```
/*	Copyright (c) 1990, 1991, 1992, 1993 UNIX System Laboratories, Inc.	*/
/*	Copyright (c) 1988, 1990 AT&T	*/
/*	  All Rights Reserved  	*/

/*	THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF     	*/
/*	UNIX System Laboratories, Inc.                     	*/
/*	The copyright notice above does not evidence any   	*/
/*	actual or intended publication of such source code.	*/

#ifndef _LIBELF_H
#define _LIBELF_H

#ident	"@(#)sgs-inc:common/libelf.h	1.8.3.1"

#include <sys/types.h>
#include "sys/elf.h"


#undef _
#ifdef __STDC__
	typedef void		Elf_Void;
#	define _(a)		a
#else
	typedef char		Elf_Void;
#	define _(a)		()
#	ifndef _SIZE_T
#		define _SIZE_T
#		ifndef size_t
#			define size_t	unsigned int
#		endif
#	endif
#	undef const
#	define const
#endif


/*	commands
 */

typedef enum {
	ELF_C_NULL = 0,	/* must be first, 0 */
	ELF_C_READ,
	ELF_C_WRITE,
	ELF_C_IMPURE_WRITE,
	ELF_C_CLR,
	ELF_C_SET,
	ELF_C_FDDONE,
	ELF_C_FDREAD,
	ELF_C_RDWR,
	ELF_C_NUM	/* must be last */
} Elf_Cmd;


/*	flags
 */

#define ELF_F_DIRTY	0x1
#define ELF_F_LAYOUT	0x4


/*	file types
 */

typedef enum {
	ELF_K_NONE = 0,	/* must be first, 0 */
	ELF_K_AR,
	ELF_K_COFF,
	ELF_K_ELF,
	ELF_K_NUM	/* must be last */
} Elf_Kind;


/*	translation types
 */

typedef enum {
	ELF_T_BYTE = 0,	/* must be first, 0 */
	ELF_T_ADDR,
```

```
/* Interface for libelf.
   Copyright (C) 1998, 1999, 2000, 2002, 2004 Red Hat, Inc.

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, version 2.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software Foundation,
   Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.  */

#ifndef _LIBELF_H
#define _LIBELF_H 1

#include <sys/types.h>

/* Get the ELF types.  */
#include <elf.h>


/* Known translation types.  */
typedef enum
{
  ELF_T_BYTE,                   /* unsigned char */
  ELF_T_ADDR,                   /* Elf32_Addr, Elf64_Addr, ... */
  ELF_T_DYN,                    /* Dynamic section record.  */
  ELF_T_EHDR,                   /* ELF header.  */
  ELF_T_HALF,                   /* Elf32_Half, Elf64_Half, ... */
  ELF_T_OFF,                    /* Elf32_Off, Elf64_Off, ... */
  ELF_T_PHDR,                   /* Program header.  */
  ELF_T_RELA,                   /* Relocation entry with addend.  */
  ELF_T_REL,                    /* Relocation entry.  */
  ELF_T_SHDR,                   /* Section header.  */
  ELF_T_SWORD,                  /* Elf32_Sword, Elf64_Sword, ... */
  ELF_T_SYM,                    /* Symbol record.  */
  ELF_T_WORD,                   /* Elf32_Word, Elf64_Word, ... */
  ELF_T_XWORD,                  /* Elf32_Xword, Elf64_Xword, ... */
  ELF_T_SXWORD,                 /* Elf32_Sxword, Elf64_Sxword, ... */
  ELF_T_VDEF,                   /* Elf32_Verdef, Elf64_Verdef, ... */
  ELF_T_VDAUX,                  /* Elf32_Verdaux, Elf64_Verdaux, ... */
  ELF_T_VNEED,                  /* Elf32_Verneed, Elf64_Verneed, ... */
  ELF_T_VNAUX,                  /* Elf32_Vernaux, Elf64_Vernaux, ... */
  ELF_T_NHDR,                   /* Elf32_Nhdr, Elf64_Nhdr, ... */
  ELF_T_SYMINFO,        /* Elf32_Syminfo, Elf64_Syminfo, ... */
  ELF_T_MOVE,            /* Elf32_Move, Elf64_Move, ... */
  ELF_T_LIB,            /* Elf32_Lib, Elf64_Lib, ... */
  /* Keep this the last entry.  */
  ELF_T_NUM
} Elf_Type;

/* Descriptor for data to be converted to or from memory format.  */
typedef struct
{
  void *d_buf;            /* Pointer to the actual data.  */
  Elf_Type d_type;        /* Type of this piece of data.  */
  unsigned int d_version;    /* ELF version.  */
  size_t d_size;        /* Size in bytes.  */
  off_t d_off;            /* Offset into section.  */
  size_t d_align;        /* Alignment in section.  */
} Elf_Data;


/* Commands for `...'.  */
typedef enum
{
  ELF_C_NULL,            /* Nothing, terminate, or compute only.  */
  ELF_C_READ,            /* Read .. */
  ELF_C_RDWR,            /* Read and write .. */
  ELF_C_WRITE,            /* Write .. */
  ELF_C_CLR,            /* Clear flag.  */
  ELF_C_SET,            /* Set flag.  */
  ELF_C_FDDONE,            /* Signal that file descriptor will not be
                 used anymore.  */
```

/Users/Shared/FileFormatter/libelf/libelf-header/src2/SVR4.2-ES-MP-LIBELF.H

```
 79:     ELF_T_DYN,
 80:     ELF_T_EHDR,
 81:     ELF_T_HALF,
 82:     ELF_T_OFF,
 83:     ELF_T_PHDR,
 84:     ELF_T_RELA,
 85:     ELF_T_REL,
 86:     ELF_T_SHDR,
 87:     ELF_T_SWORD,
 88:     ELF_T_SYM,
 89:     ELF_T_WORD,
 90:     ELF_T_NUM    /* must be last */
 91: } Elf_Type;
 92:
 93:
 94: typedef struct Elf    Elf;
 95: typedef struct Elf_Scn    Elf_Scn;
 96:
 97:
 98: /*    archive member header
 99:  */
100:
101: typedef struct {
102:     char        *ar_name;
103:     time_t        ar_date;
104:     long         ar_uid;
105:     long         ar_gid;
106:     unsigned long    ar_mode;
107:     off_t        ar_size;
108:     char         *ar_rawname;
109: } Elf_Arhdr;
110:
111:
112: /*    archive symbol table
113:  */
114:
115: typedef struct {
116:     char        *as_name;
117:     size_t        as_off;
118:     unsigned long    as_hash;
119: } Elf_Arsym;
120:
121:
122: /*    data descriptor
123:  */
124:
125: typedef struct {
126:     Elf_Void    *d_buf;
127:     Elf_Type    d_type;
128:     size_t        d_size;
129:     off_t        d_off;        /* offset into section */
130:     size_t        d_align;    /* alignment in section */
131:     unsigned    d_version;    /* elf version */
132: } Elf_Data;
133:
134:
135: /*    function declarations
136:  */
137:
138: Elf        *elf_begin    _((int, Elf_Cmd, Elf *));
139: int        elf_cntl    _((Elf *, Elf_Cmd));
140: int        elf_end        _((Elf *));
141: const char    *elf_errmsg    _((int));
142: int        elf_errno    _((void));
143: void        elf_fill    _((int));
144: unsigned    elf_flagdata    _((Elf_Data *, Elf_Cmd, unsigned));
145: unsigned    elf_flagehdr    _((Elf *, Elf_Cmd, unsigned));
146: unsigned    elf_flagelf    _((Elf *, Elf_Cmd, unsigned));
147: unsigned    elf_flagphdr    _((Elf *, Elf_Cmd, unsigned));
148: unsigned    elf_flagscn    _((Elf_Scn *, Elf_Cmd, unsigned));
149: unsigned    elf_flagshdr    _((Elf_Scn *, Elf_Cmd, unsigned));
150: size_t        elf32_fsize    _((Elf_Type, size_t, unsigned));
151: Elf_Arhdr    *elf_getarhdr    _((Elf *));
152: Elf_Arsym    *elf_getarsym    _((Elf *, size_t *));
153: off_t        elf_getbase    _((Elf *));
154: Elf_Data    *elf_getdata    _((Elf_Scn *, Elf_Data *));
155: Elf32_Ehdr    *elf32_getehdr    _((Elf *));
156: char        *elf_getident    _((Elf *, size_t *));
```

5:327 5:314 5:320 4:267 4:277 4:273 4:253 4:238 4:286 4:235 4:241 4:247 4:244

/Users/Shared/FileFormatter/libelf/libelf-header/src2/redhat-elfutils0.97-libelf.h

```
 79:   ELF_C_FDREAD,            /* Read rest of data so that file descriptor
 80:                  is not used anymore.  */
 81:   /* The following are extensions.  */
 82:   ELF_C_READ_MMAP,        /* Read, but mmap the file if possible.  */
 83:   ELF_C_RDWR_MMAP,        /* Read and write, with mmap.  */
 84:   ELF_C_WRITE_MMAP,        /* Write, with mmap.  */
 85:   ELF_C_READ_MMAP_PRIVATE,    /* Read, but memory is writable, results are
 86:                  not written to the file.  */
 87:   ELF_C_EMPTY,            /* Copy basic file data but not the content. */
 88:   /* Keep this the last entry.  */
 89:   ELF_C_NUM
 90: } Elf_Cmd;
 91:
 92:
 93: /* Flags for the ELF structures.  */
 94: enum
 95: {
 96:   ELF_F_DIRTY = 0x1,
 97: #define ELF_F_DIRTY        ELF_F_DIRTY
 98:   ELF_F_LAYOUT = 0x4,
 99: #define ELF_F_LAYOUT        ELF_F_LAYOUT
100:   ELF_F_PERMISSIVE = 0x8
101: #define ELF_F_PERMISSIVE    ELF_F_PERMISSIVE
102: };
103:
104:
105: /* Identification values for recognized object files.  */
106: typedef enum
107: {
108:   ELF_K_NONE,            /* Unknown.  */
109:   ELF_K_AR,            /* Archive.  */
110:   ELF_K_COFF,            /* Stupid old COFF.  */
111:   ELF_K_ELF,            /* ELF file.  */
112:   /* Keep this the last entry.  */
113:   ELF_K_NUM
114: } Elf_Kind;
115:
116:
117: /* Archive member header.  */
118: typedef struct
119: {
120:   char *ar_name;        /* Name of archive member.  */
121:   time_t ar_date;        /* File date.  */
122:   uid_t ar_uid;            /* User ID.  */
123:   gid_t ar_gid;            /* Group ID.  */
124:   mode_t ar_mode;        /* File mode.  */
125:   off_t ar_size;        /* File size.  */
126:   char *ar_rawname;        /* Original name of archive member.  */
127: } Elf_Arhdr;
128:
129:
130: /* Archive symbol table entry.  */
131: typedef struct
132: {
133:   char *as_name;        /* Symbol name.  */
134:   size_t as_off;        /* Offset for this file in the archive.  */
135:   unsigned long int as_hash;    /* Hash value of the name.  */
136: } Elf_Arsym;
137:
138:
139: /* Descriptor for the ELF file.  */
140: typedef struct Elf Elf;
141:
142: /* Descriptor for ELF file section.  */
143: typedef struct Elf_Scn Elf_Scn;
144:
145:
146: #ifdef __cplusplus
147: extern "C" {
148: #endif
149:
150: /* Return descriptor for ELF file to work according to CMD.  */
151: extern Elf *elf_begin (int __fildes, Elf_Cmd __cmd, Elf *__ref);
152:
153: /* Create a clone of an existing ELF descriptor.  */
154:   extern Elf *elf_clone (Elf *__elf, Elf_Cmd __cmd);
155:
156: /* Create descriptor for memory region.  */
```

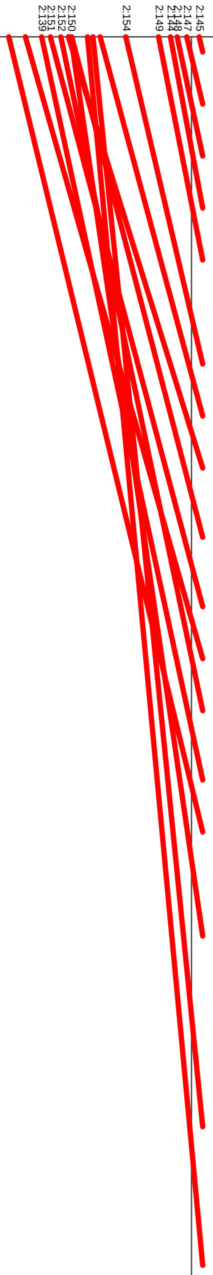
(For readability, not all matches are indicated.)

```
157: Elf32_Phdr    *elf32_getphdr    _((Elf *));
158: Elf_Scn        *elf_getscn    _((Elf *elf, size_t));
159: Elf32_Shdr    *elf32_getshdr    _((Elf_Scn *));
160: unsigned long    elf_hash    _((const char *));
161: Elf_Kind    elf_kind    _((Elf *));
162: size_t        elf_ndxscn    _((Elf_Scn *));
163: Elf_Data    *elf_newdata    _((Elf_Scn *));
164: Elf32_Ehdr    *elf32_newehdr    _((Elf *));
165: Elf32_Phdr    *elf32_newphdr    _((Elf *, size_t));
166: Elf_Scn        *elf_newscn    _((Elf *));
167: Elf_Scn        *elf_nextscn    _((Elf *, Elf_Scn *));
168: Elf_Cmd        elf_next    _((Elf *));
169: size_t        elf_rand    _((Elf *, size_t));
170: Elf_Data    *elf_rawdata    _((Elf_Scn *, Elf_Data *));
171: char        *elf_rawfile    _((Elf *, size_t *));
172: char        *elf_strptr    _((Elf *, size_t, size_t));
173: off_t        elf_update    _((Elf *, Elf_Cmd));
174: unsigned    elf_version    _((unsigned));
175: Elf_Data    *elf32_xlatetof    _((Elf_Data *, const Elf_Data *, unsigned));
176: Elf_Data    *elf32_xlatetom    _((Elf_Data *, const Elf_Data *, unsigned));
177:
178: #undef    _
179:
180: #endif
```

5.324 5.330

```
157: extern Elf *elf_memory (char *__image, size_t __size);
158:
159: /* Advance archive descriptor to next element.  */
160: extern Elf_Cmd elf_next (Elf *__elf);
161:
162: /* Free resources allocated for ELF.  */
163: extern int elf_end (Elf *__elf);
164:
165: /* Update ELF descriptor and write file to disk.  */
166: extern off_t elf_update (Elf *__elf, Elf_Cmd __cmd);
167:
168: /* Determine what kind of file is associated with ELF.  */
169: extern Elf_Kind elf_kind (Elf *__elf) __attribute__ ((__pure__));
170:
171: /* Get the base offset for an object file.  */
172: extern off_t elf_getbase (Elf *__elf);
173:
174:
175: /* Retrieve file identification data.  */
176: extern char *elf_getident (Elf *__elf, size_t *__ptr);
177:
178: /* Retrieve class-dependent object file header.  */
179: extern Elf32_Ehdr *elf32_getehdr (Elf *__elf);
180: /* Similar but this time the binary calls is ELFCLASS64.  */
181: extern Elf64_Ehdr *elf64_getehdr (Elf *__elf);
182:
183: /* Create ELF header if none exists.  */
184: extern Elf32_Ehdr *elf32_newehdr (Elf *__elf);
185: /* Similar but this time the binary calls is ELFCLASS64.  */
186: extern Elf64_Ehdr *elf64_newehdr (Elf *__elf);
187:
188: /* Retrieve class-dependent program header table.  */
189: extern Elf32_Phdr *elf32_getphdr (Elf *__elf);
190: /* Similar but this time the binary calls is ELFCLASS64.  */
191: extern Elf64_Phdr *elf64_getphdr (Elf *__elf);
192:
193: /* Create ELF program header.  */
194: extern Elf32_Phdr *elf32_newphdr (Elf *__elf, size_t __cnt);
195: /* Similar but this time the binary calls is ELFCLASS64.  */
196: extern Elf64_Phdr *elf64_newphdr (Elf *__elf, size_t __cnt);
197:
198:
199: /* Get section at INDEX.  */
200: extern Elf_Scn *elf_getscn (Elf *__elf, size_t __index);
201:
202: /* Get index of section.  */
203: extern size_t elf_ndxscn (Elf_Scn *__scn);
204:
205: /* Get section with next section index.  */
206: extern Elf_Scn *elf_nextscn (Elf *__elf, Elf_Scn *__scn);
207:
208: /* Create a new section and append it at the end of the table.  */
209: extern Elf_Scn *elf_newscn (Elf *__elf);
210:
211: /* Get the number of sections in the ELF file.  If the file uses more
212:    sections than can be represented in the e_shnum field of the ELF
213:    header the information from the sh_size field in the zeroth section
214:    header is used.  */
215: extern int elf_getshnum (Elf *__elf, size_t *__dst);
216:
217:
218: /* Get the section index of the section header string table in the ELF
219:    file.  If the index cannot be represented in the e_shnum field of
220:    the ELF header the information from the sh_link field in the zeroth
221:    section header is used.  */
222: extern int elf_getshstrndx (Elf *__elf, size_t *__dst);
223:
224:
225: /* Retrieve section header of ELFCLASS32 binary.  */
226: extern Elf32_Shdr *elf32_getshdr (Elf_Scn *__scn);
227: /* Similar for ELFCLASS64.  */
228: extern Elf64_Shdr *elf64_getshdr (Elf_Scn *__scn);
229:
230:
231: /* Set or clear flags for ELF file.  */
232: extern unsigned int elf_flagelf (Elf *__elf, Elf_Cmd __cmd,
233:                  unsigned int __flags);
234: /* Similarly for the ELF header.  */
```

(For readability, not all matches are indicated.)



```
235: extern unsigned int elf_flagehdr (Elf *__elf, Elf_Cmd __cmd,
236:                   unsigned int __flags);
237: /* Similarly for the ELF program header.  */
238: extern unsigned int elf_flagphdr (Elf *__elf, Elf_Cmd __cmd,
239:                   unsigned int __flags);
240: /* Similarly for the given ELF section.  */
241: extern unsigned int elf_flagscn (Elf_Scn *__scn, Elf_Cmd __cmd,
242:                  unsigned int __flags);
243: /* Similarly for the given ELF data.  */
244: extern unsigned int elf_flagdata (Elf_Data *__data, Elf_Cmd __cmd,
245:                   unsigned int __flags);
246: /* Similarly for the given ELF section header.  */
247: extern unsigned int elf_flagshdr (Elf_Scn *__scn, Elf_Cmd __cmd,
248:                   unsigned int __flags);
249:
250:
251: /* Get data from section while translating from file representation
252:    to memory representation.  */
253: extern Elf_Data *elf_getdata (Elf_Scn *__scn, Elf_Data *__data);
254:
255: /* Get uninterpreted section content.  */
256: extern Elf_Data *elf_rawdata (Elf_Scn *__scn, Elf_Data *__data);
257:
258: /* Create new data descriptor for section SCN.  */
259: extern Elf_Data *elf_newdata (Elf_Scn *__scn);
260:
261:
262: /* Return pointer to string at OFFSET in section INDEX.  */
263: extern char *elf_strptr (Elf *__elf, size_t __index, size_t __offset);
264:
265:
266: /* Return header of archive.  */
267: extern Elf_Arhdr *elf_getarhdr (Elf *__elf);
268:
269: /* Select archive element at OFFSET.  */
270: extern size_t elf_rand (Elf *__elf, size_t __offset);
271:
272: /* Get symbol table of archhive.  */
273: extern Elf_Arsym *elf_getarsym (Elf *__elf, size_t *__ptr);
274:
275:
276: /* Control ELF descriptor.  */
277: extern int elf_cntl (Elf *__elf, Elf_Cmd __cmd);
278:
279: /* Retrieve uninterpreted file contents.  */
280: extern char *elf_rawfile (Elf *__elf, size_t *__ptr);
281:
282:
283: /* Return size of array of COUNT elements of the type denoted by TYPE
284:    in the external representation.  The binary class is taken from ELF.
285:    The result is based on version VERSION of the ELF standard.  */
286: extern size_t elf32_fsize (Elf_Type __type, size_t __count,
287:                unsigned int __version)
288:        __attribute__ ((__const__));
289: /* Similar but this time the binary calls is ELFCLASS64.  */
290: extern size_t elf64_fsize (Elf_Type __type, size_t __count,
291:                unsigned int __version)
292:        __attribute__ ((__const__));
293:
294:
295: /* Convert data structure from the representation in the file represented
296:    by ELF to their memory representation.  */
297: extern Elf_Data *elf32_xlatetom (Elf_Data *__dest, const Elf_Data *__src,
298:                  unsigned int __encode);
299: /* Same for 64 bit class.  */
300: extern Elf_Data *elf64_xlatetom (Elf_Data *__dest, const Elf_Data *__src,
301:                  unsigned int __encode);
302:
303: /* Convert data structure from to the representation in memory
304:    represented by ELF file representation.  */
305: extern Elf_Data *elf32_xlatetof (Elf_Data *__dest, const Elf_Data *__src,
306:                  unsigned int __encode);
307: /* Same for 64 bit class.  */
308: extern Elf_Data *elf64_xlatetof (Elf_Data *__dest, const Elf_Data *__src,
309:                  unsigned int __encode);
310:
311:
312: /* Return error code of last failing function call.  This value is kept
```

(For readability, not all matches are indicated.)

```
   separately for each thread.  */
extern int elf_errno (void);

/* Return error string for ERROR.  If ERROR is zero, return error string
   for most recent error or NULL is none occurred.  If ERROR is -1 the
   behaviour is similar to the last case except that not NULL but a legal
   string is returned.  */
extern const char *elf_errmsg (int __error);


/* Coordinate ELF library and application versions.  */
extern unsigned int elf_version (unsigned int __version);

/* Set fill bytes used to fill holes in data structures.  */
extern void elf_fill (int __fill);

/* Compute hash value.  */
extern unsigned long int elf_hash (const char *__string)
       __attribute__ ((__pure__));


/* Compute simple checksum from permanent parts of the ELF file.  */
extern long int elf32_checksum (Elf *__elf);
/* Similar but this time the binary calls is ELFCLASS64.  */
extern long int elf64_checksum (Elf *__elf);

#ifdef __cplusplus
}
#endif

#endif  /* libelf.h */
```