```
#ifndef lint
static char sccsid[] = "@(#)vgrindefs.c 4.3 (Berkeley) 12/11/84";
#endif

/* $Id: vgrindefs.c,v 1.1 1998/12/27 23:50:16 elf Exp $*/

/* Copyright (c) 1979 Regents of the University of California */

#define BUFSIZ  4096                    /* 1024 gives "Vgrind entry too long" for tcl -elf */
#define MAXHOP  32      /* max number of tc= indirections */

#include <ctype.h>
/*
 * grindcap - routines for dealing with the language definitions data base
 *      (code stolen almost totally from termcap)
 *
 * BUG:         Should use a "last" pointer in tbuf, so that searching
 *              for capabilities alphabetically would not be a n**2/2
 *              process when large numbers of capabilities are given.
 * Note:        If we add a last pointer now we will screw up the
 *              tc capability. We really should compile termcap.
 *
 * Essentially all the work here is scanning and decoding escapes
 * in string capabilities.  We don't use stdio because the editor
 * doesn't, and because living w/o it is not hard.
 */

static  char *tbuf;
static  char *filename;
static  int hopcount;   /* detect infinite loops in termcap, init 0 */

/* some of these fns were re-declared as static, which generated
   compile-time warnings; deleted static keyword -elf*/

char    *tskip();
char    *tgetstr();
char    *tdecode();
char    *getenv();

/*
 * Get an entry for terminal name in buffer bp,
 * from the termcap file.  Parse is very rudimentary;
 * we just notice escaped newlines.
 */
tgetent(bp, name, file)                                          tgetent
        char *bp, *name, *file;
{
        register char *cp;
        register int c;
        register int i = 0, cnt = 0;
        char ibuf[BUFSIZ];
        char *cp2;
        int tf;
```

```c
	tbuf = bp;
	tf = 0;
	filename = file;
	tf = open(filename, 0);
	if (tf < 0)
		return (-1);
	for (;;) {
		cp = bp;
		for (;;) {
			if (i == cnt) {
				cnt = read(tf, ibuf, BUFSIZ);
				if (cnt <= 0) {
					close(tf);
					return (0);
				}
				i = 0;
			}
			c = ibuf[i++];
			if (c == '\n') {
				if (cp > bp && cp[-1] == '\\'){
					cp--;
					continue;
				}
				break;
			}
			if (cp >= bp+BUFSIZ) {
				write(2, "Vgrind entry too long\n", 23);
				break;
			} else
				*cp++ = c;
		}
		*cp = 0;

		/*
		 * The real work for the match.
		 */
		if (tnamatch(name)) {
			close(tf);
			return(tnchktc());
		}
	}
}

/*
 * tnchktc: check the last entry, see if it's tc=xxx. If so,
 * recursively find xxx and append that entry (minus the names)
 * to take the place of the tc=xxx entry. This allows termcap
 * entries to say "like an HP2621 but doesn't turn on the labels".
 * Note that this works because of the left to right scan.
 */
tnchktc()
{
```

tnchktc

```
	register char *p, *q;
	char tcname[16];		/* name of similar terminal */
	char tcbuf[BUFSIZ];
	char *holdtbuf = tbuf;
	int l;

	p = tbuf + strlen(tbuf) - 2;	/* before the last colon */
	while (*--p != ':')
		if (p<tbuf) {
			write(2, "Bad vgrind entry\n", 18);
			return (0);
		}
	p++;
	/* p now points to beginning of last field */
	if (p[0] != 't' || p[1] != 'c')
		return(1);
	strcpy(tcname,p+3);
	q = tcname;
	while (q && *q != ':')
		q++;
	*q = 0;
	if (++hopcount > MAXHOP) {
		write(2, "Infinite tc= loop\n", 18);
		return (0);
	}
	if (tgetent(tcbuf, tcname, filename) != 1)
		return(0);
	for (q=tcbuf; *q != ':'; q++)
		;
	l = p - holdtbuf + strlen(q);
	if (l > BUFSIZ) {
		write(2, "Vgrind entry too long\n", 23);
		q[BUFSIZ - (p-tbuf)] = 0;
	}
	strcpy(p, q+1);
	tbuf = holdtbuf;
	return(1);
}

/*
 * Tnamatch deals with name matching.  The first field of the termcap
 * entry is a sequence of names separated by |'s, so we compare
 * against each such name.  The normal : terminator after the last
 * name (before the first field) stops us.
 */
```

tnamatch

```
tnamatch(np)
	char *np;
{
	register char *Np, *Bp;

	Bp = tbuf;
	if (*Bp == '#')
		return(0);
```

```
	for (;;) {
		for (Np = np; *Np && *Bp == *Np; Bp++, Np++)
			continue;
		if (*Np == 0 && (*Bp == '|' || *Bp == ':' || *Bp == 0))
			return (1);
		while (*Bp && *Bp != ':' && *Bp != '|')
			Bp++;
		if (*Bp == 0 || *Bp == ':')
			return (0);
		Bp++;
	}
}

/*
 * Skip to the next field.
 */
char *
tskip(bp)                                                    tskip
	register char *bp;
{

	register char c;

	while (c = *bp++) {
	    if (c == ':' && *(bp-1) != '\\')
		break;
	}

	if (*bp == ':')
		bp++;
	return (bp);
}

/*
 * Return the (numeric) option id.
 * Numeric options look like
 *	li#80
 * i.e. the option string is separated from the numeric value by
 * a # character.  If the option is not found we return -1.
 * Note that we handle octal numbers beginning with 0.
 */
tgetnum(id)                                                  tgetnum
	char *id;
{
	register int i, base;
	register char *bp = tbuf;

	for (;;) {
		bp = tskip(bp);
		if (*bp == 0)
			return (-1);
		if (*bp++ != id[0] || *bp == 0 || *bp++ != id[1])
			continue;
```

```
			if (*bp == '@')
				return(-1);
			if (*bp != '#')
				continue;
			bp++;
			base = 10;
			if (*bp == '0')
				base = 8;
			i = 0;
			while (isdigit(*bp))
				i *= base, i += *bp++ - '0';
			return (i);
		}
}

/*
 * Handle a flag option.
 * Flag options are given "naked", i.e. followed by a : or the end
 * of the buffer.  Return 1 if we find the option, or 0 if it is
 * not given.
 */
tgetflag(id)                                                    tgetflag
	char *id;
{
	register char *bp = tbuf;

	for (;;) {
		bp = tskip(bp);
		if (!*bp)
			return (0);
		if (*bp++ == id[0] && *bp != 0 && *bp++ == id[1]) {
			if (!*bp || *bp == ':')
				return (1);
			else if (*bp == '@')
				return(0);
		}
	}
}

/*
 * Get a string valued option.
 * These are given as
 *	cl=^Z
 * Much decoding is done on the strings, and the strings are
 * placed in area, which is a ref parameter which is updated.
 * No checking on area overflow.
 */
char *
tgetstr(id, area)                                               tgetstr
	char *id, **area;
{
	register char *bp = tbuf;
	for (;;) {
```

```
		bp = tskip(bp);
		if (!*bp)
			return (0);
		if (*bp++ != id[0] || *bp == 0 || *bp++ != id[1])
			continue;
		if (*bp == '@')
			return(0);
		if (*bp != '=')
			continue;
		bp++;
		return (tdecode(bp, area));
	}
}

/*
 * Tdecode does the grung work to decode the
 * string capability escapes.
 */
char *
tdecode(str, area)
	register char *str;
	char **area;
{
	register char *cp;
	register int c;
	int i;

	cp = *area;
	while (c = *str++) {
	    if (c == ':' && *(cp-1) != '\\')
		break;
	    *cp++ = c;
	}
	*cp++ = 0;
	str = *area;
	*area = cp;
	return (str);
}
```

tdecode