```
**********************************************************
    2303 Mon Mar 23 21:41:44 2015
new/usr/src/cmd/sgs/elfdump/Makefile.com
5688 ELF tools need to be more careful with dwarf data
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1997, 2010, Oracle and/or its affiliates. All rights reserved.
#

PROG=           elfdump

include         $(SRC)/cmd/Makefile.cmd
include         $(SRC)/cmd/sgs/Makefile.com

COMOBJ =        main.o                  corenote.o \
                dwarf.o                 struct_layout.o \
                struct_layout_i386.o    struct_layout_amd64.o \
                struct_layout_sparc.o   struct_layout_sparcv9.o

COMOBJ32 =      elfdump32.o fake_shdr32.o

COMOBJ64 =      elfdump64.o fake_shdr64.o

TOOLOBJ =       leb128.o

BLTOBJ =        msg.o

OBJS=           $(BLTOBJ) $(COMOBJ) $(COMOBJ32) $(COMOBJ64) $(TOOLOBJ)

MAPFILE=        $(MAPFILE.NGB)
MAPOPT=         $(MAPFILE:%=-M%)

CPPFLAGS=       -I. -I../common -I../../include -I../../include/$(MACH) \
                -I$(SRCBASE)/lib/libc/inc -I$(SRCBASE)/uts/$(ARCH)/sys \
                $(CPPFLAGS.master) -I$(ELFCAP)
LLDFLAGS =      $(VAR_ELFDUMP_LLDFLAGS)
LLDFLAGS64 =    $(VAR_ELFDUMP_LLDFLAGS64)
LDFLAGS +=      $(VERSREF) $(CC_USE_PROTO) $(MAPOPT) $(LLDFLAGS)
LDLIBS +=       $(ELFLIBDIR) -lelf $(LDDBGLIBDIR) $(LDDBG_LIB) \
                    $(CONVLIBDIR) $(CONV_LIB)

LINTFLAGS +=    -x
LINTFLAGS64 +=  -x

CERRWARN +=     -_gcc=-Wno-uninitialized
```



```
CERRWARN +=     -_gcc=-Wno-switch

BLTDEFS =       msg.h
BLTDATA =       msg.c
BLTMESG =       $(SGSMSGDIR)/elfdump

BLTFILES =      $(BLTDEFS) $(BLTDATA) $(BLTMESG)

SGSMSGCOM =     ../common/elfdump.msg
SGSMSGTARG =    $(SGSMSGCOM)
SGSMSGALL =     $(SGSMSGCOM)
SGSMSGFLAGS +=  -h $(BLTDEFS) -d $(BLTDATA) -m $(BLTMESG) -n elfdump_msg

SRCS =          $(COMOBJ:%.o=../common/%.c) \
                $(COMOBJ32:%32.o=../common/%.c) \
                $(TOOLOBJ:%.o=../../tools/common/%.c) $(BLTDATA)
LINTSRCS =      $(SRCS) ../common/lintsup.c

CLEANFILES +=   $(LINTOUTS) $(BLTFILES) gen_struct_layout
```

```
**********************************************************
    7201 Mon Mar 23 21:41:45 2015
new/usr/src/cmd/sgs/elfdump/common/_elfdump.h
5688 ELF tools need to be more careful with dwarf data
**********************************************************
______unchanged_portion_omitted_

 172 /*
 173  * Define various elfdump() functions into their 32-bit and 64-bit variants.
 174  */
 175 #if     defined(_ELF64)
 176 #define cap                     cap64
 177 #define checksum                checksum64
 178 #define dynamic                 dynamic64
 179 #define fake_shdr_cache         fake_shdr_cache64
 180 #define fake_shdr_cache_free    fake_shdr_cache_free64
 181 #define got                     got64
 182 #define group                   group64
 183 #define hash                    hash64
 184 #define interp                  interp64
 185 #define move                    move64
 186 #define note                    note64
 187 #define note_entry              note_entry64
 188 #define regular                 regular64
 189 #define reloc                   reloc64
 190 #define sections                sections64
 191 #define string                  string64
 192 #define symbols                 symbols64
 193 #define syminfo                 syminfo64
 194 #define symlookup               symlookup64
 195 #define unwind                  unwind64
 196 #define versions                versions64
 197 #define version_def             version_def64
 198 #define version_need            version_need64
 199 #else
 200 #define cap                     cap32
 201 #define checksum                checksum32
 202 #define dynamic                 dynamic32
 203 #define fake_shdr_cache         fake_shdr_cache32
 204 #define fake_shdr_cache_free    fake_shdr_cache_free32
 205 #define got                     got32
 206 #define group                   group32
 207 #define hash                    hash32
 208 #define interp                  interp32
 209 #define move                    move32
 210 #define note                    note32
 211 #define note_entry              note_entry32
 212 #define regular                 regular32
 213 #define reloc                   reloc32
 214 #define sections                sections32
 215 #define string                  string32
 216 #define symbols                 symbols32
 217 #define syminfo                 syminfo32
 218 #define symlookup               symlookup32
 219 #define unwind                  unwind32
 220 #define versions                versions32
 221 #define version_def             version_def32
 222 #define version_need            version_need32
 223 #endif

 225 extern  corenote_ret_t  corenote(Half, int, Word, const char *, Word);
 226 extern  void    dump_eh_frame(const char *, char *, uchar_t *, size_t, uint64_t,
 227                     Half e_machine, uchar_t *e_ident, uint64_t gotaddr);
 226 extern  void    dump_eh_frame(uchar_t *, size_t, uint64_t, Half e_machine,
 227                     uchar_t *e_ident, uint64_t gotaddr);
 228 extern  void    dump_hex_bytes(const void *, size_t, int, int, int);
```



```
 230 extern  int     fake_shdr_cache32(const char *, int, Elf *, Elf32_Ehdr *,
 231                     Cache **, size_t *);
 232 extern  int     fake_shdr_cache64(const char *, int, Elf *, Elf64_Ehdr *,
 233                     Cache **, size_t *);

 235 extern  void    fake_shdr_cache_free32(Cache *, size_t);
 236 extern  void    fake_shdr_cache_free64(Cache *, size_t);

 238 extern  int     regular32(const char *, int, Elf *, uint_t, const char *, int,
 239                     uchar_t);
 240 extern  int     regular64(const char *, int, Elf *, uint_t, const char *, int,
 241                     uchar_t);

 243 #ifdef  __cplusplus
 244 }
______unchanged_portion_omitted_
```

```
**********************************************************
   27536 Mon Mar 23 21:41:46 2015
new/usr/src/cmd/sgs/elfdump/common/dwarf.c
5688 ELF tools need to be more careful with dwarf data
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include        <_libelf.h>
#include        <dwarf.h>
#include        <stdio.h>
#include        <unistd.h>
#include        <errno.h>
#include        <strings.h>
#include        <debug.h>
#include        <conv.h>
#include        <msg.h>
#include        <_elfdump.h>


/*
 * Data from eh_frame section used by dump_cfi()
 */
typedef struct {
        const char      *file;
        const char      *sh_name;
#endif /* ! codereview */
        Half            e_machine;      /* ehdr->e_machine */
        uchar_t         *e_ident;       /* ehdr->e_ident */
        uint64_t        sh_addr;        /* Address of eh_frame section */
        int             do_swap;        /* True if object and system byte */
                                        /*      order differs */
        int             cieRflag;       /* R flag from current CIE */
        uint64_t        ciecalign;      /* CIE code align factor */
        int64_t         ciedalign;      /* CIE data align factor */
        uint64_t        fdeinitloc;     /* FDE initial location */
        uint64_t        gotaddr;        /* Address of the GOT */
} dump_cfi_state_t;


/*
 * Extract an unsigned integer value from an .eh_frame section, converting it
 * from its native byte order to that of the running machine if necessary.
```



```
 *
 * entry:
 *      data - Base address from which to extract datum
 *      ndx - Address of variable giving index to start byte in data.
 *      size - # of bytes in datum. Must be one of: 1, 2, 4, 8
 *      do_swap - True if the data is in a different byte order than that
 *              of the host system.
 *
 * exit:
 *      *ndx is incremented by the size of the extracted datum.
 *
 *      The requested datum is extracted, byte swapped if necessary,
 *      and returned.
 */
static dwarf_error_t
dwarf_extract_uint(uchar_t *data, size_t len, uint64_t *ndx, int size,
    int do_swap, uint64_t *ret)
static uint64_t
dwarf_extract_uint(uchar_t *data, uint64_t *ndx, int size, int do_swap)
{
        if (((*ndx + size) > len) ||
            ((*ndx + size) < *ndx))
                return (DW_OVERFLOW);

#endif /* ! codereview */
        switch (size) {
        case 1:
                *ret = (data[(*ndx)++]);
                return (DW_SUCCESS);
                return (data[(*ndx)++]);
        case 2:
                {
                        Half    r;
                        uchar_t *p = (uchar_t *)&r;

                        data += *ndx;
                        if (do_swap)
                                UL_ASSIGN_BSWAP_HALF(p, data);
                        else
                                UL_ASSIGN_HALF(p, data);

                        (*ndx) += 2;
                        *ret = r;
                        return (DW_SUCCESS);
                        return (r);
                }
        case 4:
                {
                        Word    r;
                        uchar_t *p = (uchar_t *)&r;

                        data += *ndx;
                        if (do_swap)
                                UL_ASSIGN_BSWAP_WORD(p, data);
                        else
                                UL_ASSIGN_WORD(p, data);

                        (*ndx) += 4;
                        *ret = r;
                        return (DW_SUCCESS);
                        return (r);
                }

        case 8:
                {
                        uint64_t        r;
```

```
                        uchar_t         *p = (uchar_t *)&r;

                        data += *ndx;
                        if (do_swap)
                                UL_ASSIGN_BSWAP_LWORD(p, data);
                        else
                                UL_ASSIGN_LWORD(p, data);

                        (*ndx) += 8;
                        *ret = r;
                        return (DW_SUCCESS);
                        return (r);
                }
        default:
                return (DW_BAD_ENCODING);
#endif /* ! codereview */
        }

        /* NOTREACHED */
        /* If here, an invalid size was specified */
        assert(0);
        return (0);
}
```

***unchanged_portion_omitted_***

```
/*
 * Decode eh_frame Call Frame Instructions, printing each one on a
 * separate line.
 *
 * entry:
 *      data - Address of base of eh_frame section being processed
 *      off - Offset of current FDE within eh_frame
 *      ndx - Index of current position within current FDE
 *      len - Length of FDE
 *      len - Length of eh_frame section
 *      state - Object, CIE, and FDE state for current request
 *      msg - Header message to issue before producing output.
 *      indent - # of indentation characters issued for each line of output.
 *
 * exit:
 *      The Call Frame Instructions have been decoded and printed.
 *
 *      *ndx has been incremented to contain the index of the next
 *              byte of data to be processed in eh_frame.
 *
 * note:
 *      The format of Call Frame Instructions in .eh_frame sections is based
 *      on the DWARF specification.
 */
static void
dump_cfi(uchar_t *data, uint64_t off, uint64_t *ndx, uint_t len,
    dump_cfi_state_t *state, const char *msg, int indent)
{
        /*
         * We use %*s%s to insert leading whitespace and the op name.
         * PREFIX supplies these arguments.
         */
#define PREFIX  indent, MSG_ORIG(MSG_STR_EMPTY), opname

        /* Hide boilerplate clutter in calls to dwarf_regname() */
#define REGNAME(_rnum, _buf) \
        dwarf_regname(state->e_machine, _rnum, _buf, sizeof (_buf))

        /* Extract the lower 6 bits from an op code */
#define LOW_OP(_op) (_op & 0x3f)
```



```
        char            rbuf1[32], rbuf2[32];
        Conv_inv_buf_t  inv_buf;
        uchar_t         op;
        const char      *opname;
        uint64_t        oper1, oper2, cur_pc;
        int64_t         soper;
        const char      *loc_str;
        int             i;

        dbg_print(0, msg);

        /*
         * In a CIE/FDE, the length field does not include it's own
         * size. Hence, the value passed in is 4 less than the index
         * of the actual final location.
         */
        len += 4;

        /*
         * There is a concept of the 'current location', which is the PC
         * to which the current item applies. It starts out set to the
         * FDE initial location, and can be set or incremented by
         * various OP codes. cur_pc is used to track this.
         *
         * We want to use 'initloc' in the output the first time the location
         * is referenced, and then switch to 'loc' for subsequent references.
         * loc_str is used to manage that.
         */
        cur_pc = state->fdeinitloc;
        loc_str = MSG_ORIG(MSG_STR_INITLOC);

        while (*ndx < len) {
                /*
                 * The first byte contains the primary op code in the top
                 * 2 bits, so there are 4 of them. Primary OP code
                 * 0 uses the lower 6 bits to specify a sub-opcode, allowing
                 * for 64 of them. The other 3 primary op codes use the
                 * lower 6 bits to hold an operand (a register #, or value).
                 *
                 * Check the primary OP code. If it's 1-3, handle it
                 * and move to the next loop iteration. For OP code 0,
                 * fall through to decode the sub-code.
                 */
                op = data[off + (*ndx)++];
                opname = conv_dwarf_cfa(op, 0, &inv_buf);
                switch (op >> 6) {
                case 0x1:               /* v2: DW_CFA_advance_loc, delta */
                        oper1 = state->ciecalign * LOW_OP(op);
                        cur_pc += oper1;
                        dbg_print(0, MSG_ORIG(MSG_CFA_ADV_LOC), PREFIX,
                            loc_str, EC_XWORD(oper1), EC_XWORD(cur_pc));
                        loc_str = MSG_ORIG(MSG_STR_LOC);
                        continue;

                case 0x2:               /* v2: DW_CFA_offset, reg, offset */
                        if (uleb_extract(&data[off], ndx, len, &oper1) ==
                            DW_OVERFLOW) {
                                (void) fprintf(stderr,
                                    MSG_INTL(MSG_ERR_DWOVRFLW),
                                    state->file, state->sh_name);
                                return;
                        }

                        oper1 *= state->ciedalign;
                        soper = uleb_extract(&data[off], ndx) *
```

```
233                             state->ciedalign;
289                     dbg_print(0, MSG_ORIG(MSG_CFA_CFAOFF), PREFIX,
290                         REGNAME(LOW_OP(op), rbuf1), EC_XWORD(oper1));
235                         REGNAME(LOW_OP(op), rbuf1), EC_SXWORD(soper));
291                     continue;

293             case 0x3:               /* v2: DW_CFA_restore, reg */
294                     dbg_print(0, MSG_ORIG(MSG_CFA_REG), PREFIX,
295                         REGNAME(LOW_OP(op), rbuf1));
296                     continue;
297             }

299             /*
300              * If we're here, the high order 2 bits are 0. The low 6 bits
301              * specify a sub-opcode defining the operation.
302              */
303             switch (op) {
304             case 0x00:              /* v2: DW_CFA_nop */
305                     /*
306                      * No-ops are used to fill unused space required
307                      * for alignment. It is common for there to be
308                      * multiple adjacent nops. It saves space to report
309                      * them all with a single line of output.
310                      */
311                     for (i = 1;
312                         (*ndx < len) && (data[off + *ndx] == 0);
313                         i++, (*ndx)++)
314                             ;
315                     dbg_print(0, MSG_ORIG(MSG_CFA_SIMPLEREP), PREFIX, i);
316                     break;

318             case 0x0a:              /* v2: DW_CFA_remember_state */
319             case 0x0b:              /* v2: DW_CFA_restore_state */
320             case 0x2d:              /* GNU: DW_CFA_GNU_window_save */
321                     dbg_print(0, MSG_ORIG(MSG_CFA_SIMPLE), PREFIX);
322                     break;

324             case 0x01:              /* v2: DW_CFA_set_loc, address */
325                     switch (dwarf_ehe_extract(&data[off], len, ndx,
326                         &cur_pc, state->cieRflag, state->e_ident, B_FALSE,
327                         state->sh_addr, off + *ndx, state->gotaddr)) {
328                     case DW_OVERFLOW:
329                             (void) fprintf(stderr,
330                                 MSG_INTL(MSG_ERR_DWOVRFLW),
331                                 state->file, state->sh_name);
332                             return;
333                     case DW_BAD_ENCODING:
334                             (void) fprintf(stderr,
335                                 MSG_INTL(MSG_ERR_DWBADENC),
336                                 state->file, state->sh_name,
337                                 state->cieRflag);
338                             return;
339                     case DW_SUCCESS:
340                             break;
341                     }
270                     cur_pc = dwarf_ehe_extract(&data[off], ndx,
271                         state->cieRflag, state->e_ident, B_FALSE,
272                         state->sh_addr, off + *ndx, state->gotaddr);
342                     dbg_print(0, MSG_ORIG(MSG_CFA_CFASET), PREFIX,
343                         EC_XWORD(cur_pc));
344                     break;

346             case 0x02:      /* v2: DW_CFA_advance_loc_1, 1-byte delta */
347             case 0x03:      /* v2: DW_CFA_advance_loc_2, 2-byte delta */
348             case 0x04:      /* v2: DW_CFA_advance_loc_4, 4-byte delta */
349                     /*
```



```
350                      * Since the codes are contiguous, and the sizes are
351                      * powers of 2, we can compute the word width from
352                      * the code.
353                      */
354                     i = 1 << (op - 0x02);
355                     switch (dwarf_extract_uint(data + off, len,
356                         ndx, i, state->do_swap, &oper1)) {
357                     case DW_BAD_ENCODING:
358                             (void) fprintf(stderr,
359                                 MSG_INTL(MSG_ERR_DWBADENC),
360                                 state->file, state->sh_name,
361                                 i);
362                             return;
363                     case DW_OVERFLOW:
364                             (void) fprintf(stderr,
365                                 MSG_INTL(MSG_ERR_DWOVRFLW),
366                                 state->file, state->sh_name);
367                             return;
368                     case DW_SUCCESS:
369                             break;
370                     }
371                     oper1 *= state->ciecalign;
286                     oper1 = dwarf_extract_uint(data + off, ndx, i,
287                         state->do_swap) * state->ciecalign;
372                     cur_pc += oper1;
373                     dbg_print(0, MSG_ORIG(MSG_CFA_ADV_LOC), PREFIX,
374                         loc_str, EC_XWORD(oper1), EC_XWORD(cur_pc));
375                     loc_str = MSG_ORIG(MSG_STR_LOC);
376                     break;

378             case 0x05:              /* v2: DW_CFA_offset_extended,reg,off */
379                     if (uleb_extract(&data[off], ndx, len, &oper1) ==
380                         DW_OVERFLOW) {
381                             (void) fprintf(stderr,
382                                 MSG_INTL(MSG_ERR_DWOVRFLW),
383                                 state->file, state->sh_name);
384                             return;
385                     }

387                     if (sleb_extract(&data[off], ndx, len, &soper) ==
388                         DW_OVERFLOW) {
389                             (void) fprintf(stderr,
390                                 MSG_INTL(MSG_ERR_DWOVRFLW),
391                                 state->file, state->sh_name);
392                             return;
393                     }

395                     soper *= state->ciedalign;
295                     oper1 = uleb_extract(&data[off], ndx);
296                     soper = uleb_extract(&data[off], ndx) *
297                         state->ciedalign;
396                     dbg_print(0, MSG_ORIG(MSG_CFA_CFAOFF), PREFIX,
397                         REGNAME(oper1, rbuf1), EC_SXWORD(soper));
398                     break;

400             case 0x06:              /* v2: DW_CFA_restore_extended, reg */
401             case 0x0d:              /* v2: DW_CFA_def_cfa_register, reg */
402             case 0x08:              /* v2: DW_CFA_same_value, reg */
403             case 0x07:              /* v2: DW_CFA_undefined, reg */
404                     if (uleb_extract(&data[off], ndx, len, &oper1) ==
405                         DW_OVERFLOW) {
406                             (void) fprintf(stderr,
407                                 MSG_INTL(MSG_ERR_DWOVRFLW),
408                                 state->file, state->sh_name);
409                             return;
410                     }
```

```
306                         oper1 = uleb_extract(&data[off], ndx);
412                         dbg_print(0, MSG_ORIG(MSG_CFA_REG), PREFIX,
413                             REGNAME(oper1, rbuf1));
414                         break;


417                 case 0x09:              /* v2: DW_CFA_register, reg, reg */
418                         if (uleb_extract(&data[off], ndx, len, &oper1) ==
419                             DW_OVERFLOW) {
420                                 (void) fprintf(stderr,
421                                     MSG_INTL(MSG_ERR_DWOVRFLW),
422                                     state->file, state->sh_name);
423                                 return;
424                         }

426                         if (uleb_extract(&data[off], ndx, len, &oper2) ==
427                             DW_OVERFLOW) {
428                                 (void) fprintf(stderr,
429                                     MSG_INTL(MSG_ERR_DWOVRFLW),
430                                     state->file, state->sh_name);
431                                 return;
432                         }
313                         oper1 = uleb_extract(&data[off], ndx);
314                         oper2 = uleb_extract(&data[off], ndx);
433                         dbg_print(0, MSG_ORIG(MSG_CFA_REG_REG), PREFIX,
434                             REGNAME(oper1, rbuf1), REGNAME(oper2, rbuf2));
435                         break;

437                 case 0x0c:              /* v2: DW_CFA_def_cfa, reg, offset */
438                         if (uleb_extract(&data[off], ndx, len, &oper1) ==
439                             DW_OVERFLOW) {
440                                 (void) fprintf(stderr,
441                                     MSG_INTL(MSG_ERR_DWOVRFLW),
442                                     state->file, state->sh_name);
443                                 return;
444                         }

446                         if (uleb_extract(&data[off], ndx, len, &oper2) ==
447                             DW_OVERFLOW) {
448                                 (void) fprintf(stderr,
449                                     MSG_INTL(MSG_ERR_DWOVRFLW),
450                                     state->file, state->sh_name);
451                                 return;
452                         }
320                         oper1 = uleb_extract(&data[off], ndx);
321                         oper2 = uleb_extract(&data[off], ndx);
453                         dbg_print(0, MSG_ORIG(MSG_CFA_REG_OFFLLU), PREFIX,
454                             REGNAME(oper1, rbuf1), EC_XWORD(oper2));
455                         break;

457                 case 0x0e:              /* v2: DW_CFA_def_cfa_offset, offset */
458                         if (uleb_extract(&data[off], ndx, len, &oper1) ==
459                             DW_OVERFLOW) {
460                                 (void) fprintf(stderr,
461                                     MSG_INTL(MSG_ERR_DWOVRFLW),
462                                     state->file, state->sh_name);
463                                 return;
464                         }
327                         oper1 = uleb_extract(&data[off], ndx);
465                         dbg_print(0, MSG_ORIG(MSG_CFA_LLU), PREFIX,
466                             EC_XWORD(oper1));
467                         break;

469                 case 0x0f:              /* v3: DW_CFA_def_cfa_expression, blk */
470                         if (uleb_extract(&data[off], ndx, len, &oper1) ==
```



```
471                             DW_OVERFLOW) {
472                                 (void) fprintf(stderr,
473                                     MSG_INTL(MSG_ERR_DWOVRFLW),
474                                     state->file, state->sh_name);
475                                 return;
476                         }
333                         oper1 = uleb_extract(&data[off], ndx);
477                         dbg_print(0, MSG_ORIG(MSG_CFA_EBLK), PREFIX,
478                             EC_XWORD(oper1));
479                         /* We currently do not decode the expression block */
480                         *ndx += oper1;
481                         break;

483                 case 0x10:              /* v3: DW_CFA_expression, reg, blk */
484                 case 0x16:              /* v3: DW_CFA_val_expression,reg,blk */
485                         if (uleb_extract(&data[off], ndx, len, &oper1) ==
486                             DW_OVERFLOW) {
487                                 (void) fprintf(stderr,
488                                     MSG_INTL(MSG_ERR_DWOVRFLW),
489                                     state->file, state->sh_name);
490                                 return;
491                         }

493                         if (uleb_extract(&data[off], ndx, len, &oper2) ==
494                             DW_OVERFLOW) {
495                                 (void) fprintf(stderr,
496                                     MSG_INTL(MSG_ERR_DWOVRFLW),
497                                     state->file, state->sh_name);
498                                 return;
499                         }
342                         oper1 = uleb_extract(&data[off], ndx);
343                         oper2 = uleb_extract(&data[off], ndx);
500                         dbg_print(0, MSG_ORIG(MSG_CFA_REG_EBLK), PREFIX,
501                             REGNAME(oper1, rbuf1), EC_XWORD(oper2));
502                         /* We currently do not decode the expression block */
503                         *ndx += oper2;
504                         break;

506                 case 0x11:      /* v3: DW_CFA_offset_extended_sf, reg, off */
507                         if (uleb_extract(&data[off], ndx, len, &oper1) ==
508                             DW_OVERFLOW) {
509                                 (void) fprintf(stderr,
510                                     MSG_INTL(MSG_ERR_DWOVRFLW),
511                                     state->file, state->sh_name);
512                                 return;
513                         }

515                         if (sleb_extract(&data[off], ndx, len, &soper) ==
516                             DW_OVERFLOW) {
517                                 (void) fprintf(stderr,
518                                     MSG_INTL(MSG_ERR_DWOVRFLW),
519                                     state->file, state->sh_name);
520                                 return;
521                         }

523                         soper *= state->ciedalign;
351                         oper1 = uleb_extract(&data[off], ndx);
352                         soper = sleb_extract(&data[off], ndx) *
353                             state->ciedalign;
524                         dbg_print(0, MSG_ORIG(MSG_CFA_CFAOFF), PREFIX,
525                             REGNAME(oper1, rbuf1), EC_SXWORD(soper));
526                         break;

528                 case 0x12:              /* v3: DW_CFA_def_cfa_sf, reg, offset */
529                         if (uleb_extract(&data[off], ndx, len, &oper1) ==
530                             DW_OVERFLOW) {
```

```
531                             (void) fprintf(stderr,
532                                 MSG_INTL(MSG_ERR_DWOVRFLW),
533                                 state->file, state->sh_name);
534                             return;
535                         }

537                         if (sleb_extract(&data[off], ndx, len, &soper) ==
538                             DW_OVERFLOW) {
539                             (void) fprintf(stderr,
540                                 MSG_INTL(MSG_ERR_DWOVRFLW),
541                                 state->file, state->sh_name);
542                             return;
543                         }

545                         soper *= state->ciedalign;
359                         oper1 = uleb_extract(&data[off], ndx);
360                         soper = sleb_extract(&data[off], ndx) *
361                             state->ciedalign;
546                         dbg_print(0, MSG_ORIG(MSG_CFA_REG_OFFLLD), PREFIX,
547                             REGNAME(oper1, rbuf1), EC_SXWORD(soper));
548                         break;

550                 case 0x13:              /* DW_CFA_def_cfa_offset_sf, offset */
551                         if (sleb_extract(&data[off], ndx, len, &soper) ==
552                             DW_OVERFLOW) {
553                             (void) fprintf(stderr,
554                                 MSG_INTL(MSG_ERR_DWOVRFLW),
555                                 state->file, state->sh_name);
556                             return;
557                         }

559                         soper *= state->ciedalign;
367                         soper = sleb_extract(&data[off], ndx) *
368                             state->ciedalign;
560                         dbg_print(0, MSG_ORIG(MSG_CFA_LLD), PREFIX,
561                             EC_SXWORD(soper));
562                         break;

564                 case 0x14:              /* v3: DW_CFA_val_offset, reg, offset */
565                         if (uleb_extract(&data[off], ndx, len, &oper1) ==
566                             DW_OVERFLOW) {
567                             (void) fprintf(stderr,
568                                 MSG_INTL(MSG_ERR_DWOVRFLW),
569                                 state->file, state->sh_name);
570                             return;
571                         }

573                         if (sleb_extract(&data[off], ndx, len, &soper) ==
574                             DW_OVERFLOW) {
575                             (void) fprintf(stderr,
576                                 MSG_INTL(MSG_ERR_DWOVRFLW),
577                                 state->file, state->sh_name);
578                             return;
579                         }

581                         soper *= state->ciedalign;
374                         oper1 = uleb_extract(&data[off], ndx);
375                         soper = uleb_extract(&data[off], ndx) *
376                             state->ciedalign;
582                         dbg_print(0, MSG_ORIG(MSG_CFA_REG_OFFLLD), PREFIX,
583                             REGNAME(oper1, rbuf1), EC_SXWORD(soper));
584                         break;

586                 case 0x15:      /* v3: DW_CFA_val_offset_sf, reg, offset */
587                         if (uleb_extract(&data[off], ndx, len, &oper1) ==
588                             DW_OVERFLOW) {
```



```
589                             (void) fprintf(stderr,
590                                 MSG_INTL(MSG_ERR_DWOVRFLW),
591                                 state->file, state->sh_name);
592                             return;
593                         }

595                         if (sleb_extract(&data[off], ndx, len, &soper) ==
596                             DW_OVERFLOW) {
597                             (void) fprintf(stderr,
598                                 MSG_INTL(MSG_ERR_DWOVRFLW),
599                                 state->file, state->sh_name);
600                             return;
601                         }

603                         soper *= state->ciedalign;
382                         oper1 = uleb_extract(&data[off], ndx);
383                         soper = sleb_extract(&data[off], ndx) *
384                             state->ciedalign;
604                         dbg_print(0, MSG_ORIG(MSG_CFA_REG_OFFLLD), PREFIX,
605                             REGNAME(oper1, rbuf1), EC_SXWORD(soper));
606                         break;

608                 case 0x1d:      /* GNU: DW_CFA_MIPS_advance_loc8, delta */
609                         switch (dwarf_extract_uint(data + off, len,
610                             ndx, 8, state->do_swap, &oper1)) {
611                         case DW_BAD_ENCODING:
612                             (void) fprintf(stderr,
613                                 MSG_INTL(MSG_ERR_DWBADENC),
614                                 state->file, state->sh_name,
615                                 8);
616                             return;
617                         case DW_OVERFLOW:
618                             (void) fprintf(stderr,
619                                 MSG_INTL(MSG_ERR_DWOVRFLW),
620                                 state->file, state->sh_name);
621                             return;
622                         case DW_SUCCESS:
623                             break;
624                         }
625                         oper1 *= state->ciecalign;
390                         oper1 = dwarf_extract_uint(data + off, ndx, i,
391                             state->do_swap) * state->ciecalign;
626                         cur_pc += oper1;
627                         dbg_print(0, MSG_ORIG(MSG_CFA_ADV_LOC), PREFIX,
628                             loc_str, EC_XWORD(oper1), EC_XWORD(cur_pc));
629                         loc_str = MSG_ORIG(MSG_STR_LOC);
630                         break;

632                 case 0x2e:              /* GNU: DW_CFA_GNU_args_size, size */
633                         if (uleb_extract(&data[off], ndx, len, &oper1) ==
634                             DW_OVERFLOW) {
635                             (void) fprintf(stderr,
636                                 MSG_INTL(MSG_ERR_DWOVRFLW),
637                                 state->file, state->sh_name);
638                             return;
639                         }

399                         oper1 = uleb_extract(&data[off], ndx);
641                         dbg_print(0, MSG_ORIG(MSG_CFA_LLU), PREFIX,
642                             EC_XWORD(oper1));

644                         break;

646                 case 0x2f: /* GNU:DW_CFA_GNU_negative_offset_extended,reg,off */
647                         if (uleb_extract(&data[off], ndx, len, &oper1) ==
648                             DW_OVERFLOW) {
```

```
649                                 (void) fprintf(stderr,
650                                     MSG_INTL(MSG_ERR_DWOVRFLW),
651                                     state->file, state->sh_name);
652                                 return;
653                         }

655                         if (sleb_extract(&data[off], ndx, len, &soper) ==
656                             DW_OVERFLOW) {
657                                 (void) fprintf(stderr,
658                                     MSG_INTL(MSG_ERR_DWOVRFLW),
659                                     state->file, state->sh_name);
660                                 return;
661                         }
662                         soper = -soper * state->ciedalign;
663                         soper *= state->ciedalign;
406                         oper1 = uleb_extract(&data[off], ndx);
407                         soper = -uleb_extract(&data[off], ndx) *
408                             state->ciedalign;
664                         dbg_print(0, MSG_ORIG(MSG_CFA_CFAOFF), PREFIX,
665                             REGNAME(oper1, rbuf1), EC_SXWORD(soper));
666                         break;

668                 default:
669                         /*
670                          * Unrecognized OP code: DWARF data is variable length,
671                          * so we don't know how many bytes to skip in order to
672                          * advance to the next item. We cannot decode beyond
673                          * this point, so dump the remainder in hex.
674                          */
675                         (*ndx)--;       /* Back up to unrecognized opcode */
676                         dump_hex_bytes(data + off + *ndx, len - *ndx,
677                             indent, 8, 1);
678                         (*ndx) = len;
679                         break;
680                 }
681         }

683 #undef PREFIX
684 #undef REGNAME
685 #undef LOW_OP
686 }

688 void
689 dump_eh_frame(const char *file, char *sh_name, uchar_t *data, size_t datasize,
690     uint64_t sh_addr, Half e_machine, uchar_t *e_ident, uint64_t gotaddr)
434 dump_eh_frame(uchar_t *data, size_t datasize, uint64_t sh_addr,
435     Half e_machine, uchar_t *e_ident, uint64_t gotaddr)
691 {
692         Conv_dwarf_ehe_buf_t    dwarf_ehe_buf;
693         dump_cfi_state_t        cfi_state;
694         uint64_t        off, ndx, length, id;
439         uint64_t        off, ndx;
695         uint_t          cieid, cielength, cieversion, cieretaddr;
696         int             ciePflag = 0, cieZflag = 0, cieLflag = 0;
697         int             cieLflag_present = 0;
698         uint_t          cieaugndx;
699         char            *cieaugstr = NULL;
700         boolean_t       have_cie = B_FALSE;
441         int             ciePflag, cieZflag, cieLflag, cieLflag_present;
442         uint_t          cieaugndx, length, id;
443         char            *cieaugstr;

702         cfi_state.file = file;
703         cfi_state.sh_name = sh_name;
704 #endif /* ! codereview */
705         cfi_state.e_machine = e_machine;
```



```
706         cfi_state.e_ident = e_ident;
707         cfi_state.sh_addr = sh_addr;
708         cfi_state.do_swap = _elf_sys_encoding() != e_ident[EI_DATA];
709         cfi_state.gotaddr = gotaddr;

711         off = 0;
712         while (off < datasize) {
713                 ndx = 0;

715                 /*
716                  * Extract length in native format.  A zero length indicates
717                  * that this CIE is a terminator and that processing for this
718                  * unwind information should end. However, skip this entry and
719                  * keep processing, just in case there is any other information
720                  * remaining in this section.  Note, ld(1) will terminate the
721                  * processing of the .eh_frame contents for this file after a
722                  * zero length CIE, thus any information that does follow is
723                  * ignored by ld(1), and is therefore questionable.
724                  */
725                 if (dwarf_extract_uint(data + off, datasize - off,
726                     &ndx, 4, cfi_state.do_swap, &length) == DW_OVERFLOW) {
727                         (void) fprintf(stderr,
728                             MSG_INTL(MSG_ERR_DWOVRFLW),
729                             file, sh_name);
730                         return;
731                 }

445                 length = (uint_t)dwarf_extract_uint(data + off, &ndx,
446                     4, cfi_state.do_swap);
733                 if (length == 0) {
734                         dbg_print(0, MSG_ORIG(MSG_UNW_ZEROTERM));
735                         off += 4;
736                         continue;
737                 }

739                 if (length > (datasize - off)) {
740                         (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADCIEFDELEN),
741                             file, sh_name, EC_XWORD(length),
742                             EC_XWORD(sh_addr + off));
743                         /*
744                          * If length is wrong, we have no means to find the
745                          * next entry, just give up
746                          */
747                         return;
748                 }

750 #endif /* ! codereview */
751                 /*
752                  * extract CIE id in native format
753                  */
754                 if (dwarf_extract_uint(data + off, datasize - off, &ndx,
755                     4, cfi_state.do_swap, &id) == DW_OVERFLOW) {
756                         (void) fprintf(stderr,
757                             MSG_INTL(MSG_ERR_DWOVRFLW),
758                             file, sh_name);
759                         return;
760                 }
453                 id = (uint_t)dwarf_extract_uint(data + off, &ndx,
454                     4, cfi_state.do_swap);

762                 /*
763                  * A CIE record has an id of '0', otherwise this is a
764                  * FDE entry and the 'id' is the CIE pointer.
765                  */
766                 if (id == 0) {
767                         uint64_t        persVal, ndx_save = 0;
```

```
768                         uint64_t        axsize;

461                         uint64_t        persVal, ndx_save;
462                         uint_t          axsize;

771                         have_cie = B_TRUE;
772 #endif /* ! codereview */
773                         cielength = length;
774                         cieid = id;
775                         ciePflag = cfi_state.cieRflag = cieZflag = 0;
776                         cieLflag = cieLflag_present = 0;

778                         dbg_print(0, MSG_ORIG(MSG_UNW_CIE),
779                             EC_XWORD(sh_addr + off));
780                         dbg_print(0, MSG_ORIG(MSG_UNW_CIELNGTH),
781                             cielength, cieid);

783                         cieversion = data[off + ndx];
784                         ndx += 1;
785                         cieaugstr = (char *)(&data[off + ndx]);
786                         ndx += strlen(cieaugstr) + 1;

788                         dbg_print(0, MSG_ORIG(MSG_UNW_CIEVERS),
789                             cieversion, cieaugstr);

791                         if (uleb_extract(&data[off], &ndx, datasize - off,
792                             &cfi_state.ciecalign) == DW_OVERFLOW) {
793                                 (void) fprintf(stderr,
794                                     MSG_INTL(MSG_ERR_DWOVRFLW),
795                                     file, sh_name);
796                                 return;
797                         }

799                         if (sleb_extract(&data[off], &ndx, datasize - off,
800                             &cfi_state.ciedalign) == DW_OVERFLOW) {
801                                 (void) fprintf(stderr,
802                                     MSG_INTL(MSG_ERR_DWOVRFLW),
803                                     file, sh_name);
804                                 return;
805                         }
464                         cfi_state.ciecalign = uleb_extract(&data[off], &ndx);
465                         cfi_state.ciedalign = sleb_extract(&data[off], &ndx);
806                         cieretaddr = data[off + ndx];
807                         ndx += 1;

809                         dbg_print(0, MSG_ORIG(MSG_UNW_CIECALGN),
810                             EC_XWORD(cfi_state.ciecalign),
811                             EC_XWORD(cfi_state.ciedalign), cieretaddr);

813                         if (cieaugstr[0])
814                                 dbg_print(0, MSG_ORIG(MSG_UNW_CIEAXVAL));

816                         for (cieaugndx = 0; cieaugstr[cieaugndx]; cieaugndx++) {
817                                 switch (cieaugstr[cieaugndx]) {
818                                 case 'z':
819                                         if (uleb_extract(&data[off], &ndx,
820                                             datasize - off, &axsize) ==
821                                             DW_OVERFLOW) {
822                                                 (void) fprintf(stderr,
823                                                     MSG_INTL(MSG_ERR_DWOVRFLW),
824                                                     file, sh_name);
825                                                 return;
826                                         }

479                                         axsize = uleb_extract(&data[off], &ndx);
828                                         dbg_print(0, MSG_ORIG(MSG_UNW_CIEAXSIZ),
```



```
829                                             EC_XWORD(axsize));
481                                             axsize);
830                                         cieZflag = 1;
831                                         /*
832                                          * The auxiliary section can contain
833                                          * unused padding bytes at the end, so
834                                          * save the current index. Along with
835                                          * axsize, we will use it to set ndx to
836                                          * the proper continuation index after
837                                          * the aux data has been processed.
838                                          */
839                                         ndx_save = ndx;
840                                         break;
841                                 case 'P':
842                                         ciePflag = data[off + ndx];
843                                         ndx += 1;

845                                         switch (dwarf_ehe_extract(&data[off],
846                                             datasize - off, &ndx, &persVal,
847                                             ciePflag, e_ident, B_FALSE, sh_addr,
848                                             off + ndx, gotaddr)) {
849                                         case DW_OVERFLOW:
850                                                 (void) fprintf(stderr,
851                                                     MSG_INTL(MSG_ERR_DWOVRFLW),
852                                                     file, sh_name);
853                                                 return;
854                                         case DW_BAD_ENCODING:
855                                                 (void) fprintf(stderr,
856                                                     MSG_INTL(MSG_ERR_DWBADENC),
857                                                     file, sh_name, ciePflag);
858                                                 return;
859                                         case DW_SUCCESS:
860                                                 break;
861                                         }
497                                         persVal = dwarf_ehe_extract(&data[off],
498                                             &ndx, ciePflag, e_ident, B_FALSE,
499                                             sh_addr, off + ndx, gotaddr);
862                                         dbg_print(0,
863                                             MSG_ORIG(MSG_UNW_CIEAXPERS));
864                                         dbg_print(0,
865                                             MSG_ORIG(MSG_UNW_CIEAXPERSENC),
866                                             ciePflag, conv_dwarf_ehe(ciePflag,
867                                             &dwarf_ehe_buf));
868                                         dbg_print(0,
869                                             MSG_ORIG(MSG_UNW_CIEAXPERSRTN),
870                                             EC_XWORD(persVal));
871                                         break;
872                                 case 'R':
873                                         cfi_state.cieRflag = data[off + ndx];
874                                         ndx += 1;
875                                         dbg_print(0,
876                                             MSG_ORIG(MSG_UNW_CIEAXCENC),
877                                             cfi_state.cieRflag,
878                                             conv_dwarf_ehe(cfi_state.cieRflag,
879                                             &dwarf_ehe_buf));
880                                         break;
881                                 case 'L':
882                                         cieLflag_present = 1;
883                                         cieLflag = data[off + ndx];
884                                         ndx += 1;
885                                         dbg_print(0,
886                                             MSG_ORIG(MSG_UNW_CIEAXLSDA),
887                                             cieLflag, conv_dwarf_ehe(
888                                             cieLflag, &dwarf_ehe_buf));
889                                         break;
890                                 default:
```

```
891                                     dbg_print(0,
892                                         MSG_ORIG(MSG_UNW_CIEAXUNEC),
893                                         cieaugstr[cieaugndx]);
894                                     break;
895                             }
896                     }

898                     /*
899                      * If the z flag was present, reposition ndx using the
900                      * length given. This will safely move us past any
901                      * unaccessed padding bytes in the auxiliary section.
902                      */
903                     if (cieZflag)
904                             ndx = ndx_save + axsize;

906                     /*
907                      * Any remaining data are Call Frame Instructions
908                      */
909                     if ((cielength + 4) > ndx)
910                             dump_cfi(data, off, &ndx, cielength, &cfi_state,
911                                 MSG_ORIG(MSG_UNW_CIECFI), 3);
912                     off += cielength + 4;

914             } else {
915                     uint_t      fdelength = length;
916                     int         fdecieptr = id;
917                     uint64_t    fdeaddrrange;

919                     if (!have_cie) {
920                             (void) fprintf(stderr,
921                                 MSG_INTL(MSG_ERR_DWNOCIE), file, sh_name);
922                             return;
923                     }

925 #endif /* ! codereview */
926                     dbg_print(0, MSG_ORIG(MSG_UNW_FDE),
927                         EC_XWORD(sh_addr + off));
928                     dbg_print(0, MSG_ORIG(MSG_UNW_FDELNGTH),
929                         fdelength, fdecieptr);

931                     switch (dwarf_ehe_extract(&data[off], datasize - off,
932                         &ndx, &cfi_state.fdeinitloc, cfi_state.cieRflag,
933                         e_ident, B_FALSE, sh_addr, off + ndx, gotaddr)) {
934                     case DW_OVERFLOW:
935                             (void) fprintf(stderr,
936                                 MSG_INTL(MSG_ERR_DWOVRFLW), file, sh_name);
937                             return;
938                     case DW_BAD_ENCODING:
939                             (void) fprintf(stderr,
940                                 MSG_INTL(MSG_ERR_DWBADENC), file, sh_name,
941                                 cfi_state.cieRflag);
942                             return;
943                     case DW_SUCCESS:
944                             break;
945                     }

947                     switch (dwarf_ehe_extract(&data[off], datasize - off,
948                         &ndx, &fdeaddrrange,
949                         (cfi_state.cieRflag & ~DW_EH_PE_pcrel), e_ident,
950                         B_FALSE, sh_addr, off + ndx, gotaddr)) {
951                     case DW_OVERFLOW:
952                             (void) fprintf(stderr,
953                                 MSG_INTL(MSG_ERR_DWOVRFLW), file, sh_name);
954                             return;
955                     case DW_BAD_ENCODING:
956                             (void) fprintf(stderr,
```



```
957                                 MSG_INTL(MSG_ERR_DWBADENC), file, sh_name,
958                                 (cfi_state.cieRflag & ~DW_EH_PE_pcrel));
959                             return;
960                     case DW_SUCCESS:
961                             break;
962                     }
557                     cfi_state.fdeinitloc = dwarf_ehe_extract(&data[off],
558                         &ndx, cfi_state.cieRflag, e_ident, B_FALSE,
559                         sh_addr, off + ndx, gotaddr);
560                     fdeaddrrange = dwarf_ehe_extract(&data[off], &ndx,
561                         (cfi_state.cieRflag & ~DW_EH_PE_pcrel),
562                         e_ident, B_FALSE, sh_addr, off + ndx, gotaddr);

964                     dbg_print(0, MSG_ORIG(MSG_UNW_FDEINITLOC),
965                         EC_XWORD(cfi_state.fdeinitloc),
966                         EC_XWORD(fdeaddrrange),
967                         EC_XWORD(cfi_state.fdeinitloc + fdeaddrrange - 1));

969                     if ((cieaugstr != NULL) && (cieaugstr[0] != '\0'))
569                     if (cieaugstr[0])
970                             dbg_print(0, MSG_ORIG(MSG_UNW_FDEAXVAL));
971                     if (cieZflag) {
972                             uint64_t    val;
973                             uint64_t    lndx;

975                             if (uleb_extract(&data[off], &ndx,
976                                 datasize - off, &val) == DW_OVERFLOW) {
977                                     (void) fprintf(stderr,
978                                         MSG_INTL(MSG_ERR_DWOVRFLW),
979                                         file, sh_name);
980                                     return;
981                             }
575                             val = uleb_extract(&data[off], &ndx);
982                             lndx = ndx;
983                             ndx += val;
984                             dbg_print(0, MSG_ORIG(MSG_UNW_FDEAXSIZE),
985                                 EC_XWORD(val));
986                             if (val && cieLflag_present) {
987                                     uint64_t    lsda;

989                                     switch (dwarf_ehe_extract(&data[off],
990                                         datasize - off, &lndx, &lsda,
991                                         cieLflag, e_ident, B_FALSE, sh_addr,
992                                         off + lndx, gotaddr)) {
993                                     case DW_OVERFLOW:
994                                             (void) fprintf(stderr,
995                                                 MSG_INTL(MSG_ERR_DWOVRFLW),
996                                                 file, sh_name);
997                                             return;
998                                     case DW_BAD_ENCODING:
999                                             (void) fprintf(stderr,
1000                                                MSG_INTL(MSG_ERR_DWBADENC),
1001                                                file, sh_name, cieLflag);
1002                                            return;
1003                                    case DW_SUCCESS:
1004                                            break;
1005                                    }
583                                    lsda = dwarf_ehe_extract(&data[off],
584                                        &lndx, cieLflag, e_ident,
585                                        B_FALSE, sh_addr, off + lndx,
586                                        gotaddr);
1006                                    dbg_print(0,
1007                                        MSG_ORIG(MSG_UNW_FDEAXLSDA),
1008                                        EC_XWORD(lsda));
1009                            }
1010                    }
```

```
1011                        if ((fdelength + 4) > ndx)
1012                                dump_cfi(data, off, &ndx, fdelength, &cfi_state,
1013                                    MSG_ORIG(MSG_UNW_FDECFI), 6);
1014                        off += fdelength + 4;
1015                }
1016        }
1017 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
  146656 Mon Mar 23 21:41:46 2015
new/usr/src/cmd/sgs/elfdump/common/elfdump.c
5688 ELF tools need to be more careful with dwarf data
**********************************************************
______unchanged_portion_omitted_

 517 /*
 518  * Display the contents of GNU/amd64 .eh_frame and .eh_frame_hdr
 519  * sections.
 520  *
 521  * entry:
 522  *      cache - Cache of all section headers
 523  *      shndx - Index of .eh_frame or .eh_frame_hdr section to be displayed
 524  *      shnum - Total number of sections which exist
 525  *      uphdr - NULL, or unwind program header associated with
 526  *              the .eh_frame_hdr section.
 527  *      ehdr - ELF header for file
 528  *      eh_state - Data used across calls to this routine. The
 529  *              caller should zero it before the first call, and
 530  *              pass it on every call.
 531  *      osabi - OSABI to use in displaying information
 532  *      file - Name of file
 533  *      flags - Command line option flags
 534  */
 535 static void
 536 unwind_eh_frame(Cache *cache, Word shndx, Word shnum, Phdr *uphdr, Ehdr *ehdr,
 537     gnu_eh_state_t *eh_state, uchar_t osabi, const char *file, uint_t flags)
 538 {
 539 #if     defined(_ELF64)
 540 #define MSG_UNW_BINSRTAB2       MSG_UNW_BINSRTAB2_64
 541 #define MSG_UNW_BINSRTABENT     MSG_UNW_BINSRTABENT_64
 542 #else
 543 #define MSG_UNW_BINSRTAB2       MSG_UNW_BINSRTAB2_32
 544 #define MSG_UNW_BINSRTABENT     MSG_UNW_BINSRTABENT_32
 545 #endif

 547         Cache                   *_cache = &cache[shndx];
 548         Shdr                    *shdr = _cache->c_shdr;
 549         uchar_t                 *data = (uchar_t *)(_cache->c_data->d_buf);
 550         size_t                  datasize = _cache->c_data->d_size;
 551         Conv_dwarf_ehe_buf_t    dwarf_ehe_buf;
 552         uint64_t                ndx, frame_ptr, fde_cnt, tabndx;
 553         uint_t                  vers, frame_ptr_enc, fde_cnt_enc, table_enc;
 554         uint64_t                initloc, initloc0 = 0;
 554         uint64_t                initloc, initloc0;
 555         uint64_t                gotaddr = 0;
 556         int                     cnt;

 558         for (cnt = 1; cnt < shnum; cnt++) {
 559                 if (strncmp(cache[cnt].c_name, MSG_ORIG(MSG_ELF_GOT),
 560                     MSG_ELF_GOT_SIZE) == 0) {
 561                         gotaddr = cache[cnt].c_shdr->sh_addr;
 562                         break;
 563                 }
 564         }

 566         if ((data == NULL) || (datasize == 0)) {
 567                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSZ),
 568                     file, _cache ->c_name);
 569                 return;
 570         }

 572 #endif /* ! codereview */
 573         /*
 574          * Is this a .eh_frame_hdr?
```



```
 575          */
 576         if ((uphdr && (shdr->sh_addr == uphdr->p_vaddr)) ||
 577             (strncmp(_cache->c_name, MSG_ORIG(MSG_SCN_FRMHDR),
 578             MSG_SCN_FRMHDR_SIZE) == 0)) {
 579                 /*
 580                  * There can only be a single .eh_frame_hdr.
 581                  * Flag duplicates.
 582                  */
 583                 if (++eh_state->hdr_cnt > 1)
 584                         (void) fprintf(stderr, MSG_INTL(MSG_ERR_MULTEHFRMHDR),
 585                             file, EC_WORD(shndx), _cache->c_name);

 587                 dbg_print(0, MSG_ORIG(MSG_UNW_FRMHDR));
 588                 ndx = 0;

 590                 vers = data[ndx++];
 591                 frame_ptr_enc = data[ndx++];
 592                 fde_cnt_enc = data[ndx++];
 593                 table_enc = data[ndx++];

 595                 dbg_print(0, MSG_ORIG(MSG_UNW_FRMVERS), vers);

 597                 switch (dwarf_ehe_extract(data, datasize, &ndx,
 598                     &frame_ptr, frame_ptr_enc, ehdr->e_ident, B_TRUE,
 599                     shdr->sh_addr, ndx, gotaddr)) {
 600                 case DW_OVERFLOW:
 601                         (void) fprintf(stderr, MSG_INTL(MSG_ERR_DWOVRFLW),
 602                             file, _cache->c_name);
 603                         return;
 604                 case DW_BAD_ENCODING:
 605                         (void) fprintf(stderr, MSG_INTL(MSG_ERR_DWBADENC),
 606                             file, _cache->c_name, frame_ptr_enc);
 607                         return;
 608                 case DW_SUCCESS:
 609                         break;
 610                 }
 566                 frame_ptr = dwarf_ehe_extract(data, &ndx, frame_ptr_enc,
 567                     ehdr->e_ident, B_TRUE, shdr->sh_addr, ndx, gotaddr);
 611                 if (eh_state->hdr_cnt == 1) {
 612                         eh_state->hdr_ndx = shndx;
 613                         eh_state->frame_ptr = frame_ptr;
 614                 }

 616                 dbg_print(0, MSG_ORIG(MSG_UNW_FRPTRENC),
 617                     conv_dwarf_ehe(frame_ptr_enc, &dwarf_ehe_buf),
 618                     EC_XWORD(frame_ptr));

 620                 switch (dwarf_ehe_extract(data, datasize, &ndx, &fde_cnt,
 621                     fde_cnt_enc, ehdr->e_ident, B_TRUE, shdr->sh_addr, ndx,
 622                     gotaddr)) {
 623                 case DW_OVERFLOW:
 624                         (void) fprintf(stderr, MSG_INTL(MSG_ERR_DWOVRFLW),
 625                             file, _cache->c_name);
 626                         return;
 627                 case DW_BAD_ENCODING:
 628                         (void) fprintf(stderr, MSG_INTL(MSG_ERR_DWBADENC),
 629                             file, _cache->c_name, fde_cnt_enc);
 630                         return;
 631                 case DW_SUCCESS:
 632                         break;
 633                 }
 577                 fde_cnt = dwarf_ehe_extract(data, &ndx, fde_cnt_enc,
 578                     ehdr->e_ident, B_TRUE, shdr->sh_addr, ndx, gotaddr);

 635                 dbg_print(0, MSG_ORIG(MSG_UNW_FDCNENC),
 636                     conv_dwarf_ehe(fde_cnt_enc, &dwarf_ehe_buf),
```

```
637                     EC_XWORD(fde_cnt));
638                 dbg_print(0, MSG_ORIG(MSG_UNW_TABENC),
639                     conv_dwarf_ehe(table_enc, &dwarf_ehe_buf));
640                 dbg_print(0, MSG_ORIG(MSG_UNW_BINSRTAB1));
641                 dbg_print(0, MSG_ORIG(MSG_UNW_BINSRTAB2));

643                 for (tabndx = 0; tabndx < fde_cnt; tabndx++) {
644                         uint64_t table;

646                         switch (dwarf_ehe_extract(data, datasize, &ndx,
647                             &initloc, table_enc, ehdr->e_ident, B_TRUE,
648                             shdr->sh_addr, ndx, gotaddr)) {
649                         case DW_OVERFLOW:
650                                 (void) fprintf(stderr,
651                                     MSG_INTL(MSG_ERR_DWOVRFLW), file,
652                                     _cache->c_name);
653                                 return;
654                         case DW_BAD_ENCODING:
655                                 (void) fprintf(stderr,
656                                     MSG_INTL(MSG_ERR_DWBADENC), file,
657                                     _cache->c_name, table_enc);
658                                 return;
659                         case DW_SUCCESS:
660                                 break;
661                         }
589                         initloc = dwarf_ehe_extract(data, &ndx, table_enc,
590                             ehdr->e_ident, B_TRUE, shdr->sh_addr, ndx, gotaddr);
591                         /*LINTED:E_VAR_USED_BEFORE_SET*/
662                         if ((tabndx != 0) && (initloc0 > initloc))
663                                 (void) fprintf(stderr,
664                                     MSG_INTL(MSG_ERR_BADSORT), file,
665                                     _cache->c_name, EC_WORD(tabndx));
666                         switch (dwarf_ehe_extract(data, datasize, &ndx, &table,
667                             table_enc, ehdr->e_ident, B_TRUE, shdr->sh_addr,
668                             ndx, gotaddr)) {
669                         case DW_OVERFLOW:
670                                 (void) fprintf(stderr,
671                                     MSG_INTL(MSG_ERR_DWOVRFLW), file,
672                                     _cache->c_name);
673                                 return;
674                         case DW_BAD_ENCODING:
675                                 (void) fprintf(stderr,
676                                     MSG_INTL(MSG_ERR_DWBADENC), file,
677                                     _cache->c_name, table_enc);
678                                 return;
679                         case DW_SUCCESS:
680                                 break;
681                         }

683 #endif /* ! codereview */
684                         dbg_print(0, MSG_ORIG(MSG_UNW_BINSRTABENT),
685                             EC_XWORD(initloc),
686                             EC_XWORD(table));
596                             EC_XWORD(dwarf_ehe_extract(data, &ndx,
597                             table_enc, ehdr->e_ident, B_TRUE, shdr->sh_addr,
598                             ndx, gotaddr)));
687                         initloc0 = initloc;
688                 }
689         } else {                /* Display the .eh_frame section */
690                 eh_state->frame_cnt++;
691                 if (eh_state->frame_cnt == 1) {
692                         eh_state->frame_ndx = shndx;
693                         eh_state->frame_base = shdr->sh_addr;
694                 } else if ((eh_state->frame_cnt >  1) &&
695                     (ehdr->e_type != ET_REL)) {
696                         Conv_inv_buf_t  inv_buf;
```



```
698                         (void) fprintf(stderr, MSG_INTL(MSG_WARN_MULTEHFRM),
699                             file, EC_WORD(shndx), _cache->c_name,
700                             conv_ehdr_type(osabi, ehdr->e_type, 0, &inv_buf));
701                 }
702                 dump_eh_frame(file, _cache->c_name, data, datasize,
703                     shdr->sh_addr, ehdr->e_machine, ehdr->e_ident, gotaddr);
614                 dump_eh_frame(data, datasize, shdr->sh_addr,
615                     ehdr->e_machine, ehdr->e_ident, gotaddr);
704         }

706         /*
707          * If we've seen the .eh_frame_hdr and the first .eh_frame section,
708          * compare the header frame_ptr to the address of the actual frame
709          * section to ensure the link-editor got this right.  Note, this
710          * diagnostic is only produced when unwind information is explicitly
711          * asked for, as shared objects built with an older ld(1) may reveal
712          * this inconsistency.  Although an inconsistency, it doesn't seem to
713          * have any adverse effect on existing tools.
714          */
715         if (((flags & FLG_MASK_SHOW) != FLG_MASK_SHOW) &&
716             (eh_state->hdr_cnt > 0) && (eh_state->frame_cnt > 0) &&
717             (eh_state->frame_ptr != eh_state->frame_base))
718                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADEHFRMPTR),
719                     file, EC_WORD(eh_state->hdr_ndx),
720                     cache[eh_state->hdr_ndx].c_name,
721                     EC_XWORD(eh_state->frame_ptr),
722                     EC_WORD(eh_state->frame_ndx),
723                     cache[eh_state->frame_ndx].c_name,
724                     EC_XWORD(eh_state->frame_base));
725 #undef MSG_UNW_BINSRTAB2
726 #undef MSG_UNW_BINSRTABENT
727 }
_____unchanged_portion_omitted_

770 /*
771  * Display exception_range_entry items from the .exception_ranges section
772  * of a Sun C++ object.
773  */
774 static void
775 unwind_exception_ranges(Cache *_cache, const char *file, int do_swap)
776 {
777         /*
778          * Translate a PTRDIFF_T self-relative address field of
779          * an exception_range_entry struct into an address.
780          *
781          * entry:
782          *      exc_addr - Address of base of exception_range_entry struct
783          *      cur_ent - Pointer to data in the struct to be translated
784          *
785          *      _f - Field of struct to be translated
786          */
787 #define SRELPTR(_f) \
788         srelptr(exc_addr + offsetof(exception_range_entry, _f), cur_ent->_f)

790 #if     defined(_ELF64)
791 #define MSG_EXR_TITLE   MSG_EXR_TITLE_64
792 #define MSG_EXR_ENTRY   MSG_EXR_ENTRY_64
793 #else
794 #define MSG_EXR_TITLE   MSG_EXR_TITLE_32
795 #define MSG_EXR_ENTRY   MSG_EXR_ENTRY_32
796 #endif

798         exception_range_entry   scratch, *ent, *cur_ent = &scratch;
799         char                    index[MAXNDXSIZE];
800         Word                    i, nelts;
```

```
801         Addr                    addr, addr0 = 0, offset = 0;
713         Addr                    addr, addr0, offset = 0;
802         Addr                    exc_addr = _cache->c_shdr->sh_addr;

804         dbg_print(0, MSG_INTL(MSG_EXR_TITLE));
805         ent = (exception_range_entry *)(_cache->c_data->d_buf);
806         nelts = _cache->c_data->d_size / sizeof (exception_range_entry);

808         for (i = 0; i < nelts; i++, ent++) {
809                 if (do_swap) {
810                         /*
811                          * Copy byte swapped values into the scratch buffer.
812                          * The reserved field is not used, so we skip it.
813                          */
814                         scratch.ret_addr = swap_ptrdiff(ent->ret_addr);
815                         scratch.length = BSWAP_XWORD(ent->length);
816                         scratch.handler_addr = swap_ptrdiff(ent->handler_addr);
817                         scratch.type_block = swap_ptrdiff(ent->type_block);
818                 } else {
819                         cur_ent = ent;
820                 }

822                 /*
823                  * The table is required to be sorted by the address
824                  * derived from ret_addr, to allow binary searching. Ensure
825                  * that addresses grow monotonically.
826                  */
827                 addr = SRELPTR(ret_addr);
740                 /*LINTED:E_VAR_USED_BEFORE_SET*/
828                 if ((i != 0) && (addr0 > addr))
829                         (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSORT),
830                             file, _cache->c_name, EC_WORD(i));

832                 (void) snprintf(index, MAXNDXSIZE, MSG_ORIG(MSG_FMT_INDEX),
833                     EC_XWORD(i));
834                 dbg_print(0, MSG_INTL(MSG_EXR_ENTRY), index, EC_ADDR(offset),
835                     EC_ADDR(addr), EC_ADDR(cur_ent->length),
836                     EC_ADDR(SRELPTR(handler_addr)),
837                     EC_ADDR(SRELPTR(type_block)));

839                 addr0 = addr;
840                 exc_addr += sizeof (exception_range_entry);
841                 offset += sizeof (exception_range_entry);
842         }

844 #undef SRELPTR
845 #undef MSG_EXR_TITLE
846 #undef MSG_EXR_ENTRY
847 }
_____unchanged_portion_omitted_

3658 /*
3659  * Traverse a note section analyzing each note information block.
3660  * The data buffers size is used to validate references before they are made,
3661  * and is decremented as each element is processed.
3662  */
3663 void
3664 note_entry(Cache *cache, Word *data, size_t size, Ehdr *ehdr, const char *file)
3665 {
3666         int             cnt = 0;
3667         int             is_corenote;
3668         int             do_swap;
3669         Conv_inv_buf_t  inv_buf;
3670         parse_note_t    pnstate;

3672         pnstate.pns_file = file;
```



```
3673         pnstate.pns_cache = cache;
3674         pnstate.pns_size = size;
3675         pnstate.pns_data = data;
3676         do_swap = _elf_sys_encoding() != ehdr->e_ident[EI_DATA];

3678         /*
3679          * Print out a single ‘note’ information block.
3680          */
3681         while (pnstate.pns_size > 0) {

3683                 if (parse_note_entry(&pnstate) == 0)
3684                         return;

3686                 /*
3687                  * Is this a Solaris core note? Such notes all have
3688                  * the name "CORE".
3689                  */
3690                 is_corenote = (ehdr->e_type == ET_CORE) &&
3691                     (pnstate.pn_namesz == (MSG_STR_CORE_SIZE + 1)) &&
3692                     (strncmp(MSG_ORIG(MSG_STR_CORE), pnstate.pn_name,
3693                     MSG_STR_CORE_SIZE + 1) == 0);

3695                 dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
3696                 dbg_print(0, MSG_INTL(MSG_FMT_NOTEENTNDX), EC_WORD(cnt));
3697                 cnt++;
3698                 dbg_print(0, MSG_ORIG(MSG_NOTE_NAMESZ),
3699                     EC_WORD(pnstate.pn_namesz));
3700                 dbg_print(0, MSG_ORIG(MSG_NOTE_DESCSZ),
3701                     EC_WORD(pnstate.pn_descsz));

3703                 if (is_corenote)
3704                         dbg_print(0, MSG_ORIG(MSG_NOTE_TYPE_STR),
3705                             conv_cnote_type(pnstate.pn_type, 0, &inv_buf));
3706                 else
3707                         dbg_print(0, MSG_ORIG(MSG_NOTE_TYPE),
3708                             EC_WORD(pnstate.pn_type));
3709                 if (pnstate.pn_namesz) {
3710                         dbg_print(0, MSG_ORIG(MSG_NOTE_NAME));
3711                         /*
3712                          * The name string can contain embedded ‘null’
3713                          * bytes and/or unprintable characters. Also,
3714                          * the final NULL is documented in the ELF ABI
3715                          * as being included in the namesz. So, display
3716                          * the name using C literal string notation, and
3717                          * include the terminating NULL in the output.
3718                          * We don’t show surrounding double quotes, as
3719                          * that implies the termination that we are showing
3720                          * explicitly.
3721                          */
3722                         (void) fwrite(MSG_ORIG(MSG_STR_8SP),
3723                             MSG_STR_8SP_SIZE, 1, stdout);
3724                         conv_str_to_c_literal(pnstate.pn_name,
3725                             pnstate.pn_namesz, c_literal_cb, NULL);
3726                         dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
3727                 }

3729                 if (pnstate.pn_descsz) {
3730                         int             hexdump = 1;

3732                         /*
3733                          * If this is a core note, let the corenote()
3734                          * function handle it.
3735                          */
3736                         if (is_corenote) {
3737                                 /* We only issue the bad arch error once */
3738                                 static int      badnote_done = 0;
```

```
3739                                corenote_ret_t  corenote_ret;

3741                                corenote_ret = corenote(ehdr->e_machine,
3742                                    do_swap, pnstate.pn_type, pnstate.pn_desc,
3743                                    pnstate.pn_descsz);
3744                                switch (corenote_ret) {
3745                                case CORENOTE_R_OK_DUMP:
3746                                        hexdump = 1;
3747                                        break;
3748 #endif /* ! codereview */
3749                                case CORENOTE_R_OK:
3750                                        hexdump = 0;
3751                                        break;
3752                                case CORENOTE_R_BADDATA:
3753                                        (void) fprintf(stderr,
3754                                            MSG_INTL(MSG_NOTE_BADCOREDATA),
3755                                            file);
3756                                        break;
3757                                case CORENOTE_R_BADARCH:
3758                                        if (badnote_done)
3759                                                break;
3760                                        (void) fprintf(stderr,
3761                                            MSG_INTL(MSG_NOTE_BADCOREARCH),
3762                                            file,
3763                                            conv_ehdr_mach(ehdr->e_machine,
3764                                            0, &inv_buf));
3765                                        break;
3766                                case CORENOTE_R_BADTYPE:
3767                                        (void) fprintf(stderr,
3768                                            MSG_INTL(MSG_NOTE_BADCORETYPE),
3769                                            file,
3770                                            EC_WORD(pnstate.pn_type));
3771                                        break;

3773 #endif /* ! codereview */
3774                                }
3775                        }

3777                        /*
3778                         * The default thing when we don't understand
3779                         * the note data is to display it as hex bytes.
3780                         */
3781                        if (hexdump) {
3782                                dbg_print(0, MSG_ORIG(MSG_NOTE_DESC));
3783                                dump_hex_bytes(pnstate.pn_desc,
3784                                    pnstate.pn_descsz, 8, 4, 4);
3785                        }
3786                }
3787        }
3788 }

3790 /*
3791  * Search for and process .note sections.
3792  *
3793  * Returns the number of note sections seen.
3794  */
3795 static Word
3796 note(Cache *cache, Word shnum, Ehdr *ehdr, const char *file)
3797 {
3798         Word    cnt, note_cnt = 0;

3800         /*
3801          * Otherwise look for any .note sections.
3802          */
3803         for (cnt = 1; cnt < shnum; cnt++) {
3804                 Cache   *_cache = &cache[cnt];
```



```
3805                 Shdr    *shdr = _cache->c_shdr;

3807                 if (shdr->sh_type != SHT_NOTE)
3808                         continue;
3809                 note_cnt++;
3810                 if (!match(MATCH_F_ALL, _cache->c_name, cnt, shdr->sh_type))
3811                         continue;

3813                 /*
3814                  * As these sections are often hand rolled, make sure they're
3815                  * properly aligned before proceeding, and issue an error
3816                  * as necessary.
3817                  *
3818                  * Note that we will continue on to display the note even
3819                  * if it has bad alignment. We can do this safely, because
3820                  * libelf knows the alignment required for SHT_NOTE, and
3821                  * takes steps to deliver a properly aligned buffer to us
3822                  * even if the actual file is misaligned.
3823                  */
3824                 if (shdr->sh_offset & (sizeof (Word) - 1))
3825                         (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADALIGN),
3826                             file, _cache->c_name);

3828                 if (_cache->c_data == NULL)
3829                         continue;

3831                 dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
3832                 dbg_print(0, MSG_INTL(MSG_ELF_SCN_NOTE), _cache->c_name);
3833                 note_entry(_cache, (Word *)_cache->c_data->d_buf,
3834                 /* LINTED */
3835                     (Word)_cache->c_data->d_size, ehdr, file);
3836         }

3838         return (note_cnt);
3839 }

3841 /*
3842  * The Linux Standard Base defines a special note named .note.ABI-tag
3843  * that is used to maintain Linux ABI information. Presence of this section
3844  * is a strong indication that the object should be considered to be
3845  * ELFOSABI_LINUX.
3846  *
3847  * This function returns True (1) if such a note is seen, and False (0)
3848  * otherwise.
3849  */
3850 static int
3851 has_linux_abi_note(Cache *cache, Word shnum, const char *file)
3852 {
3853         Word    cnt;

3855         for (cnt = 1; cnt < shnum; cnt++) {
3856                 parse_note_t    pnstate;
3857                 Cache           *_cache = &cache[cnt];
3858                 Shdr            *shdr = _cache->c_shdr;

3860                 /*
3861                  * Section must be SHT_NOTE, must have the name
3862                  * .note.ABI-tag, and must have data.
3863                  */
3864                 if ((shdr->sh_type != SHT_NOTE) ||
3865                     (strcmp(MSG_ORIG(MSG_STR_NOTEABITAG),
3866                     _cache->c_name) != 0) || (_cache->c_data == NULL))
3867                         continue;

3869                 pnstate.pns_file = file;
3870                 pnstate.pns_cache = _cache;
```

```
3871                pnstate.pns_size = _cache->c_data->d_size;
3872                pnstate.pns_data = (Word *)_cache->c_data->d_buf;

3874                while (pnstate.pns_size > 0) {
3875                        Word *w;

3877                        if (parse_note_entry(&pnstate) == 0)
3878                                break;

3880                        /*
3881                         * The type must be 1, and the name must be "GNU".
3882                         * The descsz must be at least 16 bytes.
3883                         */
3884                        if ((pnstate.pn_type != 1) ||
3885                            (pnstate.pn_namesz != (MSG_STR_GNU_SIZE + 1)) ||
3886                            (strncmp(MSG_ORIG(MSG_STR_GNU), pnstate.pn_name,
3887                            MSG_STR_CORE_SIZE + 1) != 0) ||
3888                            (pnstate.pn_descsz < 16))
3889                                continue;

3891                        /*
3892                         * desc contains 4 32-bit fields. Field 0 must be 0,
3893                         * indicating Linux. The second, third, and fourth
3894                         * fields represent the earliest Linux kernel
3895                         * version compatible with this object.
3896                         */
3897                        /*LINTED*/
3898                        w = (Word *) pnstate.pn_desc;
3899                        if (*w == 0)
3900                                return (1);
3901                }
3902        }

3904        return (0);
3905 }

3907 /*
3908  * Determine an individual hash entry.  This may be the initial hash entry,
3909  * or an associated chain entry.
3910  */
3911 static void
3912 hash_entry(Cache *refsec, Cache *strsec, const char *hsecname, Word hashndx,
3913     Word symndx, Word symn, Sym *syms, const char *file, ulong_t bkts,
3914     uint_t flags, int chain)
3915 {
3916        Sym             *sym;
3917        const char      *symname, *str;
3918        char            _bucket[MAXNDXSIZE], _symndx[MAXNDXSIZE];
3919        ulong_t         nbkt, nhash;

3921        if (symndx > symn) {
3922                (void) fprintf(stderr, MSG_INTL(MSG_ERR_HSBADSYMNDX), file,
3923                    EC_WORD(symndx), EC_WORD(hashndx));
3924                symname = MSG_INTL(MSG_STR_UNKNOWN);
3925        } else {
3926                sym = (Sym *)(syms + symndx);
3927                symname = string(refsec, symndx, strsec, file, sym->st_name);
3928        }

3930        if (chain == 0) {
3931                (void) snprintf(_bucket, MAXNDXSIZE, MSG_ORIG(MSG_FMT_INTEGER),
3932                    hashndx);
3933                str = (const char *)_bucket;
3934        } else
3935                str = MSG_ORIG(MSG_STR_EMPTY);
```



```
3937        (void) snprintf(_symndx, MAXNDXSIZE, MSG_ORIG(MSG_FMT_INDEX2),
3938            EC_WORD(symndx));
3939        dbg_print(0, MSG_ORIG(MSG_FMT_HASH_INFO), str, _symndx,
3940            demangle(symname, flags));

3942        /*
3943         * Determine if this string is in the correct bucket.
3944         */
3945        nhash = elf_hash(symname);
3946        nbkt = nhash % bkts;

3948        if (nbkt != hashndx) {
3949                (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADHASH), file,
3950                    hsecname, symname, EC_WORD(hashndx), nbkt);
3951        }
3952 }

3954 #define MAXCOUNT        500

3956 static void
3957 hash(Cache *cache, Word shnum, const char *file, uint_t flags)
3958 {
3959        static int      count[MAXCOUNT];
3960        Word            cnt;
3961        ulong_t         ndx, bkts;
3962        char            number[MAXNDXSIZE];

3964        for (cnt = 1; cnt < shnum; cnt++) {
3965                uint_t          *hash, *chain;
3966                Cache           *_cache = &cache[cnt];
3967                Shdr            *sshdr, *hshdr = _cache->c_shdr;
3968                char            *ssecname, *hsecname = _cache->c_name;
3969                Sym             *syms;
3970                Word            symn;

3972                if (hshdr->sh_type != SHT_HASH)
3973                        continue;

3975                /*
3976                 * Determine the hash table data and size.
3977                 */
3978                if ((hshdr->sh_entsize == 0) || (hshdr->sh_size == 0)) {
3979                        (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSZ),
3980                            file, hsecname);
3981                        continue;
3982                }
3983                if (_cache->c_data == NULL)
3984                        continue;

3986                hash = (uint_t *)_cache->c_data->d_buf;
3987                bkts = *hash;
3988                chain = hash + 2 + bkts;
3989                hash += 2;

3991                /*
3992                 * Get the data buffer for the associated symbol table.
3993                 */
3994                if ((hshdr->sh_link == 0) || (hshdr->sh_link >= shnum)) {
3995                        (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSHLINK),
3996                            file, hsecname, EC_WORD(hshdr->sh_link));
3997                        continue;
3998                }

4000                _cache = &cache[hshdr->sh_link];
4001                ssecname = _cache->c_name;
```

```
4003                if (_cache->c_data == NULL)
4004                        continue;
4005
4006                if ((syms = (Sym *)_cache->c_data->d_buf) == NULL) {
4007                        (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSZ),
4008                            file, ssecname);
4009                        continue;
4010                }
4011
4012                sshdr = _cache->c_shdr;
4013                /* LINTED */
4014                symn = (Word)(sshdr->sh_size / sshdr->sh_entsize);
4015
4016                /*
4017                 * Get the associated string table section.
4018                 */
4019                if ((sshdr->sh_link == 0) || (sshdr->sh_link >= shnum)) {
4020                        (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSHLINK),
4021                            file, ssecname, EC_WORD(sshdr->sh_link));
4022                        continue;
4023                }
4024
4025                dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
4026                dbg_print(0, MSG_INTL(MSG_ELF_SCN_HASH), hsecname);
4027                dbg_print(0, MSG_INTL(MSG_ELF_HASH_INFO));
4028
4029                /*
4030                 * Loop through the hash buckets, printing the appropriate
4031                 * symbols.
4032                 */
4033                for (ndx = 0; ndx < bkts; ndx++, hash++) {
4034                        Word    _ndx, _cnt;
4035
4036                        if (*hash == 0) {
4037                                count[0]++;
4038                                continue;
4039                        }
4040
4041                        hash_entry(_cache, &cache[sshdr->sh_link], hsecname,
4042                            ndx, *hash, symn, syms, file, bkts, flags, 0);
4043
4044                        /*
4045                         * Determine if any other symbols are chained to this
4046                         * bucket.
4047                         */
4048                        _ndx = chain[*hash];
4049                        _cnt = 1;
4050                        while (_ndx) {
4051                                hash_entry(_cache, &cache[sshdr->sh_link],
4052                                    hsecname, ndx, _ndx, symn, syms, file,
4053                                    bkts, flags, 1);
4054                                _ndx = chain[_ndx];
4055                                _cnt++;
4056                        }
4057
4058                        if (_cnt >= MAXCOUNT) {
4059                                (void) fprintf(stderr,
4060                                    MSG_INTL(MSG_HASH_OVERFLW), file,
4061                                    _cache->c_name, EC_WORD(ndx),
4062                                    EC_WORD(_cnt));
4063                        } else
4064                                count[_cnt]++;
4065                }
4066                break;
4067        }
```



```
4069        /*
4070         * Print out the count information.
4071         */
4072        bkts = cnt = 0;
4073        dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
4074
4075        for (ndx = 0; ndx < MAXCOUNT; ndx++) {
4076                Word    _cnt;
4077
4078                if ((_cnt = count[ndx]) == 0)
4079                        continue;
4080
4081                (void) snprintf(number, MAXNDXSIZE,
4082                    MSG_ORIG(MSG_FMT_INTEGER), _cnt);
4083                dbg_print(0, MSG_INTL(MSG_ELF_HASH_BKTS1), number,
4084                    EC_WORD(ndx));
4085                bkts += _cnt;
4086                cnt += (Word)(ndx * _cnt);
4087        }
4088        if (cnt) {
4089                (void) snprintf(number, MAXNDXSIZE, MSG_ORIG(MSG_FMT_INTEGER),
4090                    bkts);
4091                dbg_print(0, MSG_INTL(MSG_ELF_HASH_BKTS2), number,
4092                    EC_WORD(cnt));
4093        }
4094 }
4095
4096 static void
4097 group(Cache *cache, Word shnum, const char *file, uint_t flags)
4098 {
4099        Word    scnt;
4100
4101        for (scnt = 1; scnt < shnum; scnt++) {
4102                Cache           *_cache = &cache[scnt];
4103                Shdr            *shdr = _cache->c_shdr;
4104                Word            *grpdata, gcnt, grpcnt, symnum, unknown;
4105                Cache           *symsec, *strsec;
4106                Sym             *syms, *sym;
4107                char            flgstrbuf[MSG_GRP_COMDAT_SIZE + 10];
4108                const char      *grpnam;
4109
4110                if (shdr->sh_type != SHT_GROUP)
4111                        continue;
4112                if (!match(MATCH_F_ALL, _cache->c_name, scnt, shdr->sh_type))
4113                        continue;
4114                if ((_cache->c_data == NULL) ||
4115                    ((grpdata = (Word *)_cache->c_data->d_buf) == NULL))
4116                        continue;
4117                grpcnt = shdr->sh_size / sizeof (Word);
4118
4119                /*
4120                 * Get the data buffer for the associated symbol table and
4121                 * string table.
4122                 */
4123                if (stringtbl(cache, 1, scnt, shnum, file,
4124                    &symnum, &symsec, &strsec) == 0)
4125                        return;
4126
4127                syms = symsec->c_data->d_buf;
4128
4129                dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
4130                dbg_print(0, MSG_INTL(MSG_ELF_SCN_GRP), _cache->c_name);
4131                dbg_print(0, MSG_INTL(MSG_GRP_TITLE));
4132
4133                /*
4134                 * The first element of the group defines the group.  The
```

```
4135                 * associated symbol is defined by the sh_link field.
4136                 */
4137                if ((shdr->sh_info == SHN_UNDEF) || (shdr->sh_info > symnum)) {
4138                        (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSHINFO),
4139                            file, _cache->c_name, EC_WORD(shdr->sh_info));
4140                        return;
4141                }

4143                (void) strcpy(flgstrbuf, MSG_ORIG(MSG_STR_OSQBRKT));
4144                if (grpdata[0] & GRP_COMDAT) {
4145                        (void) strcat(flgstrbuf, MSG_ORIG(MSG_GRP_COMDAT));
4146                }
4147                if ((unknown = (grpdata[0] & ~GRP_COMDAT)) != 0) {
4148                        size_t  len = strlen(flgstrbuf);

4150                        (void) snprintf(&flgstrbuf[len],
4151                            (MSG_GRP_COMDAT_SIZE + 10 - len),
4152                            MSG_ORIG(MSG_GRP_UNKNOWN), unknown);
4153                }
4154                (void) strcat(flgstrbuf, MSG_ORIG(MSG_STR_CSQBRKT));
4155                sym = (Sym *)(syms + shdr->sh_info);

4157                /*
4158                 * The GNU assembler can use section symbols as the signature
4159                 * symbol as described by this comment in the gold linker
4160                 * (found via google):
4161                 *
4162                 *      It seems that some versions of gas will create a
4163                 *      section group associated with a section symbol, and
4164                 *      then fail to give a name to the section symbol.  In
4165                 *      such a case, use the name of the section.
4166                 *
4167                 * In order to support such objects, we do the same.
4168                 */
4169                grpnam = string(_cache, 0, strsec, file, sym->st_name);
4170                if (((sym->st_name == 0) || (*grpnam == '\0')) &&
4171                    (ELF_ST_TYPE(sym->st_info) == STT_SECTION))
4172                        grpnam = cache[sym->st_shndx].c_name;

4174                dbg_print(0, MSG_INTL(MSG_GRP_SIGNATURE), flgstrbuf,
4175                    demangle(grpnam, flags));

4177                for (gcnt = 1; gcnt < grpcnt; gcnt++) {
4178                        char            index[MAXNDXSIZE];
4179                        const char      *name;

4181                        (void) snprintf(index, MAXNDXSIZE,
4182                            MSG_ORIG(MSG_FMT_INDEX), EC_XWORD(gcnt));

4184                        if (grpdata[gcnt] >= shnum)
4185                                name = MSG_INTL(MSG_GRP_INVALSCN);
4186                        else
4187                                name = cache[grpdata[gcnt]].c_name;

4189                        (void) printf(MSG_ORIG(MSG_GRP_ENTRY), index, name,
4190                            EC_XWORD(grpdata[gcnt]));
4191                }
4192        }
4193 }

4195 static void
4196 got(Cache *cache, Word shnum, Ehdr *ehdr, const char *file)
4197 {
4198        Cache           *gotcache = NULL, *symtab = NULL;
4199        Addr            gotbgn, gotend;
4200        Shdr            *gotshdr;
```



```
4201        Word            cnt, gotents, gotndx;
4202        size_t          gentsize;
4203        Got_info        *gottable;
4204        char            *gotdata;
4205        Sym             *gotsym;
4206        Xword           gotsymaddr;
4207        uint_t          sys_encoding;

4209        /*
4210         * First, find the got.
4211         */
4212        for (cnt = 1; cnt < shnum; cnt++) {
4213                if (strncmp(cache[cnt].c_name, MSG_ORIG(MSG_ELF_GOT),
4214                    MSG_ELF_GOT_SIZE) == 0) {
4215                        gotcache = &cache[cnt];
4216                        break;
4217                }
4218        }
4219        if (gotcache == NULL)
4220                return;

4222        /*
4223         * A got section within a relocatable object is suspicious.
4224         */
4225        if (ehdr->e_type == ET_REL) {
4226                (void) fprintf(stderr, MSG_INTL(MSG_GOT_UNEXPECTED), file,
4227                    gotcache->c_name);
4228        }

4230        gotshdr = gotcache->c_shdr;
4231        if (gotshdr->sh_size == 0) {
4232                (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSZ),
4233                    file, gotcache->c_name);
4234                return;
4235        }

4237        gotbgn = gotshdr->sh_addr;
4238        gotend = gotbgn + gotshdr->sh_size;

4240        /*
4241         * Some architectures don't properly set the sh_entsize for the GOT
4242         * table.  If it's not set, default to a size of a pointer.
4243         */
4244        if ((gentsize = gotshdr->sh_entsize) == 0)
4245                gentsize = sizeof (Xword);

4247        if (gotcache->c_data == NULL)
4248                return;

4250        /* LINTED */
4251        gotents = (Word)(gotshdr->sh_size / gentsize);
4252        gotdata = gotcache->c_data->d_buf;

4254        if ((gottable = calloc(gotents, sizeof (Got_info))) == 0) {
4255                int err = errno;
4256                (void) fprintf(stderr, MSG_INTL(MSG_ERR_MALLOC), file,
4257                    strerror(err));
4258                return;
4259        }

4261        /*
4262         * Now we scan through all the sections looking for any relocations
4263         * that may be against the GOT.  Since these may not be isolated to a
4264         * .rel[a].got section we check them all.
4265         * While scanning sections save the symbol table entry (a symtab
4266         * overriding a dynsym) so that we can lookup _GLOBAL_OFFSET_TABLE_.
```

```
4267          */
4268         for (cnt = 1; cnt < shnum; cnt++) {
4269                 Word            type, symnum;
4270                 Xword           relndx, relnum, relsize;
4271                 void            *rels;
4272                 Sym             *syms;
4273                 Cache           *symsec, *strsec;
4274                 Cache           *_cache = &cache[cnt];
4275                 Shdr            *shdr;

4277                 shdr = _cache->c_shdr;
4278                 type = shdr->sh_type;

4280                 if ((symtab == 0) && (type == SHT_DYNSYM)) {
4281                         symtab = _cache;
4282                         continue;
4283                 }
4284                 if (type == SHT_SYMTAB) {
4285                         symtab = _cache;
4286                         continue;
4287                 }
4288                 if ((type != SHT_RELA) && (type != SHT_REL))
4289                         continue;

4291                 /*
4292                  * Decide entry size.
4293                  */
4294                 if (((relsize = shdr->sh_entsize) == 0) ||
4295                     (relsize > shdr->sh_size)) {
4296                         if (type == SHT_RELA)
4297                                 relsize = sizeof (Rela);
4298                         else
4299                                 relsize = sizeof (Rel);
4300                 }

4302                 /*
4303                  * Determine the number of relocations available.
4304                  */
4305                 if (shdr->sh_size == 0) {
4306                         (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSZ),
4307                             file, _cache->c_name);
4308                         continue;
4309                 }
4310                 if (_cache->c_data == NULL)
4311                         continue;

4313                 rels = _cache->c_data->d_buf;
4314                 relnum = shdr->sh_size / relsize;

4316                 /*
4317                  * Get the data buffer for the associated symbol table and
4318                  * string table.
4319                  */
4320                 if (stringtbl(cache, 1, cnt, shnum, file,
4321                     &symnum, &symsec, &strsec) == 0)
4322                         continue;

4324                 syms = symsec->c_data->d_buf;

4326                 /*
4327                  * Loop through the relocation entries.
4328                  */
4329                 for (relndx = 0; relndx < relnum; relndx++,
4330                     rels = (void *)((char *)rels + relsize)) {
4331                         char            section[BUFSIZ];
4332                         Addr            offset;
```



```
4333                         Got_info        *gip;
4334                         Word            symndx, reltype;
4335                         Rela            *rela;
4336                         Rel             *rel;

4338                         /*
4339                          * Unravel the relocation.
4340                          */
4341                         if (type == SHT_RELA) {
4342                                 rela = (Rela *)rels;
4343                                 symndx = ELF_R_SYM(rela->r_info);
4344                                 reltype = ELF_R_TYPE(rela->r_info,
4345                                     ehdr->e_machine);
4346                                 offset = rela->r_offset;
4347                         } else {
4348                                 rel = (Rel *)rels;
4349                                 symndx = ELF_R_SYM(rel->r_info);
4350                                 reltype = ELF_R_TYPE(rel->r_info,
4351                                     ehdr->e_machine);
4352                                 offset = rel->r_offset;
4353                         }

4355                         /*
4356                          * Only pay attention to relocations against the GOT.
4357                          */
4358                         if ((offset < gotbgn) || (offset >= gotend))
4359                                 continue;

4361                         /* LINTED */
4362                         gotndx = (Word)((offset - gotbgn) /
4363                             gotshdr->sh_entsize);
4364                         gip = &gottable[gotndx];

4366                         if (gip->g_reltype != 0) {
4367                                 (void) fprintf(stderr,
4368                                     MSG_INTL(MSG_GOT_MULTIPLE), file,
4369                                     EC_WORD(gotndx), EC_ADDR(offset));
4370                                 continue;
4371                         }

4373                         if (symndx)
4374                                 gip->g_symname = relsymname(cache, _cache,
4375                                     strsec, symndx, symnum, relndx, syms,
4376                                     section, BUFSIZ, file);
4377                         gip->g_reltype = reltype;
4378                         gip->g_rel = rels;
4379                 }
4380         }

4382         if (symlookup(MSG_ORIG(MSG_SYM_GOT), cache, shnum, &gotsym, NULL,
4383             symtab, file))
4384                 gotsymaddr = gotsym->st_value;
4385         else
4386                 gotsymaddr = gotbgn;

4388         dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
4389         dbg_print(0, MSG_INTL(MSG_ELF_SCN_GOT), gotcache->c_name);
4390         Elf_got_title(0);

4392         sys_encoding = _elf_sys_encoding();
4393         for (gotndx = 0; gotndx < gotents; gotndx++) {
4394                 Got_info        *gip;
4395                 Sword           gindex;
4396                 Addr            gaddr;
4397                 Xword           gotentry;
```

```
4399                 gip = &gottable[gotndx];

4401                 gaddr = gotbgn + (gotndx * gentsize);
4402                 gindex = (Sword)(gaddr - gotsymaddr) / (Sword)gentsize;

4404                 if (gentsize == sizeof (Word))
4405                         /* LINTED */
4406                         gotentry = (Xword)(*((Word *)(gotdata) + gotndx));
4407                 else
4408                         /* LINTED */
4409                         gotentry = *((Xword *)(gotdata) + gotndx);

4411                 Elf_got_entry(0, gindex, gaddr, gotentry, ehdr->e_machine,
4412                     ehdr->e_ident[EI_DATA], sys_encoding,
4413                     gip->g_reltype, gip->g_rel, gip->g_symname);
4414         }
4415         free(gottable);
4416 }

4418 void
4419 checksum(Elf *elf)
4420 {
4421         dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
4422         dbg_print(0, MSG_INTL(MSG_STR_CHECKSUM), elf_checksum(elf));
4423 }

4425 /*
4426  * This variable is used by regular() to communicate the address of
4427  * the section header cache to sort_shdr_ndx_arr(). Unfortunately,
4428  * the qsort() interface does not include a userdata argument by which
4429  * such arbitrary data can be passed, so we are stuck using global data.
4430  */
4431 static Cache *sort_shdr_ndx_arr_cache;


4434 /*
4435  * Used with qsort() to sort the section indices so that they can be
4436  * used to access the section headers in order of increasing data offset.
4437  *
4438  * entry:
4439  *      sort_shdr_ndx_arr_cache - Contains address of
4440  *              section header cache.
4441  *      v1, v2 - Point at elements of sort_shdr_bits array to be compared.
4442  *
4443  * exit:
4444  *      Returns -1 (less than), 0 (equal) or 1 (greater than).
4445  */
4446 static int
4447 sort_shdr_ndx_arr(const void *v1, const void *v2)
4448 {
4449         Cache   *cache1 = sort_shdr_ndx_arr_cache + *((size_t *)v1);
4450         Cache   *cache2 = sort_shdr_ndx_arr_cache + *((size_t *)v2);

4452         if (cache1->c_shdr->sh_offset < cache2->c_shdr->sh_offset)
4453                 return (-1);

4455         if (cache1->c_shdr->sh_offset > cache2->c_shdr->sh_offset)
4456                 return (1);

4458         return (0);
4459 }


4462 static int
4463 shdr_cache(const char *file, Elf *elf, Ehdr *ehdr, size_t shstrndx,
4464     size_t shnum, Cache **cache_ret, Word flags)
```



```
4465 {
4466         Elf_Scn         *scn;
4467         Elf_Data        *data;
4468         size_t          ndx;
4469         Shdr            *nameshdr;
4470         char            *names = NULL;
4471         Cache           *cache, *_cache;
4472         size_t          *shdr_ndx_arr, shdr_ndx_arr_cnt;


4475         /*
4476          * Obtain the .shstrtab data buffer to provide the required section
4477          * name strings.
4478          */
4479         if (shstrndx == SHN_UNDEF) {
4480                 /*
4481                  * It is rare, but legal, for an object to lack a
4482                  * header string table section.
4483                  */
4484                 names = NULL;
4485                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_NOSHSTRSEC), file);
4486         } else if ((scn = elf_getscn(elf, shstrndx)) == NULL) {
4487                 failure(file, MSG_ORIG(MSG_ELF_GETSCN));
4488                 (void) fprintf(stderr, MSG_INTL(MSG_ELF_ERR_SHDR),
4489                     EC_XWORD(shstrndx));

4491         } else if ((data = elf_getdata(scn, NULL)) == NULL) {
4492                 failure(file, MSG_ORIG(MSG_ELF_GETDATA));
4493                 (void) fprintf(stderr, MSG_INTL(MSG_ELF_ERR_DATA),
4494                     EC_XWORD(shstrndx));

4496         } else if ((nameshdr = elf_getshdr(scn)) == NULL) {
4497                 failure(file, MSG_ORIG(MSG_ELF_GETSHDR));
4498                 (void) fprintf(stderr, MSG_INTL(MSG_ELF_ERR_SCN),
4499                     EC_WORD(elf_ndxscn(scn)));

4501         } else if ((names = data->d_buf) == NULL)
4502                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_SHSTRNULL), file);

4504         /*
4505          * Allocate a cache to maintain a descriptor for each section.
4506          */
4507         if ((*cache_ret = cache = malloc(shnum * sizeof (Cache))) == NULL) {
4508                 int err = errno;
4509                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_MALLOC),
4510                     file, strerror(err));
4511                 return (0);
4512         }

4514         *cache = cache_init;
4515         _cache = cache;
4516         _cache++;

4518         /*
4519          * Allocate an array that will hold the section index for
4520          * each section that has data in the ELF file:
4521          *
4522          *      - Is not a NOBITS section
4523          *      - Data has non-zero length
4524          *
4525          * Note that shnum is an upper bound on the size required. It
4526          * is likely that we won't use a few of these array elements.
4527          * Allocating a modest amount of extra memory in this case means
4528          * that we can avoid an extra loop to count the number of needed
4529          * items, and can fill this array immediately in the first loop
4530          * below.
```

```
4531          */
4532         if ((shdr_ndx_arr = malloc(shnum * sizeof (*shdr_ndx_arr))) == NULL) {
4533                 int err = errno;
4534                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_MALLOC),
4535                     file, strerror(err));
4536                 return (0);
4537         }
4538         shdr_ndx_arr_cnt = 0;
4539
4540         /*
4541          * Traverse the sections of the file.  This gathering of data is
4542          * carried out in two passes.  First, the section headers are captured
4543          * and the section header names are evaluated.  A verification pass is
4544          * then carried out over the section information.  Files have been
4545          * known to exhibit overlapping (and hence erroneous) section header
4546          * information.
4547          *
4548          * Finally, the data for each section is obtained.  This processing is
4549          * carried out after section verification because should any section
4550          * header overlap occur, and a file needs translating (ie. xlate'ing
4551          * information from a non-native architecture file), then the process
4552          * of translation can corrupt the section header information.  Of
4553          * course, if there is any section overlap, the data related to the
4554          * sections is going to be compromised.  However, it is the translation
4555          * of this data that has caused problems with elfdump()'s ability to
4556          * extract the data.
4557          */
4558         for (ndx = 1, scn = NULL; scn = elf_nextscn(elf, scn);
4559             ndx++, _cache++) {
4560                 char    scnndxnm[100];
4561
4562                 _cache->c_ndx = ndx;
4563                 _cache->c_scn = scn;
4564
4565                 if ((_cache->c_shdr = elf_getshdr(scn)) == NULL) {
4566                         failure(file, MSG_ORIG(MSG_ELF_GETSHDR));
4567                         (void) fprintf(stderr, MSG_INTL(MSG_ELF_ERR_SCN),
4568                             EC_WORD(elf_ndxscn(scn)));
4569                 }
4570
4571                 /*
4572                  * If this section has data in the file, include it in
4573                  * the array of sections to check for address overlap.
4574                  */
4575                 if ((_cache->c_shdr->sh_size != 0) &&
4576                     (_cache->c_shdr->sh_type != SHT_NOBITS))
4577                         shdr_ndx_arr[shdr_ndx_arr_cnt++] = ndx;
4578
4579                 /*
4580                  * If a shstrtab exists, assign the section name.
4581                  */
4582                 if (names && _cache->c_shdr) {
4583                         if (_cache->c_shdr->sh_name &&
4584                             /* LINTED */
4585                             (nameshdr->sh_size > _cache->c_shdr->sh_name)) {
4586                                 const char      *symname;
4587                                 char            *secname;
4588
4589                                 secname = names + _cache->c_shdr->sh_name;
4590
4591                                 /*
4592                                  * A SUN naming convention employs a "%" within
4593                                  * a section name to indicate a section/symbol
4594                                  * name.  This originated from the compilers
4595                                  * -xF option, that places functions into their
4596                                  * own sections.  This convention (which has no
```



```
4597                                  * formal standard) has also been followed for
4598                                  * COMDAT sections.  To demangle the symbol
4599                                  * name, the name must be separated from the
4600                                  * section name.
4601                                  */
4602                                 if (((flags & FLG_CTL_DEMANGLE) == 0) ||
4603                                     ((symname = strchr(secname, '%')) == NULL))
4604                                         _cache->c_name = secname;
4605                                 else {
4606                                         size_t  secsz = ++symname - secname;
4607                                         size_t  strsz;
4608
4609                                         symname = demangle(symname, flags);
4610                                         strsz = secsz + strlen(symname) + 1;
4611
4612                                         if ((_cache->c_name =
4613                                             malloc(strsz)) == NULL) {
4614                                                 int err = errno;
4615                                                 (void) fprintf(stderr,
4616                                                     MSG_INTL(MSG_ERR_MALLOC),
4617                                                     file, strerror(err));
4618                                                 return (0);
4619                                         }
4620                                         (void) snprintf(_cache->c_name, strsz,
4621                                             MSG_ORIG(MSG_FMT_SECSYM),
4622                                             EC_WORD(secsz), secname, symname);
4623                                 }
4624
4625                                 continue;
4626                         }
4627
4628                         /*
4629                          * Generate an error if the section name index is zero
4630                          * or exceeds the shstrtab data.  Fall through to
4631                          * fabricate a section name.
4632                          */
4633                         if ((_cache->c_shdr->sh_name == 0) ||
4634                             /* LINTED */
4635                             (nameshdr->sh_size <= _cache->c_shdr->sh_name)) {
4636                                 (void) fprintf(stderr,
4637                                     MSG_INTL(MSG_ERR_BADSHNAME), file,
4638                                     EC_WORD(ndx),
4639                                     EC_XWORD(_cache->c_shdr->sh_name));
4640                         }
4641                 }
4642
4643                 /*
4644                  * If there exists no shstrtab data, or a section header has no
4645                  * name (an invalid index of 0), then compose a name for the
4646                  * section.
4647                  */
4648                 (void) snprintf(scnndxnm, sizeof (scnndxnm),
4649                     MSG_INTL(MSG_FMT_SCNNDX), ndx);
4650
4651                 if ((_cache->c_name = malloc(strlen(scnndxnm) + 1)) == NULL) {
4652                         int err = errno;
4653                         (void) fprintf(stderr, MSG_INTL(MSG_ERR_MALLOC),
4654                             file, strerror(err));
4655                         return (0);
4656                 }
4657                 (void) strcpy(_cache->c_name, scnndxnm);
4658         }
4659
4660         /*
4661          * Having collected all the sections, validate their address range.
4662          * Cases have existed where the section information has been invalid.
```

```
4663          * This can lead to all sorts of other, hard to diagnose errors, as
4664          * each section is processed individually (ie. with elf_getdata()).
4665          * Here, we carry out some address comparisons to catch a family of
4666          * overlapping memory issues we have observed (likely, there are others
4667          * that we have yet to discover).
4668          *
4669          * Note, should any memory overlap occur, obtaining any additional
4670          * data from the file is questionable.  However, it might still be
4671          * possible to inspect the ELF header, Programs headers, or individual
4672          * sections, so rather than bailing on an error condition, continue
4673          * processing to see if any data can be salvaged.
4674          */
4675         if (shdr_ndx_arr_cnt > 1) {
4676                 sort_shdr_ndx_arr_cache = cache;
4677                 qsort(shdr_ndx_arr, shdr_ndx_arr_cnt,
4678                     sizeof (*shdr_ndx_arr), sort_shdr_ndx_arr);
4679         }
4680         for (ndx = 0; ndx < shdr_ndx_arr_cnt; ndx++) {
4681                 Cache   *_cache = cache + shdr_ndx_arr[ndx];
4682                 Shdr    *shdr = _cache->c_shdr;
4683                 Off     bgn1, bgn = shdr->sh_offset;
4684                 Off     end1, end = shdr->sh_offset + shdr->sh_size;
4685                 size_t  ndx1;

4687                 /*
4688                  * Check the section against all following ones, reporting
4689                  * any overlaps. Since we've sorted the sections by offset,
4690                  * we can stop after the first comparison that fails. There
4691                  * are no overlaps in a properly formed ELF file, in which
4692                  * case this algorithm runs in O(n) time. This will degenerate
4693                  * to O(n^2) for a completely broken file. Such a file is
4694                  * (1) highly unlikely, and (2) unusable, so it is reasonable
4695                  * for the analysis to take longer.
4696                  */
4697                 for (ndx1 = ndx + 1; ndx1 < shdr_ndx_arr_cnt; ndx1++) {
4698                         Cache   *_cache1 = cache + shdr_ndx_arr[ndx1];
4699                         Shdr    *shdr1 = _cache1->c_shdr;

4701                         bgn1 = shdr1->sh_offset;
4702                         end1 = shdr1->sh_offset + shdr1->sh_size;

4704                         if (((bgn1 <= bgn) && (end1 > bgn)) ||
4705                             ((bgn1 < end) && (end1 >= end))) {
4706                                 (void) fprintf(stderr,
4707                                     MSG_INTL(MSG_ERR_SECMEMOVER), file,
4708                                     EC_WORD(elf_ndxscn(_cache->c_scn)),
4709                                     _cache->c_name, EC_OFF(bgn), EC_OFF(end),
4710                                     EC_WORD(elf_ndxscn(_cache1->c_scn)),
4711                                     _cache1->c_name, EC_OFF(bgn1),
4712                                     EC_OFF(end1));
4713                         } else {        /* No overlap, so can stop */
4714                                 break;
4715                         }
4716                 }

4718                 /*
4719                  * In addition to checking for sections overlapping
4720                  * each other (done above), we should also make sure
4721                  * the section doesn't overlap the section header array.
4722                  */
4723                 bgn1 = ehdr->e_shoff;
4724                 end1 = ehdr->e_shoff + (ehdr->e_shentsize * ehdr->e_shnum);

4726                 if (((bgn1 <= bgn) && (end1 > bgn)) ||
4727                     ((bgn1 < end) && (end1 >= end))) {
4728                         (void) fprintf(stderr,
```



```
4729                             MSG_INTL(MSG_ERR_SHDRMEMOVER), file, EC_OFF(bgn1),
4730                             EC_OFF(end1),
4731                             EC_WORD(elf_ndxscn(_cache->c_scn)),
4732                             _cache->c_name, EC_OFF(bgn), EC_OFF(end));
4733                 }
4734         }

4736         /*
4737          * Obtain the data for each section.
4738          */
4739         for (ndx = 1; ndx < shnum; ndx++) {
4740                 Cache   *_cache = &cache[ndx];
4741                 Elf_Scn *scn = _cache->c_scn;

4743                 if ((_cache->c_data = elf_getdata(scn, NULL)) == NULL) {
4744                         failure(file, MSG_ORIG(MSG_ELF_GETDATA));
4745                         (void) fprintf(stderr, MSG_INTL(MSG_ELF_ERR_SCNDATA),
4746                             EC_WORD(elf_ndxscn(scn)));
4747                 }

4749                 /*
4750                  * If a string table, verify that it has NULL first and
4751                  * final bytes.
4752                  */
4753                 if ((_cache->c_shdr->sh_type == SHT_STRTAB) &&
4754                     (_cache->c_data->d_buf != NULL) &&
4755                     (_cache->c_data->d_size > 0)) {
4756                         const char *s = _cache->c_data->d_buf;

4758                         if ((*s != '\0') ||
4759                             (*(s + _cache->c_data->d_size - 1) != '\0'))
4760                                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_MALSTR),
4761                                     file, _cache->c_name);
4762                 }
4763         }

4765         return (1);
4766 }



4770 /*
4771  * Generate a cache of section headers and related information
4772  * for use by the rest of elfdump. If requested (or the file
4773  * contains no section headers), we generate a fake set of
4774  * headers from the information accessible from the program headers.
4775  * Otherwise, we use the real section headers contained in the file.
4776  */
4777 static int
4778 create_cache(const char *file, int fd, Elf *elf, Ehdr *ehdr, Cache **cache,
4779     size_t shstrndx, size_t *shnum, uint_t *flags)
4780 {
4781         /*
4782          * If there are no section headers, then resort to synthesizing
4783          * section headers from the program headers. This is normally
4784          * only done by explicit request, but in this case there's no
4785          * reason not to go ahead, since the alternative is simply to quit.
4786          */
4787         if ((*shnum <= 1) && ((*flags & FLG_CTL_FAKESHDR) == 0)) {
4788                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_NOSHDR), file);
4789                 *flags |= FLG_CTL_FAKESHDR;
4790         }

4792         if (*flags & FLG_CTL_FAKESHDR) {
4793                 if (fake_shdr_cache(file, fd, elf, ehdr, cache, shnum) == 0)
4794                         return (0);
```

```
4795         } else {
4796                 if (shdr_cache(file, elf, ehdr, shstrndx, *shnum,
4797                     cache, *flags) == 0)
4798                         return (0);
4799         }

4801         return (1);
4802 }

4804 int
4805 regular(const char *file, int fd, Elf *elf, uint_t flags,
4806     const char *wname, int wfd, uchar_t osabi)
4807 {
4808         enum { CACHE_NEEDED, CACHE_OK, CACHE_FAIL} cache_state = CACHE_NEEDED;
4809         Elf_Scn         *scn;
4810         Ehdr            *ehdr;
4811         size_t          ndx, shstrndx, shnum, phnum;
4812         Shdr            *shdr;
4813         Cache           *cache;
4814         VERSYM_STATE    versym = { 0 };
4815         int             ret = 0;
4816         int             addr_align;

4818         if ((ehdr = elf_getehdr(elf)) == NULL) {
4819                 failure(file, MSG_ORIG(MSG_ELF_GETEHDR));
4820                 return (ret);
4821         }

4823         if (elf_getshdrnum(elf, &shnum) == -1) {
4824                 failure(file, MSG_ORIG(MSG_ELF_GETSHDRNUM));
4825                 return (ret);
4826         }

4828         if (elf_getshdrstrndx(elf, &shstrndx) == -1) {
4829                 failure(file, MSG_ORIG(MSG_ELF_GETSHDRSTRNDX));
4830                 return (ret);
4831         }

4833         if (elf_getphdrnum(elf, &phnum) == -1) {
4834                 failure(file, MSG_ORIG(MSG_ELF_GETPHDRNUM));
4835                 return (ret);
4836         }
4837         /*
4838          * If the user requested section headers derived from the
4839          * program headers (-P option) and this file doesn't have
4840          * any program headers (i.e. ET_REL), then we can't do it.
4841          */
4842         if ((phnum == 0) && (flags & FLG_CTL_FAKESHDR)) {
4843                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_PNEEDSPH), file);
4844                 return (ret);
4845         }


4848         if ((scn = elf_getscn(elf, 0)) != NULL) {
4849                 if ((shdr = elf_getshdr(scn)) == NULL) {
4850                         failure(file, MSG_ORIG(MSG_ELF_GETSHDR));
4851                         (void) fprintf(stderr, MSG_INTL(MSG_ELF_ERR_SCN), 0);
4852                         return (ret);
4853                 }
4854         } else
4855                 shdr = NULL;

4857         /*
4858          * Print the elf header.
4859          */
4860         if (flags & FLG_SHOW_EHDR)
```



```
4861                 Elf_ehdr(0, ehdr, shdr);

4863         /*
4864          * If the section headers or program headers have inadequate
4865          * alignment for the class of object, print a warning. libelf
4866          * can handle such files, but programs that use them can crash
4867          * when they dereference unaligned items.
4868          *
4869          * Note that the AMD64 ABI, although it is a 64-bit architecture,
4870          * allows access to data types smaller than 128-bits to be on
4871          * word alignment.
4872          */
4873         if (ehdr->e_machine == EM_AMD64)
4874                 addr_align = sizeof (Word);
4875         else
4876                 addr_align = sizeof (Addr);

4878         if (ehdr->e_phoff & (addr_align - 1))
4879                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADPHDRALIGN), file);
4880         if (ehdr->e_shoff & (addr_align - 1))
4881                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSHDRALIGN), file);


4884         /*
4885          * Determine the Operating System ABI (osabi) we will use to
4886          * interpret the object.
4887          */
4888         if (flags & FLG_CTL_OSABI) {
4889                 /*
4890                  * If the user explicitly specifies '-O none', we need
4891                  * to display a completely generic view of the file.
4892                  * However, libconv is written to assume that ELFOSABI_NONE
4893                  * is equivalent to ELFOSABI_SOLARIS. To get the desired
4894                  * effect, we use an osabi that libconv has no knowledge of.
4895                  */
4896                 if (osabi == ELFOSABI_NONE)
4897                         osabi = ELFOSABI_UNKNOWN4;
4898         } else {
4899                 /* Determine osabi from file */
4900                 osabi = ehdr->e_ident[EI_OSABI];
4901                 if (osabi == ELFOSABI_NONE) {
4902                         /*
4903                          * Chicken/Egg scenario:
4904                          *
4905                          * Ideally, we wait to create the section header cache
4906                          * until after the program headers are printed. If we
4907                          * only output program headers, we can skip building
4908                          * the cache entirely.
4909                          *
4910                          * Proper interpretation of program headers requires
4911                          * the osabi, which is supposed to be in the ELF header.
4912                          * However, many systems (Solaris and Linux included)
4913                          * have a history of setting the osabi to the generic
4914                          * SysV ABI (ELFOSABI_NONE). We assume ELFOSABI_SOLARIS
4915                          * in such cases, but would like to check the object
4916                          * to see if it has a Linux .note.ABI-tag section,
4917                          * which implies ELFOSABI_LINUX. This requires a
4918                          * section header cache.
4919                          *
4920                          * To break the cycle, we create section headers now
4921                          * if osabi is ELFOSABI_NONE, and later otherwise.
4922                          * If it succeeds, we use them, if not, we defer
4923                          * exiting until after the program headers are out.
4924                          */
4925                         if (create_cache(file, fd, elf, ehdr, &cache,
4926                             shstrndx, &shnum, &flags) == 0) {
```

```
4927                                cache_state = CACHE_FAIL;
4928                        } else {
4929                                cache_state = CACHE_OK;
4930                                if (has_linux_abi_note(cache, shnum, file)) {
4931                                        Conv_inv_buf_t  ibuf1, ibuf2;

4933                                        (void) fprintf(stderr,
4934                                            MSG_INTL(MSG_INFO_LINUXOSABI), file,
4935                                            conv_ehdr_osabi(osabi, 0, &ibuf1),
4936                                            conv_ehdr_osabi(ELFOSABI_LINUX,
4937                                            0, &ibuf2));
4938                                        osabi = ELFOSABI_LINUX;
4939                                }
4940                        }
4941                }
4942                /*
4943                 * We treat ELFOSABI_NONE identically to ELFOSABI_SOLARIS.
4944                 * Mapping NONE to SOLARIS simplifies the required test.
4945                 */
4946                if (osabi == ELFOSABI_NONE)
4947                        osabi = ELFOSABI_SOLARIS;
4948        }

4950        /*
4951         * Print the program headers.
4952         */
4953        if ((flags & FLG_SHOW_PHDR) && (phnum != 0)) {
4954                Phdr    *phdr;

4956                if ((phdr = elf_getphdr(elf)) == NULL) {
4957                        failure(file, MSG_ORIG(MSG_ELF_GETPHDR));
4958                        return (ret);
4959                }

4961                for (ndx = 0; ndx < phnum; phdr++, ndx++) {
4962                        if (!match(MATCH_F_PHDR| MATCH_F_NDX | MATCH_F_TYPE,
4963                            NULL, ndx, phdr->p_type))
4964                                continue;

4966                        dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
4967                        dbg_print(0, MSG_INTL(MSG_ELF_PHDR), EC_WORD(ndx));
4968                        Elf_phdr(0, osabi, ehdr->e_machine, phdr);
4969                }
4970        }

4972        /*
4973         * If we have flag bits set that explicitly require a show or calc
4974         * operation, but none of them require the section headers, then
4975         * we are done and can return now.
4976         */
4977        if (((flags & (FLG_MASK_SHOW | FLG_MASK_CALC)) != 0) &&
4978            ((flags & (FLG_MASK_SHOW_SHDR | FLG_MASK_CALC_SHDR)) == 0))
4979                return (ret);

4981        /*
4982         * Everything from this point on requires section headers.
4983         * If we have no section headers, there is no reason to continue.
4984         *
4985         * If we tried above to create the section header cache and failed,
4986         * it is time to exit. Otherwise, create it if needed.
4987         */
4988        switch (cache_state) {
4989        case CACHE_NEEDED:
4990                if (create_cache(file, fd, elf, ehdr, &cache, shstrndx,
4991                    &shnum, &flags) == 0)
4992                        return (ret);
```



```
4993                    break;
4994            case CACHE_OK:
4995                    break;
4996    #endif /* ! codereview */
4997            case CACHE_FAIL:
4998                    return (ret);
4999            }
5000            if (shnum <= 1)
5001                    goto done;

5003            /*
5004             * If -w was specified, find and write out the section(s) data.
5005             */
5006            if (wfd) {
5007                    for (ndx = 1; ndx < shnum; ndx++) {
5008                            Cache   *_cache = &cache[ndx];

5010                            if (match(MATCH_F_STRICT | MATCH_F_ALL, _cache->c_name,
5011                                ndx, _cache->c_shdr->sh_type) &&
5012                                _cache->c_data && _cache->c_data->d_buf) {
5013                                    if (write(wfd, _cache->c_data->d_buf,
5014                                        _cache->c_data->d_size) !=
5015                                        _cache->c_data->d_size) {
5016                                            int err = errno;
5017                                            (void) fprintf(stderr,
5018                                                MSG_INTL(MSG_ERR_WRITE), wname,
5019                                                strerror(err));
5020                                            /*
5021                                             * Return an exit status of 1, because
5022                                             * the failure is not related to the
5023                                             * ELF file, but by system resources.
5024                                             */
5025                                            ret = 1;
5026                                            goto done;
5027                                    }
5028                            }
5029                    }
5030            }

5032            /*
5033             * If we have no flag bits set that explicitly require a show or calc
5034             * operation, but match options (-I, -N, -T) were used, then run
5035             * through the section headers and see if we can't deduce show flags
5036             * from the match options given.
5037             *
5038             * We don't do this if -w was specified, because (-I, -N, -T) used
5039             * with -w in lieu of some other option is supposed to be quiet.
5040             */
5041            if ((wfd == 0) && (flags & FLG_CTL_MATCH) &&
5042                ((flags & (FLG_MASK_SHOW | FLG_MASK_CALC)) == 0)) {
5043                    for (ndx = 1; ndx < shnum; ndx++) {
5044                            Cache   *_cache = &cache[ndx];

5046                            if (!match(MATCH_F_STRICT | MATCH_F_ALL, _cache->c_name,
5047                                ndx, _cache->c_shdr->sh_type))
5048                                    continue;

5050                            switch (_cache->c_shdr->sh_type) {
5051                            case SHT_PROGBITS:
5052                                    /*
5053                                     * Heuristic time: It is usually bad form
5054                                     * to assume the meaning/format of a PROGBITS
5055                                     * section based on its name. However, there
5056                                     * are ABI mandated exceptions. Check for
5057                                     * these special names.
5058                                     */
```

```
5060                                /* The ELF ABI specifies .interp and .got */
5061                                if (strcmp(_cache->c_name,
5062                                    MSG_ORIG(MSG_ELF_INTERP)) == 0) {
5063                                        flags |= FLG_SHOW_INTERP;
5064                                        break;
5065                                }
5066                                if (strcmp(_cache->c_name,
5067                                    MSG_ORIG(MSG_ELF_GOT)) == 0) {
5068                                        flags |= FLG_SHOW_GOT;
5069                                        break;
5070                                }
5071                                /*
5072                                 * The GNU compilers, and amd64 ABI, define
5073                                 * .eh_frame and .eh_frame_hdr. The Sun
5074                                 * C++ ABI defines .exception_ranges.
5075                                 */
5076                                if ((strncmp(_cache->c_name,
5077                                    MSG_ORIG(MSG_SCN_FRM),
5078                                    MSG_SCN_FRM_SIZE) == 0) ||
5079                                    (strncmp(_cache->c_name,
5080                                    MSG_ORIG(MSG_SCN_EXRANGE),
5081                                    MSG_SCN_EXRANGE_SIZE) == 0)) {
5082                                        flags |= FLG_SHOW_UNWIND;
5083                                        break;
5084                                }
5085                                break;

5087                        case SHT_SYMTAB:
5088                        case SHT_DYNSYM:
5089                        case SHT_SUNW_LDYNSYM:
5090                        case SHT_SUNW_versym:
5091                        case SHT_SYMTAB_SHNDX:
5092                                flags |= FLG_SHOW_SYMBOLS;
5093                                break;

5095                        case SHT_RELA:
5096                        case SHT_REL:
5097                                flags |= FLG_SHOW_RELOC;
5098                                break;

5100                        case SHT_HASH:
5101                                flags |= FLG_SHOW_HASH;
5102                                break;

5104                        case SHT_DYNAMIC:
5105                                flags |= FLG_SHOW_DYNAMIC;
5106                                break;

5108                        case SHT_NOTE:
5109                                flags |= FLG_SHOW_NOTE;
5110                                break;

5112                        case SHT_GROUP:
5113                                flags |= FLG_SHOW_GROUP;
5114                                break;

5116                        case SHT_SUNW_symsort:
5117                        case SHT_SUNW_tlssort:
5118                                flags |= FLG_SHOW_SORT;
5119                                break;

5121                        case SHT_SUNW_cap:
5122                                flags |= FLG_SHOW_CAP;
5123                                break;
```



```
5125                        case SHT_SUNW_move:
5126                                flags |= FLG_SHOW_MOVE;
5127                                break;

5129                        case SHT_SUNW_syminfo:
5130                                flags |= FLG_SHOW_SYMINFO;
5131                                break;

5133                        case SHT_SUNW_verdef:
5134                        case SHT_SUNW_verneed:
5135                                flags |= FLG_SHOW_VERSIONS;
5136                                break;

5138                        case SHT_AMD64_UNWIND:
5139                                flags |= FLG_SHOW_UNWIND;
5140                                break;
5141                        }
5142                }
5143        }


5146        if (flags & FLG_SHOW_SHDR)
5147                sections(file, cache, shnum, ehdr, osabi);

5149        if (flags & FLG_SHOW_INTERP)
5150                interp(file, cache, shnum, phnum, elf);

5152        if ((osabi == ELFOSABI_SOLARIS) || (osabi == ELFOSABI_LINUX))
5153                versions(cache, shnum, file, flags, &versym);

5155        if (flags & FLG_SHOW_SYMBOLS)
5156                symbols(cache, shnum, ehdr, osabi, &versym, file, flags);

5158        if ((flags & FLG_SHOW_SORT) && (osabi == ELFOSABI_SOLARIS))
5159                sunw_sort(cache, shnum, ehdr, osabi, &versym, file, flags);

5161        if (flags & FLG_SHOW_HASH)
5162                hash(cache, shnum, file, flags);

5164        if (flags & FLG_SHOW_GOT)
5165                got(cache, shnum, ehdr, file);

5167        if (flags & FLG_SHOW_GROUP)
5168                group(cache, shnum, file, flags);

5170        if (flags & FLG_SHOW_SYMINFO)
5171                syminfo(cache, shnum, ehdr, osabi, file);

5173        if (flags & FLG_SHOW_RELOC)
5174                reloc(cache, shnum, ehdr, file);

5176        if (flags & FLG_SHOW_DYNAMIC)
5177                dynamic(cache, shnum, ehdr, osabi, file);

5179        if (flags & FLG_SHOW_NOTE) {
5180                Word    note_cnt;
5181                size_t  note_shnum;
5182                Cache   *note_cache;

5184                note_cnt = note(cache, shnum, ehdr, file);

5186                /*
5187                 * Solaris core files have section headers, but these
5188                 * headers do not include SHT_NOTE sections that reference
5189                 * the core note sections. This means that note() won't
5190                 * find the core notes. Fake section headers (-P option)
```

```
5191                     * recover these sections, but it is inconvenient to require
5192                     * users to specify -P in this situation. If the following
5193                     * are all true:
5194                     *
5195                     *      - No note sections were found
5196                     *      - This is a core file
5197                     *      - We are not already using fake section headers
5198                     *
5199                     * then we will automatically generate fake section headers
5200                     * and then process them in a second call to note().
5201                     */
5202                    if ((note_cnt == 0) && (ehdr->e_type == ET_CORE) &&
5203                        !(flags & FLG_CTL_FAKESHDR) &&
5204                        (fake_shdr_cache(file, fd, elf, ehdr,
5205                        &note_cache, &note_shnum) != 0)) {
5206                            (void) note(note_cache, note_shnum, ehdr, file);
5207                            fake_shdr_cache_free(note_cache, note_shnum);
5208                    }
5209            }

5211            if ((flags & FLG_SHOW_MOVE) && (osabi == ELFOSABI_SOLARIS))
5212                    move(cache, shnum, file, flags);

5214            if (flags & FLG_CALC_CHECKSUM)
5215                    checksum(elf);

5217            if ((flags & FLG_SHOW_CAP) && (osabi == ELFOSABI_SOLARIS))
5218                    cap(file, cache, shnum, phnum, ehdr, osabi, elf, flags);

5220            if ((flags & FLG_SHOW_UNWIND) &&
5221                ((osabi == ELFOSABI_SOLARIS) || (osabi == ELFOSABI_LINUX)))
5222                    unwind(cache, shnum, phnum, ehdr, osabi, file, elf, flags);


5225            /* Release the memory used to cache section headers */
5226    done:
5227            if (flags & FLG_CTL_FAKESHDR)
5228                    fake_shdr_cache_free(cache, shnum);
5229            else
5230                    free(cache);

5232            return (ret);
5233    }
```

```
**********************************************************
   26207 Mon Mar 23 21:41:47 2015
new/usr/src/cmd/sgs/elfdump/common/elfdump.msg
5688 ELF tools need to be more careful with dwarf data
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1997, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2012 DEY Storage Systems, Inc.  All rights reserved.
#

@ _START_

# Message file for cmd/sgs/elfdump.

@ MSG_ID_ELFDUMP

# Usage Messages

@ MSG_USAGE_BRIEF       "usage: %s [-cCdegGhHiklmnrSsuvy] [-I index] \
                         [-N name] [-O osabi] [-T type] [-p | -w outfile] \
                         file...\n"
@ MSG_USAGE_DETAIL1     "\t[-c]\t\tdump section header information\n"
@ MSG_USAGE_DETAIL2     "\t[-C]\t\tdemangle C++ symbol names\n"
@ MSG_USAGE_DETAIL3     "\t[-d]\t\tdump the contents of the .dynamic section\n"
@ MSG_USAGE_DETAIL4     "\t[-e]\t\tdump the elf header\n"
@ MSG_USAGE_DETAIL5     "\t[-g]\t\tdump the contents of the .group sections\n"
@ MSG_USAGE_DETAIL6     "\t[-G]\t\tdump the contents of the .got section\n"
@ MSG_USAGE_DETAIL7     "\t[-h]\t\tdump the contents of the .hash section\n"
@ MSG_USAGE_DETAIL8     "\t[-H]\t\tdump the contents of the .SUNW_cap section\n"
@ MSG_USAGE_DETAIL9     "\t[-i]\t\tdump the contents of the .interp section\n"
@ MSG_USAGE_DETAIL10    "\t[-I index]\tqualify an option with an index\n"
@ MSG_USAGE_DETAIL11    "\t[-l]\t\tdump with no truncated section names\n"
@ MSG_USAGE_DETAIL12    "\t[-k]\t\tcalculate elf checksum\n"
@ MSG_USAGE_DETAIL13    "\t[-m]\t\tdump the contents of the .SUNW_move \
                         section\n"
@ MSG_USAGE_DETAIL14    "\t[-n]\t\tdump the contents of the .note section\n"
@ MSG_USAGE_DETAIL15    "\t[-N name]\tqualify an option with a 'name'\n"
@ MSG_USAGE_DETAIL16    "\t[-O osabi]\tuse given osabi to interpret object\n"
@ MSG_USAGE_DETAIL17    "\t[-P]\t\tuse program headers to generate \
                         section headers\n"
@ MSG_USAGE_DETAIL18    "\t[-p]\t\tdump the program headers\n"
@ MSG_USAGE_DETAIL19    "\t[-r]\t\tdump the contents of the relocation \
                         sections\n"
@ MSG_USAGE_DETAIL20    "\t[-S]\t\tdump the contents of the sort index \
                         sections\n"
```



```
@ MSG_USAGE_DETAIL21    "\t[-s]\t\tdump the contents of the symbol table \
                         sections\n"
@ MSG_USAGE_DETAIL22    "\t[-T type]\tqualify an option with a section or \
                         program header type\n"
@ MSG_USAGE_DETAIL23    "\t[-u]\t\tdump the contents of a frame unwind \
                         section\n"
@ MSG_USAGE_DETAIL24    "\t[-v]\t\tdump the contents of the version sections\n"
@ MSG_USAGE_DETAIL25    "\t[-w file]\twrite the contents of specified section \
                         to 'file'\n"
@ MSG_USAGE_DETAIL26    "\t[-y]\t\tdump the contents of the .SUNW_syminfo \
                         section\n"

# Errors

@ MSG_ERR_BADFILE       "%s: invalid file type\n"
@ MSG_ERR_BADREL1       "%s: bad relocation entry: %s: relocation requires \
                         symbol\n"
@ MSG_ERR_NOSHDR        "%s: section headers are not present: synthesizing \
                         from program headers (-P option)\n"
@ MSG_ERR_PNEEDSPH      "%s: file without program headers is incompatible \
                         with -P option\n"
@ MSG_ERR_NOSHSTRSEC    "%s: no header string table section (shstrtab). \
                         header names are not available\n"
@ MSG_ERR_SHSTRNULL     "%s: zero size section header string table \
                         (shstrtab). header names are not available\n"
@ MSG_ERR_FAILURE       "%s: %s failed: %s\n"
@ MSG_ERR_BADSHDRALIGN  "%s: bad section header array alignment\n"
@ MSG_ERR_BADPHDRALIGN  "%s: bad program header array alignment\n"
@ MSG_ERR_BADSHNAME     "%s: section[%d]: invalid sh_name: %lld\n"
@ MSG_ERR_BADSHLINK     "%s: %s: invalid sh_link: %d\n"
@ MSG_ERR_BADSHINFO     "%s: %s: invalid sh_info: %d\n"
@ MSG_ERR_BADSHTYPE     "%s: %s: invalid sh_type: %d\n"
@ MSG_ERR_BADALIGN      "%s: %s: bad sh_offset alignment\n"
@ MSG_ERR_BADSYM2       "%s: %s: index[%d]: bad symbol entry: %s: must be \
                         SHN_COMMON or defined in SHT_NOBITS section\n"
@ MSG_ERR_BADSYM3       "%s: %s: index[%d]: bad symbol entry: %s: must be \
                         defined in a SHF_TLS section\n"
@ MSG_ERR_BADSYM4       "%s: %s: index[%d]: bad symbol entry: %s: must be \
                         defined in a non-SHF_TLS section\n"
@ MSG_ERR_BADSYM5       "%s: %s: index[%d]: bad symbol entry: %s: \
                         invalid shndx: %d\n"
@ MSG_ERR_BADSYM6       "%s: %s: index[%d]: bad symbol entry: %s: section[%d] \
                         size: %#llx: symbol (address %#llx, size %#llx) \
                         lies outside of containing section\n"
@ MSG_ERR_BADSYM7       "%s: %s: index[%d]: suspicious global symbol entry: \
                         %s: lies within local symbol range (index < %lld)\n"
@ MSG_ERR_BADSYM8       "%s: %s: index[%d]: suspicious local symbol entry: \
                         %s: lies within global symbol range (index >= %lld)\n"

@ MSG_ERR_RELBADSYMNDX  "%s: bad symbol reference %d: from relocation \
                         entry: %d\n"
@ MSG_ERR_HSBADSYMNDX   "%s: bad symbol reference %d: from hash entry: %d\n"
@ MSG_ERR_BADSYMXINDEX1 "%s: %s: index[%d]: invalid SHN_XINDEX reference: \
                         SHT_SYMTAB_SHNDX section truncated: no entry for this \
                         symbol\n"
@ MSG_ERR_BADSYMXINDEX2 "%s: %s: index[%d]: invalid SHN_XINDEX reference: \
                         bad SHT_SYMTAB_SHNDX entry: invalid shndx: 0x%x\n"
@ MSG_ERR_BADSYMXINDEX3 "%s: %s: index[%d]: invalid SHN_XINDEX reference: \
                         SHT_SYMTAB_SHNDX section not found\n"

@ MSG_ERR_BADSTOFF      "%s: %s: index[%d]: bad %s offset: 0x%x: max 0x%x\n"
@ MSG_ERR_BADSZ         "%s: %s: zero size or zero entry size information\n"
@ MSG_ERR_BADMINFO      "%s: %s: invalid m_info: 0x%llx\n"
@ MSG_ERR_BADHASH       "%s: %s: bad hash entry: symbol %s: exists in bucket \
                         %d, should be bucket %ld\n"
@ MSG_ERR_NODYNSYM      "%s: %s: associated SHT_DYNSYM section not found\n"
```

```
128 @ MSG_ERR_BADNDXSEC     "%s: %s: unexpected section type associated with \
129                          index section: %s\n"
130 @ MSG_ERR_BADSYMNDX     "%s: %s: bad symbol index: %d\n"
131 @ MSG_ERR_BADVER        "%s: %s: index[%d]: version %d is out of range: \
132                          version definitions available: 0-%d\n"
133 @ MSG_ERR_NOTSTRTAB     "%s: section[%d] is not a string table as expected \
134                          by section[%d]\n";

136 @ MSG_ERR_LDYNNOTADJ    "%s: bad dynamic symbol table layout: %s and %s \
137                          sections are not adjacent\n"
138 @ MSG_ERR_SECMEMOVER    "%s: memory overlap between section[%d]: %s: %llx:%llx \
139                          and section[%d]: %s: %llx:%llx\n"
140 @ MSG_ERR_SHDRMEMOVER   "%s: memory overlap between section header table: \
141                          %llx:%llx and section[%d]: %s: %llx:%llx\n"
142 @ MSG_ERR_MULTDYN       "%s: %d dynamic sections seen (1 expected)\n"
143 @ MSG_ERR_DYNNOBCKSEC   "%s: object lacks %s section required by %s dynamic \
144                          entry\n"
145 @ MSG_ERR_DYNBADADDR    "%s: %s (%#llx) does not match \
146                          shdr[%d: %s].sh_addr (%#llx)\n"
147 @ MSG_ERR_DYNBADSIZE    "%s: %s (%#llx) does not match \
148                          shdr[%d: %s].sh_size (%#llx)\n"
149 @ MSG_ERR_DYNBADENTSIZE "%s: %s (%#llx) does not match \
150                          shdr[%d: %s].sh_entsize (%#llx)\n"
151 @ MSG_ERR_DYNSYMVAL     "%s: %s: symbol value does not match \
152                          %s entry: %s: value: %#llx\n"
153 @ MSG_ERR_MALSTR        "%s: %s: malformed string table, initial or final \
154                          byte\n"
155 @ MSG_ERR_MULTEHFRMHDR  "%s: [%d: %s] multiple .eh_frame_hdr sections seen \
156                          (1 expected)\n"
157 @ MSG_ERR_BADEHFRMPTR   "%s: section[%d: %s] FramePtr (%#llx) does not match \
158                          shdr[%d: %s].sh_addr (%#llx)\n"
159 @ MSG_ERR_BADSORT       "%s: %s: index[%d]: invalid sort order\n"
160 @ MSG_ERR_BADSIDYNNDX   "%s: [%d: %s][%d]: dynamic section index out of \
161                          range (0 - %d): %d\n";
162 @ MSG_ERR_BADSIDYNTAG   "%s: [%d: %s][%d]: dynamic element \
163                          [%d: %s][%d] should have type %s: %s\n";
164 @ MSG_ERR_BADCIEFDELEN  "%s: %s: invalid CIE/FDE length: %#llx at %#llx\n"
165 #endif /* ! codereview */


168 @ MSG_WARN_INVINTERP1   "%s: PT_INTERP header has no associated section\n"
169 @ MSG_WARN_INVINTERP2   "%s: interp section: %s: and PT_INTERP program \
170                          header have conflicting size or offsets\n"
171 @ MSG_WARN_INVCAP1      "%s: PT_SUNWCAP header has no associated section\n"
172 @ MSG_WARN_INVCAP2      "%s: capabilities section[%d]: %s: requires PT_CAP \
173                          program header\n"
174 @ MSG_WARN_INVCAP3      "%s: capabilities section[%d]: %s: and PT_CAP program \
175                          header have conflicting size or offsets\n"
176 @ MSG_WARN_INVCAP4      "%s: capabilities section[%d]: %s: requires string \
177                          table: invalid sh_info: %d\n";
178 @ MSG_WARN_INADDR32SF1  "%s: capabilities section %s: software capability \
179                          ADDR32: is ineffective within a 32-bit object\n"
180 @ MSG_WARN_MULTEHFRM    "%s: section[%d: %s]: %s object has multiple \
181                          .eh_frame sections\n"

183 @ MSG_INFO_LINUXOSABI   "%s: %s object has Linux .note.ABI-tag section. \
184                          Assuming %s\n"

186 @ MSG_ERR_DWOVRFLW      "%s: %s: encoded DWARF data exceeds section size\n"
187 @ MSG_ERR_DWBADENC      "%s: %s: bad DWARF encoding: %#x\n"
188 @ MSG_ERR_DWNOCIE       "%s: %s: no CIE prior to FDE\n"

190 #endif /* ! codereview */
191 # exception_range_entry table entries.
192 # TRANSLATION_NOTE - the following entries provide for a series of one or more
193 # standard 32-bit and 64-bit .exception_ranges table entries that align with
```



```
194 # the initial title.

196 @ MSG_EXR_TITLE_32      "     index    offset     ret_addr   \
197                          length     handler    type_blk"
198 @ MSG_EXR_ENTRY_32      "%10.10s  0x%8.8llx 0x%8.8llx 0x%8.8llx 0x%8.8llx \
199                          0x%8.8llx"
200 @ MSG_EXR_TITLE_64      "     index    offset             ret_addr           \
201                          length             handler            type_blk"
202 @ MSG_EXR_ENTRY_64      "%10.10s  0x%16.16llx 0x%16.16llx 0x%16.16llx \
203                          0x%16.16llx 0x%16.16llx"

205 # Elf Output Messages

207 @ MSG_ELF_SHDR          "Section Header[%d]:  sh_name: %s"
208 @ MSG_ELF_PHDR          "Program Header[%d]:"

210 @ MSG_ELF_SCN_CAP       "Capabilities Section:  %s"
211 @ MSG_ELF_SCN_CAPCHAIN  "Capabilities Chain Section:  %s"
212 @ MSG_ELF_SCN_INTERP    "Interpreter Section:  %s"
213 @ MSG_ELF_SCN_VERDEF    "Version Definition Section:  %s"
214 @ MSG_ELF_SCN_VERNEED   "Version Needed Section:  %s"
215 @ MSG_ELF_SCN_SYMTAB    "Symbol Table Section:  %s"
216 @ MSG_ELF_SCN_RELOC     "Relocation Section:  %s"
217 @ MSG_ELF_SCN_UNWIND    "Unwind Section: %s"
218 @ MSG_ELF_SCN_DYNAMIC   "Dynamic Section:  %s"
219 @ MSG_ELF_SCN_NOTE      "Note Section:  %s"
220 @ MSG_ELF_SCN_HASH      "Hash Section:  %s"
221 @ MSG_ELF_SCN_SYMINFO   "Syminfo Section:  %s"
222 @ MSG_ELF_SCN_GOT       "Global Offset Table Section:  %s"
223 @ MSG_ELF_SCN_GRP       "Group Section:  %s"
224 @ MSG_ELF_SCN_MOVE      "Move Section:  %s"
225 @ MSG_ELF_SCN_SYMSORT1  "Symbol Sort Section:  %s (%s)"
226 @ MSG_ELF_SCN_SYMSORT2  "Symbol Sort Section:  %s (%s / %s)"

228 @ MSG_OBJ_CAP_TITLE     " Object Capabilities:"
229 @ MSG_SYM_CAP_TITLE     " Symbol Capabilities:"
230 @ MSG_CAPINFO_ENTRIES   "  Symbols:"
231 @ MSG_CAPCHAIN_TITLE    " Capabilities family: %s"
232 @ MSG_CAPCHAIN_ENTRY    "  chainndx  symndx      name"
233 @ MSG_ERR_INVCAP        "%s: capabilities section: %s: contains symbol \
234                          capabilities groups, but no capabilities information \
235                          section is defined: invalid sh_link: %d\n"
236 @ MSG_ERR_INVCAPINFO1   "%s: capabilities information section: %s: no symbol \
237                          table is defined: invalid sh_link: %d\n"
238 @ MSG_ERR_INVCAPINFO2   "%s: capabilities information section: %s: no \
239                          capabilities chain is defined: invalid sh_info: %d\n"
240 @ MSG_ERR_INVCAPINFO3   "%s: capabilities information section: %s: index %d: \
241                          bad capabilities chain index defined: %d\n"
242 @ MSG_ERR_CHBADSYMNDX   "%s: bad symbol reference %d: from capability chain: \
243                          %s entry: %d\n"

245 @ MSG_ELF_HASH_BKTS1    "%10.10s  buckets contain %8d symbols"
246 @ MSG_ELF_HASH_BKTS2    "%10.10s  buckets         %8d symbols (globals)"
247 @ MSG_ELF_HASH_INFO     "    bucket  symndx      name"
248 @ MSG_HASH_OVERFLW      "%s: warning: section %s: too many symbols to count, \
249                          bucket=%d count=%d"
250 @ MSG_ELF_ERR_SHDR      "\tunable to obtain section header: shstrtab[%lld]\n"
251 @ MSG_ELF_ERR_DATA      "\tunable to obtain section data: shstrtab[%lld]\n"
252 @ MSG_ELF_ERR_SCN       "\tunable to obtain section header: section[%d]\n"
253 @ MSG_ELF_ERR_SCNDATA   "\tunable to obtain section data: section[%d]\n"
254 @ MSG_ARCHIVE_SYMTAB_32 "\nSymbol Table: (archive, 32-bit offsets)"
255 @ MSG_ARCHIVE_SYMTAB_64 "\nSymbol Table: (archive, 64-bit offsets)"
256 @ MSG_ARCHIVE_FIELDS_32 "     index    offset    member name and symbol"
257 @ MSG_ARCHIVE_FIELDS_64 "     index    offset            member name and symbol"

259 @ MSG_GOT_MULTIPLE      "%s: multiple relocations against \
```

```
260                          the same GOT entry ndx: %d addr: 0x%llx\n"
261 @ MSG_GOT_UNEXPECTED    "%s: warning: section %s: section unexpected within \
262                          relocatable object\n"

264 # Miscellaneous clutter

266 @ MSG_STR_NULL          "(null)"
267 @ MSG_STR_DEPRECATED    "(deprecated value)"
268 @ MSG_STR_UNKNOWN       "<unknown>"
269 @ MSG_STR_SECTION       "%s (section)"
270 @ MSG_STR_CHECKSUM      "elf checksum: 0x%lx"

272 @ MSG_FMT_SCNNDX        "section[%d]"
273 @ MSG_FMT_NOTEENTNDX    "   entry [%d]";


276 @ MSG_ERR_MALLOC        "%s: malloc: %s\n"
277 @ MSG_ERR_OPEN          "%s: open: %s\n"
278 @ MSG_ERR_READ          "%s: read: %s\n"
279 @ MSG_ERR_WRITE         "%s: write: %s\n"
280 @ MSG_ERR_BAD_T_SHT     "%s: unrecognized section header type: %s\n"
281 @ MSG_ERR_BAD_T_PT      "%s: unrecognized program header type: %s\n"
282 @ MSG_ERR_BAD_T_OSABI   "%s: unrecognized operating system ABI: %s\n"
283 @ MSG_ERR_AMBIG_MATCH   "%s: ambiguous use of -I, -N, or -T. Remove \
284                          -p option or section selection option(s)\n"

286 #
287 # SHT_MOVE messages
288 #
289 @ MSG_MOVE_TITLE        "    symndx     offset   size repeat stride           \
290                          value  with respect to"
291 @ MSG_MOVE_ENTRY        "%10.10s %#10llx %6d %6d %6d %#16llx  %s"

293 #
294 # SHT_GROUP messages
295 #
296 @ MSG_GRP_TITLE         "     index    flags / section         signature symbol"
297 @ MSG_GRP_SIGNATURE     "       [0]   %-24s %s"
298 @ MSG_GRP_INVALSCN      "<invalid section>"

300 #
301 # SHT_NOTE messages
302 #
303 @ MSG_NOTE_BADDATASZ    "%s: %s: note header exceeds section size. \
304                          offset: 0x%x\n"
305 @ MSG_NOTE_BADNMSZ      "%s: %s: note name value exceeds section size. \
306                          offset: 0x%x namesize: 0x%x\n"
307 @ MSG_NOTE_BADDESZ      "%s: %s: note data size exceeds section size. \
308                          offset: 0x%x datasize: 0x%x\n"
309 @ MSG_NOTE_BADCOREARCH  "%s: elfdump core file note support not available for \
310                          architecture: %s\n"
311 @ MSG_NOTE_BADCOREDATA  "%s: elfdump core file note data truncated or \
312                          otherwise malformed\n"
313 @ MSG_NOTE_BADCORETYPE  "%s: unknown note type %#x\n"
314 #endif /* ! codereview */

316 @ _END_

318 # The following strings represent reserved words, files, pathnames and symbols.
319 # Reference to this strings is via the MSG_ORIG() macro, and thus no message
320 # translation is required.

322 @ MSG_STR_OSQBRKT       "["
323 @ MSG_STR_CSQBRKT       "]"

325 @ MSG_GRP_COMDAT        " COMDAT "
```



```
326 @ MSG_GRP_ENTRY         "%10.10s   %s [%lld]\n"
327 @ MSG_GRP_UNKNOWN       " 0x%x "

329 @ MSG_ELF_GOT           ".got"
330 @ MSG_ELF_INIT          ".init"
331 @ MSG_ELF_FINI          ".fini"
332 @ MSG_ELF_INTERP        ".interp"

334 @ MSG_ELF_GETEHDR       "elf_getehdr"
335 @ MSG_ELF_GETPHDR       "elf_getphdr"
336 @ MSG_ELF_GETSHDR       "elf_getshdr"
337 @ MSG_ELF_GETSCN        "elf_getscn"
338 @ MSG_ELF_GETDATA       "elf_getdata"
339 @ MSG_ELF_GETARHDR      "elf_getarhdr"
340 @ MSG_ELF_GETARSYM      "elf_getarsym"
341 @ MSG_ELF_RAND          "elf_rand"
342 @ MSG_ELF_BEGIN         "elf_begin"
343 @ MSG_ELF_GETPHDRNUM    "elf_getphdrnum"
344 @ MSG_ELF_GETSHDRNUM    "elf_getshdrnum"
345 @ MSG_ELF_GETSHDRSTRNDX "elf_getshdrstrndx"
346 @ MSG_ELF_XLATETOM      "elf_xlatetom"
347 @ MSG_ELF_ARSYM         "ARSYM"

349 @ MSG_SYM_INIT          "_init"
350 @ MSG_SYM_FINI          "_fini"
351 @ MSG_SYM_GOT           "_GLOBAL_OFFSET_TABLE_"

353 @ MSG_STR_OPTIONS       "CcdeGgHhiI:klmN:nO:PprSsT:uvw:y"

355 @ MSG_STR_8SP           "        "
356 @ MSG_STR_EMPTY         ""
357 @ MSG_STR_CORE          "CORE"
358 @ MSG_STR_NOTEABITAG    ".note.ABI-tag"
359 @ MSG_STR_GNU           "GNU"
360 @ MSG_STR_LOC           "loc"
361 @ MSG_STR_INITLOC       "initloc"

363 @ MSG_FMT_INDENT        "       %s"
364 @ MSG_FMT_INDEX         " [%lld]"
365 @ MSG_FMT_INDEX2        "[%d]"
366 @ MSG_FMT_ASRINDEX      "[ asr%d ]"
367 @ MSG_FMT_INDEXRNG      "[%d-%d]"
368 @ MSG_FMT_INTEGER       " %d"
369 @ MSG_FMT_HASH_INFO     "%10.10s  %-10s  %s"
370 @ MSG_FMT_CHAIN_INFO    "%10.10s  %-10s  %s"
371 @ MSG_FMT_ARSYM1_32     "%10.10s  0x%8.8llx  (%s):%s"
372 @ MSG_FMT_ARSYM2_32     "%10.10s  0x%8.8llx"
373 @ MSG_FMT_ARSYM1_64     "%10.10s  0x%16.16llx  (%s):%s"
374 @ MSG_FMT_ARSYM2_64     "%10.10s  0x%16.16llx"
375 @ MSG_FMT_ARNAME        "%s(%s)"
376 @ MSG_FMT_NLSTR         "\n%s:"
377 @ MSG_FMT_NLSTRNL       "\n%s:\n"
378 @ MSG_FMT_SECSYM        "%.*s%s"

380 @ MSG_HEXDUMP_ROW       "%*s%-*s%s"
381 @ MSG_HEXDUMP_TOK       "%2.2x"

383 @ MSG_SUNW_OST_SGS      "SUNW_OST_SGS"

385 # Unwind info

387 @ MSG_SCN_FRM           ".eh_frame"
388 @ MSG_SCN_FRMHDR        ".eh_frame_hdr"
389 @ MSG_SCN_EXRANGE       ".exception_ranges"

391 @ MSG_UNW_FRMHDR        "Frame Header:"
```

```
392 @ MSG_UNW_FRMVERS       "  Version: %d"
393 @ MSG_UNW_FRPTRENC      "  FramePtrEnc: %-20s  FramePtr: %#llx"
394 @ MSG_UNW_FDCNENC       "  FdeCntEnc:   %-20s  FdeCnt: %lld"
395 @ MSG_UNW_TABENC        "  TableEnc:    %-20s"
396 @ MSG_UNW_BINSRTAB1     "  Binary Search Table:"
397 @ MSG_UNW_BINSRTAB2_32  "      InitialLoc    FdeLoc"
398 @ MSG_UNW_BINSRTAB2_64  "      InitialLoc          FdeLoc"
399 @ MSG_UNW_BINSRTABENT_32 "    0x%08llx    0x%08llx"
400 @ MSG_UNW_BINSRTABENT_64 "    0x%016llx  0x%016llx"
401 @ MSG_UNW_ZEROTERM      "ZERO terminator: [0x00000000]"
402 @ MSG_UNW_CIE           "CIE: [%#llx]"
403 @ MSG_UNW_CIELNGTH      " length: 0x%02x cieid: %d"
404 @ MSG_UNW_CIEVERS       " version: %d  augmentation: '%s'"
405 @ MSG_UNW_CIECALGN      " codealign: %#llx  dataalign: %lld  \
406                         retaddr: %d"
407 @ MSG_UNW_CIEAXVAL      " Augmentation Data:"
408 @ MSG_UNW_CIEAXSIZ      "   size: %lld"
164 @ MSG_UNW_CIEAXSIZ      "   size: %d"
409 @ MSG_UNW_CIEAXPERS     "   personality:"
410 @ MSG_UNW_CIEAXPERSENC  "       encoding: 0x%02x %s"
411 @ MSG_UNW_CIEAXPERSRTN  "       routine:  %#08llx"
412 @ MSG_UNW_CIEAXCENC     "   code pointer encoding: 0x%02x %s"
413 @ MSG_UNW_CIEAXLSDA     "   lsda encoding: 0x%02x %s"
414 @ MSG_UNW_CIEAXUNEC     "   Unexpected aug val: %c"
415 @ MSG_UNW_CIECFI        " CallFrameInstructions:"

417 @ MSG_UNW_FDE           " FDE: [%#llx]"
418 @ MSG_UNW_FDELNGTH      "    length:  %#x  cieptr: %#x"
419 @ MSG_UNW_FDEINITLOC    "    initloc: %#llx  addrrange: %#llx  endloc: %#llx"
420 @ MSG_UNW_FDEAXVAL      "   Augmentation Data:"
421 @ MSG_UNW_FDEAXSIZE     "      size: %#llx"
422 @ MSG_UNW_FDEAXLSDA     "      lsda: %#llx"
423 @ MSG_UNW_FDECFI        "   CallFrameInstructions:"

425 # Unwind section Call Frame Instructions. These all start with a leading
426 # "%*s%s", used to insert leading white space and the opcode name.

428 @ MSG_CFA_ADV_LOC       "%*s%s: %s + %llu => %#llx"
429 @ MSG_CFA_CFAOFF        "%*s%s: %s, cfa%+lld"
430 @ MSG_CFA_CFASET        "%*s%s: cfa=%#llx"
431 @ MSG_CFA_LLD           "%*s%s: %lld"
432 @ MSG_CFA_LLU           "%*s%s: %llu"
433 @ MSG_CFA_REG           "%*s%s: %s"
434 @ MSG_CFA_REG_OFFLLD    "%*s%s: %s, offset=%lld"
435 @ MSG_CFA_REG_OFFLLU    "%*s%s: %s, offset=%llu"
436 @ MSG_CFA_REG_REG       "%*s%s: %s, %s"
437 @ MSG_CFA_SIMPLE        "%*s%s"
438 @ MSG_CFA_SIMPLEREP     "%*s%s [%d]"
439 @ MSG_CFA_EBLK          "%*s%s: expr(%llu bytes)"
440 @ MSG_CFA_REG_EBLK      "%*s%s: %s, expr(%llu bytes)"

442 # Architecture specific register name formats

444 @ MSG_REG_FMT_BASIC     "r%d"
445 @ MSG_REG_FMT_NAME      "r%d (%s)"


448 # Note messages

450 @ MSG_NOTE_TYPE         "    type:   %#x"
451 @ MSG_NOTE_TYPE_STR     "    type:   %s"
452 @ MSG_NOTE_NAMESZ       "    namesz: %#x"
453 @ MSG_NOTE_NAME         "    name:"
454 @ MSG_NOTE_DESCSZ       "    descsz: %#x"

456 @ MSG_NOTE_DESC                        "    desc:"
```



```
457 @ MSG_CNOTE_DESC_ASRSET_T              "desc: (asrset_t)"
458 @ MSG_CNOTE_DESC_AUXV_T                "desc: (auxv_t)"
459 @ MSG_CNOTE_DESC_CORE_CONTENT_T        "desc: (core_content_t)"
460 @ MSG_CNOTE_DESC_LWPSINFO_T            "desc: (lwpsinfo_t)"
461 @ MSG_CNOTE_DESC_LWPSTATUS_T           "desc: (lwpstatus_t)"
462 @ MSG_CNOTE_DESC_PRCRED_T              "desc: (prcred_t)"
463 @ MSG_CNOTE_DESC_PRIV_IMPL_INFO_T      "desc: (priv_impl_info_t)"
464 @ MSG_CNOTE_DESC_PRPRIV_T              "desc: (prpriv_t)"
465 @ MSG_CNOTE_DESC_PRPSINFO_T            "desc: (prpsinfo_t)"
466 @ MSG_CNOTE_DESC_PRSTATUS_T            "desc: (prstatus_t)"
467 @ MSG_CNOTE_DESC_PSINFO_T              "desc: (psinfo_t)"
468 @ MSG_CNOTE_DESC_PSTATUS_T             "desc: (pstatus_t)"
469 @ MSG_CNOTE_DESC_STRUCT_UTSNAME        "desc: (struct utsname)"
470 @ MSG_CNOTE_DESC_PRFDINFO_T            "desc: (prfdinfo_t)"


473 @ MSG_CNOTE_FMT_LINE            "%*s%-*s%s"
474 @ MSG_CNOTE_FMT_LINE_2UP        "%*s%-*s%-*s%-*s%s"
475 @ MSG_CNOTE_FMT_D               "%d"
476 @ MSG_CNOTE_FMT_LLD             "%lld"
477 @ MSG_CNOTE_FMT_U               "%u"
478 @ MSG_CNOTE_FMT_LLU             "%llu"
479 @ MSG_CNOTE_FMT_X               "%#x"
480 @ MSG_CNOTE_FMT_LLX             "%#llx"
481 @ MSG_CNOTE_FMT_Z2X             "0x%2.2x"
482 @ MSG_CNOTE_FMT_Z4X             "0x%4.4x"
483 @ MSG_CNOTE_FMT_Z8X             "0x%8.8x"
484 @ MSG_CNOTE_FMT_Z16LLX          "0x%16.16llx"
485 @ MSG_CNOTE_FMT_TITLE           "%*s%s"
486 @ MSG_CNOTE_FMT_AUXVLINE        "%*s%10.10s  %-*s  %s"
487 @ MSG_CNOTE_FMT_PRTPCT          "%u.%u%%"

489 @ MSG_CNOTE_T_PRIV_FLAGS                "priv_flags:"
490 @ MSG_CNOTE_T_PRIV_GLOBALINFOSIZE       "priv_globalinfosize:"
491 @ MSG_CNOTE_T_PRIV_HEADERSIZE           "priv_headersize:"
492 @ MSG_CNOTE_T_PRIV_INFOSIZE             "priv_infosize:"
493 @ MSG_CNOTE_T_PRIV_MAX                  "priv_max:"
494 @ MSG_CNOTE_T_PRIV_NSETS                "priv_nsets:"
495 @ MSG_CNOTE_T_PRIV_SETSIZE              "priv_setsize:"
496 @ MSG_CNOTE_T_PR_ACTION                 "pr_action:"
497 @ MSG_CNOTE_T_PR_ADDR                   "pr_addr:"
498 @ MSG_CNOTE_T_PR_AGENTID                "pr_agentid:"
499 @ MSG_CNOTE_T_PR_ALTSTACK               "pr_altstack:"
500 @ MSG_CNOTE_T_PR_ARGC                   "pr_argc:"
501 @ MSG_CNOTE_T_PR_ARGV                   "pr_argv:"
502 @ MSG_CNOTE_T_PR_ASLWPID                "pr_aslwpid:"
503 @ MSG_CNOTE_T_PR_BIND                   "pr_bind:"
504 @ MSG_CNOTE_T_PR_BINDPRO                "pr_bindpro:"
505 @ MSG_CNOTE_T_PR_BINDPSET               "pr_bindpset:"
506 @ MSG_CNOTE_T_PR_BRKBASE                "pr_brkbase:"
507 @ MSG_CNOTE_T_PR_BRKSIZE                "pr_brksize:"
508 @ MSG_CNOTE_T_PR_BYRSSIZE               "pr_byrssize:"
509 @ MSG_CNOTE_T_PR_BYSIZE                 "pr_bysize:"
510 @ MSG_CNOTE_T_PR_CLNAME                 "pr_clname:"
511 @ MSG_CNOTE_T_PR_CONTRACT               "pr_contract:"
512 @ MSG_CNOTE_T_PR_CPU                    "pr_cpu:"
513 @ MSG_CNOTE_T_PR_CSTIME                 "pr_cstime:"
514 @ MSG_CNOTE_T_PR_CTIME                  "pr_ctime:"
515 @ MSG_CNOTE_T_PR_CURSIG                 "pr_cursig:"
516 @ MSG_CNOTE_T_PR_CUTIME                 "pr_cutime:"
517 @ MSG_CNOTE_T_PR_DMODEL                 "pr_dmodel:"
518 @ MSG_CNOTE_T_PR_EGID                   "pr_egid:"
519 @ MSG_CNOTE_T_PR_ENVP                   "pr_envp:"
520 @ MSG_CNOTE_T_PR_ERRNO                  "pr_errno:"
521 @ MSG_CNOTE_T_PR_ERRPRIV                "pr_errpriv:"
522 @ MSG_CNOTE_T_PR_EUID                   "pr_euid:"
```

```
523 @ MSG_CNOTE_T_PR_FLAG               "pr_flag:"
524 @ MSG_CNOTE_T_PR_FLAGS              "pr_flags:"
525 @ MSG_CNOTE_T_PR_FLTTRACE           "pr_flttrace:"
526 @ MSG_CNOTE_T_PR_FNAME              "pr_fname:"
527 @ MSG_CNOTE_T_PR_FPREG              "pr_fpreg:"
528 @ MSG_CNOTE_T_PR_GID                "pr_gid:"
529 @ MSG_CNOTE_T_PR_GROUPS             "pr_groups:"
530 @ MSG_CNOTE_T_PR_INFO               "pr_info:"
531 @ MSG_CNOTE_T_PR_INFOSIZE           "pr_infosize:"
532 @ MSG_CNOTE_T_PR_INSTR              "pr_instr:"
533 @ MSG_CNOTE_T_PR_LGRP               "pr_lgrp:"
534 @ MSG_CNOTE_T_PR_LTTYDEV            "pr_lttydev:"
535 @ MSG_CNOTE_T_PR_LWP                "pr_lwp:"
536 @ MSG_CNOTE_T_PR_LWPHOLD            "pr_lwphold:"
537 @ MSG_CNOTE_T_PR_LWPID              "pr_lwpid:"
538 @ MSG_CNOTE_T_PR_LWPPEND            "pr_lwppend:"
539 @ MSG_CNOTE_T_PR_NAME               "pr_name:"
540 @ MSG_CNOTE_T_PR_NGROUPS            "pr_ngroups:"
541 @ MSG_CNOTE_T_PR_NICE               "pr_nice:"
542 @ MSG_CNOTE_T_PR_NLWP               "pr_nlwp:"
543 @ MSG_CNOTE_T_PR_NSETS              "pr_nsets:"
544 @ MSG_CNOTE_T_PR_NSYSARG            "pr_nsysarg:"
545 @ MSG_CNOTE_T_PR_NZOMB              "pr_nzomb:"
546 @ MSG_CNOTE_T_PR_OLDCONTEXT         "pr_oldcontext:"
547 @ MSG_CNOTE_T_PR_OLDPRI             "pr_oldpri:"
548 @ MSG_CNOTE_T_PR_ONPRO              "pr_onpro:"
549 @ MSG_CNOTE_T_PR_OTTYDEV            "pr_ottydev:"
550 @ MSG_CNOTE_T_PR_PCTCPU             "pr_pctcpu:"
551 @ MSG_CNOTE_T_PR_PCTMEM             "pr_pctmem:"
552 @ MSG_CNOTE_T_PR_PGID               "pr_pgid:"
553 @ MSG_CNOTE_T_PR_PGRP               "pr_pgrp:"
554 @ MSG_CNOTE_T_PR_PID                "pr_pid:"
555 @ MSG_CNOTE_T_PR_POOLID             "pr_poolid:"
556 @ MSG_CNOTE_T_PR_PPID               "pr_ppid:"
557 @ MSG_CNOTE_T_PR_PRI                "pr_pri:"
558 @ MSG_CNOTE_T_PR_PROCESSOR          "pr_processor:"
559 @ MSG_CNOTE_T_PR_PROJID             "pr_projid:"
560 @ MSG_CNOTE_T_PR_PSARGS             "pr_psargs:"
561 @ MSG_CNOTE_T_PR_REG                "pr_reg:"
562 @ MSG_CNOTE_T_PR_RGID               "pr_rgid:"
563 @ MSG_CNOTE_T_PR_RSSIZE             "pr_rssize:"
564 @ MSG_CNOTE_T_PR_RUID               "pr_ruid:"
565 @ MSG_CNOTE_T_PR_RVAL1              "pr_rval1:"
566 @ MSG_CNOTE_T_PR_RVAL2              "pr_rval2:"
567 @ MSG_CNOTE_T_PR_SETS               "pr_sets:"
568 @ MSG_CNOTE_T_PR_SETSIZE            "pr_setsize:"
569 @ MSG_CNOTE_T_PR_SGID               "pr_sgid:"
570 @ MSG_CNOTE_T_PR_SID                "pr_sid:"
571 @ MSG_CNOTE_T_PR_SIGHOLD            "pr_sighold:"
572 @ MSG_CNOTE_T_PR_SIGPEND            "pr_sigpend:"
573 @ MSG_CNOTE_T_PR_SIGTRACE           "pr_sigtrace:"
574 @ MSG_CNOTE_T_PR_SIZE               "pr_size:"
575 @ MSG_CNOTE_T_PR_SNAME              "pr_sname:"
576 @ MSG_CNOTE_T_PR_START              "pr_start:"
577 @ MSG_CNOTE_T_PR_STATE              "pr_state:"
578 @ MSG_CNOTE_T_PR_STIME              "pr_stime:"
579 @ MSG_CNOTE_T_PR_STKBASE            "pr_stkbase:"
580 @ MSG_CNOTE_T_PR_STKSIZE            "pr_stksize:"
581 @ MSG_CNOTE_T_PR_STYPE              "pr_stype:"
582 @ MSG_CNOTE_T_PR_SUID               "pr_suid:"
583 @ MSG_CNOTE_T_PR_SYSARG             "pr_sysarg:"
584 @ MSG_CNOTE_T_PR_SYSCALL            "pr_syscall:"
585 @ MSG_CNOTE_T_PR_SYSENTRY           "pr_sysentry:"
586 @ MSG_CNOTE_T_PR_SYSEXIT            "pr_sysexit:"
587 @ MSG_CNOTE_T_PR_TASKID             "pr_taskid:"
588 @ MSG_CNOTE_T_PR_TIME               "pr_time:"
```



```
589 @ MSG_CNOTE_T_PR_TSTAMP              "pr_tstamp:"
590 @ MSG_CNOTE_T_PR_TTYDEV              "pr_ttydev:"
591 @ MSG_CNOTE_T_PR_UID                 "pr_uid:"
592 @ MSG_CNOTE_T_PR_USTACK              "pr_ustack:"
593 @ MSG_CNOTE_T_PR_UTIME               "pr_utime:"
594 @ MSG_CNOTE_T_PR_WCHAN               "pr_wchan:"
595 @ MSG_CNOTE_T_PR_WHAT                "pr_what:"
596 @ MSG_CNOTE_T_PR_WHO                 "pr_who:"
597 @ MSG_CNOTE_T_PR_WHY                 "pr_why:"
598 @ MSG_CNOTE_T_PR_WSTAT               "pr_wstat:"
599 @ MSG_CNOTE_T_PR_ZOMB                "pr_zomb:"
600 @ MSG_CNOTE_T_PR_ZONEID              "pr_zoneid:"
601 @ MSG_CNOTE_T_SA_FLAGS               "sa_flags:"
602 @ MSG_CNOTE_T_SA_HANDLER             "sa_handler:"
603 @ MSG_CNOTE_T_SA_MASK                "sa_mask:"
604 @ MSG_CNOTE_T_SA_SIGACTION           "sa_sigaction:"
605 @ MSG_CNOTE_T_SIVAL_INT              "sival_int:"
606 @ MSG_CNOTE_T_SIVAL_PTR              "sival_ptr:"
607 @ MSG_CNOTE_T_SI_ADDR                "si_addr:"
608 @ MSG_CNOTE_T_SI_BAND                "si_band:"
609 @ MSG_CNOTE_T_SI_CODE                "si_code:"
610 @ MSG_CNOTE_T_SI_CTID                "si_ctid:"
611 @ MSG_CNOTE_T_SI_ENTITY              "si_entity"
612 @ MSG_CNOTE_T_SI_ERRNO               "si_errno:"
613 @ MSG_CNOTE_T_SI_PID                 "si_pid:"
614 @ MSG_CNOTE_T_SI_SIGNO               "si_signo:"
615 @ MSG_CNOTE_T_SI_STATUS              "si_status:"
616 @ MSG_CNOTE_T_SI_UID                 "si_uid:"
617 @ MSG_CNOTE_T_SI_VALUE               "si_value:"
618 @ MSG_CNOTE_T_SI_ZONEID              "si_zoneid:"
619 @ MSG_CNOTE_T_SS_FLAGS               "ss_flags:"
620 @ MSG_CNOTE_T_SS_SIZE                "ss_size:"
621 @ MSG_CNOTE_T_SS_SP                  "ss_sp:"
622 @ MSG_CNOTE_T_TV_NSEC                "tv_nsec:"
623 @ MSG_CNOTE_T_TV_SEC                 "tv_sec:"
624 @ MSG_CNOTE_T_UTS_MACHINE            "machine:"
625 @ MSG_CNOTE_T_UTS_NODENAME           "nodename:"
626 @ MSG_CNOTE_T_UTS_RELEASE            "release:"
627 @ MSG_CNOTE_T_UTS_SYSNAME            "sysname:"
628 @ MSG_CNOTE_T_UTS_VERSION            "version:"
629 @ MSG_CNOTE_T_PR_FD                  "pr_fd:"
630 @ MSG_CNOTE_T_PR_MODE                "pr_mode:"
631 @ MSG_CNOTE_T_PR_PATH                "pr_path:"
632 @ MSG_CNOTE_T_PR_MAJOR               "pr_major:"
633 @ MSG_CNOTE_T_PR_MINOR               "pr_minor:"
634 @ MSG_CNOTE_T_PR_RMAJOR              "pr_rmajor:"
635 @ MSG_CNOTE_T_PR_RMINOR              "pr_rminor:"
636 @ MSG_CNOTE_T_PR_OFFSET              "pr_offset:"
637 @ MSG_CNOTE_T_PR_INO                 "pr_ino:"
638 @ MSG_CNOTE_T_PR_FILEFLAGS           "pr_fileflags:"
639 @ MSG_CNOTE_T_PR_FDFLAGS             "pr_fdflags:"


642 # Names of fake sections generated from program header data
643 @ MSG_PHDRNAM_CAP               ".SUNW_cap(phdr)"
644 @ MSG_PHDRNAM_CAPINFO           ".SUNW_capinfo(phdr)"
645 @ MSG_PHDRNAM_CAPCHAIN          ".SUNW_capchain(phdr)"
646 @ MSG_PHDRNAM_DYN               ".dynamic(phdr)"
647 @ MSG_PHDRNAM_DYNSTR            ".dynstr(phdr)"
648 @ MSG_PHDRNAM_DYNSYM            ".dynsym(phdr)"
649 @ MSG_PHDRNAM_FINIARR           ".fini_array(phdr)"
650 @ MSG_PHDRNAM_HASH              ".hash(phdr)"
651 @ MSG_PHDRNAM_INITARR           ".init_array(phdr)"
652 @ MSG_PHDRNAM_INTERP            ".interp(phdr)"
653 @ MSG_PHDRNAM_LDYNSYM           ".SUNW_ldynsym(phdr)"
654 @ MSG_PHDRNAM_MOVE              ".move(phdr)"
```

```
 655 @ MSG_PHDRNAM_NOTE             ".note(phdr)"
 656 @ MSG_PHDRNAM_PREINITARR       ".preinit_array(phdr)"
 657 @ MSG_PHDRNAM_REL              ".rel(phdr)"
 658 @ MSG_PHDRNAM_RELA             ".rela(phdr)"
 659 @ MSG_PHDRNAM_SYMINFO          ".syminfo(phdr)"
 660 @ MSG_PHDRNAM_SYMSORT          ".SUNW_symsort(phdr)"
 661 @ MSG_PHDRNAM_TLSSORT          ".SUNW_tlssort(phdr)"
 662 @ MSG_PHDRNAM_UNWIND           ".eh_frame_hdr(phdr)"
 663 @ MSG_PHDRNAM_VER              ".SUNW_version(phdr)"
```

***********************************************************
***9455 Mon Mar 23 21:41:48 2015***
***new/usr/src/cmd/sgs/include/dwarf.h***
***5688 ELF tools need to be more careful with dwarf data***
***********************************************************
_____***unchanged_portion_omitted_***

```
250 typedef enum {
251         DW_SUCCESS = 0,
252         DW_BAD_ENCODING,
253         DW_OVERFLOW,
254 } dwarf_error_t;

256 #endif /* ! codereview */
257 /*
258  * Little Endian Base 128 (leb128) encoding/decoding routines
259  */
260 extern  dwarf_error_t   uleb_extract(unsigned char *, uint64_t *, size_t,
261                             uint64_t *);
262 extern  dwarf_error_t   sleb_extract(unsigned char *, uint64_t *, size_t,
263                             int64_t *);
264 extern  dwarf_error_t   dwarf_ehe_extract(unsigned char *, size_t, uint64_t *,
265                             uint64_t *, uint_t, unsigned char *, boolean_t,
266                             uint64_t, uint64_t, uint64_t);
250 extern  uint64_t        uleb_extract(unsigned char *, uint64_t *);
251 extern  int64_t         sleb_extract(unsigned char *, uint64_t *);
252 extern  uint64_t        dwarf_ehe_extract(unsigned char *, uint64_t *,
253                             uint_t, unsigned char *, boolean_t, uint64_t,
254                             uint64_t, uint64_t);

268 #ifdef  __cplusplus
269 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   60422 Mon Mar 23 21:41:48 2015
new/usr/src/cmd/sgs/libld/common/libld.msg
5688 ELF tools need to be more careful with dwarf data
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright (c) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
  24 #

  26 #
  27 # Copyright (c) 2012, Joyent, Inc.  All rights reserved.
  28 #

  30 @ _START_

  32 # Message file for cmd/sgs/libld.

  34 @ MSG_ID_LIBLD

  36 #
  37 # TRANSLATION_NOTE -- Beginning of USAGE message
  38 #       The following messages are the usage messages for the ld command.
  39 #       Tab characters (\t) are used to align the messages.
  40 #
  41 #       Each usage message starts with \t, and if the message has more than one
  42 #               line, the following messages are aligned by 3 tab characters.
  43 #               When you see \n\t\t\t, the first \n is used to change the line,
  44 #               and following 3 tab characters are used to align the line.
  45 #
  46 #       Each usage message option is surrounded by [ and ].  Then the
  47 #       description of the option follows. The descriptions should be aligned,
  48 #       so tab characters are padded as needed after the closing bracket ].
  49 #
  50 #       How to align the messages are up to the translators and the
  51 #       localization engineers.
  52 #
  53 #       In C locale, the first 3 messages would look like the following:
  54 #
  55 #       usage: ld [-6:abc:.....] file(s)
  56 #               [-a]            create an absolute file
  57 #               [-b]            do not do special PIC relocations in a.out
  58 #               [-c file]       record configuration 'file'
  59 #
  60 @ MSG_ARG_USAGE         "usage: ld [-%s] file(s)\n"
  61 @ MSG_ARG_DETAIL_3      "\t[-32]\t\tenforce a 32-bit link-edit\n"
```



```
  62 @ MSG_ARG_DETAIL_6      "\t[-64]\t\tenforce a 64-bit link-edit\n"
  63 @ MSG_ARG_DETAIL_A      "\t[-a]\t\tcreate an absolute file\n"
  64 @ MSG_ARG_DETAIL_B      "\t[-b]\t\tdo not do special PIC relocations in a.out\n"
  65 @ MSG_ARG_DETAIL_CBDR   "\t[-B direct | nodirect]\n\
  66                          \t\t\testablish direct bindings, or inhibit direct \
  67                          binding\n\
  68                          \t\t\tto, the object being created\n"
  69 @ MSG_ARG_DETAIL_CBDY   "\t[-B dynamic | static]\n\
  70                          \t\t\tsearch for shared libraries|archives\n"
  71 @ MSG_ARG_DETAIL_CBE    "\t[-B eliminate]\teliminate unqualified global \
  72                          symbols from the\n\t\t\tsymbol table\n"
  73 @ MSG_ARG_DETAIL_CBG    "\t[-B group]\trelocate object from within group\n"
  74 @ MSG_ARG_DETAIL_CBL    "\t[-B local]\treduce unqualified global symbols to \
  75                          local\n"
  76 @ MSG_ARG_DETAIL_CBR    "\t[-B reduce]\tprocess symbol reductions\n"
  77 @ MSG_ARG_DETAIL_CBS    "\t[-B symbolic]\tbind external references to \
  78                          definitions when creating\n\
  79                          \t\t\tshared objects\n"
  80 @ MSG_ARG_DETAIL_C      "\t[-c name]\trecord configuration file 'name'\n"
  81 @ MSG_ARG_DETAIL_CC     "\t[-C]\t\tdemangle C++ symbol name diagnostics\n"
  82 @ MSG_ARG_DETAIL_D      "\t[-d y | n]\toperate in dynamic|static mode\n"
  83 @ MSG_ARG_DETAIL_CD     "\t[-D token,...]\tprint diagnostic messages\n"
  84 @ MSG_ARG_DETAIL_E      "\t[-e epsym], [--entry epsym]\n\
  85                          \t\t\tuse 'epsym' as entry point address\n"
  86 @ MSG_ARG_DETAIL_F      "\t[-f name], [--auxiliary name]\n\
  87                          \t\t\tspecify library for which this file is an \
  88                          auxiliary\n\t\t\tfilter\n"
  89 @ MSG_ARG_DETAIL_CF     "\t[-F name], [--filter name]\n\
  90                          \t\t\tspecify library for which this file is a filter\n
  91 @ MSG_ARG_DETAIL_CG     "\t[-G], [-shared]\n\
  92                          \t\t\tcreate a shared object\n"
  93 @ MSG_ARG_DETAIL_H      "\t[-h name], [--soname name]\n\
  94                          \t\t\tuse 'name' as internal shared object identifier\n
  95 @ MSG_ARG_DETAIL_I      "\t[-i]\t\tignore LD_LIBRARY_PATH setting\n"
  96 @ MSG_ARG_DETAIL_CI     "\t[-I name]\tuse 'name' as path of interpreter\n"
  97 @ MSG_ARG_DETAIL_L      "\t[-l x], [--library x]\n\
  98                          \t\t\tsearch for libx.so or libx.a\n"
  99 @ MSG_ARG_DETAIL_CL     "\t[-L path], [--library-path path]\n\
 100                          \t\t\tsearch for libraries in directory 'path'\n"
 101 @ MSG_ARG_DETAIL_M      "\t[-m]\t\tprint memory map\n"
 102 @ MSG_ARG_DETAIL_CM     "\t[-M mapfile]\tuse processing directives contained \
 103                          in 'mapfile'\n"
 104 @ MSG_ARG_DETAIL_CN     "\t[-N string]\tcreate a dynamic dependency for \
 105                          'string'\n"
 106 @ MSG_ARG_DETAIL_O      "\t[-o outfile], [--output outfile]\n\
 107                          \t\t\tname the output file 'outfile'\n"
 108 @ MSG_ARG_DETAIL_P      "\t[-p auditlib]\tidentify audit library to accompany \
 109                          this object\n"
 110 @ MSG_ARG_DETAIL_CP     "\t[-P auditlib]\tidentify audit library for \
 111                          processing the dependencies\n\
 112                          \t\t\tof this object\n"
 113 @ MSG_ARG_DETAIL_CQ     "\t[-Q y | n]\tdo|do not place version information in \
 114                          output file\n"
 115 @ MSG_ARG_DETAIL_R      "\t[-r], [--relocatable]\n\
 116                          \t\t\tcreate a relocatable object\n"
 117 @ MSG_ARG_DETAIL_CR     "\t[-R path], [-rpath path]\n\
 118                          \t\t\tspecify a library search path to be used at run \
 119                          time\n"
 120 @ MSG_ARG_DETAIL_S      "\t[-s], [--strip-all]\n\
 121                          \t\t\tstrip any symbol and debugging information\n"
 122 @ MSG_ARG_DETAIL_CS     "\t[-S supportlib]\n\
 123                          \t\t\tspecify a link-edit support library\n"
 124 @ MSG_ARG_DETAIL_T      "\t[-t]\t\tdo not warn of multiply-defined symbols \
 125                          that have\n\t\t\tdifferent sizes or alignments\n"
 126 @ MSG_ARG_DETAIL_U      "\t[-u symname], [--undefined symname]\n\
 127                          \t\t\tcreate an undefined symbol 'symname'\n"
```

```
128 @ MSG_ARG_DETAIL_CV     "\t[-V], [--version]\n\
129                          \t\t\tprint version information\n"
130 @ MSG_ARG_DETAIL_CY     "\t[-Y P,dirlist]\tuse 'dirlist' as a default path \
131                          when searching for\n\
132                          \t\t\tlibraries\n"
133 @ MSG_ARG_DETAIL_ZA     "\t[-z absexec]\twhen building an executable absolute \
134                          symbols\n \
135                          \t\t\treferenced in dynamic objects are promoted to\n \
136                          \t\t\tthe executable\n"
137 @ MSG_ARG_DETAIL_ZAE    "\t[-z allextract | defaultextract | weakextract],\n\
138                          \t[--whole-archive | --no-whole-archive]\n\
139                          \t\t\textract all member files, only members that \
140                          resolve\n\
141                          \t\t\tundefined or tentative symbols, or \
142                          allow extraction of\n\
143                          \t\t\tarchive members to resolve weak references from \
144                          \n\t\t\t\tarchive files\n"
145 @ MSG_ARG_DETAIL_ZAL    "\t[-z altexec64]\texecute the 64-bit link-editor\n"
146 @ MSG_ARG_DETAIL_ZADLIB "\t[-z assert-deflib]\n\
147                         \t\t\tenables warnings for linking with libraries in \
148                         the \n\t\t\tdefault search path\n\
149                         \t[-z assert-deflib=libname]\n\
150                         \t\t\tenables warnings for linking with libraries in \
151                         the \n\t\t\tdefault search path, but 'libname' is exempt
152 @ MSG_ARG_DETAIL_ZC     "\t[-z combreloc | nocombreloc]\n\
153                          \t\t\tcombine|do not combine multiple relocation \
154                          sections\n"
155 @ MSG_ARG_DETAIL_ZNC    "\t[-z nocompstrtab]\n\t\t\tdisable compression of \
156                          string tables\n"
157 @ MSG_ARG_DETAIL_ZDEF   "\t[-z deferred | nodeferred]\n\
158                          \t\t\tenable|disable deferred identification of \
159                          shared object\n\t\t\tdependencies\n"
160 @ MSG_ARG_DETAIL_ZDFS   "\t[-z defs], [--no-undefined]\n\
161                          \t\t\tdisallow undefined symbol references\n"
162 @ MSG_ARG_DETAIL_ZDRS   "\t[-z direct | nodirect]\n\
163                          \t\t\tenable|disable direct binding to shared object\n\
164                          \t\t\tdependencies\n"
165 @ MSG_ARG_DETAIL_ZE     "\t[-z endfiltee]\tmarks a filtee such that it will \
166                          terminate a filters\n\t\t\tsearch\n"
167 @ MSG_ARG_DETAIL_ZFATW  "\t[-z fatal-warnings | nofatal-warnings],\n\
168                          \t[--fatal-warnings | --no-fatal-warnings]\n\
169                          \t\tenable|disable treatment of warnings as fatal\n"
170 @ MSG_ARG_DETAIL_ZFA    "\t[-z finiarray=function]\n\
171                          \t\t\tname of function to be appended to the \
172                          .fini_array\n"
173 @ MSG_ARG_DETAIL_ZGP    "\t[-z groupperm | nogroupperm]\n\
174                          \t\t\tenable|disable setting of group permissions\n\
175                          \t\t\ton dynamic dependencies\n"
176 @ MSG_ARG_DETAIL_ZGUIDE "\t[-z guidance | -z guidance=item1,item2,...]\n\
177                          \t\t\tenable guidance warnings. items: \
178                          noall, nodefs,\n\
179                          \t\t\tnodirect, nolazyload, nomapfile, notext, \
180                          nounused\n"
181 @ MSG_ARG_DETAIL_ZH     "\t[-z help], [--help]\n\
182                          \t\t\tprint this usage message\n"
183 @ MSG_ARG_DETAIL_ZIG    "\t[-z ignore | record]\n\
184                          \t\t\tignore|record unused dynamic dependencies\n"
185 @ MSG_ARG_DETAIL_ZINA   "\t[-z initarray=function]\n\
186                          \t\t\tname of function to be appended to the \
187                          .init_array\n"
188 @ MSG_ARG_DETAIL_ZINI   "\t[-z initfirst]\tmark object to indicate that its \
189                          .init section should\n\
190                          \t\t\tbe executed before the .init section of any \
191                          other\n\t\t\tobjects\n"
192 @ MSG_ARG_DETAIL_ZINT   "\t[-z interpose]\
193                          \tdynamic object is to be an 'interposer' on direct\n\
```



```
194                          \t\t\tbindings\n"
195 @ MSG_ARG_DETAIL_ZLAZY  "\t[-z lazyload | nolazyload]\n\
196                          \t\t\tenable|disable delayed loading of shared \
197                          object\n\t\t\tdependencies\n"
198 @ MSG_ARG_DETAIL_ZLD32  "\t[-z ld32=arg1,arg2,...]\n\
199                          \t\t\tdefine arguments applicable to the \
200                          32-bit class of ld(1)\n"
201 @ MSG_ARG_DETAIL_ZLD64  "\t[-z ld64=arg1,arg2,...]\n\
202                          \t\t\tdefine arguments applicable to the \
203                          64-bit class of ld(1)\n"
204 @ MSG_ARG_DETAIL_ZLO    "\t[-z loadfltr]\tmark filter as requiring immediate \
205                          loading of its\n\
206                          \t\t\tfiltees at runtime\n"
207 @ MSG_ARG_DETAIL_ZM     "\t[-z muldefs], [--allow-multiple-definition]\n\
208                          \t\t\tallow multiply-defined symbols\n"
209 @ MSG_ARG_DETAIL_ZNDFS  "\t[-z nodefs]\tallow undefined symbol references\n"
210 @ MSG_ARG_DETAIL_ZNDEF  "\t[-z nodefaultlib]\n\
211                          \t\t\tmark object to ignore any default library \
212                          search path\n"
213 @ MSG_ARG_DETAIL_ZNDEL  "\t[-z nodelete]\tmark object as non-deletable\n"
214 @ MSG_ARG_DETAIL_ZNDLO  "\t[-z nodlopen]\tmark object as non-dlopen()'able\n"
215 @ MSG_ARG_DETAIL_ZNDU   "\t[-z nodump]\tmark object as non-dldump()'able\n"
216 @ MSG_ARG_DETAIL_ZNLD   "\t[-z noldynsym]\tdo not add a .SUNW_ldynsym section\n"
217 @ MSG_ARG_DETAIL_ZNPA   "\t[-z nopartial]\texpand any partially initialized \
218                          symbols\n"
219 @ MSG_ARG_DETAIL_ZNV    "\t[-z noversion]\tdo not record any version sections\n"
220 @ MSG_ARG_DETAIL_ZNOW   "\t[-z now]\tmark object as requiring non-lazy \
221                          binding\n"
222 @ MSG_ARG_DETAIL_ZO     "\t[-z origin]\tmark object as requiring $ORIGIN \
223                          processing\n"
224 @ MSG_ARG_DETAIL_ZPIA   "\t[-z preinitarray=function]\n\
225                          \t\tname of function to be appended to the \
226                          .preinit_array\n"
227 @ MSG_ARG_DETAIL_ZRL    "\t[-z redlocsym]\treduce local syms in .symtab to \
228                          a minimum\n"
229 @ MSG_ARG_DETAIL_ZRREL  "\t[-z relaxreloc]\trelax rules used for relocations \
230                          against COMDAT sections\n"
231 @ MSG_ARG_DETAIL_ZRS    "\t[-z rescan]\tafter processing all arguments, rescan \
232                          archive list\n\
233                          \t\tuntil no further member extraction occurs\n"
234 @ MSG_ARG_DETAIL_ZRSN   "\t[-z rescan-now]\timmediately rescan archive list \
235                          until\n\
236                          \t\tno further member extraction occurs\n"
237 @ MSG_ARG_DETAIL_ZRSGRP "\t[-z rescan-start archives... -z rescan-end],\n\
238                          \t[--start-group archives... --end-group], \
239                          [-( archives... -)]\n\
240                          \t\trescan specified archive group upon reaching\n\
241                          \t\tthe end of the group, until no further\n\
242                          \t\tmember extraction occurs\n"
243 @ MSG_ARG_DETAIL_ZSCAP  "\t[-z symbolcap]\tconvert object capabilities to \
244                          symbol capabilities\n"
245 @ MSG_ARG_DETAIL_ZTARG  "\t[-z target=platform]\n\
246                          \t\ttarget machine for cross linking\n"
247 @ MSG_ARG_DETAIL_ZT     "\t[-z text]\tdisallow output relocations against \
248                          text\n"
249 @ MSG_ARG_DETAIL_ZTO    "\t[-z textoff]\tallow output relocations against \
250                          text\n"
251 @ MSG_ARG_DETAIL_ZTW    "\t[-z textwarn]\twarn if there are relocations \
252                          against text\n"
253 @ MSG_ARG_DETAIL_ZWRAP  "\t[-z wrap=symbol], [-wrap=symbol], [--wrap=symbol]\n\
254                          \t\twrap symbol references\n"
255 @ MSG_ARG_DETAIL_ZVER   "\t[-z verbose]\t\
256                          generate warnings for suspicious processings\n"

258 #
259 # TRANSLATION_NOTE -- End of USAGE message
```

```
260 #
261 @ MSG_GRP_INVALNDX      "file %s: group section [%u]%s: entry %d: \
262                          invalid section index: %d"
263 @ MSG_GRP_INVALSYM      "file %s: group section [%u]%s: invalid group symbol %s"

265 # Relocation processing messages (some of these are required to satisfy
266 # do_reloc(), which is common code used by cmd/sgs/rtld - make sure both
267 # message files remain consistent).

269 @ MSG_REL_NOFIT         "relocation error: %s: file %s: symbol %s: \
270                          value 0x%llx does not fit"
271 @ MSG_REL_NONALIGN      "relocation error: %s: file %s: symbol %s: \
272                          offset 0x%llx is non-aligned"
273 @ MSG_REL_NULL          "relocation error: file %s: section [%u]%s: \
274                          skipping null relocation record"
275 @ MSG_REL_NOTSUP        "relocation error: %s: file %s: section [%u]%s: \
276                          relocation not currently supported"
277 @ MSG_REL_PICREDLOC     "relocation error: %s: file %s symbol %s: \
278                          -z redlocsym may not be used for pic code"
279 @ MSG_REL_TLSLE         "relocation error: %s: file %s: symbol %s: \
280                          relocation illegal when building a shared object"
281 @ MSG_REL_TLSBND        "relocation error: %s: file %s: symbol %s: \
282                          bound to: %s: relocation illegal when not bound \
283                          to object being created"
284 @ MSG_REL_TLSSTAT       "relocation error: %s: file %s: symbol %s: \
285                          relocation illegal when building a static object"
286 @ MSG_REL_TLSBADSYM     "relocation error: %s: file %s: symbol %s: \
287                          bad symbol type %s: symbol type must be TLS"
288 @ MSG_REL_BADTLS        "relocation error: %s: file %s: symbol %s: \
289                          relocation illegal for TLS symbol"
290 @ MSG_REL_BADGOTBASED   "relocation error: %s: file %s: symbol %s: a GOT \
291                          relative relocation must reference a local symbol"
292 @ MSG_REL_UNKNWSYM      "relocation error: %s: file %s: section [%u]%s: \
293                          attempt to relocate with respect to unknown \
294                          symbol %s: offset 0x%llx, symbol index %d"
295 @ MSG_REL_UNSUPSZ       "relocation error: %s: file %s: symbol %s: \
296                          offset size (%d bytes) is not supported"
297 @ MSG_REL_INVALOFFSET   "relocation error: %s: file %s section [%u]%s: \
298                          invalid offset symbol '%s': offset 0x%llx"
299 @ MSG_REL_INVALRELT     "relocation error: file %s: section [%u]%s: \
300                          invalid relocation type: 0x%x"
301 @ MSG_REL_EMPTYSEC      "relocation error: %s: file %s: symbol %s: \
302                          attempted against empty section [%u]%s"
303 @ MSG_REL_EXTERNSYM     "relocation error: %s: file %s: symbol %s: \
304                          external symbolic relocation against non-allocatable \
305                          section %s; cannot be processed at runtime: \
306                          relocation ignored"
307 @ MSG_REL_UNEXPREL      "relocation error: %s: file %s: symbol %s: \
308                          unexpected relocation; generic processing performed"
309 @ MSG_REL_UNEXPSYM      "relocation error: %s: file %s: symbol %s: \
310                          unexpected symbol referenced from file %s"
311 @ MSG_REL_SYMDISC       "relocation error: %s: file %s: section [%u]%s: \
312                          symbol %s: symbol has been discarded with discarded \
313                          section: [%u]%s"
314 @ MSG_REL_NOSYMBOL      "relocation error: %s: file %s: section: [%u]%s: \
315                          offset: 0x%llx: relocation requires reference symbol"
316 @ MSG_REL_DISPREL1      "relocation error: %s: file %s: symbol %s: \
317                          displacement relocation applied to the symbol \
318                          %s at 0x%llx: symbol %s is a copy relocated symbol"
319 @ MSG_REL_UNSUPSIZE     "relocation error: %s: file %s: section [%u]%s: \
320                          relocation against section symbol unsupported"

322 @ MSG_REL_DISPREL2      "relocation warning: %s: file %s: symbol %s: \
323                          may contain displacement relocation"
324 @ MSG_REL_DISPREL3      "relocation warning: %s: file %s: symbol %s: \
325                          displacement relocation applied to the symbol \
```



```
326                          %s: at 0x%llx: displacement relocation will not be \
327                          visible in output image"
328 @ MSG_REL_DISPREL4      "relocation warning: %s: file %s: symbol %s: \
329                          displacement relocation to be applied to the symbol \
330                          %s: at 0x%llx: displacement relocation will be \
331                          visible in output image"
332 @ MSG_REL_COPY          "relocation warning: %s: file %s: symbol %s: \
333                          relocation bound to a symbol with STV_PROTECTED \
334                          visibility"
335 @ MSG_RELINVSEC         "relocation warning: %s: file %s: section: [%u]%s: \
336                          against suspicious section [%u]%s; relocation ignored"
337 @ MSG_REL_TLSIE         "relocation warning: %s: file %s: symbol %s: \
338                          relocation has restricted use when building a shared \
339                          object"

341 @ MSG_REL_SLOPCDATNONAM "relocation warning: %s: file %s: section [%u]%s: \
342                          relocation against discarded COMDAT section [%u]%s: \
343                          redirected to file %s"
344 @ MSG_REL_SLOPCDATNAM   "relocation warning: %s: file %s: section [%u]%s: \
345                          symbol %s: relocation against discarded COMDAT \
346                          section [%u]%s: redirected to file %s"
347 @ MSG_REL_SLOPCDATNOSYM "relocation warning: %s: file %s: section [%u]%s: \
348                          symbol %s: relocation against discarded COMDAT \
349                          section [%u]%s: symbol not found, relocation ignored"

351 @ MSG_REL_NOREG         "relocation error: REGISTER relocation not supported \
352                          on target architecture"

354 #
355 # TRANSLATION_NOTE
356 #       The following 7 messages are the message to print the
357 #       following example messages.
358 #
359 #Text relocation remains                         referenced
360 #    against symbol                offset      in file
361 #str                                0x14        main.o
362 #printf                             0x1c        main.o
363 #
364 #       The first two lines are the header, and the next msgid
365 #       is the format string for the header.
366 #       Tabs and spaces are used for alignment.
367 #       The first and third %s are for: "Text relocation remains against symbol"
368 #       The second %s and fourth %s are for: "referenced in file"
369 #       The third %s is for: "offset"
370 #
371 @ MSG_REL_REMAIN_FMT_1  "%-40s\t%s\n    %s\t\t    %s\t%s"
372 #
373 # TRANSLATION_NOTE
374 #       The next two msdid make a sentence. So translate:
375 #               "Text relocation remain against symbol"
376 #       And separate them into two msgstr considering the proper
377 #       alignment.
378 @ MSG_REL_RMN_ITM_11    "Text relocation remains"
379 @ MSG_REL_RMN_ITM_12    "against symbol"
380 @ MSG_REL_RMN_ITM_13    "warning: Text relocation remains"

382 @ MSG_REL_RMN_ITM_2     "offset"

384 #
385 # TRANSLATION_NOTE
386 #       The next two msdid make a sentence. So translate:
387 #               "referenced in file"
388 #       And separate them into two msgstr considering the proper
389 #       alignment.
390 @ MSG_REL_RMN_ITM_31    "referenced"
391 @ MSG_REL_RMN_ITM_32    "in file"
```

```
392 @ MSG_REL_REMAIN_2      "%-35s 0x%-8llx\t%s"
393 @ MSG_REL_REMAIN_3      "relocations remain against allocatable but \
394                          non-writable sections"

396 # Files processing messages

398 @ MSG_FIL_MULINC_1      "file %s: attempted multiple inclusion of file"
399 @ MSG_FIL_MULINC_2      "file %s: linked to %s: attempted multiple inclusion \
400                          of file"
401 @ MSG_FIL_SOINSTAT      "input of shared object '%s' in static mode"
402 @ MSG_FIL_INVALSEC      "file %s: section [%u]%s has invalid type %s"
403 @ MSG_FIL_NOTFOUND      "file %s: required by %s, not found"
404 @ MSG_FIL_MALSTR        "file %s: section [%u]%s: malformed string table, \
405                          initial or final byte"
406 @ MSG_FIL_PTHTOLONG     "'%s/%s' pathname too long"
407 @ MSG_FIL_EXCLUDE       "file %s: section [%u]%s contains both SHF_EXCLUDE and \
408                          SHF_ALLOC flags: SHF_EXCLUDE ignored"
409 @ MSG_FIL_INTERRUPT     "file %s: creation interrupted: %s"
410 @ MSG_FIL_INVRELOC1     "file %s: section [%u]%s: relocations can not be \
411                          applied against section [%u]%s"
412 @ MSG_FIL_INVSHINFO     "file %s: section [%u]%s: has invalid sh_info: %lld"
413 @ MSG_FIL_INVSHLINK     "file %s: section [%u]%s: has invalid sh_link: %lld"
414 @ MSG_FIL_INVSHENTSIZE  "file %s: section [%u]%s: has invalid sh_entsize: %lld"
415 @ MSG_FIL_NOSTRTABLE    "file %s: section [%u]%s: symbol[%d]: specifies string \
416                          table offset 0x%llx: no string table is available"
417 @ MSG_FIL_EXCSTRTABLE   "file %s: section [%u]%s: symbol[%d]: specifies string \
418                          table offset 0x%llx: exceeds string table %s: \
419                          size 0x%llx"
420 @ MSG_FIL_NONAMESYM     "file %s: section [%u]%s: symbol[%d]: global symbol has
421                          no name"
422 @ MSG_FIL_UNKCAP        "file %s: section [%u]%s: unknown capability tag: %d"
423 @ MSG_FIL_BADSF1        "file %s: section [%u]%s: unknown software \
424                          capabilities: 0x%llx; ignored"
425 @ MSG_FIL_INADDR32SF1   "file %s: section [%u]%s: software capability ADDR32: is
426                          ineffective when building 32-bit object; ignored"
427 @ MSG_FIL_EXADDR32SF1   "file %s: section [%u]%s: software capability ADDR32: \
428                          requires executable be built with ADDR32 capability"

430 @ MSG_FIL_BADORDREF     "file %s: section [%u]%s: contains illegal reference \
431                          to discarded section: [%u]%s"

433 # Recording name conflicts

435 @ MSG_REC_OPTCNFLT      "recording name conflict: file '%s' and %s provide \
436                          identical dependency names: %s"
437 @ MSG_REC_OBJCNFLT      "recording name conflict: file '%s' and file '%s' \
438                          provide identical dependency names: %s  %s"
439 @ MSG_REC_CNFLTHINT     "(possible multiple inclusion of the same file)"

441 # System call messages

443 @ MSG_SYS_OPEN          "file %s: open failed: %s"
444 @ MSG_SYS_UNLINK        "file %s: unlink failed: %s"
445 @ MSG_SYS_MMAPANON      "mmap anon failed: %s"
446 @ MSG_SYS_MALLOC        "malloc failed: %s"


449 # Messages related to platform support

451 @ MSG_TARG_UNSUPPORTED  "unsupported ELF machine type: %s"


454 # ELF processing messages

456 @ MSG_ELF_LIBELF        "libelf: version not supported: %d"
```



```
458 @ MSG_ELF_ARMEM         "file %s: unable to locate archive member;\n\t\
459                          offset=%x, symbol=%s"

461 @ MSG_ELF_ARSYM         "file %s ignored: unable to locate archive symbol table"

463 @ MSG_ELF_VERSYM        "file %s: version symbol section entry mismatch:\n\t\
464                          (section [%u]%s entries=%d; section [%u]%s entries=%d)"

466 @ MSG_ELF_NOGROUPSECT   "file %s: section [%u]%s: SHF_GROUP flag set, but no \
467                          corresponding SHT_GROUP section found"

469 # Section processing errors

471 @ MSG_SCN_NONALLOC      "%s: non-allocatable section '%s' directed to a \
472                          loadable segment: %s"

474 @ MSG_SCN_MULTICOMDAT   "file %s: section [%u]%s: cannot be susceptible to multi
475                          COMDAT mechanisms: %s"

477 @ MSG_SCN_DWFOVRFLW     "%s: section %s: encoded DWARF data exceeds \
478                          section size"
479 @ MSG_SCN_DWFBADENC     "%s: section %s: invalid DWARF encoding: %#x"

481 #endif /* ! codereview */
482 # Symbol processing errors

484 @ MSG_SYM_NOSECDEF      "symbol '%s' in file %s has no section definition"
485 @ MSG_SYM_INVSEC        "symbol '%s' in file %s associated with invalid \
486                          section[%lld]"
487 @ MSG_SYM_TLS           "symbol '%s' in file %s (STT_TLS), is defined \
488                          in a non-SHF_TLS section"
489 @ MSG_SYM_BADADDR       "symbol '%s' in file %s: section [%u]%s: size %#llx: \
490                          symbol (address %#llx, size %#llx) lies outside \
491                          of containing section"
492 @ MSG_SYM_BADADDR_ROTXT "symbol '%s' in file %s: readonly text section \
493                          [%u]%s: size %#llx: symbol (address %#llx, \
494                          size %#llx) lies outside of containing section"
495 @ MSG_SYM_MULDEF        "symbol '%s' is multiply-defined:"
496 @ MSG_SYM_CONFVIS       "symbol '%s' has conflicting visibilities:"
497 @ MSG_SYM_DIFFTYPE      "symbol '%s' has differing types:"
498 @ MSG_SYM_DIFFATTR      "symbol '%s' has differing %s:\n\
499                          \t(file %s value=0x%llx; file %s value=0x%llx);"
500 @ MSG_SYM_FILETYPES     "\t(file %s type=%s; file %s type=%s);"
501 @ MSG_SYM_VISTYPES      "\t(file %s visibility=%s; file %s visibility=%s);"
502 @ MSG_SYM_DEFTAKEN      "\t%s definition taken"
503 @ MSG_SYM_DEFUPDATE     "\t%s definition taken and updated with larger size"
504 @ MSG_SYM_LARGER        "\tlargest value applied"
505 @ MSG_SYM_TENTERR       "\ttentative symbol cannot override defined symbol \
506                          of smaller size"

508 @ MSG_SYM_INVSHNDX      "symbol %s has invalid section index; \
509                          ignored:\n\t(file %s value=%s);"
510 @ MSG_SYM_NONGLOB       "global symbol %s has non-global binding:\n\
511                          \t(file %s value=%s);"
512 @ MSG_SYM_RESERVE       "reserved symbol '%s' already defined in file %s"
513 @ MSG_SYM_NOTNULL       "undefined symbol '%s' with non-zero value encountered \
514                          from file %s"
515 @ MSG_SYM_DUPSORTADDR   "section %s: symbol '%s' and symbol '%s' have the \
516                          same address: %#llx: remove duplicate with \
517                          NOSORTSYM mapfile directive"

519 @ MSG_PSYM_INVMINFO1    "file %s: section [%u]%s: entry[%d] has invalid m_info:
520                          0x%llx for symbol index"
521 @ MSG_PSYM_INVMINFO2    "file %s: section [%u]%s: entry[%d] has invalid m_info:
522                          0x%llx for size"
523 @ MSG_PSYM_INVMREPEAT   "file %s: section [%u]%s: entry[%d] has invalid m_repeat
```

```
524                          0x%llx"
525 @ MSG_PSYM_CANNOTEXPND  "file %s: section [%u]%s: entry[%d] can not be expanded:
526                          associated symbol size is unknown %s"
527 @ MSG_PSYM_NOSTATIC     "and partial initialization cannot be deferred to \
528                          a static object"
529 @ MSG_MOVE_OVERLAP      "file %s: section [%u]%s: symbol '%s' overlapping move \
530                          initialization: start=0x%llx, length=0x%llx: \
531                          start=0x%llx, length=0x%llx"
532 @ MSG_PSYM_EXPREASON1   "output file is static object"
533 @ MSG_PSYM_EXPREASON2   "-z nopartial option in effect"
534 @ MSG_PSYM_EXPREASON3   "move infrastructure size is greater than move data"

536 #
537 # Support library failures
538 #
539 @ MSG_SUP_NOLOAD        "dlopen() of support library (%s) failed with \
540                          error: %s"
541 @ MSG_SUP_BADVERSION    "initialization of support library (%s) failed with \
542                          bad version.  supported: %d returned: %d"


545 #
546 # TRANSLATION_NOTE
547 #       The following 7 messages are the message to print the
548 #       following example messages.
549 #
550 #Undefined                      first referenced
551 # symbol                           in file
552 #inquire                          halt_hold.o
553 #
554 @ MSG_SYM_FMT_UNDEF     "%s\t\t\t%s\
555                          \n %s  \t\t\t    %s"

557 #
558 # TRANSLATION_NOTE
559 #       The next two msdid make a sentence. So translate:
560 #               "Undefined symbol"
561 #       And separate them into two msgstr considering the proper
562 #       alignment.
563 @ MSG_SYM_UNDEF_ITM_11  "Undefined"
564 @ MSG_SYM_UNDEF_ITM_12  "symbol"
565 #
566 # TRANSLATION_NOTE
567 #       The next two msdid make a sentence. So translate:
568 #               "first referenced in file"
569 #       And separate them into two msgstr considering the proper
570 #       alignment.
571 @ MSG_SYM_UNDEF_ITM_21  "first referenced"
572 @ MSG_SYM_UNDEF_ITM_22  "in file"
573 #

575 @ MSG_SYM_UND_UNDEF     "%-35s %s"
576 @ MSG_SYM_UND_NOVER     "%-35s %s  (symbol has no version assigned)"
577 @ MSG_SYM_UND_IMPL      "%-35s %s  (symbol belongs to implicit dependency %s)"
578 @ MSG_SYM_UND_NOTA      "%-35s %s  (symbol belongs to unavailable version %s \
579                          (%s))"
580 @ MSG_SYM_UND_BNDLOCAL  "%-35s %s  (symbol scope specifies local binding)"

582 @ MSG_SYM_ENTRY         "entry point"
583 @ MSG_SYM_UNDEF         "%s symbol '%s' is undefined"
584 @ MSG_SYM_EXTERN        "%s symbol '%s' is undefined  (symbol belongs to \
585                          dependency %s)"
586 @ MSG_SYM_NOCRT         "symbol '%s' not found, but %s section exists - \
587                          possible link-edit without using the compiler driver"

589 # Output file update messages
```



```
591 @ MSG_UPD_NOREADSEG     "No read-only segments found.  Setting '_etext' to 0"
592 @ MSG_UPD_NORDWRSEG     "No read-write segments found.  Setting '_edata' to 0"
593 @ MSG_UPD_NOSEG         "Setting 'end' and '_end' to 0"

595 @ MSG_UPD_SEGOVERLAP    "%s: segment address overlap;\n\
596                          \tprevious segment ending at address 0x%llx overlaps\n\
597                          \tuser defined segment '%s' starting at address 0x%llx"
598 @ MSG_UPD_LARGSIZE      "%s: segment %s calculated size 0x%llx\n\
599                          \tis larger than user-defined size 0x%llx"

601 @ MSG_UPD_NOBITS        "NOBITS section found before end of initialized data"
602 @ MSG_SEG_FIRNOTLOAD    "First segment has type %s, PT_LOAD required: %s"
603 @ MSG_UPD_MULEHFRAME    "file %s; section [%u]%s and file %s; section [%u]%s \
604                          have incompatibile attributes and cannot \
605                          be merged into a single output section"


608 # Version processing messages

610 @ MSG_VER_HIGHER        "file %s: version revision %d is higher than \
611                          expected %d"
612 @ MSG_VER_NOEXIST       "file %s: version '%s' does not exist:\n\
613                          \trequired by file %s"
614 @ MSG_VER_UNDEF         "version '%s' undefined, referenced by version '%s':\n\
615                          \trequired by file %s"
616 @ MSG_VER_UNAVAIL       "file %s: version '%s' is unavailable:\n\
617                          \trequired by file %s"
618 @ MSG_VER_DEFINED       "version symbol '%s' already defined in file %s"
619 @ MSG_VER_INVALNDX      "version symbol '%s' from file %s has an invalid \
620                          version index (%d)"
621 @ MSG_VER_ADDVERS       "unused $ADDVERS specification from file '%s' \
622                          for object '%s'\nversion(s):"
623 @ MSG_VER_ADDVER        "\t%s"
624 @ MSG_VER_CYCLIC        "following versions generate cyclic dependency:"

626 # Capabilities messages

628 @ MSG_CAP_MULDEF        "capabilities symbol '%s' has multiply-defined members:"
629 @ MSG_CAP_MULDEFSYMS    "\t(file %s symbol '%s'; file %s symbol '%s');"
630 @ MSG_CAP_REDUNDANT     "file %s: section [%u]%s: symbol capabilities \
631                          redundant, as object capabilities are more restrictive"
632 @ MSG_CAP_NOSYMSFOUND   "no global symbols have been found that are associated \
633                          with capabilities identified relocatable objects: \
634                          -z symbolcap has no effect"

636 @ MSG_CAPINFO_INVALSYM  "file %s: capabilities info section [%u]%s: index %d: \
637                          family member symbol '%s': invalid"
638 @ MSG_CAPINFO_INVALLEAD "file %s: capabilities info section [%u]%s: index %d: \
639                          family lead symbol '%s': invalid symbol index %d"

641 # Basic strings

643 @ MSG_STR_ALIGNMENTS    "alignments"
644 @ MSG_STR_COMMAND       "(command line)"
645 @ MSG_STR_TLSREL        "(internal TLS relocation requirement)"
646 @ MSG_STR_SIZES         "sizes"
647 @ MSG_STR_UNKNOWN       "<unknown>"
648 @ MSG_STR_SECTION       "%s (section)"
649 @ MSG_STR_SECTION_MSTR  "%s (merged string section)"

651 #
652 # TRANSLATION_NOTE
653 #       The elf_ function name represents a man page reference and should not
654 #       be translated.
655 @ MSG_ELF_BEGIN         "file %s: elf_begin"
```

```
656 @ MSG_ELF_CNTL          "file %s: elf_cntl"
657 @ MSG_ELF_GETARHDR      "file %s: elf_getarhdr"
658 @ MSG_ELF_GETARSYM      "file %s: elf_getarsym"
659 @ MSG_ELF_GETDATA       "file %s: elf_getdata"
660 @ MSG_ELF_GETEHDR       "file %s: elf_getehdr"
661 @ MSG_ELF_GETPHDR       "file %s: elf_getphdr"
662 @ MSG_ELF_GETSCN        "file %s: elf_getscn: scnndx: %d"
663 @ MSG_ELF_GETSHDR       "file %s: elf_getshdr"
664 @ MSG_ELF_MEMORY        "file %s: elf_memory"
665 @ MSG_ELF_NDXSCN        "file %s: elf_ndxscn"
666 @ MSG_ELF_NEWDATA       "file %s: elf_newdata"
667 @ MSG_ELF_NEWEHDR       "file %s: elf_newehdr"
668 @ MSG_ELF_NEWSCN        "file %s: elf_newscn"
669 @ MSG_ELF_NEWPHDR       "file %s: elf_newphdr"
670 @ MSG_ELF_STRPTR        "file %s: elf_strptr"
671 @ MSG_ELF_UPDATE        "file %s: elf_update"
672 @ MSG_ELF_SWAP_WRIMAGE  "file %s: _elf_swap_wrimage"


675 @ MSG_REJ_MACH          "file %s: wrong ELF machine type: %s"
676 @ MSG_REJ_CLASS         "file %s: wrong ELF class: %s"
677 @ MSG_REJ_DATA          "file %s: wrong ELF data format: %s"
678 @ MSG_REJ_TYPE          "file %s: bad ELF type: %s"
679 @ MSG_REJ_BADFLAG       "file %s: bad ELF flags value: %s"
680 @ MSG_REJ_MISFLAG       "file %s: mismatched ELF flags value: %s"
681 @ MSG_REJ_VERSION       "file %s: mismatched ELF/lib version: %s"
682 @ MSG_REJ_HAL           "file %s: HAL R1 extensions required"
683 @ MSG_REJ_US3           "file %s: Sun UltraSPARC III extensions required"
684 @ MSG_REJ_STR           "file %s: %s"
685 @ MSG_REJ_UNKFILE       "file %s: unknown file type"
686 @ MSG_REJ_UNKCAP        "file=%s; unknown capability: %d"
687 @ MSG_REJ_HWCAP_1       "file %s: hardware capability (CA_SUNW_HW_1) \
688                          unsupported: %s"
689 @ MSG_REJ_SFCAP_1       "file %s: software capability (CA_SUNW_SF_1) \
690                          unsupported: %s"
691 @ MSG_REJ_MACHCAP       "file %s: machine capability (CA_SUNW_MACH) \
692                          unsupported: %s"
693 @ MSG_REJ_PLATCAP       "file %s: platform capability (CA_SUNW_PLAT) \
694                          unsupported: %s"
695 @ MSG_REJ_HWCAP_2       "file %s: hardware capability (CA_SUNW_HW_2) \
696                          unsupported: %s"
697 @ MSG_REJ_ARCHIVE       "file %s: invalid archive use"

699 # Guidance messages
700 @ MSG_GUIDE_SUMMARY     "see ld(1) -z guidance for more information"
701 @ MSG_GUIDE_DEFS        "-z defs option recommended for shared objects"
702 @ MSG_GUIDE_DIRECT      "-B direct or -z direct option recommended before \
703                          first dependency"
704 @ MSG_GUIDE_LAZYLOAD    "-z lazyload option recommended before \
705                          first dependency"
706 @ MSG_GUIDE_MAPFILE     "version 2 mapfile syntax recommended: %s"
707 @ MSG_GUIDE_TEXT        "position independent (PIC) code recommended for \
708                          shared objects"
709 @ MSG_GUIDE_UNUSED      "removal of unused dependency recommended: %s"

711 @ _END_


714 # The following strings represent reserved names.  Reference to these strings
715 # is via the MSG_ORIG() macro, and thus translations are not required.

717 @ MSG_STR_EOF           "<eof>"
718 @ MSG_STR_ERROR         "<error>"
719 @ MSG_STR_EMPTY         ""
720 @ MSG_QSTR_BANG         "'!'"
721 @ MSG_STR_COLON         ":"
```



```
722 @ MSG_QSTR_COLON        "':'"
723 @ MSG_QSTR_SEMICOLON    "';'"
724 @ MSG_QSTR_EQUAL        "'='"
725 @ MSG_QSTR_PLUSEQ       "'+='"
726 @ MSG_QSTR_MINUSEQ      "'-='"
727 @ MSG_QSTR_ATSIGN       "'@'"
728 @ MSG_QSTR_DASH         "'-'"
729 @ MSG_QSTR_LEFTBKT      "'{'"
730 @ MSG_QSTR_RIGHTBKT     "'}'"
731 @ MSG_QSTR_PIPE         "'|'"
732 @ MSG_QSTR_STAR         "'*'"
733 @ MSG_STR_DOT           "."
734 @ MSG_STR_SLASH         "/"
735 @ MSG_STR_DYNAMIC       "(.dynamic)"
736 @ MSG_STR_ORIGIN        "$ORIGIN"
737 @ MSG_STR_MACHINE       "$MACHINE"
738 @ MSG_STR_PLATFORM      "$PLATFORM"
739 @ MSG_STR_ISALIST       "$ISALIST"
740 @ MSG_STR_OSNAME        "$OSNAME"
741 @ MSG_STR_OSREL         "$OSREL"
742 @ MSG_STR_UU_REAL_U     "__real_"
743 @ MSG_STR_UU_WRAP_U     "__wrap_"
744 @ MSG_STR_UELF32        "_ELF32"
745 @ MSG_STR_UELF64        "_ELF64"
746 @ MSG_STR_USPARC        "_sparc"
747 @ MSG_STR_UX86          "_x86"
748 @ MSG_STR_TRUE          "true"

750 @ MSG_STR_CDIR_ADD      "$add"
751 @ MSG_STR_CDIR_CLEAR    "$clear"
752 @ MSG_STR_CDIR_ERROR    "$error"
753 @ MSG_STR_CDIR_MFVER    "$mapfile_version"
754 @ MSG_STR_CDIR_IF       "$if"
755 @ MSG_STR_CDIR_ELIF     "$elif"
756 @ MSG_STR_CDIR_ELSE     "$else"
757 @ MSG_STR_CDIR_ENDIF    "$endif"

759 @ MSG_STR_GROUP         "GROUP"
760 @ MSG_STR_SUNW_COMDAT   "SUNW_COMDAT"

762 @ MSG_FMT_ARMEM         "%s(%s)"
763 @ MSG_FMT_COLPATH       "%s:%s"
764 @ MSG_FMT_SYMNAM        "'%s'"
765 @ MSG_FMT_NULLSYMNAM    "%s[%d]"
766 @ MSG_FMT_STRCAT        "%s%s"

768 @ MSG_PTH_RTLD          "/usr/lib/ld.so.1"

770 @ MSG_SUNW_OST_SGS      "SUNW_OST_SGS"


773 # Section strings

775 @ MSG_SCN_BSS           ".bss"
776 @ MSG_SCN_DATA          ".data"
777 @ MSG_SCN_COMMENT       ".comment"
778 @ MSG_SCN_DEBUG         ".debug"
779 @ MSG_SCN_DEBUG_INFO    ".debug_info"
780 @ MSG_SCN_DYNAMIC       ".dynamic"
781 @ MSG_SCN_DYNSYMSORT    ".SUNW_dynsymsort"
782 @ MSG_SCN_DYNTLSSORT    ".SUNW_dyntlssort"
783 @ MSG_SCN_DYNSTR        ".dynstr"
784 @ MSG_SCN_DYNSYM        ".dynsym"
785 @ MSG_SCN_DYNSYM_SHNDX  ".dynsym_shndx"
786 @ MSG_SCN_LDYNSYM       ".SUNW_ldynsym"
787 @ MSG_SCN_LDYNSYM_SHNDX ".SUNW_ldynsym_shndx"
```

```
788 @ MSG_SCN_EX_SHARED     ".ex_shared"
789 @ MSG_SCN_EX_RANGES     ".exception_ranges"
790 @ MSG_SCN_EXCL          ".excl"
791 @ MSG_SCN_FINI          ".fini"
792 @ MSG_SCN_FINIARRAY     ".fini_array"
793 @ MSG_SCN_GOT           ".got"
794 @ MSG_SCN_GNU_LINKONCE  ".gnu.linkonce."
795 @ MSG_SCN_HASH          ".hash"
796 @ MSG_SCN_INDEX         ".index"
797 @ MSG_SCN_INIT          ".init"
798 @ MSG_SCN_INITARRAY     ".init_array"
799 @ MSG_SCN_INTERP        ".interp"
800 @ MSG_SCN_LBSS          ".lbss"
801 @ MSG_SCN_LDATA         ".ldata"
802 @ MSG_SCN_LINE          ".line"
803 @ MSG_SCN_LRODATA       ".lrodata"
804 @ MSG_SCN_PLT           ".plt"
805 @ MSG_SCN_PREINITARRAY  ".preinit_array"
806 @ MSG_SCN_REL           ".rel"
807 @ MSG_SCN_RELA          ".rela"
808 @ MSG_SCN_RODATA        ".rodata"
809 @ MSG_SCN_SBSS          ".sbss"
810 @ MSG_SCN_SBSS2         ".sbss2"
811 @ MSG_SCN_SDATA         ".sdata"
812 @ MSG_SCN_SDATA2        ".sdata2"
813 @ MSG_SCN_SHSTRTAB      ".shstrtab"
814 @ MSG_SCN_STAB          ".stab"
815 @ MSG_SCN_STABEXCL      ".stab.exclstr"
816 @ MSG_SCN_STRTAB        ".strtab"
817 @ MSG_SCN_SUNWMOVE      ".SUNW_move"
818 @ MSG_SCN_SUNWRELOC     ".SUNW_reloc"
819 @ MSG_SCN_SUNWSYMINFO   ".SUNW_syminfo"
820 @ MSG_SCN_SUNWVERSION   ".SUNW_version"
821 @ MSG_SCN_SUNWVERSYM    ".SUNW_versym"
822 @ MSG_SCN_SUNWCAP       ".SUNW_cap"
823 @ MSG_SCN_SUNWCAPINFO   ".SUNW_capinfo"
824 @ MSG_SCN_SUNWCAPCHAIN  ".SUNW_capchain"
825 @ MSG_SCN_SYMTAB        ".symtab"
826 @ MSG_SCN_SYMTAB_SHNDX  ".symtab_shndx"
827 @ MSG_SCN_TBSS          ".tbss"
828 @ MSG_SCN_TDATA         ".tdata"
829 @ MSG_SCN_TEXT          ".text"

831 @ MSG_SYM_FINIARRAY     "finiarray"
832 @ MSG_SYM_INITARRAY     "initarray"
833 @ MSG_SYM_PREINITARRAY  "preinitarray"

835 #
836 # GNU section names
837 #
838 @ MSG_SCN_CTORS         ".ctors"
839 @ MSG_SCN_DTORS         ".dtors"
840 @ MSG_SCN_EHFRAME       ".eh_frame"
841 @ MSG_SCN_EHFRAME_HDR   ".eh_frame_hdr"
842 @ MSG_SCN_GCC_X_TBL     ".gcc_except_table"
843 @ MSG_SCN_JCR           ".jcr"

845 # Segment names for segments referenced by entrance criteria

847 @ MSG_ENT_BSS           "bss"
848 @ MSG_ENT_DATA          "data"
849 @ MSG_ENT_EXTRA         "extra"
850 @ MSG_ENT_LDATA         "ldata"
851 @ MSG_ENT_LRODATA       "lrodata"
852 @ MSG_ENT_NOTE          "note"
853 @ MSG_ENT_TEXT          "text"
```



```
855 # Symbol names

857 @ MSG_SYM_START         "_start"
858 @ MSG_SYM_MAIN          "main"

860 @ MSG_SYM_FINI_U        "_fini"
861 @ MSG_SYM_INIT_U        "_init"
862 @ MSG_SYM_DYNAMIC       "DYNAMIC"
863 @ MSG_SYM_DYNAMIC_U     "_DYNAMIC"
864 @ MSG_SYM_EDATA         "edata"
865 @ MSG_SYM_EDATA_U       "_edata"
866 @ MSG_SYM_END           "end"
867 @ MSG_SYM_END_U         "_end"
868 @ MSG_SYM_ETEXT         "etext"
869 @ MSG_SYM_ETEXT_U       "_etext"
870 @ MSG_SYM_GOFTBL        "GLOBAL_OFFSET_TABLE_"
871 @ MSG_SYM_GOFTBL_U      "_GLOBAL_OFFSET_TABLE_"
872 @ MSG_SYM_PLKTBL        "PROCEDURE_LINKAGE_TABLE_"
873 @ MSG_SYM_PLKTBL_U      "_PROCEDURE_LINKAGE_TABLE_"
874 @ MSG_SYM_TLSGETADDR_U  "__tls_get_addr"
875 @ MSG_SYM_TLSGETADDR_UU "___tls_get_addr"

877 @ MSG_SYM_L_END         "END_"
878 @ MSG_SYM_L_END_U       "_END_"
879 @ MSG_SYM_L_START       "START_"
880 @ MSG_SYM_L_START_U     "_START_"

882 # Support functions

884 @ MSG_SUP_VERSION       "ld_version"
885 @ MSG_SUP_INPUT_DONE    "ld_input_done"

887 @ MSG_SUP_START_64      "ld_start64"
888 @ MSG_SUP_ATEXIT_64     "ld_atexit64"
889 @ MSG_SUP_OPEN_64       "ld_open64"
890 @ MSG_SUP_FILE_64       "ld_file64"
891 @ MSG_SUP_INSEC_64      "ld_input_section64"
892 @ MSG_SUP_SEC_64        "ld_section64"

894 @ MSG_SUP_START         "ld_start"
895 @ MSG_SUP_ATEXIT        "ld_atexit"
896 @ MSG_SUP_OPEN          "ld_open"
897 @ MSG_SUP_FILE          "ld_file"
898 @ MSG_SUP_INSEC         "ld_input_section"
899 @ MSG_SUP_SEC           "ld_section"

901 #
902 # Message previously in ‘ld’
903 #
904 #
905 @ _START_

907 # System error messages

909 @ MSG_SYS_STAT          "file %s: stat failed: %s"
910 @ MSG_SYS_READ          "file %s: read failed: %s"
911 @ MSG_SYS_NOTREG        "file %s: is not a regular file"

913 # Argument processing messages

915 @ MSG_ARG_DY_INCOMP     "%s option is incompatible with building a dynamic \
916                          executable"
917 @ MSG_MARG_DY_INCOMP    "%s is incompatible with building a dynamic \
918                          executable"
919 @ MSG_ARG_ST_INCOMP     "%s option is incompatible with building a static \
```

```
920                          object (-dn, -r, --relocatable)"
921 @ MSG_MARG_ST_INCOMP    "%s is incompatible with building a static \
922                          object (-dn, -r, --relocatable)"
923 @ MSG_MARG_ST_ONLYAVL   "%s is only available when building a shared object"
924 @ MSG_ARG_INCOMP        "option %s and %s are incompatible"
925 @ MSG_MARG_INCOMP       "%s and %s are incompatible"
926 @ MSG_ARG_MTONCE        "option %s appears more than once, first setting taken"
927 @ MSG_MARG_MTONCE       "%s appears more than once, first setting taken"
928 @ MSG_ARG_ILLEGAL       "option %s has illegal argument '%s'"
929 @ MSG_ARG_YP            "option -YP and -Y%c may not be specified concurrently"
930 @ MSG_ARG_STRIP         "%s specified with %s; only debugging \
931                          information stripped"
932 @ MSG_ARG_NOFILES       "no files on input command line"
933 @ MSG_ARG_NOFLTR        "option %s is only meaningful when building a filter"
934 @ MSG_ARG_NODEFLIB      "the default library search path has been suppressed, \
935                          but no runpaths have been specified via %s"
936 @ MSG_ARG_NOENTRY       "entry point symbol '%s' is undefined"
937 @ MSG_ARG_UNSUPPORTED   "option %s is no longer supported; ignored"
938 @ MSG_MARG_ONLY         "option %s can only be used with a %s"
939 @ MSG_ARG_UNKNOWN       "unrecognized option '-%c'"
940 @ MSG_ARG_LONG_UNKNOWN  "unrecognized option '%s'"
941 @ MSG_ARG_USEHELP       "use the -z help option for usage information"


944 @ MSG_ARG_FLAGS         "flags processing errors"
945 @ MSG_ARG_FILES         "file processing errors. No output written to %s"
946 @ MSG_ARG_SYM_WARN      "symbol referencing errors"
947 @ MSG_ARG_SYM_FATAL     "symbol referencing errors. No output written to %s"
948 @ MSG_ARG_AR_GRP_OLAP   "%s cannot be nested"
949 @ MSG_ARG_AR_GRP_BAD    "%s used without corresponding %s"


952 # Messages used to refer to options where there is more than
953 # one name accepted.

955 @ MSG_MARG_AR_GRPS      "archive rescan groups \
956                          (-z rescan-start, -(, --start-group)"
957 @ MSG_MARG_AR_GRP_END   "archive rescan group end option \
958                          (-z rescan-end, -), --end-group)"
959 @ MSG_MARG_AR_GRP_START "archive rescan group start option \
960                          (-z rescan-start, -(, --start-group)"
961 @ MSG_MARG_ENTRY        "entry point option (-e, --entry)"
962 @ MSG_MARG_FILTER_AUX   "auxiliary filter option (-f, --auxiliary)"
963 @ MSG_MARG_FILTER       "filter option (-F, --filter)"
964 @ MSG_MARG_OUTFILE      "output object option (-o, --output)"
965 @ MSG_MARG_REL          "relocatable object option (-r, --relocatable)"
966 @ MSG_MARG_RPATH        "runpath option (-R, -rpath)"
967 @ MSG_MARG_SO           "shared object option (-G, -shared)"
968 @ MSG_MARG_SONAME       "soname option (-h, --soname)"
969 @ MSG_MARG_STRIP        "strip option (-s, --strip-all)"

971 # Entrance criteria messages

973 @ MSG_ENT_MAP_FMT_TIL_1 "\t\t%s\n\n"
974 @ MSG_ENT_MAP_TITLE_1   "LINK EDITOR MEMORY MAP"

976 #
977 # TRANSLATION_NOTE -- Entry map header
978 #
979 # The next message is a format string for a title. The title is composed of
980 # two lines. In C locale, it would look like:
981 #
982 #       output          input           new
983 #       section         section         displacement    size
984 #
985 # The \t characters are used for alignment.  (output section), (input section),
```



```
986 # and (new displacement) have to be aligned.
987 #
988 @ MSG_ENT_MAP_FMT_TIL_2 "\n%s\t\t%s\t\t%s\n%s\t\t%s\t\t%s\t%s\n\n"
989 @ MSG_ENT_MAP_FMT_TIL_3 "\n%s\t\t%s\t\t%s\n%s\t\t%s\t\t%s\t%s\n\n"
990 @ MSG_ENT_ITM_OUTPUT    "output"
991 @ MSG_ENT_ITM_INPUT     "input"
992 @ MSG_ENT_ITM_NEW       "new"
993 @ MSG_ENT_ITM_SECTION   "section"
994 @ MSG_ENT_ITM_DISPMNT   "displacement"
995 @ MSG_ENT_ITM_SIZE      "size"
996 @ MSG_ENT_ITM_VIRTUAL   "virtual"
997 @ MSG_ENT_ITM_ADDRESS   "address"

999 @ MSG_ENT_MAP_ENTRY_1   "%-8.8s\t\t\t%08.2llx\t%08.2llx\n"
1000 @ MSG_ENT_MAP_ENTRY_2   "\t\t%-8.8s\t%08.2llx\t%08.2llx %s\n"

1002 #
1003 # TRANSLATION_NOTE -- multiple defined symbol table header
1004 #
1005 # In C locale, an example output is:
1006 #
1007 #               MULTIPLY DEFINED SYMBOLS
1008 #
1009 #
1010 #symbol                          definition used     also defined in
1011 #
1012 #variable1                          main.o
1013 #                                                      ./libfred.so
1014 @ MSG_ENT_MUL_FMT_TIL_0 "\n\n\t\t%s\n\n\n"
1015 @ MSG_ENT_MUL_TIL_0     "MULTIPLY DEFINED SYMBOLS"

1017 #
1018 # TRANSLATION_NOTE -- This is the format string for:
1019 #
1020 #symbol                          definition used     also defined in
1021 #
1022 @ MSG_ENT_MUL_FMT_TIL_1 "%s\t\t\t\t  %s     %s\n\n"
1023 @ MSG_ENT_MUL_ITM_SYM   "symbol"
1024 @ MSG_ENT_MUL_ITM_DEF_0 "definition used"
1025 @ MSG_ENT_MUL_ITM_DEF_1 "also defined in"

1027 #
1028 # TRANSLATION_NOTE -- This is the format string for the second item:
1029 #
1030 @ MSG_ENT_MUL_ENTRY_1   "%-35s %s\n"

1032 #
1033 # TRANSLATION_NOTE -- This is the format string for the third item:
1034 #
1035 @ MSG_ENT_MUL_ENTRY_2   "\t\t\t\t\t\t\t%s\n"

1037 @ MSG_ENT_NOSEC_1       "mapfile: %s segment: section '%s' does not appear \
1038                          in mapfile specified input file(s)"
1039 @ MSG_ENT_NOSEC_2       "mapfile: %s segment: section '%s' does not appear \
1040                          in any input file"

1042 # Library messages

1044 @ MSG_LIB_NOTFOUND      "library -l%s: not found"
1045 @ MSG_LIB_MALFORM       "LD_LIBRARY_PATH malformed"
1046 @ MSG_LIB_BADYP         "-YP library path malformed"


1049 # Mapfile processing messages

1051 @ MSG_MAP_BADAUTORED    "%s: %llu: auto-reduction ('*') can only be used in \
```

```
1052                            hidden/local, or eliminate scope"
1053 @ MSG_MAP_BADFLAG        "%s: %llu: badly formed section flags '%s'"
1054 @ MSG_MAP_BADBNAME       "%s: %llu: basename cannot contain path \
1055                             separator ('/'): %s"
1056 @ MSG_MAP_BADONAME       "%s: %llu: object name cannot contain path \
1057                             separator ('/'): %s"
1058 @ MSG_MAP_REDEFATT       "%s: %llu: redefining %s attribute for '%s'"
1059 @ MSG_MAP_PREMEOF        "%s: %llu: premature EOF"
1060 @ MSG_MAP_ILLCHAR        "%s: %llu: illegal character '\\%03o'"
1061 @ MSG_MAP_MALFORM        "%s: %llu: malformed entry"
1062 @ MSG_MAP_NONLOAD        "%s: %llu: %s not allowed on non-LOAD segments"
1063 @ MSG_MAP_NOSTACK1       "%s: %llu: %s not allowed on STACK segment"
1064 @ MSG_MAP_MOREONCE       "%s: %llu: %s set more than once on same line"
1065 @ MSG_MAP_NOTERM         "%s: %llu: unterminated quoted string: %s"
1066 @ MSG_MAP_SECINSEG       "%s: %llu: section within segment ordering done on \
1067                             a non-existent segment '%s'"
1068 @ MSG_MAP_UNEXINHERIT    "%s: %llu: unnamed version cannot inherit from other \
1069                             versions: %s"
1070 @ MSG_MAP_UNEXTOK        "%s: %llu: unexpected occurrence of '%c' token"

1072 @ MSG_MAP_SEGEMPLOAD     "%s: %llu: empty segment must be of type LOAD or NULL"
1073 @ MSG_MAP_SEGEMPEXE      "%s: %llu: a LOAD empty segment definition is only \
1074                             allowed when creating a dynamic executable"
1075 @ MSG_MAP_SEGEMPATT      "%s: %llu: a LOAD empty segment must have an address \
1076                             and size"
1077 @ MSG_MAP_SEGEMPNOATT    "%s: %llu: a NULL empty segment must not have an \
1078                             address or size"
1079 @ MSG_MAP_SEGEMPSEC      "%s: %llu: empty segment can not have sections \
1080                             assigned to it"
1081 @ MSG_MAP_SEGEMNOPERM    "%s: %llu: empty segment must not have \
1082                             p_flags set: 0x%x"

1084 @ MSG_MAP_CNTADDRORDER   "%s: %llu: segment cannot have an explicit address \
1085                             and also be in the SEGMENT_ORDER list: %s"
1086 @ MSG_MAP_CNTDISSEG      "%s: %llu: segment cannot be disabled: %s"
1087 @ MSG_MAP_DUPNAMENT      "%s: %llu: cannot redefine entrance criteria: %s"
1088 @ MSG_MAP_DUPORDSEG      "%s: %llu: segment is already in %s list: %s"
1089 @ MSG_MAP_DUP_OS_ORD     "%s: %llu: section is already in OS_ORDER list: %s"
1090 @ MSG_MAP_DUP_IS_ORD     "%s: %llu: entrance criteria is already in \
1091                             IS_ORDER list: %s"
1092 @ MSG_MAP_UNKENT         "%s: %llu: unknown entrance criteria \
1093                             (ASSIGN_SECTION): %s"
1094 @ MSG_MAP_UNKSEG         "%s: %llu: unknown segment: %s"
1095 @ MSG_MAP_UNKSYMDEF      "%s: %llu: unknown symbol definition: %s"
1096 @ MSG_MAP_UNKSEGTYP      "%s: %llu: unknown internal segment type %d"
1097 @ MSG_MAP_UNKSOTYP       "%s: %llu: unknown shared object type: %s"
1098 @ MSG_MAP_UNKSEGATT      "%s: %llu: unknown segment attribute: %s"
1099 @ MSG_MAP_UNKSEGFLG      "%s: %llu: unknown segment flag: ?%c"
1100 @ MSG_MAP_UNKSECTYP      "%s: %llu: unknown section type: %s"

1102 @ MSG_MAP_SEGSIZE        "%s: %lld: existing segment size symbols cannot \
1103                             be reset: %s"
1104 @ MSG_MAP_SEGADDR        "%s: %llu: segment address or length '%s' %s"
1105 @ MSG_MAP_BADCAPVAL      "%s: %llu: bad capability value: %s"
1106 @ MSG_MAP_UNKCAPATTR     "%s: %llu: unknown capability attribute '%s'"
1107 @ MSG_MAP_EMPTYCAP       "%s: %llu: empty capability definition; ignored"

1109 @ MSG_MAP_SYMDEF1        "%s: %llu: symbol '%s' is already defined in file: \
1110                             %s: %s"
1111 @ MSG_MAP_SYMDEF2        "%s: %llu: symbol '%s': %s"

1113 @ MSG_MAP_EXPSCOL        "%s: %llu: expected a ';'"
1114 @ MSG_MAP_EXPEQU         "%s: %llu: expected a '=', ':', '|', or '@'"
1115 @ MSG_MAP_EXPSEGATT      "%s: %llu: expected one or more segment attributes \
1116                             after an '='"
1117 @ MSG_MAP_EXPSEGNAM      "%s: %llu: expected a segment name at the beginning \
```



```
1118                            of a line"
1119 @ MSG_MAP_EXPSEGTYPE     "%s: %llu: %s segment cannot be used with %s \
1120                             directive: %s"
1121 @ MSG_MAP_EXPSYM_1       "%s: %llu: expected a symbol name after '@'"
1122 @ MSG_MAP_EXPSYM_2       "%s: %llu: expected a symbol name after '{'"
1123 @ MSG_MAP_EXPSEC         "%s: %llu: expected a section name after '|'"
1124 @ MSG_MAP_EXPSO          "%s: %llu: expected a shared object definition \
1125                             after '-'"
1126 @ MSG_MAP_MULTFILTEE     "%s: %llu: multiple filtee definitions are unsupported"
1127 @ MSG_MAP_NOFILTER       "%s: %llu: filtee definition required"
1128 @ MSG_MAP_BADSF1         "%s: %llu: unknown software capabilities: 0x%llx; \
1129                             ignored"
1130 @ MSG_MAP_INADDR32SF1    "%s: %llu: software capability ADDR32: is ineffective \
1131                             when building 32-bit object: ignored"
1132 @ MSG_MAP_NOINTPOSE      "%s: %llu: interposition symbols can only be defined \
1133                             when building a dynamic executable"
1134 @ MSG_MAP_NOEXVLSZ       "%s: %llu: value and size attributes are incompatible \
1135                             with extern or parent symbols"
1136 @ MSG_MAP_FLTR_ONLYAVL   "%s: %llu: symbol filtering is only available when \
1137                             building a shared object"

1139 @ MSG_MAP_SEGSAME        "segments '%s' and '%s' have the same assigned \
1140                             virtual address"
1141 @ MSG_MAP_EXCLIMIT       "exceeds internal limit"
1142 @ MSG_MAP_NOBADFRM       "number is badly formed"

1144 @ MSG_MAP_SEGTYP         "segment type"
1145 @ MSG_MAP_SEGVADDR       "segment virtual address"
1146 @ MSG_MAP_SEGPHYS        "segment physical address"
1147 @ MSG_MAP_SEGLEN         "segment length"
1148 @ MSG_MAP_SEGFLAG        "segment flags"
1149 @ MSG_MAP_SEGALIGN       "segment alignment"
1150 @ MSG_MAP_SEGROUND       "segment rounding"

1152 @ MSG_MAP_SECTYP         "section type"
1153 @ MSG_MAP_SECFLAG        "section flags"
1154 @ MSG_MAP_SECNAME        "section name"

1156 @ MSG_MAP_SYMVAL         "symbol value"
1157 @ MSG_MAP_SYMSIZE        "symbol size"

1159 @ MSG_MAP_DIFF_SYMVAL    "symbol values differ"
1160 @ MSG_MAP_DIFF_SYMSZ     "symbol sizes differ"
1161 @ MSG_MAP_DIFF_SYMTYP    "symbol types differ"
1162 @ MSG_MAP_DIFF_SYMNDX    "symbol indexes differ"
1163 @ MSG_MAP_DIFF_SYMLCL    "symbol scope conflict against local and non-local"
1164 @ MSG_MAP_DIFF_SYMGLOB   "symbol scope conflict against singleton/exported"
1165 @ MSG_MAP_DIFF_SYMPROT   "symbol scope conflict against protected"
1166 @ MSG_MAP_DIFF_SYMVER    "symbol version conflict"
1167 @ MSG_MAP_DIFF_SYMMUL    "symbol multiple definition"
1168 @ MSG_MAP_DIFF_SNGLDIR   "singleton scope and direct declaration are \
1169                             incompatible"
1170 @ MSG_MAP_DIFF_PROTNDIR "protected scope and no-direct declaration \
1171                             are incompatible"


1174 @ MSG_MAP_SECORDER       "section ordering requested, but no matching section \
1175                             found: segment: %s section: %s"


1178 # Mapfile Directives

1180 @ MSG_MAP_EXP_ATTR       "%s: %llu: expected attribute name (%s), or \
1181                             terminator (';', '}'): %s"
1182 @ MSG_MAP_EXP_CAPMASK    "%s: %llu: expected capability name, integer value, or \
1183                             terminator (';', '}'): %s"
```

```
1184 @ MSG_MAP_EXP_CAPNAME   "%s: %llu: expected name, or terminator (';', '}'): %s"
1185 @ MSG_MAP_EXP_CAPID     "%s: %llu: expected name, or '{' following %s: %s"
1186 @ MSG_MAP_EXP_CAPHW     "%s: %llu: expected hardware capability, or \
1187                          terminator (';', '}'): %s"
1188 @ MSG_MAP_EXP_CAPSF     "%s: %llu: expected software capability, or \
1189                          terminator (';', '}'): %s"
1190 @ MSG_MAP_EXP_EQ        "%s: %llu: expected '=' following %s: %s"
1191 @ MSG_MAP_EXP_EQ_ALL    "%s: %llu: expected '=', '+=', or '-=' following %s: %s"
1192 @ MSG_MAP_EXP_EQ_PEQ    "%s: %llu: expected '=' following %s: %s"
1193 @ MSG_MAP_EXP_DIR       "%s: %llu: expected mapfile directive (%s): %s"
1194 @ MSG_MAP_SFLG_EXBANG   "%s: %llu: '!' appears without corresponding flag"
1195 @ MSG_MAP_EXP_FILNAM    "%s: %llu: expected file name following %s: %s"
1196 @ MSG_MAP_EXP_FILPATH   "%s: %llu: expected file path following %s: %s"
1197 @ MSG_MAP_EXP_INT       "%s: %llu: expected integer value following %s: %s"
1198 @ MSG_MAP_EXP_LBKT      "%s: %llu: expected '{' following %s: %s"
1199 @ MSG_MAP_EXP_OBJNAM    "%s: %llu: expected object name following %s: %s"
1200 @ MSG_MAP_SFLG_ONEBANG  "%s: %llu: '!' can only be specified once per flag"
1201 @ MSG_MAP_EXP_SECFLAG   "%s: %llu: expected section flag (%s), '!', or \
1202                          terminator (';', '}'): %s"
1203 @ MSG_MAP_EXP_SECNAM    "%s: %llu: expected section name following %s: %s"
1204 @ MSG_MAP_EXP_SEGFLAG   "%s: %llu: expected segment flag (%s), or \
1205                          terminator (';', '}'): %s"
1206 @ MSG_MAP_EXP_ECNAM     "%s: %llu: expected entrance criteria (ASSIGN_SECTION) \
1207                          name, or terminator (';', '}'): %s"
1208 @ MSG_MAP_EXP_SEGNAM    "%s: %llu: expected segment name following %s: %s"
1209 @ MSG_MAP_EXP_SEM       "%s: %llu: expected ';' to terminate %s: %s"
1210 @ MSG_MAP_EXP_SEMLBKT   "%s: %llu: expected ';' or '{' following %s: %s"
1211 @ MSG_MAP_EXP_SEMRBKT   "%s: %llu: expected ';' or '}' to terminate %s: %s"
1212 @ MSG_MAP_EXP_SHTYPE    "%s: %llu: expected section type: %s"
1213 @ MSG_MAP_EXP_SYM       "%s: %llu: expected symbol name, symbol scope, \
1214                          or '*': %s"
1215 @ MSG_MAP_EXP_SYMEND    "%s: %llu: expected inherited version name, or \
1216                          terminator (';'): %s"
1217 @ MSG_MAP_EXP_SYMDELIM  "%s: %llu: expected one of ':', ';', or '{': %s"
1218 @ MSG_MAP_EXP_SYMFLAG   "%s: %llu: expected symbol flag (%s), or \
1219                          terminator (';', '}'): %s"
1220 @ MSG_MAP_EXP_SYMNAM    "%s: %llu: expected symbol name following %s: %s"
1221 @ MSG_MAP_EXP_SYMSCOPE  "%s: %llu: expected symbol scope (%s): %s"
1222 @ MSG_MAP_EXP_SYMTYPE   "%s: %llu: expected symbol type (%s): %s"
1223 @ MSG_MAP_EXP_VERSION   "%s: %llu: expected version name following %s: %s"
1224 @ MSG_MAP_BADEXTRA      "%s: %llu: unexpected text found following %s directive"
1225 @ MSG_MAP_VALUELIMIT    "%s: %llu: numeric value exceeds word size: %s"
1226 @ MSG_MAP_MALVALUE      "%s: %llu: malformed numeric value: %s"
1227 @ MSG_MAP_BADVALUETAIL  "%s: %llu: unexpected characters following numeric \
1228                          constant: %s"
1229 @ MSG_MAP_WSNEEDED      "%s: %llu: whitespace needed before token: %s"
1230 @ MSG_MAP_BADCHAR       "%s: %llu: unexpected text: %s"
1231 @ MSG_MAP_BADKWQUOTE    "%s: %llu: mapfile keywords should not be quoted: %s"
1232 @ MSG_MAP_CDIR_NOTBOL   "%s: %llu: mapfile control directive not at start of \
1233                          line: %s"
1234 @ MSG_MAP_NOATTR        "%s: %llu: %s specified no attributes (empty {})"
1235 @ MSG_MAP_NOVALUES      "%s: %llu: %s specified without values"
1236 @ MSG_MAP_INTERR        "<internal error>"
1237 @ MSG_MAP_ISORDVER      "%s: %llu: version 0 mapfile ?O flag and version 1 \
1238                          segment IS_ORDER attribute are mutually exclusive: %s"
1239 @ MSG_MAP_SYMATTR       "symbol attributes";

1241 # Mapfile Control Directives

1243 @ MSG_MAP_CDIR_BADVDIR  "%s: %llu: $mapfile_version directive must specify \
1244                          version 2 or higher: %d"
1245 @ MSG_MAP_CDIR_BADVER   "%s: %llu: unknown mapfile version: %d"
1246 @ MSG_MAP_CDIR_REPVER   "%s: %llu: $mapfile_version must be first directive \
1247                          in file"
1248 @ MSG_MAP_CDIR_REQARG   "%s: %llu: %s directive requires an argument"
1249 @ MSG_MAP_CDIR_REQNOARG "%s: %llu: %s directive does not accept arguments"
```



```
1250 @ MSG_MAP_CDIR_BAD      "%s: %llu: unrecognized mapfile control directive"
1251 @ MSG_MAP_CDIR_NOIF     "%s: %llu: %s directive used without opening $if"
1252 @ MSG_MAP_CDIR_ELSE     "%s: %llu: %s directive preceded by $else on line %d"
1253 @ MSG_MAP_CDIR_NOEND    "%s: %llu: EOF encountered without closing $endif \
1254                          for $if on line %d"
1255 @ MSG_MAP_CDIR_ERROR    "%s: %llu: error: %s"


1258 # Mapfile Conditional Expressions

1260 @ MSG_MAP_CEXP_TOKERR   "%s: %llu: syntax error in conditional expression at: %s
1261 @ MSG_MAP_CEXP_SEMERR   "%s: %llu: malformed conditional expression"
1262 @ MSG_MAP_CEXP_BADOPUSE "%s: %llu: invalid operator use in conditional \
1263                          expression"
1264 @ MSG_MAP_CEXP_UNBALPAR "%s: %llu: unbalanced parenthesis in conditional \
1265                          expression"
1266 @ MSG_MAP_BADCESC       "%s: %llu: unrecognized escape in double quoted \
1267                          token: \\%c\n"

1269 # Generic error diagnostic labels

1271 @ MSG_STR_NULL          "(null)"

1273 @ MSG_DBG_DFLT_FMT      "debug: "
1274 @ MSG_DBG_AOUT_FMT      "debug: a.out: "
1275 @ MSG_DBG_NAME_FMT      "debug: %s: "

1277 # -z assert-deflib strings

1279 @ MSG_ARG_ASSDEFLIB_MALFORMED   "library name malformed: %s"
1280 @ MSG_ARG_ASSDEFLIB_FOUND       "dynamic library found on default search path \
1281                                 (%s): lib%s.so"

1283 @ _END_


1286 # Software identification.  Note, the SGU strings is historic, and has
1287 # little relevance.  It is preserved as applications have used this
1288 # string to identify the Solaris link-editor.

1290 @ MSG_SGS_ID            "ld: Software Generation Utilities - \
1291                          Solaris Link Editors: "

1293 # The following strings represent reserved words, files, pathnames and symbols.
1294 # Reference to this strings is via the MSG_ORIG() macro, and thus no message
1295 # translation is required.

1297 @ MSG_DBG_FOPEN_MODE    "w"

1299 @ MSG_DBG_CLS32_FMT     "32: "
1300 @ MSG_DBG_CLS64_FMT     "64: "

1302 @ MSG_STR_PATHTOK       ";:"
1303 @ MSG_STR_AOUT          "a.out"

1305 @ MSG_STR_LIB_A         "%s/lib%s.a"
1306 @ MSG_STR_LIB_SO        "%s/lib%s.so"
1307 @ MSG_STR_PATH          "%s/%s"
1308 @ MSG_STR_STRNL         "%s\n"
1309 @ MSG_STR_NL            "\n"
1310 @ MSG_STR_CAPGROUPID    "CAP_GROUP_%d"

1312 @ MSG_STR_LD_DYNAMIC    "dynamic"
1313 @ MSG_STR_SYMBOLIC      "symbolic"
1314 @ MSG_STR_ELIMINATE     "eliminate"
1315 @ MSG_STR_LOCAL         "local"
```

```
@ MSG_STR_PROGBITS      "progbits"
@ MSG_STR_SYMTAB        "symtab"
@ MSG_STR_DYNSYM        "dynsym"
@ MSG_STR_REL           "rel"
@ MSG_STR_RELA          "rela"
@ MSG_STR_STRTAB        "strtab"
@ MSG_STR_HASH          "hash"
@ MSG_STR_LIB           "lib"
@ MSG_STR_NOTE          "note"
@ MSG_STR_NOBITS        "nobits"
@ MSG_STR_HWCAP_1       "hwcap_1"
@ MSG_STR_SFCAP_1       "sfcap_1"
@ MSG_STR_SOEXT         ".so"

@ MSG_STR_OPTIONS       "3:6:abc:d:e:f:h:il:mo:p:rstu:z:B:CD:F:GI:L:M:N:P:Q:R:\
                         S:VW:Y:?"

# Argument processing strings

@ MSG_ARG_3             "-3"
@ MSG_ARG_6             "-6"
@ MSG_ARG_A             "-a"
@ MSG_ARG_B             "-b"
@ MSG_ARG_CB            "-B"
@ MSG_ARG_BDIRECT       "-Bdirect"
@ MSG_ARG_BDYNAMIC      "-Bdynamic"
@ MSG_ARG_BELIMINATE    "-Beliminate"
@ MSG_ARG_BGROUP        "-Bgroup"
@ MSG_ARG_BLOCAL        "-Blocal"
@ MSG_ARG_BNODIRECT     "-Bnodirect"
@ MSG_ARG_BSYMBOLIC     "-Bsymbolic"
@ MSG_ARG_BTRANSLATOR   "-Btranslator"
@ MSG_ARG_C             "-c"
@ MSG_ARG_D             "-d"
@ MSG_ARG_DY            "-dy"
@ MSG_ARG_CI            "-I"
@ MSG_ARG_CN            "-N"
@ MSG_ARG_P             "-p"
@ MSG_ARG_CP            "-P"
@ MSG_ARG_CQ            "-Q"
@ MSG_ARG_CY            "-Y"
@ MSG_ARG_CYL           "-YL"
@ MSG_ARG_CYP           "-YP"
@ MSG_ARG_CYU           "-YU"
@ MSG_ARG_Z             "-z"
@ MSG_ARG_ZDEFNODEF     "-z[defs|nodefs]"
@ MSG_ARG_ZGUIDE        "-zguidance"
@ MSG_ARG_ZNODEF        "-znodefs"
@ MSG_ARG_ZNOINTERP     "-znointerp"
@ MSG_ARG_ZRELAXRELOC   "-zrelaxreloc"
@ MSG_ARG_ZNORELAXRELOC "-znorelaxreloc"
@ MSG_ARG_ZTEXT         "-ztext"
@ MSG_ARG_ZTEXTOFF      "-ztextoff"
@ MSG_ARG_ZTEXTWARN     "-ztextwarn"
@ MSG_ARG_ZTEXTALL      "-z[text|textwarn|textoff]"
@ MSG_ARG_ZLOADFLTR     "-zloadfltr"
@ MSG_ARG_ZCOMBRELOC    "-zcombreloc"
@ MSG_ARG_ZSYMBOLCAP    "-zsymbolcap"
@ MSG_ARG_ZFATWNOFATW   "-z[fatal-warnings|nofatalwarnings]"

@ MSG_ARG_ABSEXEC       "absexec"
@ MSG_ARG_ALTEXEC64     "altexec64"
@ MSG_ARG_NOCOMPSTRTAB  "nocompstrtab"
@ MSG_ARG_GROUPPERM     "groupperm"
@ MSG_ARG_NOGROUPPERM   "nogroupperm"
@ MSG_ARG_LAZYLOAD      "lazyload"
```



```
@ MSG_ARG_NOLAZYLOAD    "nolazyload"
@ MSG_ARG_INTERPOSE     "interpose"
@ MSG_ARG_DIRECT        "direct"
@ MSG_ARG_NODIRECT      "nodirect"
@ MSG_ARG_IGNORE        "ignore"
@ MSG_ARG_RECORD        "record"
@ MSG_ARG_INITFIRST     "initfirst"
@ MSG_ARG_INITARRAY     "initarray="
@ MSG_ARG_FINIARRAY     "finiarray="
@ MSG_ARG_PREINITARRAY  "preinitarray="
@ MSG_ARG_RTLDINFO      "rtldinfo="
@ MSG_ARG_DTRACE        "dtrace="
@ MSG_ARG_TRANSLATOR    "translator"
@ MSG_ARG_NOOPEN        "nodlopen"
@ MSG_ARG_NOW           "now"
@ MSG_ARG_ORIGIN        "origin"
@ MSG_ARG_DEFS          "defs"
@ MSG_ARG_NODEFS        "nodefs"
@ MSG_ARG_NODUMP        "nodump"
@ MSG_ARG_NOVERSION     "noversion"
@ MSG_ARG_TEXT          "text"
@ MSG_ARG_TEXTOFF       "textoff"
@ MSG_ARG_TEXTWARN      "textwarn"
@ MSG_ARG_MULDEFS       "muldefs"
@ MSG_ARG_NODELETE      "nodelete"
@ MSG_ARG_NOINTERP      "nointerp"
@ MSG_ARG_NOPARTIAL     "nopartial"
@ MSG_ARG_NORELOC       "noreloc"
@ MSG_ARG_REDLOCSYM     "redlocsym"
@ MSG_ARG_VERBOSE       "verbose"
@ MSG_ARG_WEAKEXT       "weakextract"
@ MSG_ARG_LOADFLTR      "loadfltr"
@ MSG_ARG_ALLEXTRT      "allextract"
@ MSG_ARG_DFLEXTRT      "defaultextract"
@ MSG_ARG_COMBRELOC     "combreloc"
@ MSG_ARG_NOCOMBRELOC   "nocombreloc"
@ MSG_ARG_NODEFAULTLIB  "nodefaultlib"
@ MSG_ARG_ENDFILTEE     "endfiltee"
@ MSG_ARG_LD32          "ld32="
@ MSG_ARG_LD64          "ld64="
@ MSG_ARG_RESCAN        "rescan"
@ MSG_ARG_RESCAN_NOW    "rescan-now"
@ MSG_ARG_RESCAN_START  "rescan-start"
@ MSG_ARG_RESCAN_END    "rescan-end"
@ MSG_ARG_GUIDE         "guidance"
@ MSG_ARG_NOLDYNSYM     "noldynsym"
@ MSG_ARG_RELAXRELOC    "relaxreloc"
@ MSG_ARG_NORELAXRELOC  "norelaxreloc"
@ MSG_ARG_NOSIGHANDLER  "nosighandler"
@ MSG_ARG_GLOBAUDIT     "globalaudit"
@ MSG_ARG_TARGET        "target="
@ MSG_ARG_WRAP          "wrap="
@ MSG_ARG_FATWARN       "fatal-warnings"
@ MSG_ARG_NOFATWARN     "nofatal-warnings"
@ MSG_ARG_HELP          "help"
@ MSG_ARG_GROUP         "group"
@ MSG_ARG_REDUCE        "reduce"
@ MSG_ARG_STATIC        "static"
@ MSG_ARG_SYMBOLCAP     "symbolcap"
@ MSG_ARG_DEFERRED      "deferred"
@ MSG_ARG_NODEFERRED    "nodeferred"
@ MSG_ARG_ASSDEFLIB     "assert-deflib"

@ MSG_ARG_LCOM          "L,"
@ MSG_ARG_PCOM          "P,"
@ MSG_ARG_UCOM          "U,"
```

```
1449 @ MSG_ARG_T_RPATH       "rpath"
1450 @ MSG_ARG_T_SHARED      "shared"
1451 @ MSG_ARG_T_SONAME      "soname"
1452 @ MSG_ARG_T_WL          "l,-"

1454 @ MSG_ARG_T_AUXFLTR     "-auxiliary"
1455 @ MSG_ARG_T_MULDEFS     "-allow-multiple-definition"
1456 @ MSG_ARG_T_INTERP      "-dynamic-linker"
1457 @ MSG_ARG_T_ENDGROUP    "-end-group"
1458 @ MSG_ARG_T_ENTRY       "-entry"
1459 @ MSG_ARG_T_STDFLTR     "-filter"
1460 @ MSG_ARG_T_FATWARN     "-fatal-warnings"
1461 @ MSG_ARG_T_NOFATWARN   "-no-fatal-warnings"
1462 @ MSG_ARG_T_HELP        "-help"
1463 @ MSG_ARG_T_LIBRARY     "-library"
1464 @ MSG_ARG_T_LIBPATH     "-library-path"
1465 @ MSG_ARG_T_NOUNDEF     "-no-undefined"
1466 @ MSG_ARG_T_NOWHOLEARC  "-no-whole-archive"
1467 @ MSG_ARG_T_OUTPUT      "-output"
1468 @ MSG_ARG_T_RELOCATABLE "-relocatable"
1469 @ MSG_ARG_T_STARTGROUP  "-start-group"
1470 @ MSG_ARG_T_STRIP       "-strip-all"
1471 @ MSG_ARG_T_UNDEF       "-undefined"
1472 @ MSG_ARG_T_VERSION     "-version"
1473 @ MSG_ARG_T_WHOLEARC    "-whole-archive"
1474 @ MSG_ARG_T_WRAP        "-wrap"
1475 @ MSG_ARG_T_OPAR        "("
1476 @ MSG_ARG_T_CPAR        ")"

1478 # -z guidance=item strings
1479 @ MSG_ARG_GUIDE_DELIM           ",: \t"
1480 @ MSG_ARG_GUIDE_NO_ALL          "noall"
1481 @ MSG_ARG_GUIDE_NO_DEFS         "nodefs"
1482 @ MSG_ARG_GUIDE_NO_DIRECT       "nodirect"
1483 @ MSG_ARG_GUIDE_NO_LAZYLOAD     "nolazyload"
1484 @ MSG_ARG_GUIDE_NO_MAPFILE      "nomapfile"
1485 @ MSG_ARG_GUIDE_NO_TEXT         "notext"
1486 @ MSG_ARG_GUIDE_NO_UNUSED       "nounused"

1488 # Environment variable strings

1490 @ MSG_LD_RUN_PATH       "LD_RUN_PATH"
1491 @ MSG_LD_LIBPATH_32     "LD_LIBRARY_PATH_32"
1492 @ MSG_LD_LIBPATH_64     "LD_LIBRARY_PATH_64"
1493 @ MSG_LD_LIBPATH        "LD_LIBRARY_PATH"

1495 @ MSG_LD_NOVERSION_32   "LD_NOVERSION_32"
1496 @ MSG_LD_NOVERSION_64   "LD_NOVERSION_64"
1497 @ MSG_LD_NOVERSION      "LD_NOVERSION"

1499 @ MSG_SGS_SUPPORT_32    "SGS_SUPPORT_32"
1500 @ MSG_SGS_SUPPORT_64    "SGS_SUPPORT_64"
1501 @ MSG_SGS_SUPPORT       "SGS_SUPPORT"


1504 # Symbol names

1506 @ MSG_SYM_LIBVER_U      "_lib_version"


1509 # Mapfile tokens

1511 @ MSG_MAP_LOAD          "load"
1512 @ MSG_MAP_NOTE          "note"
1513 @ MSG_MAP_NULL          "null"
```



```
1514 @ MSG_MAP_STACK         "stack"
1515 @ MSG_MAP_ADDVERS       "addvers"
1516 @ MSG_MAP_FUNCTION      "function"
1517 @ MSG_MAP_DATA          "data"
1518 @ MSG_MAP_COMMON        "common"
1519 @ MSG_MAP_PARENT        "parent"
1520 @ MSG_MAP_EXTERN        "extern"
1521 @ MSG_MAP_DIRECT        "direct"
1522 @ MSG_MAP_NODIRECT      "nodirect"
1523 @ MSG_MAP_FILTER        "filter"
1524 @ MSG_MAP_AUXILIARY     "auxiliary"
1525 @ MSG_MAP_OVERRIDE      "override"
1526 @ MSG_MAP_INTERPOSE     "interpose"
1527 @ MSG_MAP_DYNSORT       "dynsort"
1528 @ MSG_MAP_NODYNSORT     "nodynsort"

1530 @ MSG_MAPKW_ALIGN               "ALIGN"
1531 @ MSG_MAPKW_ALLOC               "ALLOC"
1532 @ MSG_MAPKW_ALLOW               "ALLOW"
1533 @ MSG_MAPKW_AMD64_LARGE         "AMD64_LARGE"
1534 @ MSG_MAPKW_ASSIGN_SECTION      "ASSIGN_SECTION"
1535 @ MSG_MAPKW_AUX                 "AUXILIARY"
1536 @ MSG_MAPKW_CAPABILITY          "CAPABILITY"
1537 @ MSG_MAPKW_COMMON              "COMMON"
1538 @ MSG_MAPKW_DATA                "DATA"
1539 @ MSG_MAPKW_DEFAULT             "DEFAULT"
1540 @ MSG_MAPKW_DEPEND_VERSIONS     "DEPEND_VERSIONS"
1541 @ MSG_MAPKW_DIRECT              "DIRECT"
1542 @ MSG_MAPKW_DISABLE             "DISABLE"
1543 @ MSG_MAPKW_DYNSORT             "DYNSORT"
1544 @ MSG_MAPKW_ELIMINATE           "ELIMINATE"
1545 @ MSG_MAPKW_EXECUTE             "EXECUTE"
1546 @ MSG_MAPKW_EXPORTED            "EXPORTED"
1547 @ MSG_MAPKW_EXTERN              "EXTERN"
1548 @ MSG_MAPKW_FILTER              "FILTER"
1549 @ MSG_MAPKW_FILE_BASENAME       "FILE_BASENAME"
1550 @ MSG_MAPKW_FILE_PATH           "FILE_PATH"
1551 @ MSG_MAPKW_FILE_OBJNAME        "FILE_OBJNAME"
1552 @ MSG_MAPKW_FUNCTION            "FUNCTION"
1553 @ MSG_MAPKW_FLAGS               "FLAGS"
1554 @ MSG_MAPKW_GLOBAL              "GLOBAL"
1555 @ MSG_MAPKW_INTERPOSE           "INTERPOSE"
1556 @ MSG_MAPKW_HIDDEN              "HIDDEN"
1557 @ MSG_MAPKW_HDR_NOALLOC         "HDR_NOALLOC"
1558 @ MSG_MAPKW_HW                  "HW"
1559 @ MSG_MAPKW_HW_1                "HW_1"
1560 @ MSG_MAPKW_HW_2                "HW_2"
1561 @ MSG_MAPKW_IS_NAME             "IS_NAME"
1562 @ MSG_MAPKW_IS_ORDER            "IS_ORDER"
1563 @ MSG_MAPKW_LOAD_SEGMENT        "LOAD_SEGMENT"
1564 @ MSG_MAPKW_LOCAL               "LOCAL"
1565 @ MSG_MAPKW_MACHINE             "MACHINE"
1566 @ MSG_MAPKW_MAX_SIZE            "MAX_SIZE"
1567 @ MSG_MAPKW_NOHDR               "NOHDR"
1568 @ MSG_MAPKW_NODIRECT            "NODIRECT"
1569 @ MSG_MAPKW_NODYNSORT           "NODYNSORT"
1570 @ MSG_MAPKW_NOTE_SEGMENT        "NOTE_SEGMENT"
1571 @ MSG_MAPKW_NULL_SEGMENT        "NULL_SEGMENT"
1572 @ MSG_MAPKW_OS_ORDER            "OS_ORDER"
1573 @ MSG_MAPKW_PADDR               "PADDR"
1574 @ MSG_MAPKW_PARENT              "PARENT"
1575 @ MSG_MAPKW_PHDR_ADD_NULL       "PHDR_ADD_NULL"
1576 @ MSG_MAPKW_PLATFORM            "PLATFORM"
1577 @ MSG_MAPKW_PROTECTED           "PROTECTED"
1578 @ MSG_MAPKW_READ                "READ"
1579 @ MSG_MAPKW_ROUND               "ROUND"
```

```
1580 @ MSG_MAPKW_REQUIRE              "REQUIRE"
1581 @ MSG_MAPKW_SEGMENT_ORDER       "SEGMENT_ORDER"
1582 @ MSG_MAPKW_SF                  "SF"
1583 @ MSG_MAPKW_SF_1                "SF_1"
1584 @ MSG_MAPKW_SINGLETON           "SINGLETON"
1585 @ MSG_MAPKW_SIZE                "SIZE"
1586 @ MSG_MAPKW_SIZE_SYMBOL         "SIZE_SYMBOL"
1587 @ MSG_MAPKW_STACK               "STACK"
1588 @ MSG_MAPKW_SYMBOL_SCOPE        "SYMBOL_SCOPE"
1589 @ MSG_MAPKW_SYMBOL_VERSION      "SYMBOL_VERSION"
1590 @ MSG_MAPKW_SYMBOLIC            "SYMBOLIC"
1591 @ MSG_MAPKW_TYPE                "TYPE"
1592 @ MSG_MAPKW_VADDR               "VADDR"
1593 @ MSG_MAPKW_VALUE               "VALUE"
1594 @ MSG_MAPKW_WRITE               "WRITE"


1597 @ MSG_STR_DTRACE        "PT_SUNWDTRACE"
```

```
**********************************************************
   23955 Mon Mar 23 21:41:49 2015
new/usr/src/cmd/sgs/libld/common/unwind.c
5688 ELF tools need to be more careful with dwarf data
**********************************************************
______unchanged_portion_omitted_

 482 uintptr_t
 483 ld_unwind_populate_hdr(Ofl_desc *ofl)
 484 {
 485         uchar_t         *hdrdata;
 486         uint_t          *binarytable;
 487         uint_t          hdroff;
 488         Aliste          idx;
 489         Addr            hdraddr;
 490         Os_desc         *hdrosp;
 491         Os_desc         *osp;
 492         Os_desc         *first_unwind;
 493         uint_t          fde_count;
 494         uint_t          *uint_ptr;
 495         int             bswap = (ofl->ofl_flags1 & FLG_OF1_ENCDIFF) != 0;

 497         /*
 498          * Are we building the unwind hdr?
 499          */
 500         if ((hdrosp = ofl->ofl_unwindhdr) == 0)
 501                 return (1);

 503         hdrdata = hdrosp->os_outdata->d_buf;
 504         hdraddr = hdrosp->os_shdr->sh_addr;
 505         hdroff = 0;

 507         /*
 508          * version == 1
 509          */
 510         hdrdata[hdroff++] = 1;
 511         /*
 512          * The encodings are:
 513          *
 514          *  eh_frameptr_enc     sdata4 | pcrel
 515          *  fde_count_enc       udata4
 516          *  table_enc           sdata4 | datarel
 517          */
 518         hdrdata[hdroff++] = DW_EH_PE_sdata4 | DW_EH_PE_pcrel;
 519         hdrdata[hdroff++] = DW_EH_PE_udata4;
 520         hdrdata[hdroff++] = DW_EH_PE_sdata4 | DW_EH_PE_datarel;

 522         /*
 523          *      Header Offsets
 524          *      ----------------------------------
 525          *      byte        version             +1
 526          *      byte        eh_frame_ptr_enc    +1
 527          *      byte        fde_count_enc       +1
 528          *      byte        table_enc           +1
 529          *      4 bytes     eh_frame_ptr        +4
 530          *      4 bytes     fde_count           +4
 531          */
 532         /* LINTED */
 533         binarytable =  (uint_t *)(hdrdata + 12);
 534         first_unwind = 0;
 535         fde_count = 0;

 537         for (APLIST_TRAVERSE(ofl->ofl_unwind, idx, osp)) {
 538                 uchar_t         *data;
 539                 size_t          size;
 540                 uint64_t        off = 0, ujunk;
```



```
 541                 int64_t         sjunk;
 540                 uint64_t        off = 0;
 542                 uint_t          cieRflag = 0, ciePflag = 0;
 543                 Shdr            *shdr;

 545                 /*
 546                  * remember first UNWIND section to
 547                  * point to in the frame_ptr entry.
 548                  */
 549                 if (first_unwind == 0)
 550                         first_unwind = osp;

 552                 data = osp->os_outdata->d_buf;
 553                 shdr = osp->os_shdr;
 554                 size = shdr->sh_size;

 556                 while (off < size) {
 557                         uint_t      length, id;
 558                         uint64_t    ndx = 0;

 560                         /*
 561                          * Extract length in lsb format.  A zero length
 562                          * indicates that this CIE is a terminator and that
 563                          * processing of unwind information is complete.
 564                          */
 565                         length = extract_uint(data + off, &ndx, bswap);
 566                         if (length == 0)
 567                                 goto done;

 569                         /*
 570                          * Extract CIE id in lsb format.
 571                          */
 572                         id = extract_uint(data + off, &ndx, bswap);

 574                         /*
 575                          * A CIE record has a id of '0'; otherwise
 576                          * this is a FDE entry and the 'id' is the
 577                          * CIE pointer.
 578                          */
 579                         if (id == 0) {
 580                                 char    *cieaugstr;
 581                                 uint_t  cieaugndx;
 582                                 uint_t  cieversion;

 584                                 ciePflag = 0;
 585                                 cieRflag = 0;
 586                                 /*
 587                                  * We need to drill through the CIE
 588                                  * to find the Rflag.  It's the Rflag
 589                                  * which describes how the FDE code-pointers
 590                                  * are encoded.
 591                                  */

 593                                 cieversion = data[off + ndx];
 594                                 ndx += 1;

 596                                 /*
 597                                  * augstr
 598                                  */
 599                                 cieaugstr = (char *)(&data[off + ndx]);
 600                                 ndx += strlen(cieaugstr) + 1;

 602                                 /*
 603                                  * calign & dalign
 604                                  */
 605                                 if (uleb_extract(&data[off], &ndx,
```

```
606                                     size - off, &ujunk) == DW_OVERFLOW) {
607                                         ld_eprintf(ofl, ERR_FATAL,
608                                             MSG_INTL(MSG_SCN_DWFOVRFLW),
609                                             ofl->ofl_name,
610                                             osp->os_name);
611                                         return (S_ERROR);
612                                 }

614                                 if (sleb_extract(&data[off], &ndx,
615                                     size - off, &sjunk) == DW_OVERFLOW) {
616                                         ld_eprintf(ofl, ERR_FATAL,
617                                             MSG_INTL(MSG_SCN_DWFOVRFLW),
618                                             ofl->ofl_name,
619                                             osp->os_name);
620                                         return (S_ERROR);
621                                 }
604                                 (void) uleb_extract(&data[off], &ndx);
605                                 (void) sleb_extract(&data[off], &ndx);

623                                 /*
624                                  * retreg
625                                  */
626                                 if (cieversion == 1) {
610                                 if (cieversion == 1)
627                                         ndx++;
628                                 } else {
629                                         if (uleb_extract(&data[off], &ndx,
630                                             size - off, &ujunk) ==
631                                             DW_OVERFLOW) {
632                                                 ld_eprintf(ofl, ERR_FATAL,
633                                                     MSG_INTL(MSG_SCN_DWFOVRFLW),
634                                                     ofl->ofl_name,
635                                                     osp->os_name);
636                                                 return (S_ERROR);
637                                         }
638                                 }
612                                 else
613                                         (void) uleb_extract(&data[off], &ndx);
639                                 /*
640                                  * we walk through the augmentation
641                                  * section now looking for the Rflag
642                                  */
643                                 for (cieaugndx = 0; cieaugstr[cieaugndx];
644                                     cieaugndx++) {
645                                         /* BEGIN CSTYLED */
646                                         switch (cieaugstr[cieaugndx]) {
647                                         case 'z':
648                                             /* size */
649                                             if (uleb_extract(&data[off],
650                                                 &ndx, size - off, &ujunk) ==
651                                                 DW_OVERFLOW) {
652                                                 ld_eprintf(ofl, ERR_FATAL,
653                                                     MSG_INTL(MSG_SCN_DWFOVRFLW),
654                                                     ofl->ofl_name,
655                                                     osp->os_name);
656                                                 return (S_ERROR);
657                                             }
624                                             (void) uleb_extract(&data[off],
625                                                 &ndx);
658                                             break;
659                                         case 'P':
660                                             /* personality */
661                                             ciePflag = data[off + ndx];
662                                             ndx++;
663                                                 /*
664                                                  * Just need to extract the
```



```
665                                                  * value to move on to the next
666                                                  * field.
667                                                  */
668                                             switch (dwarf_ehe_extract(
669                                                 &data[off], size - off,
670                                                 &ndx, &ujunk, ciePflag,
636                                             (void) dwarf_ehe_extract(
637                                                 &data[off],
638                                                 &ndx, ciePflag,
671                                                 ofl->ofl_dehdr->e_ident, B_FALSE
672                                                 shdr->sh_addr, off + ndx, 0)) {
673                                             case DW_OVERFLOW:
674                                                 ld_eprintf(ofl, ERR_FATAL,
675                                                     MSG_INTL(MSG_SCN_DWFOVRFLW),
676                                                     ofl->ofl_name,
677                                                     osp->os_name);
678                                                 return (S_ERROR);
679                                             case DW_BAD_ENCODING:
680                                                 ld_eprintf(ofl, ERR_FATAL,
681                                                     MSG_INTL(MSG_SCN_DWFBADENC),
682                                                     ofl->ofl_name,
683                                                     osp->os_name, ciePflag);
684                                                 return (S_ERROR);
685                                             case DW_SUCCESS:
686                                                 break;
687                                             }
640                                             shdr->sh_addr, off + ndx, 0);
688                                             break;
689                                         case 'R':
690                                             /* code encoding */
691                                             cieRflag = data[off + ndx];
692                                             ndx++;
693                                             break;
694                                         case 'L':
695                                             /* lsda encoding */
696                                             ndx++;
697                                             break;
698                                         }
699                                         /* END CSTYLED */
700                                 }
701                         } else {
702                                 uint_t      bintabndx;
703                                 uint64_t    initloc;
704                                 uint64_t    fdeaddr;
705                                 uint64_t    gotaddr = 0;

707                                 if (ofl->ofl_osgot != NULL)
708                                         gotaddr =
709                                             ofl->ofl_osgot->os_shdr->sh_addr;

711                                 switch (dwarf_ehe_extract(&data[off],
712                                     size - off, &ndx, &initloc, cieRflag,
713                                     ofl->ofl_dehdr->e_ident, B_FALSE,
714                                     shdr->sh_addr, off + ndx, gotaddr)) {
715                                 case DW_OVERFLOW:
716                                         ld_eprintf(ofl, ERR_FATAL,
717                                             MSG_INTL(MSG_SCN_DWFOVRFLW),
718                                             ofl->ofl_name,
719                                             osp->os_name);
720                                         return (S_ERROR);
721                                 case DW_BAD_ENCODING:
722                                         ld_eprintf(ofl, ERR_FATAL,
723                                             MSG_INTL(MSG_SCN_DWFBADENC),
724                                             ofl->ofl_name,
725                                             osp->os_name, cieRflag);
726                                         return (S_ERROR);
```

```
727                                 case DW_SUCCESS:
728                                         break;
729                                 }
664                                 initloc = dwarf_ehe_extract(&data[off],
665                                     &ndx, cieRflag, ofl->ofl_dehdr->e_ident,
666                                     B_FALSE,
667                                     shdr->sh_addr, off + ndx,
668                                     gotaddr);

731                                 /*
732                                  * Ignore FDEs with initloc set to 0.
733                                  * initloc will not be 0 unless this FDE was
734                                  * abandoned due to GNU linkonce processing.
735                                  * The 0 value occurs because we don't resolve
736                                  * sloppy relocations for unwind header target
737                                  * sections.
738                                  */
739                                 if (initloc != 0) {
740                                         bintabndx = fde_count * 2;
741                                         fde_count++;

743                                         /*
744                                          * FDEaddr is adjusted
745                                          * to account for the length & id which
746                                          * have already been consumed.
747                                          */
748                                         fdeaddr = shdr->sh_addr + off;

750                                         binarytable[bintabndx] =
751                                             (uint_t)(initloc - hdraddr);
752                                         binarytable[bintabndx + 1] =
753                                             (uint_t)(fdeaddr - hdraddr);
754                                 }
755                         }

757                         /*
758                          * the length does not include the length
759                          * itself - so account for that too.
760                          */
761                         off += length + 4;
762                 }
763         }

765 done:
766         /*
767          * Do a quicksort on the binary table. If this is a cross
768          * link from a system with the opposite byte order, xlate
769          * the resulting values into LSB order.
770          */
771         framehdr_addr = hdraddr;
772         qsort((void *)binarytable, (size_t)fde_count,
773             (size_t)(sizeof (uint_t) * 2), bintabcompare);
774         if (bswap) {
775                 uint_t  *btable = binarytable;
776                 uint_t  cnt;

778                 for (cnt = fde_count * 2; cnt-- > 0; btable++)
779                         *btable = ld_bswap_Word(*btable);
780         }

782         /*
783          * Fill in:
784          *      first_frame_ptr
785          *      fde_count
786          */
787         hdroff = 4;
```



```
788         /* LINTED */
789         uint_ptr = (uint_t *)(&hdrdata[hdroff]);
790         *uint_ptr = first_unwind->os_shdr->sh_addr -
791             (hdrosp->os_shdr->sh_addr + hdroff);
792         if (bswap)
793                 *uint_ptr = ld_bswap_Word(*uint_ptr);

795         hdroff += 4;
796         /* LINTED */
797         uint_ptr = (uint_t *)&hdrdata[hdroff];
798         *uint_ptr = fde_count;
799         if (bswap)
800                 *uint_ptr = ld_bswap_Word(*uint_ptr);

802         /*
803          * If relaxed relocations are active, then there is a chance
804          * that we didn't use all the space reserved for this section.
805          * For details, see the note at head of ld_unwind_make_hdr() above.
806          *
807          * Find the PT_SUNW_UNWIND program header, and change the size values
808          * to the size of the subset of the section that was actually used.
809          */
810         if (ofl->ofl_flags1 & FLG_OF1_RLXREL) {
811                 Word    phnum = ofl->ofl_nehdr->e_phnum;
812                 Phdr    *phdr = ofl->ofl_phdr;

814                 for (; phnum-- > 0; phdr++) {
815                         if (phdr->p_type == PT_SUNW_UNWIND) {
816                                 phdr->p_memsz = 12 + (8 * fde_count);
817                                 phdr->p_filesz = phdr->p_memsz;
818                                 break;
819                         }
820                 }
821         }

823         return (1);
824 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    8641 Mon Mar 23 21:41:49 2015
new/usr/src/cmd/sgs/tools/common/leb128.c
5688 ELF tools need to be more careful with dwarf data
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <stdio.h>
#include <dwarf.h>
#include <sys/types.h>
#include <sys/elf.h>

/*
 * Little Endian Base 128 (LEB128) numbers.
 * ----------------------------------------
 *
 * LEB128 is a scheme for encoding integers densely that exploits the
 * assumption that most integers are small in magnitude. (This encoding
 * is equally suitable whether the target machine architecture represents
 * data in big-endian or little- endian
 *
 * Unsigned LEB128 numbers are encoded as follows: start at the low order
 * end of an unsigned integer and chop it into 7-bit chunks. Place each
 * chunk into the low order 7 bits of a byte. Typically, several of the
 * high order bytes will be zero; discard them. Emit the remaining bytes in
 * a stream, starting with the low order byte; set the high order bit on
 * each byte except the last emitted byte. The high bit of zero on the last
 * byte indicates to the decoder that it has encountered the last byte.
 * The integer zero is a special case, consisting of a single zero byte.
 *
 * Signed, 2s complement LEB128 numbers are encoded in a similar except
 * that the criterion for discarding high order bytes is not whether they
 * are zero, but whether they consist entirely of sign extension bits.
 * Consider the 32-bit integer -2. The three high level bytes of the number
 * are sign extension, thus LEB128 would represent it as a single byte
 * containing the low order 7 bits, with the high order bit cleared to
 * indicate the end of the byte stream.
 *
 * Note that there is nothing within the LEB128 representation that
 * indicates whether an encoded number is signed or unsigned. The decoder
 * must know what type of number to expect.
 *
```



```
 * DWARF Exception Header Encoding
 * -------------------------------
 *
 * The DWARF Exception Header Encoding is used to describe the type of data
 * used in the .eh_frame_hdr section. The upper 4 bits indicate how the
 * value is to be applied. The lower 4 bits indicate the format of the data.
 *
 * DWARF Exception Header value format
 *
 * Name         Value Meaning
 * DW_EH_PE_omit            0xff No value is present.
 * DW_EH_PE_absptr          0x00 Value is a void*
 * DW_EH_PE_uleb128         0x01 Unsigned value is encoded using the
 *                               Little Endian Base 128 (LEB128)
 * DW_EH_PE_udata2          0x02 A 2 bytes unsigned value.
 * DW_EH_PE_udata4          0x03 A 4 bytes unsigned value.
 * DW_EH_PE_udata8          0x04 An 8 bytes unsigned value.
 * DW_EH_PE_signed          0x08 bit on for all signed encodings
 * DW_EH_PE_sleb128         0x09 Signed value is encoded using the
 *                               Little Endian Base 128 (LEB128)
 * DW_EH_PE_sdata2          0x0A A 2 bytes signed value.
 * DW_EH_PE_sdata4          0x0B A 4 bytes signed value.
 * DW_EH_PE_sdata8          0x0C An 8 bytes signed value.
 *
 * DWARF Exception Header application
 *
 * Name     Value Meaning
 * DW_EH_PE_absptr          0x00 Value is used with no modification.
 * DW_EH_PE_pcrel           0x10 Value is reletive to the location of itself
 * DW_EH_PE_textrel         0x20
 * DW_EH_PE_datarel         0x30 Value is reletive to the beginning of the
 *                               eh_frame_hdr segment ( segment type
 *                               PT_GNU_EH_FRAME )
 * DW_EH_PE_funcrel         0x40
 * DW_EH_PE_aligned         0x50 value is an aligned void*
 * DW_EH_PE_indirect        0x80 bit to signal indirection after relocation
 * DW_EH_PE_omit            0xff No value is present.
 *
 */

dwarf_error_t
uleb_extract(unsigned char *data, uint64_t *dotp, size_t len, uint64_t *ret)
uint64_t
uleb_extract(unsigned char *data, uint64_t *dotp)
{
        uint64_t        dot = *dotp;
        uint64_t        res = 0;
        int             more = 1;
        int             shift = 0;
        int             val;

        data += dot;

        while (more) {
                if (dot > len)
                        return (DW_OVERFLOW);

#endif /* ! codereview */
                /*
                 * Pull off lower 7 bits
                 */
                val = (*data) & 0x7f;

                /*
                 * Add prepend value to head of number.
                 */
```

```
126                 res = res | (val << shift);
128                 /*
129                  * Increment shift & dot pointer
130                  */
131                 shift += 7;
132                 dot++;
134                 /*
135                  * Check to see if hi bit is set - if not, this
136                  * is the last byte.
137                  */
138                 more = ((*data++) & 0x80) >> 7;
139         }
140         *dotp = dot;
141         *ret = res;
142         return (DW_SUCCESS);
114         return (res);
143 }
145 dwarf_error_t
146 sleb_extract(unsigned char *data, uint64_t *dotp, size_t len, int64_t *ret)
117 int64_t
118 sleb_extract(unsigned char *data, uint64_t *dotp)
147 {
148         uint64_t        dot = *dotp;
149         int64_t         res = 0;
150         int             more = 1;
151         int             shift = 0;
152         int             val;
154         data += dot;
156         while (more) {
157                 if (dot > len)
158                         return (DW_OVERFLOW);
160 #endif /* ! codereview */
161                 /*
162                  * Pull off lower 7 bits
163                  */
164                 val = (*data) & 0x7f;
166                 /*
167                  * Add prepend value to head of number.
168                  */
169                 res = res | (val << shift);
171                 /*
172                  * Increment shift & dot pointer
173                  */
174                 shift += 7;
175                 dot++;
177                 /*
178                  * Check to see if hi bit is set - if not, this
179                  * is the last byte.
180                  */
181                 more = ((*data++) & 0x80) >> 7;
182         }
183         *dotp = dot;
185         /*
186          * Make sure value is properly sign extended.
187          */
188         res = (res << (64 - shift)) >> (64 - shift);
```



```
189         *ret = res;
190         return (DW_SUCCESS);
130         return (res);
191 }
193 /*
194  * Extract a DWARF encoded datum
195  *
196  * entry:
197  *      data - Base of data buffer containing encoded bytes
198  *      dotp - Address of variable containing index within data
199  *             at which the desired datum starts.
200  *      ehe_flags - DWARF encoding
201  *      eident - ELF header e_ident[] array for object being processed
202  *      frame_hdr - Boolean, true if we're extracting from .eh_frame_hdr
203  *      sh_base - Base address of ELF section containing desired datum
204  *      sh_offset - Offset relative to sh_base of desired datum.
205  *      dbase - The base address to which DW_EH_PE_datarel is relative
206  *              (if frame_hdr is false)
207  */
208 dwarf_error_t
209 dwarf_ehe_extract(unsigned char *data, size_t len, uint64_t *dotp,
210     uint64_t *ret, uint_t ehe_flags, unsigned char *eident,
211     boolean_t frame_hdr, uint64_t sh_base, uint64_t sh_offset,
212     uint64_t dbase)
148 uint64_t
149 dwarf_ehe_extract(unsigned char *data, uint64_t *dotp, uint_t ehe_flags,
150     unsigned char *eident, boolean_t frame_hdr, uint64_t sh_base,
151     uint64_t sh_offset, uint64_t dbase)
213 {
214         uint64_t    dot = *dotp;
215         uint_t      lsb;
216         uint_t      wordsize;
217         uint_t      fsize;
218         uint64_t    result;
220         if (eident[EI_DATA] == ELFDATA2LSB)
221                 lsb = 1;
222         else
223                 lsb = 0;
225         if (eident[EI_CLASS] == ELFCLASS64)
226                 wordsize = 8;
227         else
228                 wordsize = 4;
230         switch (ehe_flags & 0x0f) {
231         case DW_EH_PE_omit:
232                 *ret = 0;
233                 return (DW_SUCCESS);
171                 return (0);
234         case DW_EH_PE_absptr:
235                 fsize = wordsize;
236                 break;
237         case DW_EH_PE_udata8:
238         case DW_EH_PE_sdata8:
239                 fsize = 8;
240                 break;
241         case DW_EH_PE_udata4:
242         case DW_EH_PE_sdata4:
243                 fsize = 4;
244                 break;
245         case DW_EH_PE_udata2:
246         case DW_EH_PE_sdata2:
247                 fsize = 2;
```

```
248                 break;
249         case DW_EH_PE_uleb128:
250                 return (uleb_extract(data, dotp, len, ret));
188                 return (uleb_extract(data, dotp));
251         case DW_EH_PE_sleb128:
252                 return (sleb_extract(data, dotp, len, (int64_t *)ret));
190                 return ((uint64_t)sleb_extract(data, dotp));
253         default:
254                 *ret = 0;
255                 return (DW_BAD_ENCODING);
192                 return (0);
256         }

258         if (lsb) {
259                 /*
260                  * Extract unaligned LSB formated data
261                  */
262                 uint_t  cnt;

264                 result = 0;
265                 for (cnt = 0; cnt < fsize;
266                     cnt++, dot++) {
267                         uint64_t val;

269                         if (dot > len)
270                                 return (DW_OVERFLOW);
271 #endif /* ! codereview */
272                         val = data[dot];
273                         result |= val << (cnt * 8);
274                 }
275         } else {
276                 /*
277                  * Extract unaligned MSB formated data
278                  */
279                 uint_t  cnt;
280                 result = 0;
281                 for (cnt = 0; cnt < fsize;
282                     cnt++, dot++) {
283                         uint64_t val;

285                         if (dot > len)
286                                 return (DW_OVERFLOW);
287 #endif /* ! codereview */
288                         val = data[dot];
289                         result |= val << ((fsize - cnt - 1) * 8);
290                 }
291         }
292         /*
293          * perform sign extension
294          */
295         if ((ehe_flags & DW_EH_PE_signed) &&
296             (fsize < sizeof (uint64_t))) {
297                 int64_t sresult;
298                 uint_t  bitshift;
299                 sresult = result;
300                 bitshift = (sizeof (uint64_t) - fsize) * 8;
301                 sresult = (sresult << bitshift) >> bitshift;
302                 result = sresult;
303         }

305         /*
306          * If value is relative to a base address, adjust it
307          */
308         switch (ehe_flags & 0xf0) {
309         case DW_EH_PE_pcrel:
310                 result += sh_base + sh_offset;
```



```
311                 break;

313         /*
314          * datarel is relative to .eh_frame_hdr if within .eh_frame,
315          * but GOT if not.
316          */
317         case DW_EH_PE_datarel:
318                 if (frame_hdr)
319                         result += sh_base;
320                 else
321                         result += dbase;
322                 break;
323         }

325         /* Truncate the result to its specified size */
326         result = (result << ((sizeof (uint64_t) - fsize) * 8)) >>
327             ((sizeof (uint64_t) - fsize) * 8);

329         *dotp = dot;
330         *ret = result;
331         return (DW_SUCCESS);
205         return (result);
332 }
```

_____*unchanged_portion_omitted_*