```
**********************************************************
   17910 Wed Aug 13 19:36:49 2014
new/usr/src/cmd/vi/port/ex.c
5088 it's probably ok for vi to stop working around pdp-11 bugs now
**********************************************************
```

```
/*
 * 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 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/* Copyright (c) 1981 Regents of the University of California */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include "ex.h"
#include "ex_argv.h"
#include "ex_temp.h"
#include "ex_tty.h"
#include <stdlib.h>
#include <locale.h>
#include <stdio.h>
#ifdef TRACE
unsigned char   tttrace[BUFSIZ];
#endif

#define EQ(a, b)        (strcmp(a, b) == 0)

char    *strrchr();
void    init_re(void);

/*
 * The code for ex is divided as follows:
 *
 * ex.c                 Entry point and routines handling interrupt, hangup
 *                      signals; initialization code.
 *
 * ex_addr.c            Address parsing routines for command mode decoding.
 *                      Routines to set and check address ranges on commands.
 *
 * ex_cmds.c            Command mode command decoding.
 *
 * ex_cmds2.c           Subroutines for command decoding and processing of
 *                      file names in the argument list.  Routines to print
```



```
 *                      messages and reset state when errors occur.
 *
 * ex_cmdsub.c          Subroutines which implement command mode functions
 *                      such as append, delete, join.
 *
 * ex_data.c            Initialization of options.
 *
 * ex_get.c             Command mode input routines.
 *
 * ex_io.c              General input/output processing: file i/o, unix
 *                      escapes, filtering, source commands, preserving
 *                      and recovering.
 *
 * ex_put.c             Terminal driving and optimizing routines for low-level
 *                      output (cursor-positioning); output line formatting
 *                      routines.
 *
 * ex_re.c              Global commands, substitute, regular expression
 *                      compilation and execution.
 *
 * ex_set.c             The set command.
 *
 * ex_subr.c            Loads of miscellaneous subroutines.
 *
 * ex_temp.c            Editor buffer routines for main buffer and also
 *                      for named buffers (Q registers if you will.)
 *
 * ex_tty.c             Terminal dependent initializations from termcap
 *                      data base, grabbing of tty modes (at beginning
 *                      and after escapes).
 *
 * ex_unix.c            Routines for the ! command and its variations.
 *
 * ex_v*.c              Visual/open mode routines... see ex_v.c for a
 *                      guide to the overall organization.
 */

/*
 * This sets the Version of ex/vi for both the exstrings file and
 * the version command (":ver").
 */

/* variable used by ":ver" command */
unsigned char *Version = (unsigned char *)"Version SVR4.0, Solaris 2.5.0";

/*
 * NOTE: when changing the Version number, it must be changed in the
 *       following files:
 *
 *                          port/READ_ME
 *                          port/ex.c
 *                          port/ex.news
 *
 */
#ifdef XPG4
unsigned char *savepat = (unsigned char *) NULL; /* firstpat storage    */
#endif /* XPG4 */

/*
 * Main procedure.  Process arguments and then
 * transfer control to the main command processing loop
 * in the routine commands.  We are entered as either "ex", "edit", "vi"
 * or "view" and the distinction is made here. For edit we just diddle options;
 * for vi we actually force an early visual command.
 */
static unsigned char cryptkey[19]; /* contents of encryption key */
```

```
127 static void usage(unsigned char *);

129 static int validate_exrc(unsigned char *);

131 void init(void);

133 int
134 main(int ac, char *av[])
135 {
136         extern  char    *optarg;
137         extern  int     optind;
138         unsigned char   *rcvname = 0;
139         unsigned char *cp;
140         int c;
141         unsigned char   *cmdnam;
142         bool recov = 0;
143         bool ivis = 0;
144         bool itag = 0;
145         bool fast = 0;
146         extern int verbose;
147         int argcounter = 0;
148         extern int tags_flag;   /* Set if tag file is not sorted (-S flag) */
149         unsigned char scratch [PATH_MAX+1];   /* temp for sourcing rc file(s) */
150         int vret = 0;
151         unsigned char exrcpath [PATH_MAX+1]; /* temp for sourcing rc file(s) */
152         int toptseen = 0;
153 #ifdef TRACE
154         unsigned char *tracef;
155 #endif
156         tagflg = 0;
157         (void) setlocale(LC_ALL, "");
158 #if !defined(TEXT_DOMAIN)
159 #define TEXT_DOMAIN "SYS_TEST"
160 #endif
161         (void) textdomain(TEXT_DOMAIN);

163         /*
164          * Immediately grab the tty modes so that we won't
165          * get messed up if an interrupt comes in quickly.
166          */
167         (void) gTTY(2);
168         normf = tty;
169         ppid = getpid();
170         /* Note - this will core dump if you didn't -DSINGLE in CFLAGS */
171         lines = 24;
172         columns = 80;   /* until defined right by setupterm */
173         /*
174          * Defend against d's, v's, w's, and a's in directories of
175          * path leading to our true name.
176          */
177         if ((cmdnam = (unsigned char *)strrchr(av[0], '/')) != 0)
178                 cmdnam++;
179         else
180                 cmdnam = (unsigned char *)av[0];

182         if (EQ((char *)cmdnam, "vi"))
183                 ivis = 1;
184         else if (EQ(cmdnam, "view")) {
185                 ivis = 1;
186                 value(vi_READONLY) = 1;
187         } else if (EQ(cmdnam, "vedit")) {
188                 ivis = 1;
189                 value(vi_NOVICE) = 1;
190                 value(vi_REPORT) = 1;
191                 value(vi_MAGIC) = 0;
```



```
192                 value(vi_SHOWMODE) = 1;
193         } else if (EQ(cmdnam, "edit")) {
194                 value(vi_NOVICE) = 1;
195                 value(vi_REPORT) = 1;
196                 value(vi_MAGIC) = 0;
197                 value(vi_SHOWMODE) = 1;
198         }

200 #ifdef  XPG4
201         {
202                 struct winsize jwin;
203                 char *envptr;

205                 envlines = envcolumns = -1;
206                 oldlines = oldcolumns = -1;

208                 if (ioctl(0, TIOCGWINSZ, &jwin) != -1) {
209                         oldlines = jwin.ws_row;
210                         oldcolumns = jwin.ws_col;
211                 }

213                 if ((envptr = getenv("LINES")) != NULL &&
214                     *envptr != '\0' && isdigit(*envptr)) {
215                         if ((envlines = atoi(envptr)) <= 0) {
216                                 envlines = -1;
217                         }
218                 }

220                 if ((envptr = getenv("COLUMNS")) != NULL &&
221                     *envptr != '\0' && isdigit(*envptr)) {
222                         if ((envcolumns = atoi(envptr)) <= 0) {
223                                 envcolumns = -1;
224                         }
225                 }
226         }
227 #endif /* XPG4 */

229         draino();
230         pstop();

232         /*
233          * Initialize interrupt handling.
234          */
235         oldhup = signal(SIGHUP, SIG_IGN);
236         if (oldhup == SIG_DFL)
237                 signal(SIGHUP, onhup);
238         oldquit = signal(SIGQUIT, SIG_IGN);
239         ruptible = signal(SIGINT, SIG_IGN) == SIG_DFL;
240         if (signal(SIGTERM, SIG_IGN) == SIG_DFL)
241                 signal(SIGTERM, onhup);
244         if (signal(SIGEMT, SIG_IGN) == SIG_DFL)
245                 signal(SIGEMT, onemt);
242         signal(SIGILL, oncore);
243         signal(SIGTRAP, oncore);
244         signal(SIGIOT, oncore);
245         signal(SIGFPE, oncore);
246         signal(SIGBUS, oncore);
247         signal(SIGSEGV, oncore);
248         signal(SIGPIPE, oncore);
249         init_re();
250         while (1) {
251 #ifdef TRACE
252                 while ((c = getopt(ac, (char **)av, "VU:Lc:Tvt:rlw:xRCsS")) !=
253                     EOF)
254 #else
255                 while ((c = getopt(ac, (char **)av,
```

```
256                     "VLc:vt:rlw:xRCsS")) != EOF)
257 #endif
258                         switch (c) {
259                         case 's':
260                                 hush = 1;
261                                 value(vi_AUTOPRINT) = 0;
262                                 fast++;
263                                 break;

265                         case 'R':
266                                 value(vi_READONLY) = 1;
267                                 break;
268                         case 'S':
269                                 tags_flag = 1;
270                                 break;
271 #ifdef TRACE
272                         case 'T':
273                                 tracef = (unsigned char *)"trace";
274                                 goto trace;

276                         case 'U':
277                                 tracef = tttrace;
278                                 strcpy(tracef, optarg);
279                 trace:
280                                 trace = fopen((char *)tracef, "w");
281 #define tracbuf NULL
282                                 if (trace == NULL)
283                                         viprintf("Trace create error\n");
284                                 else
285                                         setbuf(trace, (char *)tracbuf);
286                                 break;
287 #endif
288                         case 'c':
289                                 if (optarg != NULL)
290                                         firstpat = (unsigned char *)optarg;
291                                 else
292                                         firstpat = (unsigned char *)"";
293                                 break;

295                         case 'l':
296                                 value(vi_LISP) = 1;
297                                 value(vi_SHOWMATCH) = 1;
298                                 break;

300                         case 'r':
301                                 if (av[optind] && (c = av[optind][0]) &&
302                                     c != '-') {
303                                         if ((strlen(av[optind])) >=
304                                             sizeof (savedfile)) {
305                                                 (void) fprintf(stderr, gettext(
306                                                     "Recovered file name"
307                                                     " too long\n"));
308                                                 exit(1);
309                                         }

311                                         rcvname = (unsigned char *)av[optind];
312                                         optind++;
313                                 }

315                         case 'L':
316                                 recov++;
317                                 break;

319                         case 'V':
320                                 verbose = 1;
321                                 break;
```



```
323                         case 't':
324                                 if (toptseen) {
325                                         usage(cmdnam);
326                                         exit(1);
327                                 } else {
328                                         toptseen++;
329                                 }
330                                 itag = tagflg = 1; /* -t option */
331                                 if (strlcpy(lasttag, optarg,
332                                     sizeof (lasttag)) >= sizeof (lasttag)) {
333                                         (void) fprintf(stderr, gettext("Tag"
334                                             " file name too long\n"));
335                                         exit(1);
336                                 }
337                                 break;

339                         case 'w':
340                                 defwind = 0;
341                                 if (optarg[0] == NULL)
342                                         defwind = 3;
343                                 else for (cp = (unsigned char *)optarg;
344                                     isdigit(*cp); cp++)
345                                         defwind = 10*defwind + *cp - '0';
346                                 if (defwind < 0)
347                                         defwind = 3;
348                                 break;

350                         case 'C':
351                                 crflag = 1;
352                                 xflag = 1;
353                                 break;

355                         case 'x':
356                                 /* encrypted mode */
357                                 xflag = 1;
358                                 crflag = -1;
359                                 break;

361                         case 'v':
362                                 ivis = 1;
363                                 break;

365                         default:
366                                 usage(cmdnam);
367                                 exit(1);
368                         }
369                 if (av[optind] && av[optind][0] == '+' &&
370                     av[optind-1] && strcmp(av[optind-1], "--")) {
371                                 firstpat = (unsigned char *)&av[optind][1];
372                                 optind++;
373                                 continue;
374                 } else if (av[optind] && av[optind][0] == '-' &&
375                     av[optind-1] && strcmp(av[optind-1], "--")) {
376                         hush = 1;
377                         value(vi_AUTOPRINT) = 0;
378                         fast++;
379                         optind++;
380                         continue;
381                 }
382                 break;
383         }

385         if (isatty(0) == 0) {
386                 /*
387                  * If -V option is set and input is coming in via
```

```
388                  * stdin then vi behavior should be ignored. The vi
389                  * command should act like ex and only process ex commands
390                  * and echo the input ex commands to stderr
391                  */
392                 if (verbose == 1) {
393                         ivis = 0;
394                 }
395
396                 /*
397                  * If the standard input is not a terminal device,
398                  * it is as if the -s option has been specified.
399                  */
400                 if (ivis == 0) {
401                         hush = 1;
402                         value(vi_AUTOPRINT) = 0;
403                         fast++;
404                 }
405         }
406
407         ac -= optind;
408         av  = &av[optind];
409
410         for (argcounter = 0; argcounter < ac; argcounter++) {
411                 if ((strlen(av[argcounter])) >= sizeof (savedfile)) {
412                         (void) fprintf(stderr, gettext("File argument"
413                             " too long\n"));
414                         exit(1);
415                 }
416         }
417
418 #ifdef SIGTSTP
419         if (!hush && signal(SIGTSTP, SIG_IGN) == SIG_DFL)
420                 signal(SIGTSTP, onsusp), dosusp++;
421 #endif
422
423         if (xflag) {
424                 permflag = 1;
425                 if ((kflag = run_setkey(perm,
426                     (key = (unsigned char *)getpass(
427                     gettext("Enter key:")))) == -1) {
428                         kflag = 0;
429                         xflag = 0;
430                         smerror(gettext("Encryption facility not available\n"));
431                 }
432                 if (kflag == 0)
433                         crflag = 0;
434                 else {
435                         strcpy(cryptkey, "CrYpTkEy=XXXXXXXXX");
436                         strcpy(cryptkey + 9, key);
437                         if (putenv((char *)cryptkey) != 0)
438                         smerror(gettext(" Cannot copy key to environment"));
439                 }
440
441         }
442 #ifndef PRESUNEUC
443         /*
444          * Perform locale-specific initialization
445          */
446         localize();
447 #endif /* PRESUNEUC */
448
449         /*
450          * Initialize end of core pointers.
451          * Normally we avoid breaking back to fendcore after each
452          * file since this can be expensive (much core-core copying).
453          * If your system can scatter load processes you could do
```



```
454          * this as ed does, saving a little core, but it will probably
455          * not often make much difference.
456          */
457         fendcore = (line *) sbrk(0);
458         endcore = fendcore - 2;
459
460         /*
461          * If we are doing a recover and no filename
462          * was given, then execute an exrecover command with
463          * the -r option to type out the list of saved file names.
464          * Otherwise set the remembered file name to the first argument
465          * file name so the "recover" initial command will find it.
466          */
467         if (recov) {
468                 if (ac == 0 && (rcvname == NULL || *rcvname == NULL)) {
469                         ppid = 0;
470                         setrupt();
471                         execlp(EXRECOVER, "exrecover", "-r", (char *)0);
472                         filioerr(EXRECOVER);
473                         exit(++errcnt);
474                 }
475                 if (rcvname && *rcvname)
476                         (void) strlcpy(savedfile, rcvname, sizeof (savedfile));
477                 else {
478                         (void) strlcpy(savedfile, *av++, sizeof (savedfile));
479                         ac--;
480                 }
481         }
482
483         /*
484          * Initialize the argument list.
485          */
486         argv0 = (unsigned char **)av;
487         argc0 = ac;
488         args0 = (unsigned char *)av[0];
489         erewind();
490
491         /*
492          * Initialize a temporary file (buffer) and
493          * set up terminal environment.  Read user startup commands.
494          */
495         if (setexit() == 0) {
496                 setrupt();
497                 intty = isatty(0);
498                 value(vi_PROMPT) = intty;
499                 if (((cp = (unsigned char *)getenv("SHELL")) != NULL) &&
500                     (*cp != '\0')) {
501                         if (strlen(cp) < sizeof (shell)) {
502                                 (void) strlcpy(shell, cp, sizeof (shell));
503                         }
504                 }
505                 if (fast)
506                         setterm((unsigned char *)"dumb");
507                 else {
508                         gettmode();
509                         cp = (unsigned char *)getenv("TERM");
510                         if (cp == NULL || *cp == '\0')
511                                 cp = (unsigned char *)"unknown";
512                         setterm(cp);
513                 }
514         }
515
516         /*
517          * Bring up some code from init()
518          * This is still done in init later. This
519          * avoids null pointer problems
```

```
520          */

522         dot = zero = truedol = unddol = dol = fendcore;
523         one = zero+1;
524         {
525         int i;

527         for (i = 0; i <= 'z'-'a'+1; i++)
528                 names[i] = 1;
529         }

531         if (setexit() == 0 && !fast) {
532                 if ((globp =
533                     (unsigned char *) getenv("EXINIT")) && *globp) {
534                         if (ivis)
535                                 inexrc = 1;
536                         commands(1, 1);
537                         inexrc = 0;
538                 } else {
539                         globp = 0;
540                         if ((cp = (unsigned char *) getenv("HOME")) !=
541                             0 && *cp) {
542                                 strncpy(scratch, cp, sizeof (scratch) - 1);
543                                 strncat(scratch, "/.exrc",
544                                     sizeof (scratch) - 1 - strlen(scratch));
545                                 if (ivis)
546                                         inexrc = 1;
547                                 if ((vret = validate_exrc(scratch)) == 0) {
548                                         source(scratch, 1);
549                                 } else {
550                                         if (vret == -1) {
551                                                 error(gettext(
552                                                     "Not owner of .exrc "
553                                                     "or .exrc is group or "
554                                                     "world writable"));
555                                         }
556                                 }
557                                 inexrc = 0;
558                         }
559                 }

561                 /*
562                  * Allow local .exrc if the "exrc" option was set. This
563                  * loses if . is $HOME, but nobody should notice unless
564                  * they do stupid things like putting a version command
565                  * in .exrc.
566                  * Besides, they should be using EXINIT, not .exrc, right?
567                  */

569                 if (value(vi_EXRC)) {
570                         if (ivis)
571                                 inexrc = 1;
572                         if ((cp = (unsigned char *) getenv("PWD")) != 0 &&
573                             *cp) {
574                                 strncpy(exrcpath, cp, sizeof (exrcpath) - 1);
575                                 strncat(exrcpath, "/.exrc",
576                                     sizeof (exrcpath) - 1 - strlen(exrcpath));
577                                 if (strcmp(scratch, exrcpath) != 0) {
578                                         if ((vret =
579                                             validate_exrc(exrcpath)) == 0) {
580                                                 source(exrcpath, 1);
581                                         } else {
582                                                 if (vret == -1) {
583                                                         error(gettext(
584                                                             "Not owner of "
585                                                             ".exrc or .exrc "
```



```
586                                                             "is group or world "
587                                                             "writable"));
588                                                 }
589                                         }
590                                 }
591                         }
592                         inexrc = 0;
593                 }
594         }

596         init(); /* moved after prev 2 chunks to fix directory option */

598         /*
599          * Initial processing.  Handle tag, recover, and file argument
600          * implied next commands.  If going in as 'vi', then don't do
601          * anything, just set initev so we will do it later (from within
602          * visual).
603          */
604         if (setexit() == 0) {
605                 if (recov)
606                         globp = (unsigned char *)"recover";
607                 else if (itag) {
608                         globp = ivis ? (unsigned char *)"tag" :
609                             (unsigned char *)"tag|p";
610 #ifdef  XPG4
611                         if (firstpat != NULL) {
612                                 /*
613                                  * if the user specified the -t and -c
614                                  * flags together, then we service these
615                                  * commands here. -t is handled first.
616                                  */
617                                 savepat = firstpat;
618                                 firstpat = NULL;
619                                 inglobal = 1;
620                                 commands(1, 1);

622                                 /* now handle the -c argument: */
623                                 globp = savepat;
624                                 commands(1, 1);
625                                 inglobal = 0;
626                                 globp = savepat = NULL;

628                                 /* the above isn't sufficient for ex mode: */
629                                 if (!ivis) {
630                                         setdot();
631                                         nonzero();
632                                         plines(addr1, addr2, 1);
633                                 }
634                         }
635 #endif /* XPG4 */
636                 } else if (argc)
637                         globp = (unsigned char *)"next";
638                 if (ivis)
639                         initev = globp;
640                 else if (globp) {
641                         inglobal = 1;
642                         commands(1, 1);
643                         inglobal = 0;
644                 }
645         }

647         /*
648          * Vi command... go into visual.
649          */
650         if (ivis) {
651                 /*
```

```
652                  * Don't have to be upward compatible
653                  * by starting editing at line $.
654                  */
655 #ifdef  XPG4
656                 if (!itag && (dol > zero))
657 #else /* XPG4 */
658                 if (dol > zero)
659 #endif /* XPG4 */
660                         dot = one;
661                 globp = (unsigned char *)"visual";
662                 if (setexit() == 0)
663                         commands(1, 1);
664         }

666         /*
667          * Clear out trash in state accumulated by startup,
668          * and then do the main command loop for a normal edit.
669          * If you quit out of a 'vi' command by doing Q or ^\,
670          * you also fall through to here.
671          */
672         seenprompt = 1;
673         ungetchar(0);
674         globp = 0;
675         initev = 0;
676         setlastchar('\n');
677         setexit();
678         commands(0, 0);
679         cleanup(1);
680         return (errcnt);
681 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   18817 Wed Aug 13 19:36:50 2014
new/usr/src/cmd/vi/port/ex.h
5088 it's probably ok for vi to stop working around pdp-11 bugs now
**********************************************************
______unchanged_portion_omitted_

 142 #define ONOFF   0
 143 #define NUMERIC 1
 144 #define STRING  2               /* SHELL or DIRECTORY */
 145 #define OTERM   3

 147 #define value(a)        options[a].ovalue
 148 #define svalue(a)       options[a].osvalue

 150 extern   struct option options[vi_NOPTS + 1];


 153 /*
 154  * The editor does not normally use the standard i/o library.  Because
 155  * we expect the editor to be a heavily used program and because it
 156  * does a substantial amount of input/output processing it is appropriate
 157  * for it to call low level read/write primitives directly.  In fact,
 158  * when debugging the editor we use the standard i/o library.  In any
 159  * case the editor needs a printf which prints through "putchar" ala the
 160  * old version 6 printf.  Thus we normally steal a copy of the "printf.c"
 161  * and "strout" code from the standard i/o library and mung it for our
 162  * purposes to avoid dragging in the stdio library headers, etc if we
 163  * are not debugging.  Such a modified printf exists in "printf.c" here.
 164  */
 165 #ifdef TRACE
 166 #include <stdio.h>
 167         var     FILE    *trace;
 168         var     bool    trubble;
 169         var     bool    techoin;
 170         var     unsigned char   tracbuf[BUFSIZ];
 171 #undef  putchar
 172 #undef  getchar
 173 #else
 174 /*
 175  * Warning: do not change BUFSIZ without also changing LBSIZE in ex_tune.h
 176  * Running with BUFSIZ set to anything besides what is in <stdio.h> is
 177  * not recommended, if you use stdio.
 178  */
 179 #ifdef u370
 180 #define BUFSIZE 4096
 181 #else
 182 #define BUFSIZE (LINE_MAX*2)
 183 #endif
 184 #undef  NULL
 185 #define NULL    0
 186 #undef  EOF
 187 #define EOF     -1
 188 #endif

 190 /*
 191  * Character constants and bits
 192  *
 193  * The editor uses the QUOTE bit as a flag to pass on with characters
 194  * e.g. to the putchar routine.  The editor never uses a simple char variable.
 195  * Only arrays of and pointers to characters are used and parameters and
 196  * registers are never declared character.
 197  */
 198 #define QUOTE   020000000000
 199 #define TRIM    017777777777
 200 #define NL      '\n'
```



```
 201 #define CR      '\r'
 202 #define DELETE  0177            /* See also ATTN, QUIT in ex_tune.h */
 203 #define ESCAPE  033
 204 #undef  CTRL
 205 #define CTRL(c) (c & 037)

 207 /*
 208  * Miscellaneous random variables used in more than one place
 209  */
 210 var bool multibyte;
 211 var     bool    aiflag;         /* Append/change/insert with autoindent */
 212 var     bool    tagflg;         /* set for -t option and :tag command */
 213 var     bool    anymarks;       /* We have used '[a-z] */
 214 var     int     chng;           /* Warn "No write" */
 215 var     unsigned char   *Command;
 216 var     short   defwind;        /* -w# change default window size */
 217 var     int     dirtcnt;        /* When >= MAXDIRT, should sync temporary */
 218 #ifdef SIGTSTP
 219 var     bool    dosusp;         /* Do SIGTSTP in visual when ^Z typed */
 220 #endif
 221 var     bool    edited;         /* Current file is [Edited] */
 222 var     line    *endcore;       /* Last available core location */
 223 extern   bool   endline;        /* Last cmd mode command ended with \n */
 224 var     line    *fendcore;      /* First address in line pointer space */
 225 var     unsigned char   file[FNSIZE];   /* Working file name */
 226 var     unsigned char   genbuf[LBSIZE]; /* Working buffer when manipulating line
 227 var     bool    hush;           /* Command line option - was given, hush up! */
 228 var     unsigned char   *globp;         /* (Untyped) input string to command mod
 229 var     bool    holdcm;         /* Don't cursor address */
 230 var     bool    inappend;       /* in ex command append mode */
 231 var     bool    inglobal;       /* Inside g//... or v//... */
 232 var     unsigned char   *initev;        /* Initial : escape for visual */
 233 var     bool    inopen;         /* Inside open or visual */
 234 var     unsigned char   *input;         /* Current position in cmd line input bu
 235 var     bool    intty;          /* Input is a tty */
 236 var     short   io;             /* General i/o unit (auto-closed on error!) */
 237 extern   short  lastc;          /* Last character ret'd from cmd input */
 238 var     bool    laste;          /* Last command was an "e" (or "rec") */
 239 var     unsigned char   lastmac;        /* Last macro called for ** */
 240 var     unsigned char   lasttag[TAGSIZE];       /* Last argument to a tag comman
 241 var     unsigned char   *linebp;        /* Used in substituting in \n */
 242 var     unsigned char   linebuf[LBSIZE];        /* The primary line buffer */
 243 var     bool    listf;          /* Command should run in list mode */
 244 var     line    names['z'-'a'+2];       /* Mark registers a-z,' */
 245 var     int     notecnt;        /* Count for notify (to visual from cmd) */
 246 var     bool    numberf;        /* Command should run in number mode */
 247 var     unsigned char   obuf[BUFSIZE];  /* Buffer for tty output */
 248 var     short   oprompt;        /* Saved during source */
 249 var     short   ospeed;         /* Output speed (from gtty) */
 250 var     int     otchng;         /* Backup tchng to find changes in macros */
 251 var     int     peekc;          /* Peek ahead character (cmd mode input) */
 252 var     unsigned char   *pkill[2];      /* Trim for put with ragged (LISP) delet
 253 var     bool    pfast;          /* Have stty -nl'ed to go faster */
 254 var     pid_t   pid;            /* Process id of child */
 255 var     pid_t   ppid;           /* Process id of parent (e.g. main ex proc) */
 256 var     jmp_buf resetlab;       /* For error throws to top level (cmd mode) */
 257 var     pid_t   rpid;           /* Pid returned from wait() */
 258 var     bool    ruptible;       /* Interruptible is normal state */
 259 var     bool    seenprompt;     /* 1 if have gotten user input */
 260 var     bool    shudclob;       /* Have a prompt to clobber (e.g. on ^D) */
 261 var     int     status;         /* Status returned from wait() */
 262 var     int     tchng;          /* If nonzero, then [Modified] */
 263 extern  short   tfile;          /* Temporary file unit */
 264 var     bool    vcatch;         /* Want to catch an error (open/visual) */
 265 var     jmp_buf vreslab;        /* For error throws to a visual catch */
 266 var     bool    writing;        /* 1 if in middle of a file write */
```

```
267 var     int     xchng;          /* Suppresses multiple "No writes" in !cmd */
268 #ifndef PRESUNEUC
269 var     char    mc_filler;      /* Right margin filler for multicolumn char */
270 var     bool    mc_wrap;        /* Multicolumn character wrap at right margin */
271 #endif /* PRESUNEUC */
272 var     int     inexrc;         /* boolean: in .exrc initialization */

274 extern  int     termiosflag;    /* flag for using termios */

276 /*
277  * Macros
278  */
279 #define CP(a, b)        ((void)strcpy(a, b))
280                         /*
281                          * FIXUNDO: do we want to mung undo vars?
282                          * Usually yes unless in a macro or global.
283                          */
284 #define FIXUNDO         (inopen >= 0 && (inopen || !inglobal))
285 #define ckaw()          {if (chng && value(vi_AUTOWRITE) && !value(vi_READONLY))
286                                 wop(0);\
287                         }
288 #define copy(a,b,c)     Copy((char *) (a), (char *) (b), (c))
289 #define eq(a, b)        ((a) && (b) && strcmp(a, b) == 0)
290 #define getexit(a)      copy(a, resetlab, sizeof (jmp_buf))
291 #define lastchar()      lastc
292 #define outchar(c)      (*Outchar)(c)
293 #define pastwh()        ((void)skipwh())
294 #define pline(no)       (*Pline)(no)
295 #define reset()         longjmp(resetlab,1)
296 #define resexit(a)      copy(resetlab, a, sizeof (jmp_buf))
297 #define setexit()       setjmp(resetlab)
298 #define setlastchar(c)  lastc = c
299 #define ungetchar(c)    peekc = c

301 #define CATCH           vcatch = 1; if (setjmp(vreslab) == 0) {
302 #define ONERR           } else { vcatch = 0;
303 #define ENDCATCH        } vcatch = 0;

305 /*
306  * Environment like memory
307  */
308 var     unsigned char   altfile[FNSIZE];        /* Alternate file name */
309 extern  unsigned char   direct[ONMSZ];          /* Temp file goes here */
310 extern  unsigned char   shell[ONMSZ];           /* Copied to be settable */
311 var     unsigned char   uxb[UXBSIZE + 2];       /* Last !command for !! */

313 /*
314  * The editor data structure for accessing the current file consists
315  * of an incore array of pointers into the temporary file tfile.
316  * Each pointer is 15 bits (the low bit is used by global) and is
317  * padded with zeroes to make an index into the temp file where the
318  * actual text of the line is stored.
319  *
320  * To effect undo, copies of affected lines are saved after the last
321  * line considered to be in the buffer, between dol and unddol.
322  * During an open or visual, which uses the command mode undo between
323  * dol and unddol, a copy of the entire, pre-command buffer state
324  * is saved between unddol and truedol.
325  */
326 var     line    *addr1;                 /* First addressed line in a command */
327 var     line    *addr2;                 /* Second addressed line */
328 var     line    *dol;                   /* Last line in buffer */
329 var     line    *dot;                   /* Current line */
330 var     line    *one;                   /* First line */
331 var     line    *truedol;               /* End of all lines, including saves */
332 var     line    *unddol;                /* End of undo saved lines */
```



```
333 var     line    *zero;                  /* Points to empty slot before one */

335 /*
336  * Undo information
337  *
338  * For most commands we save lines changed by salting them away between
339  * dol and unddol before they are changed (i.e. we save the descriptors
340  * into the temp file tfile which is never garbage collected).  The
341  * lines put here go back after unddel, and to complete the undo
342  * we delete the lines [undap1,undap2).
343  *
344  * Undoing a move is much easier and we treat this as a special case.
345  * Similarly undoing a "put" is a special case for although there
346  * are lines saved between dol and unddol we don't stick these back
347  * into the buffer.
348  */
349 var     short   undkind;

351 var     line    *unddel;        /* Saved deleted lines go after here */
352 var     line    *undap1;        /* Beginning of new lines */
353 var     line    *undap2;        /* New lines end before undap2 */
354 var     line    *undadot;       /* If we saved all lines, dot reverts here */

356 #define UNDCHANGE       0
357 #define UNDMOVE         1
358 #define UNDALL          2
359 #define UNDNONE         3
360 #define UNDPUT          4

362 /*
363  * Various miscellaneous flags and buffers needed by the encryption routines.
364  */
365 #define KSIZE   9       /* key size for encryption */
366 var     int     xflag;          /* True if we are in encryption mode */
367 var     int     xtflag;         /* True if the temp file is being encrypted */
368 var     int     kflag;          /* True if the key has been accepted */
369 var     int     crflag;         /* True if the key has been accepted  and the fi
370                                    being read is ciphertext
371                                  */
372 var     int     perm[2];        /* pipe connection to crypt for file being edite
373 var     int     tperm[2];       /* pipe connection to crypt for temporary file *
374 var     int permflag;
375 var     int tpermflag;
376 var     unsigned char   *key;
377 var     unsigned char   crbuf[CRSIZE];
378 char    *getpass();

380 var     bool    write_quit;     /* True if executing a 'wq' command */
381 var     int     errcnt;         /* number of error/warning messages in */
382                                 /*      editing session (global flag)  */
383 /*
384  * Function type definitions
385  */
386 #define NOSTR   (char *) 0
387 #define NOLINE  (line *) 0

389 #define setterm visetterm
390 #define draino vidraino
391 #define gettmode vigettmode

393 extern  int     (*Outchar)();
394 extern  int     (*Pline)();
395 extern  int     (*Putchar)();
396 var     void    (*oldhup)();
397 int     (*setlist())();
398 int     (*setnorm())();
```

```
399 int     (*setnorm())();
400 int     (*setnumb())();
401 #ifndef PRESUNEUC
402 int     (*wdwc)(wchar_t);       /* tells kind of word character */
403 int     (*wdbdg)(wchar_t, wchar_t, int);        /* tells word binding force */
404 wchar_t *(*wddlm)(wchar_t, wchar_t, int);       /* tells desired delimiter */
405 wchar_t (*mcfllr)(void);        /* tells multicolumn filler character */
406 #endif /* PRESUNEUC */
407 line    *address();
408 unsigned char   *cgoto();
409 unsigned char   *genindent();
410 unsigned char   *getblock();
411 char    *getenv();
412 line    *getmark();
413 unsigned char   *mesg();
414 unsigned char   *place();
415 unsigned char   *plural();
416 line    *scanfor();
417 void setin(line *);
418 unsigned char   *strend();
419 unsigned char   *tailpath();
420 char    *tgetstr();
421 char    *tgoto();
422 char    *ttyname();
423 line    *vback();
424 unsigned char   *vfindcol();
425 unsigned char   *vgetline();
426 unsigned char   *vinit();
427 unsigned char   *vpastwh();
428 unsigned char   *vskipwh();
429 int     put(void);
430 int     putreg(unsigned char);
431 int     YANKreg(int);
432 int     delete(bool);
433 int     vi_filter();
434 int     getfile();
435 int     getsub();
436 int     gettty();
437 int     join(int);
438 int     listchar(wchar_t);
439 int     normchar(wchar_t);
440 int     normline(void);
441 int     numbline(int);
442 var     void    (*oldquit)();

443 #ifdef __STDC__
444 void    onhup(int);
445 void    onintr(int);
446 void    onemt(int);
446 void    oncore(int);
447 #ifdef CBREAK
448 void    vintr(int);
449 #endif
450 void    onsusp(int);
451 int     putch(char);
452 int     plodput(char);
453 int     vputch(char);
455 #else
456 void    onhup();
457 void    onintr();
458 void    onemt();
459 void    oncore();
460 #ifdef CBREAK
461 void    vintr();
462 #endif
463 void    onsusp();
```



```
464 int     putch();
465 int     plodput();
466 int     vputch();
467 #endif /* __STDC__ */

455 void    shift(int, int);
456 int     termchar(wchar_t);
457 int     vfilter();
458 int     vshftop();
459 int     yank(void);
460 unsigned char *lastchr();
461 unsigned char *nextchr();
462 bool putoctal;

464 void    error();
465 void    error0(void);
466 void error1(unsigned char *);
467 void fixol(void);
468 void resetflav(void);
469 void serror(unsigned char *, unsigned char *);
470 void setflav(void);
471 void tailprim(unsigned char *, int, bool);
472 void vcontin(bool);
473 void squish(void);
474 void move1(int, line *);
475 void pragged(bool);
476 void zop2(int, int);
477 void plines(line *, line *, bool);
478 void pofix(void);
479 void undo(bool);
480 void somechange(void);
481 void savetag(char *);
482 void unsavetag(void);
483 void checkjunk(unsigned char);
484 void getone(void);
485 void rop3(int);
486 void rop2(void);
487 void putfile(int);
488 void wrerror(void);
489 void clrstats(void);
490 void slobber(int);
491 void flush(void);
492 void flush1(void);
493 void flush2(void);
494 void fgoto(void);
495 void flusho(void);
496 void comprhs(int);
497 int dosubcon(bool, line *);
498 void ugo(int, int);
499 void dosub(void);
500 void snote(int, int);
501 void cerror(unsigned char *);
502 void unterm(void);
503 int setend(void);
504 void prall(void);
505 void propts(void);
506 void propt(struct option *);
507 void killcnt(int);
508 void markpr(line *);
509 void merror1(unsigned char *);
510 void notempty(void);
511 int qcolumn(unsigned char *, unsigned char *);
512 void netchange(int);
513 void putmk1(line *, int);
514 int nqcolumn(unsigned char *, unsigned char *);
515 void syserror(int);
```

```
 516 void cleanup(bool);
 517 void blkio(short, unsigned char *, int (*)());
 518 void tflush(void);
 519 short partreg(unsigned char);
 520 void kshift(void);
 521 void YANKline(void);
 522 void rbflush(void);
 523 void waitfor(void);
 524 void ovbeg(void);
 525 void fixzero(void);
 526 void savevis(void);
 527 void undvis(void);
 528 void setwind(void);
 529 void vok(wchar_t *, int);
 530 void vsetsiz(int);
 531 void vinslin(int, int, int);
 532 void vopenup(int, bool, int);
 533 void vadjAL(int, int);
 534 void vup1(void);
 535 void vmoveitup(int, bool);
 536 void vscroll(int);
 537 void vscrap(void);
 538 void vredraw(int);
 539 void vdellin(int, int, int);
 540 void vadjDL(int, int);
 541 void vsyncCL(void);
 542 void vsync(int);
 543 void vsync1(int);
 544 void vcloseup(int, int);
 545 void sethard(void);
 546 void vdirty(int, int);
 547 void setBUF(unsigned char *);
 548 void addto(unsigned char *, unsigned char *);
 549 void macpush();
 550 void setalarm(void);
 551 void cancelalarm(void);
 552 void grabtag(void);
 553 void prepapp(void);
 554 void vremote();
 555 void vsave(void);
 556 void vzop(bool, int, int);
 557 void warnf();
 558 int wordof(unsigned char, unsigned char *);
 559 void setpk(void);
 560 void back1(void);
 561 void vdoappend(unsigned char *);
 562 void vclrbyte(wchar_t *, int);
 563 void vclreol(void);
 564 void vsetcurs(unsigned char *);
 565 void vigoto(int, int);
 566 void vcsync(void);
 567 void vgotoCL(int);
 568 void vgoto(int, int);
 569 void vmaktop(int, wchar_t *);
 570 void vrigid(void);
 571 void vneedpos(int);
 572 void vnpins(int);
 573 void vishft(void);
 574 void viin(wchar_t);
 575 void godm(void);
 576 void enddm(void);
 577 void goim(void);
 578 void endim(void);
 579 void vjumpto(line *, unsigned char *, unsigned char);
 580 void vup(int, int, bool);
 581 void vdown(int, int, bool);
```



```
 582 void vcontext(line *, unsigned char);
 583 void vclean(void);
 584 void vshow(line *, line*);
 585 void vreset(bool);
 586 void vroll(int);
 587 void vrollR(int);
 588 void vnline(unsigned char *);
 589 void noerror();
 590 void getaline(line);
 591 void viprintf();
 592 void gettmode(void);
 593 void setterm(unsigned char *);
 594 void draino(void);
 595 int lfind();
 596 void source();
 597 void commands();
 598 void addmac();
 599 void vmoveto();
 600 void vrepaint();
 601 void getDOT(void);
 602 void vclear(void);

 604 unsigned char *lastchr();
 605 unsigned char *nextchr();
 606 bool putoctal;

 608 void setdot1(void);

 610 #ifdef __cplusplus
 611 }
```

_____***unchanged_portion_omitted_***

****
19640 Wed Aug 13 19:36:50 2014
new/usr/src/cmd/vi/port/ex_subr.c
5088 it's probably ok for vi to stop working around pdp-11 bugs now
****
_____**unchanged_portion_omitted_**

```
 926 /*
 927  * The following code is defensive programming against a bug in the
 928  * pdp-11 overlay implementation.  Sometimes it goes nuts and asks
 929  * for an overlay with some garbage number, which generates an emt
 930  * trap.  This is a less than elegant solution, but it is somewhat
 931  * better than core dumping and losing your work, leaving your tty
 932  * in a weird state, etc.
 933  */
 934 int _ovno;

 936 /*ARGSUSED*/
 937 void
 938 onemt(sig)
 939 int sig;
 940 {
 941         int oovno;

 943         signal(SIGEMT, onemt);
 944         oovno = _ovno;
 945         /* 2 and 3 are valid on 11/40 type vi, so */
 946         if (_ovno < 0 || _ovno > 3)
 947                 _ovno = 0;
 948         error(value(vi_TERSE) ? gettext("emt trap, _ovno is %d ") :
 949 gettext("emt trap, _ovno is %d   - try again"));
 950 }

 952 /*
 927  * When a hangup occurs our actions are similar to a preserve
 928  * command.  If the buffer has not been [Modified], then we do
 929  * nothing but remove the temporary files and exit.
 930  * Otherwise, we sync the temp file and then attempt a preserve.
 931  * If the preserve succeeds, we unlink our temp files.
 932  * If the preserve fails, we leave the temp files as they are
 933  * as they are a backup even without preservation if they
 934  * are not removed.
 935  */

 937 /*ARGSUSED*/
 938 void
 939 onhup(sig)
 940 int sig;
 941 {

 943         /*
 944          * USG tty driver can send multiple HUP's!!
 945          */
 946         signal(SIGINT, SIG_IGN);
 947         signal(SIGHUP, SIG_IGN);
 948         if (chng == 0) {
 949                 cleanup(1);
 950                 exit(++errcnt);
 951         }
 952         if (setexit() == 0) {
 953                 if (preserve()) {
 954                         cleanup(1);
 955                         exit(++errcnt);
 956                 }
 957         }
 958         if (kflag)
```



```
 959                 crypt_close(perm);
 960         if (xtflag)
 961                 crypt_close(tperm);
 962         exit(++errcnt);
 963 }
```
_____**unchanged_portion_omitted_**