487    }
     @@ -572,9 +505,8 @@ const z_crc_t FAR * ZEXPORT get_crc_table(void) {
572  505    #define Z_BATCH_ZEROS 0xa10d3d0c    /* computed from Z_BATCH = 3990 */
573  506    #define Z_BATCH_MIN 800             /* fewest words in a final batch */
574  507
575       - unsigned long ZEXPORT crc32_z(unsigned long crc, const unsigned char FAR
          *buf,
576       -                               z_size_t len) {
577       -    z_crc_t val;
     508  + uLong ZEXPORT crc32_z(uLong crc, const unsigned char FAR *buf, z_size_t len)
          {
     509  +    uLong val;
578  510        z_word_t crc1, crc2;
579  511        const z_word_t *word;
580  512        z_word_t val0, val1, val2;
     @@ -585,7 +517,7 @@ unsigned long ZEXPORT crc32_z(unsigned long crc,
     const unsigned char FAR *buf,
585  517        if (buf == Z_NULL) return 0;
586  518
587  519    #ifdef DYNAMIC_CRC_TABLE
588       -    once(&made, make_crc_table);
     520  +    z_once(&made, make_crc_table);
589  521    #endif /* DYNAMIC_CRC_TABLE */
590  522
591  523        /* Pre-condition the CRC */
     @@ -640,7 +572,7 @@ unsigned long ZEXPORT crc32_z(unsigned long crc,
     const unsigned char FAR *buf,
640  572            }
```

```
641  573            word += 3 * last;
642  574            num -= 3 * last;
643      -          val = x2nmodp(last, 6);
     575  +          val = x2nmodp((int)last, 6);
644  576            crc = multmodp(val, crc) ^ crc1;
645  577            crc = multmodp(val, crc) ^ crc2;
646  578        }
          @@ -691,13 +623,12 @@ local z_word_t crc_word_big(z_word_t data) {
691  623    #endif
692  624
693  625    /* ========================================================================
               */
694      - unsigned long ZEXPORT crc32_z(unsigned long crc, const unsigned char FAR
               *buf,
695      -                               z_size_t len) {
     626  + uLong ZEXPORT crc32_z(uLong crc, const unsigned char FAR *buf, z_size_t len)
               {
696  627        /* Return initial CRC, if requested. */
697  628        if (buf == Z_NULL) return 0;
698  629
699  630    #ifdef DYNAMIC_CRC_TABLE
700      -     once(&made, make_crc_table);
     631  +     z_once(&made, make_crc_table);
701  632    #endif /* DYNAMIC_CRC_TABLE */
702  633
703  634        /* Pre-condition the CRC */
          @@ -1012,38 +943,41 @@ unsigned long ZEXPORT crc32_z(unsigned long crc,
               const unsigned char FAR *buf,
1012 943    #endif
1013 944
1014 945    /* ========================================================================
               */
1015     - unsigned long ZEXPORT crc32(unsigned long crc, const unsigned char FAR *buf,
1016     -                             uInt len) {
     946  + uLong ZEXPORT crc32(uLong crc, const unsigned char FAR *buf, uInt len) {
     947  +     #ifdef HAVE_S390X_VX
     948  +     return crc32_z_hook(crc, buf, len);
     949  +     #endif
1017 950        return crc32_z(crc, buf, len);
```

```
1018  951     }
1019  952
1020  953     /* ========================================================================
                */
1021        - uLong ZEXPORT crc32_combine64(uLong crc1, uLong crc2, z_off64_t len2) {
      954   + uLong ZEXPORT crc32_combine_gen64(z_off64_t len2) {
      955   +     if (len2 < 0)
      956   +         return 0;
1022  957     #ifdef DYNAMIC_CRC_TABLE
1023        -     once(&made, make_crc_table);
      958   +     z_once(&made, make_crc_table);
1024  959     #endif /* DYNAMIC_CRC_TABLE */
1025        -     return multmodp(x2nmodp(len2, 3), crc1) ^ (crc2 & 0xffffffff);
      960   +     return x2nmodp(len2, 3);
1026  961     }
1027  962
1028  963     /* ========================================================================
                */
1029        - uLong ZEXPORT crc32_combine(uLong crc1, uLong crc2, z_off_t len2) {
1030        -     return crc32_combine64(crc1, crc2, (z_off64_t)len2);
      964   + uLong ZEXPORT crc32_combine_gen(z_off_t len2) {
      965   +     return crc32_combine_gen64((z_off64_t)len2);
1031  966     }
1032  967
1033  968     /* ========================================================================
                */
1034        - uLong ZEXPORT crc32_combine_gen64(z_off64_t len2) {
1035        - #ifdef DYNAMIC_CRC_TABLE
1036        -     once(&made, make_crc_table);
1037        - #endif /* DYNAMIC_CRC_TABLE */
1038        -     return x2nmodp(len2, 3);
      969   + uLong ZEXPORT crc32_combine_op(uLong crc1, uLong crc2, uLong op) {
      970   +     if (op == 0)
      971   +         return 0;
      972   +     return multmodp(op, crc1 & 0xffffffff) ^ (crc2 & 0xffffffff);
1039  973     }
1040  974
1041  975     /* ========================================================================
                */
1042        - uLong ZEXPORT crc32_combine_gen(z_off_t len2) {
```

```
1043      -     return crc32_combine_gen64((z_off64_t)len2);
          976 + uLong ZEXPORT crc32_combine64(uLong crc1, uLong crc2, z_off64_t len2) {
          977 +     return crc32_combine_op(crc1, crc2, crc32_combine_gen64(len2));
1044      978   }
1045      979
1046      980   /* ========================================================================
                */
1047      -   uLong ZEXPORT crc32_combine_op(uLong crc1, uLong crc2, uLong op) {
1048      -       return multmodp(op, crc1) ^ (crc2 & 0xffffffff);
          981 + uLong ZEXPORT crc32_combine(uLong crc1, uLong crc2, z_off_t len2) {
          982 +     return crc32_combine64(crc1, crc2, (z_off64_t)len2);
1049      983   }
```

Comments 0

Please sign in to comment.