```
**********************************************************
   43470 Wed Jul 11 20:34:35 2012
new/usr/src/cmd/file/file.c
2990 file(1) should have a -s flag to process special files
**********************************************************
```

```
   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  */
  21 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
  22 /*        All Rights Reserved   */


  25 /*      Copyright (c) 1987, 1988 Microsoft Corporation  */
  26 /*        All Rights Reserved   */

  28 /*
  29  * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */

  33 /*
  34  * Copyright 2012, Josef 'Jeff' Sipek <jeffpc@31bits.net>. All rights reserved.
  35  */

  37 #define _LARGEFILE64_SOURCE

  39 /* Get definitions for the relocation types supported. */
  40 #define ELF_TARGET_ALL

  42 #include <ctype.h>
  43 #include <unistd.h>
  44 #include <fcntl.h>
  45 #include <signal.h>
  46 #include <stdio.h>
  47 #include <libelf.h>
  48 #include <stdlib.h>
  49 #include <limits.h>
  50 #include <locale.h>
  51 #include <wctype.h>
  52 #include <string.h>
  53 #include <errno.h>
  54 #include <door.h>
  55 #include <sys/param.h>
  56 #include <sys/types.h>
  57 #include <sys/mkdev.h>
  58 #include <sys/stat.h>
  59 #include <sys/elf.h>
  60 #include <procfs.h>
  61 #include <sys/core.h>
```



```
  62 #include <sys/dumphdr.h>
  63 #include <netinet/in.h>
  64 #include <gelf.h>
  65 #include <elfcap.h>
  66 #include <sgsrtcid.h>
  67 #include "file.h"
  68 #include "elf_read.h"

  70 /*
  71  *      Misc
  72  */

  74 #define FBSZ            512
  75 #define MLIST_SZ        12

  77 /*
  78  * The 0x8FCA0102 magic string was used in crash dumps generated by releases
  79  * prior to Solaris 7.
  80  */
  81 #define OLD_DUMP_MAGIC  0x8FCA0102

  83 #if defined(__sparc)
  84 #define NATIVE_ISA      "SPARC"
  85 #define OTHER_ISA       "Intel"
  86 #else
  87 #define NATIVE_ISA      "Intel"
  88 #define OTHER_ISA       "SPARC"
  89 #endif

  91 /* Assembly language comment char */
  92 #ifdef pdp11
  93 #define ASCOMCHAR '/'
  94 #else
  95 #define ASCOMCHAR '!'
  96 #endif

  98 #pragma align   16(fbuf)
  99 static char     fbuf[FBSZ];

 101 /*
 102  * Magic file variables
 103  */
 104 static intmax_t maxmagicoffset;
 105 static intmax_t tmpmax;
 106 static char     *magicbuf;

 108 static char     *dfile;
 109 static char     *troff[] = {    /* new troff intermediate lang */
 110                 "x", "T", "res", "init", "font", "202", "V0", "p1", 0};

 112 static char     *fort[] = {                     /* FORTRAN */
 113                 "function", "subroutine", "common", "dimension", "block",
 114                 "integer", "real", "data", "double",
 115                 "FUNCTION", "SUBROUTINE", "COMMON", "DIMENSION", "BLOCK",
 116                 "INTEGER", "REAL", "DATA", "DOUBLE", 0};

 118 static char     *asc[] = {              /* Assembler Commands */
 119                 "sys", "mov", "tst", "clr", "jmp", "cmp", "set", "inc",
 120                 "dec", 0};

 122 static char     *c[] = {                        /* C Language */
 123                 "int", "char", "float", "double", "short", "long", "unsigned",
 124                 "register", "static", "struct", "extern", 0};

 126 static char     *as[] = {       /* Assembler Pseudo Ops, prepended with '.' */
 127                 "globl", "global", "ident", "file", "byte", "even",
```

```
128                 "text", "data", "bss", "comm", 0};

130 /*
131  * The line and debug section names are used by the strip command.
132  * Any changes in the strip implementation need to be reflected here.
133  */
134 static char     *debug_sections[] = { /* Debug sections in a ELF file */
135                 ".debug", ".stab", ".dwarf", ".line", NULL};

137 /* start for MB env */
138 static wchar_t  wchar;
139 static int      length;
140 static int      IS_ascii;
141 static int      Max;
142 /* end for MB env */
143 static int      i;      /* global index into first 'fbsz' bytes of file */
144 static int      fbsz;
145 static int      ifd = -1;
146 static int      elffd = -1;
147 static int      tret;
148 static int      sflg;
149 static int      hflg;
150 static int      dflg;
151 static int      mflg;
152 static int      M_flg;
153 static int      iflg;
154 static struct stat64    mbuf;

156 static char     **mlist1;       /* 1st ordered list of magic files */
157 static char     **mlist2;       /* 2nd ordered list of magic files */
158 static size_t   mlist1_sz;      /* number of ptrs allocated for mlist1 */
159 static size_t   mlist2_sz;      /* number of ptrs allocated for mlist2 */
160 static char     **mlist1p;      /* next entry in mlist1 */
161 static char     **mlist2p;      /* next entry in mlist2 */

163 static ssize_t  mread;

165 static void ar_coff_or_aout(int ifd);
166 static int type(char *file);
167 static int def_position_tests(char *file);
168 static void def_context_tests(void);
169 static int troffint(char *bp, int n);
170 static int lookup(char **tab);
171 static int ccom(void);
172 static int ascom(void);
173 static int sccs(void);
174 static int english(char *bp, int n);
175 static int shellscript(char buf[], struct stat64 *sb);
176 static int elf_check(char *file);
177 static int get_door_target(char *, char *, size_t);
178 static int zipfile(char *, int);
179 static int is_crash_dump(const char *, int);
180 static void print_dumphdr(const int, const dumphdr_t *, uint32_t (*)(uint32_t),
181     const char *);
182 static uint32_t swap_uint32(uint32_t);
183 static uint32_t return_uint32(uint32_t);
184 static void usage(void);
185 static void default_magic(void);
186 static void add_to_mlist(char *, int);
187 static void fd_cleanup(void);
188 static int is_rtld_config(void);

190 /* from elf_read.c */
191 int elf_read32(int elffd, Elf_Info *EInfo);
192 int elf_read64(int elffd, Elf_Info *EInfo);
```



```
194 #ifdef XPG4
195         /* SUSv3 requires a single <space> after the colon */
196 #define prf(x)  (void) printf("%s: ", x);
197 #else   /* !XPG4 */
198 #define prf(x)  (void) printf("%s:%s", x, (int)strlen(x) > 6 ? "\t" : "\t\t");
199 #endif  /* XPG4 */

201 /*
202  * Static program identifier - used to prevent localization of the name "file"
203  * within individual error messages.
204  */
205 const char *File = "file";

207 int
208 main(int argc, char **argv)
209 {
210         char    *p;
211         int     ch;
212         FILE    *fl;
213         int     cflg = 0;
214         int     eflg = 0;
215         int     fflg = 0;
216         char    *ap = NULL;
217         int     pathlen;
218         char    **filep;

220         (void) setlocale(LC_ALL, "");
221 #if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
222 #define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't */
223 #endif
224         (void) textdomain(TEXT_DOMAIN);

226         while ((ch = getopt(argc, argv, "M:cdsf:him:")) != EOF) {
221         while ((ch = getopt(argc, argv, "M:cdf:him:")) != EOF) {
227                 switch (ch) {

229                 case 'M':
230                         add_to_mlist(optarg, !dflg);
231                         M_flg++;
232                         break;

234                 case 'c':
235                         cflg++;
236                         break;

238                 case 'd':
239                         if (!dflg) {
240                                 default_magic();
241                                 add_to_mlist(dfile, 0);
242                                 dflg++;
243                         }
244                         break;

246                 case 's':
247                         sflg++;
248                         break;

250                 case 'f':
251                         fflg++;
252                         errno = 0;
253                         if ((fl = fopen(optarg, "r")) == NULL) {
254                                 int err = errno;
255                                 (void) fprintf(stderr, gettext("%s: cannot "
256                                     "open file %s: %s\n"), File, optarg,
257                                     err ? strerror(err) : "");
258                                 usage();
```

```
259                         }
260                         pathlen = pathconf("/", _PC_PATH_MAX);
261                         if (pathlen == -1) {
262                                 int err = errno;
263                                 (void) fprintf(stderr, gettext("%s: cannot "
264                                     "determine maximum path length: %s\n"),
265                                     File, strerror(err));
266                                 exit(1);
267                         }
268                         pathlen += 2; /* for null and newline in fgets */
269                         if ((ap = malloc(pathlen * sizeof (char))) == NULL) {
270                                 int err = errno;
271                                 (void) fprintf(stderr, gettext("%s: malloc "
272                                     "failed: %s\n"), File, strerror(err));
273                                 exit(2);
274                         }
275                         break;

277                 case 'h':
278                         hflg++;
279                         break;

281                 case 'i':
282                         iflg++;
283                         break;

285                 case 'm':
286                         add_to_mlist(optarg, !dflg);
287                         mflg++;
288                         break;

290                 case '?':
291                         eflg++;
292                         break;
293                 }
294         }
295         if (!cflg && !fflg && (eflg || optind == argc))
296                 usage();
297         if (iflg && (dflg || mflg || M_flg)) {
298                 usage();
299         }
300         if (iflg && cflg) {
301                 usage();
302         }
303         if (sflg && (iflg || cflg))
304                 usage();

306         if (!dflg && !mflg && !M_flg && !iflg) {
307         /* no -d, -m, nor -M option; also -i option doesn't need magic  */
308                 default_magic();
309                 if (f_mkmtab(dfile, cflg, 0) == -1) {
310                         exit(2);
311                 }
312         }

314         else if (mflg && !M_flg && !dflg) {
315         /* -m specified without -d nor -M */

317 #ifdef XPG4     /* For SUSv3 only */

319                 /*
320                  * The default position-dependent magic file tests
321                  * in /etc/magic will follow all the -m magic tests.
322                  */

324                 for (filep = mlist1; filep < mlist1p; filep++) {
```



```
325                         if (f_mkmtab(*filep, cflg, 1) == -1) {
326                                 exit(2);
327                         }
328                 }
329                 default_magic();
330                 if (f_mkmtab(dfile, cflg, 0) == -1) {
331                         exit(2);
332                 }
333 #else   /* !XPG4 */
334                 /*
335                  * Retain Solaris file behavior for -m before SUSv3,
336                  * when the new -d and -M options are not specified.
337                  * Use the -m file specified in place of the default
338                  * /etc/magic file.  Solaris file will
339                  * now allow more than one magic file to be specified
340                  * with multiple -m options, for consistency with
341                  * other behavior.
342                  *
343                  * Put the magic table(s) specified by -m into
344                  * the second magic table instead of the first
345                  * (as indicated by the last argument to f_mkmtab()),
346                  * since they replace the /etc/magic tests and
347                  * must be executed alongside the default
348                  * position-sensitive tests.
349                  */

351                 for (filep = mlist1; filep < mlist1p; filep++) {
352                         if (f_mkmtab(*filep, cflg, 0) == -1) {
353                                 exit(2);
354                         }
355                 }
356 #endif /* XPG4 */
357         } else {
358                 /*
359                  * For any other combination of -d, -m, and -M,
360                  * use the magic files in command-line order.
361                  * Store the entries from the two separate lists of magic
362                  * files, if any, into two separate magic file tables.
363                  * mlist1: magic tests executed before default magic tests
364                  * mlist2: default magic tests and after
365                  */
366                 for (filep = mlist1; filep && (filep < mlist1p); filep++) {
367                         if (f_mkmtab(*filep, cflg, 1) == -1) {
368                                 exit(2);
369                         }
370                 }
371                 for (filep = mlist2; filep && (filep < mlist2p); filep++) {
372                         if (f_mkmtab(*filep, cflg, 0) == -1) {
373                                 exit(2);
374                         }
375                 }
376         }

378         /* Initialize the magic file variables; check both magic tables */
379         tmpmax = f_getmaxoffset(1);
380         maxmagicoffset = f_getmaxoffset(0);
381         if (maxmagicoffset < tmpmax) {
382                 maxmagicoffset = tmpmax;
383         }
384         if (maxmagicoffset < (intmax_t)FBSZ)
385                 maxmagicoffset = (intmax_t)FBSZ;
386         if ((magicbuf = malloc(maxmagicoffset)) == NULL) {
387                 int err = errno;
388                 (void) fprintf(stderr, gettext("%s: malloc failed: %s\n"),
389                     File, strerror(err));
390                 exit(2);
```

```
391         }
393         if (cflg) {
394                 f_prtmtab();
395                 if (ferror(stdout) != 0) {
396                         (void) fprintf(stderr, gettext("%s: error writing to "
397                             "stdout\n"), File);
398                         exit(1);
399                 }
400                 if (fclose(stdout) != 0) {
401                         int err = errno;
402                         (void) fprintf(stderr, gettext("%s: fclose "
403                             "failed: %s\n"), File, strerror(err));
404                         exit(1);
405                 }
406                 exit(0);
407         }
409         for (; fflg || optind < argc; optind += !fflg) {
410                 register int    l;
412                 if (fflg) {
413                         if ((p = fgets(ap, pathlen, fl)) == NULL) {
414                                 fflg = 0;
415                                 optind--;
416                                 continue;
417                         }
418                         l = strlen(p);
419                         if (l > 0)
420                                 p[l - 1] = '\0';
421                 } else
422                         p = argv[optind];
423                 prf(p);                         /* print "file_name:<tab>" */
425                 if (type(p))
426                         tret = 1;
427         }
428         if (ap != NULL)
429                 free(ap);
430         if (tret != 0)
431                 exit(tret);
433         if (ferror(stdout) != 0) {
434                 (void) fprintf(stderr, gettext("%s: error writing to "
435                     "stdout\n"), File);
436                 exit(1);
437         }
438         if (fclose(stdout) != 0) {
439                 int err = errno;
440                 (void) fprintf(stderr, gettext("%s: fclose failed: %s\n"),
441                     File, strerror(err));
442                 exit(1);
443         }
444         return (0);
445 }
447 static int
448 type(char *file)
449 {
450         int     cc;
451         char    buf[BUFSIZ];
452         int     (*statf)() = hflg ? lstat64 : stat64;
454         i = 0;          /* reset index to beginning of file */
455         ifd = -1;
456         if ((*statf)(file, &mbuf) < 0) {
```



```
457                 if (statf == lstat64 || lstat64(file, &mbuf) < 0) {
458                         int err = errno;
459                         (void) printf(gettext("cannot open: %s\n"),
460                             strerror(err));
461                         return (0);             /* POSIX.2 */
462                 }
463         }
464         switch (mbuf.st_mode & S_IFMT) {
465         case S_IFREG:
466                 if (iflg) {
467                         (void) printf(gettext("regular file\n"));
468                         return (0);
469                 }
470                 break;
471         case S_IFCHR:
472                 if (sflg)
473                         break;
474                 (void) printf(gettext("character"));
475                 goto spcl;
477         case S_IFDIR:
478                 (void) printf(gettext("directory\n"));
479                 return (0);
481         case S_IFIFO:
482                 (void) printf(gettext("fifo\n"));
483                 return (0);
485         case S_IFLNK:
486                 if ((cc = readlink(file, buf, BUFSIZ)) < 0) {
487                         int err = errno;
488                         (void) printf(gettext("readlink error: %s\n"),
489                             strerror(err));
490                         return (1);
491                 }
492                 buf[cc] = '\0';
493                 (void) printf(gettext("symbolic link to %s\n"), buf);
494                 return (0);
496         case S_IFBLK:
497                 if (sflg)
498                         break;
499                 (void) printf(gettext("block"));
500                                         /* major and minor, see sys/mkdev.h */
501 spcl:
502                 (void) printf(gettext(" special (%d/%d)\n"),
503                     major(mbuf.st_rdev), minor(mbuf.st_rdev));
504                 return (0);
506         case S_IFSOCK:
507                 (void) printf("socket\n");
508                 /* FIXME, should open and try to getsockname. */
509                 return (0);
511         case S_IFDOOR:
512                 if (get_door_target(file, buf, sizeof (buf)) == 0)
513                         (void) printf(gettext("door to %s\n"), buf);
514                 else
515                         (void) printf(gettext("door\n"));
516                 return (0);
518         }
520         if (elf_version(EV_CURRENT) == EV_NONE) {
521                 (void) printf(gettext("libelf is out of date\n"));
522                 return (1);
```

```
523         }

525         ifd = open64(file, O_RDONLY);
526         if (ifd < 0) {
527                 int err = errno;
528                 (void) printf(gettext("cannot open: %s\n"), strerror(err));
529                 return (0);                     /* POSIX.2 */
530         }

532         /* need another fd for elf, since we might want to read the file too */
533         elffd = open64(file, O_RDONLY);
534         if (elffd < 0) {
535                 int err = errno;
536                 (void) printf(gettext("cannot open: %s\n"), strerror(err));
537                 (void) close(ifd);
538                 ifd = -1;
539                 return (0);                     /* POSIX.2 */
540         }
541         if ((fbsz = read(ifd, fbuf, FBSZ)) == -1) {
542                 int err = errno;
543                 (void) printf(gettext("cannot read: %s\n"), strerror(err));
544                 (void) close(ifd);
545                 ifd = -1;
546                 return (0);                     /* POSIX.2 */
547         }
548         if (fbsz == 0) {
549                 (void) printf(gettext("empty file\n"));
550                 fd_cleanup();
551                 return (0);
552         }

554         /*
555          * First try user-specified position-dependent magic tests, if any,
556          * which need to execute before the default tests.
557          */
558         if ((mread = pread(ifd, (void*)magicbuf, (size_t)maxmagicoffset,
559             (off_t)0)) == -1) {
560                 int err = errno;
561                 (void) printf(gettext("cannot read: %s\n"), strerror(err));
562                 fd_cleanup();
563                 return (0);
564         }

566         /*
567          * CheCK against Magic Table entries.
568          * Check first magic table for magic tests to be applied
569          * before default tests.
570          * If no default tests are to be applied, all magic tests
571          * should occur in this magic table.
572          */
573         switch (f_ckmtab(magicbuf, mread, 1)) {
574                 case -1:        /* Error */
575                         exit(2);
576                         break;
577                 case 0:         /* Not magic */
578                         break;
579                 default:        /* Switch is magic index */
580                         (void) putchar('\n');
581                         fd_cleanup();
582                         return (0);
583                         /* NOTREACHED */
584                         break;
585         }

587         if (dflg || !M_flg) {
588                 /*
```



```
589                  * default position-dependent tests,
590                  * plus non-default magic tests, if any
591                  */
592                 switch (def_position_tests(file)) {
593                         case -1:        /* error */
594                                 fd_cleanup();
595                                 return (1);
596                         case 1: /* matching type found */
597                                 fd_cleanup();
598                                 return (0);
599                                 /* NOTREACHED */
600                                 break;
601                         case 0:         /* no matching type found */
602                                 break;
603                 }
604                 /* default context-sensitive tests */
605                 def_context_tests();
606         } else {
607                 /* no more tests to apply; no match was found */
608                 (void) printf(gettext("data\n"));
609         }
610         fd_cleanup();
611         return (0);
612 }
```

_____*unchanged_portion_omitted_*

```
1659 static void
1660 usage(void)
1661 {
1662         (void) fprintf(stderr, gettext(
1663             "usage: file [-dhs] [-M mfile] [-m mfile] [-f ffile] file ...\n"
1664             "       file [-dhs] [-M mfile] [-m mfile] -f ffile\n"
1648             "usage: file [-dh] [-M mfile] [-m mfile] [-f ffile] file ...\n"
1649             "       file [-dh] [-M mfile] [-m mfile] -f ffile\n"
1665             "       file -i [-h] [-f ffile] file ...\n"
1666             "       file -i [-h] -f ffile\n"
1667             "       file -c [-d] [-M mfile] [-m mfile]\n"));
1668         exit(2);
1669 }
```

_____*unchanged_portion_omitted_*

```
**********
    9081 Wed Jul 11 20:34:36 2012
new/usr/src/man/man1/file.1
2990 file(1) should have a -s flag to process special files
**********
'\" te
.\" Copyright 1989 AT&T Copyright (c) 1992, X/Open Company Limited All Rights Re
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
.\" http://www.opengroup.org/bookstore/.
.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
.\"  This notice shall appear on any product containing this material.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH FILE 1 "May 15, 2006"
.SH NAME
file \- determine file type
.SH SYNOPSIS
.LP
.nf
\fB/usr/bin/file\fR [\fB-dhs\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\fR] [
\fB/usr/bin/file\fR [\fB-dh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\fR] [\
.fi

.LP
.nf
\fB/usr/bin/file\fR [\fB-dhs\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\fR] \
\fB/usr/bin/file\fR [\fB-dh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\fR] \f
.fi

.LP
.nf
\fB/usr/bin/file\fR \fB-i\fR [\fB-h\fR] [\fB-f\fR \fIffile\fR] \fIfile\fR...
.fi

.LP
.nf
\fB/usr/bin/file\fR \fB-i\fR [\fB-h\fR] \fB-f\fR \fIffile\fR
.fi

.LP
.nf
\fB/usr/bin/file\fR \fB-c\fR [\fB-d\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfil
.fi

.LP
.nf
\fB/usr/xpg4/bin/file\fR [\fB-dhs\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\
\fB/usr/xpg4/bin/file\fR [\fB-dh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\f
.fi

.LP
.nf
\fB/usr/xpg4/bin/file\fR [\fB-das\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\
\fB/usr/xpg4/bin/file\fR [\fB-dh\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \fIMfile\f
.fi

.LP
.nf
\fB/usr/xpg4/bin/file\fR \fB-i\fR [\fB-h\fR] [\fB-f\fR \fIffile\fR] \fIfile\fR..
.fi

.LP
.nf
\fB/usr/xpg4/bin/file\fR \fB-i\fR [\fB-h\fR] \fB-f\fR \fIffile\fR
.fi
```



```
.LP
.nf
\fB/usr/xpg4/bin/file\fR \fB-c\fR [\fB-d\fR] [\fB-m\fR \fImfile\fR] [\fB-M\fR \f
.fi

.SH DESCRIPTION
.sp
.LP
The \fBfile\fR utility performs a series of tests on each file supplied by
\fIfile\fR and, optionally, on each file listed in \fIffile\fR in an attempt to
classify it. If the file is not a regular file, its file type is identified.
The file types directory, \fBFIFO\fR, block special, and character special are
identified as such. If the file is a regular file and the file is zero-length,
it is identified as an empty file.
.sp
.LP
If \fIfile\fR appears to be a text file, \fBfile\fR examines the first 512
bytes and tries to determine its programming language. If \fIfile\fR is a
symbolic link, by default the link is followed and \fBfile\fR tests the file to
which the symbolic link refers.
.sp
.LP
If \fIfile\fR is a relocatable object, executable, or shared object, \fBfile\fR
prints out information about the file's execution requirements. This
information includes the machine class, byte-ordering, static/dynamic linkage,
and any software or hardware capability requirements. If \fIfile\fR is a
runtime linking configuration file, \fBfile\fR prints information about the
target platform, including the machine class and byte-ordering.
.sp
.LP
By default, \fBfile\fR will try to use the localized magic file
\fB/usr/lib/locale/\fIlocale\fR/LC_MESSAGES/magic\fR, if it exists, to identify
files that have a magic number. For example, in the Japanese locale, \fBfile\fR
will try to use \fB/usr/lib/locale/ja/LC_MESSAGES/magic\fR. If a localized
magic file does not exist, \fBfile\fR will utilize \fB/etc/magic\fR. A magic
number is a numeric or string constant that indicates the file type. See
\fBmagic\fR(4) for an explanation of the format of \fB/etc/magic\fR.
.sp
.LP
If \fIfile\fR does not exist, cannot be read, or its file status could not be
determined, it is not considered an error that affects the exit status. The
output will indicate that the file was processed, but that its type could not
be determined.
.SH OPTIONS
.sp
.LP
The following options are supported:
.sp
.ne 2
.na
\fB\fB-c\fR\fR
.ad
.RS 12n
Checks the magic file for format errors. For reasons of efficiency, this
validation is normally not carried out.
.RE

.sp
.ne 2
.na
\fB\fB-d\fR\fR
.ad
.RS 12n
Applies any position-sensitive and context-sensitive default system tests to
the file.
```

```
.RE

.sp
.ne 2
.na
\fB\fB-f\fR \fIffile\fR\fR
.ad
.RS 12n
\fIffile\fR contains a list of the files to be examined.
.RE

.sp
.ne 2
.na
\fB\fB-h\fR\fR
.ad
.RS 12n
When a symbolic link is encountered, this option identifies the file as a
symbolic link. If \fB-h\fR is not specified and \fIfile\fR is a symbolic link
that refers to a non-existent file, the \fBfile\fR utility identifies the file
as a symbolic link, as if \fB-h\fR had been specified.
.RE

.sp
.ne 2
.na
\fB\fB-i\fR\fR
.ad
.RS 12n
If a file is a regular file, this option does not attempt to classify the type
of file further, but identifies the file as a "regular file".
.RE

.sp
.ne 2
.na
\fB\fB-s\fR\fR
.ad
.RS 12n
If a file is a block device or a character device, this option attempts to
classify the contents of the device as if it were a regular file.
.RE

.sp
.ne 2
.na
\fB\fB-m\fR \fImfile\fR\fR
.ad
.RS 12n
.sp
.ne 2
.na
\fB\fB/usr/bin/file\fR\fR
.ad
.RS 22n
Uses \fImfile\fR as an alternate magic file, instead of \fB/etc/magic\fR.
.RE

.sp
.ne 2
.na
\fB\fB/usr/xpg4/bin/file\fR\fR
.ad
.RS 22n
Specifies the name of a file containing position-sensitive tests that are
applied to a file in order to classify it (see \fBmagic\fR(4)). If the \fB-m\fR
```



```
option is specified without specifying the \fB-d\fR option or the \fB-M\fR
option, position-sensitive default system tests are applied after the
position-sensitive tests specified by the \fB-m\fR option.
.RE

.RE

.sp
.ne 2
.na
\fB\fB-M\fR \fIMfile\fR\fR
.ad
.RS 12n
Specifies the name of a file containing position-sensitive tests that are
applied to a file in order to classify it (see \fBmagic\fR(4)). No
position-sensitive default system tests nor context-sensitive default system
tests are applied unless the \fB-d\fR option is also specified.
.RE

.sp
.LP
If the \fB-M\fR option is specified with the \fB-d\fR option, the \fB-m\fR
option, or both, or if the \fB-m\fR option is specified with the \fB-d\fR
option, the concatenation of the position-sensitive tests specified by these
options is applied in the order specified by the appearance of these options.
.SH OPERANDS
.sp
.LP
The following operands are supported:
.sp
.ne 2
.na
\fB\fIfile\fR\fR
.ad
.RS 8n
A path name of a file to be tested.
.RE

.SH USAGE
.sp
.LP
See \fBlargefile\fR(5) for the description of the behavior of \fBfile\fR when
encountering files greater than or equal to 2 Gbyte ( 2^31 bytes).
.SH EXAMPLES
.LP
\fBExample 1 \fRDetermining if an Argument is a Binary Executable Files
.sp
.LP
The following example determine if an argument is a binary executable file:

.sp
.in +2
.nf
file "$1" | grep \(miFq executable &&
          printf "%s is executable.\n" "$1"
.fi
.in -2
.sp

.SH ENVIRONMENT VARIABLES
.sp
.LP
See \fBenviron\fR(5) for descriptions of the following environment variables
that affect the execution of \fBfile\fR: \fBLANG\fR, \fBLC_ALL\fR,
\fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, and \fBNLSPATH\fR.
.SH EXIT STATUS
```

```
256 .sp
257 .LP
258 The following exit values are returned:
259 .sp
260 .ne 2
261 .na
262 \fB\fB0\fR\fR
263 .ad
264 .RS 6n
265 Successful completion.
266 .RE

268 .sp
269 .ne 2
270 .na
271 \fB\fB>0\fR\fR
272 .ad
273 .RS 6n
274 An error occurred.
275 .RE

277 .SH FILES
278 .sp
279 .ne 2
280 .na
281 \fB\fB/etc/magic\fR\fR
282 .ad
283 .RS 14n
284 \fBfile\fR's magic number file
285 .RE

287 .SH ATTRIBUTES
288 .sp
289 .LP
290 See \fBattributes\fR(5) for descriptions of the following attributes:
291 .sp

293 .sp
294 .TS
295 box;
296 c | c
297 l | l .
298 ATTRIBUTE TYPE  ATTRIBUTE VALUE
299 _
300 CSI     Enabled
301 _
302 Interface Stability     Standard
303 .TE

305 .SH SEE ALSO
306 .sp
307 .LP
308 \fBcrle\fR(1), \fBelfdump\fR(1), \fBls\fR(1), \fBmagic\fR(4),
309 \fBattributes\fR(5), \fBenviron\fR(5), \fBlargefile\fR(5), \fBstandards\fR(5)
```