```
**********************************************************
    1544 Sat May 28 21:32:37 2011
new/usr/src/cmd/iconv/Makefile
30 Need iconv
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
#

PROG=iconv
SHFILES=iconv_list

include ../Makefile.cmd

OBJS = iconv.o charmap.o parser.tab.o scanner.o

SRCS    = $(OBJS:%.o=%.c)

C99MODE=        $(C99_ENABLE)
LDLIBS          += -lgen
LDLIBS          += -lavl
YFLAGS          = -d -b parser
$(RELEASE_BUILD) CPPFLAGS += -DNDEBUG

CLEANFILES      = $(OBJS) parser.tab.c parser.tab.h
CLOBBERFILES    = $(PROG) $(POFILE)
PIFILES         = $(OBJS:%.o=%.i)
POFILE          = iconv_cmd.po

all: $(PROG) $(SHFILES)

ROOTLIBICONV    = $(ROOT)/usr/lib/iconv
ROOTLIBICONVSH  = $(SHFILES:%=$(ROOTLIBICONV)/%)
$(ROOTLIBICONVSH) := FILEMODE = 0555

install: all $(ROOTPROG) $(ROOTLIBICONV) $(ROOTLIBICONVSH)

$(PROG): $(OBJS)
        $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
        $(POST_PROCESS)

$(OBJS):        parser.tab.h

parser.tab.c parser.tab.h: parser.y
        $(YACC) $(YFLAGS) parser.y

lint:   $(SRCS)
        $(LINT.c) $(CPPFLAGS) $(SRCS)

clean:
        $(RM) $(CLEANFILES)

$(POFILE):      $(PIFILES)
        $(RM) $@
        $(RM) messages.po
```



```
        $(XGETTEXT) -s $(PIFILES)
        $(SED) -e '/domain/d' messages.po > $@
        $(RM) $(PIFILES) messages.po

$(ROOTLIBICONV):
        $(INS.dir)

$(ROOTLIBICONV)/%: %
        $(INS.file)

.KEEP_STATE:

include ../Makefile.targ
```

```
**********************************************************
   11277 Sat May 28 21:32:37 2011
new/usr/src/cmd/iconv/charmap.c
30 Need iconv
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

/*
 * CHARMAP file handling for iconv.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <limits.h>
#include <unistd.h>
#include <alloca.h>
#include <sys/avl.h>
#include <stddef.h>
#include <unistd.h>
#include "charmap.h"
#include "parser.tab.h"
#include <assert.h>

enum cmap_pass cmap_pass;
static avl_tree_t       cmap_sym;
static avl_tree_t       cmap_mbs;

typedef struct charmap {
        const char *name;
        struct charmap *alias_of;
        avl_node_t avl_sym;
        avl_node_t avl_mbs;
        int warned;
        int mbs_len;
        int tombs_len;
        char mbs[MB_LEN_MAX + 1]; /* input */
        char tombs[MB_LEN_MAX + 1]; /* output */
} charmap_t;

static void add_charmap_impl_fr(char *sym, char *mbs, int mbs_len, int nodups);
static void add_charmap_impl_to(char *sym, char *mbs, int mbs_len, int nodups);

/*
 * Array of POSIX specific portable characters.
 */
static const struct {
        char    *name;
        int     ch;
} portable_chars[] = {
        { "NUL",                '\0' },
        { "alert",              '\a' },
```



```
        { "backspace",          '\b' },
        { "tab",                '\t' },
        { "carriage-return",    '\r' },
        { "newline",            '\n' },
        { "vertical-tab",       '\v' },
        { "form-feed",          '\f' },
        { "space",              ' ' },
        { "exclamation-mark",   '!' },
        { "quotation-mark",     '"' },
        { "number-sign",        '#' },
        { "dollar-sign",        '$' },
        { "percent-sign",       '%' },
        { "ampersand",          '&' },
        { "apostrophe",         '\'' },
        { "left-parenthesis",   '(' },
        { "right-parenthesis",  ')' },
        { "asterisk",           '*' },
        { "plus-sign",          '+' },
        { "comma",              ',' },
        { "hyphen-minus",       '-' },
        { "hyphen",             '-' },
        { "full-stop",          '.' },
        { "period",             '.' },
        { "slash",              '/' },
        { "solidus",            '/' },
        { "zero",               '0' },
        { "one",                '1' },
        { "two",                '2' },
        { "three",              '3' },
        { "four",               '4' },
        { "five",               '5' },
        { "six",                '6' },
        { "seven",              '7' },
        { "eight",              '8' },
        { "nine",               '9' },
        { "colon",              ':' },
        { "semicolon",          ';' },
        { "less-than-sign",     '<' },
        { "equals-sign",        '=' },
        { "greater-than-sign",  '>' },
        { "question-mark",      '?' },
        { "commercial-at",      '@' },
        { "left-square-bracket", '[' },
        { "backslash",          '\\' },
        { "reverse-solidus",    '\\' },
        { "right-square-bracket", ']' },
        { "circumflex",         '^' },
        { "circumflex-accent",  '^' },
        { "low-line",           '_' },
        { "underscore",         '_' },
        { "grave-accent",       '`' },
        { "left-brace",         '{' },
        { "left-curly-bracket", '{' },
        { "vertical-line",      '|' },
        { "right-brace",        '}' },
        { "right-curly-bracket", '}' },
        { "tilde",              '~' },
        { "A", 'A' },
        { "B", 'B' },
        { "C", 'C' },
        { "D", 'D' },
        { "E", 'E' },
        { "F", 'F' },
        { "G", 'G' },
        { "H", 'H' },
        { "I", 'I' },
```

```
128         { "J", 'J' },
129         { "K", 'K' },
130         { "L", 'L' },
131         { "M", 'M' },
132         { "N", 'N' },
133         { "O", 'O' },
134         { "P", 'P' },
135         { "Q", 'Q' },
136         { "R", 'R' },
137         { "S", 'S' },
138         { "T", 'T' },
139         { "U", 'U' },
140         { "V", 'V' },
141         { "W", 'W' },
142         { "X", 'X' },
143         { "Y", 'Y' },
144         { "Z", 'Z' },
145         { "a", 'a' },
146         { "b", 'b' },
147         { "c", 'c' },
148         { "d", 'd' },
149         { "e", 'e' },
150         { "f", 'f' },
151         { "g", 'g' },
152         { "h", 'h' },
153         { "i", 'i' },
154         { "j", 'j' },
155         { "k", 'k' },
156         { "l", 'l' },
157         { "m", 'm' },
158         { "n", 'n' },
159         { "o", 'o' },
160         { "p", 'p' },
161         { "q", 'q' },
162         { "r", 'r' },
163         { "s", 's' },
164         { "t", 't' },
165         { "u", 'u' },
166         { "v", 'v' },
167         { "w", 'w' },
168         { "x", 'x' },
169         { "y", 'y' },
170         { "z", 'z' },
171         { NULL, 0 }
172 };

173 static int
174 cmap_compare_sym(const void *n1, const void *n2)
175 {
176         const charmap_t *c1 = n1;
177         const charmap_t *c2 = n2;
178         int rv;

179         rv = strcmp(c1->name, c2->name);
180         return ((rv < 0) ? -1 : (rv > 0) ? 1 : 0);
181 }

182 /*
183  * In order for partial match searches to work,
184  * we need these sorted by mbs contents.
185  */
186 static int
187 cmap_compare_mbs(const void *n1, const void *n2)
188 {
189         const charmap_t *c1 = n1;
190         const charmap_t *c2 = n2;
```



```
191         int len, rv;

192         len = c1->mbs_len;
193         if (len < c2->mbs_len)
194                 len = c2->mbs_len;
195         rv = memcmp(c1->mbs, c2->mbs, len);
196         if (rv < 0)
197                 return (-1);
198         if (rv > 0)
199                 return (1);
200         /* they match through length */
201         if (c1->mbs_len < c2->mbs_len)
202                 return (-1);
203         if (c2->mbs_len < c1->mbs_len)
204                 return (1);
205         return (0);
206 }

207 void
208 charmap_init(char *to_map, char *from_map)
209 {
210         avl_create(&cmap_sym, cmap_compare_sym, sizeof (charmap_t),
211             offsetof(charmap_t, avl_sym));

212         avl_create(&cmap_mbs, cmap_compare_mbs, sizeof (charmap_t),
213             offsetof(charmap_t, avl_mbs));

214         cmap_pass = CMAP_PASS_FROM;
215         reset_scanner(from_map);
216         (void) yyparse();
217         add_charmap_posix();

218         cmap_pass = CMAP_PASS_TO;
219         reset_scanner(to_map);
220         (void) yyparse();
221 }

222 void
223 charmap_dump()
224 {
225         charmap_t *cm;
226         int i;

227         cm = avl_first(&cmap_mbs);
228         while (cm != NULL) {
229                 (void) printf("name=\"%s\"\n", cm->name);

230                 (void) printf("\timbs=\"");
231                 for (i = 0; i < cm->mbs_len; i++)
232                         (void) printf("\\x%02x", cm->mbs[i] & 0xFF);
233                 (void) printf("\"\n");

234                 (void) printf("\tombs=\"");
235                 for (i = 0; i < cm->tombs_len; i++)
236                         (void) printf("\\x%02x", cm->tombs[i] & 0xFF);
237                 (void) printf("\"\n");

238                 cm = AVL_NEXT(&cmap_mbs, cm);
239         }
240 }

241 /*
242  * We parse two charmap files:  First the "from" map, where we build
243  * cmap_mbs and cmap_sym which we'll later use to translate the input
244  * stream (mbs encodings) to symbols.  Second, we parse the "to" map,
245  * where we fill in the tombs members of entries in cmap_sym, (which
```

```
 * must alread exist) used later to write the output encoding.
 */
static void
add_charmap_impl(char *sym, char *mbs, int mbs_len, int nodups)
{

        /*
         * While parsing both the "from" and "to" cmaps,
         * require both the symbol and encoding.
         */
        if (sym == NULL || mbs == NULL) {
                errf(_("invalid charmap entry"));
                return;
        }

        switch (cmap_pass) {
        case CMAP_PASS_FROM:
                add_charmap_impl_fr(sym, mbs, mbs_len, nodups);
                break;
        case CMAP_PASS_TO:
                add_charmap_impl_to(sym, mbs, mbs_len, nodups);
                break;
        default:
                assert(0);
                break;
        }
}

static void
add_charmap_impl_fr(char *sym, char *mbs, int mbs_len, int nodups)
{
        charmap_t       *m, *n, *s;
        avl_index_t     where_sym, where_mbs;

        if ((n = calloc(1, sizeof (*n))) == NULL) {
                errf(_("out of memory"));
                return;
        }
        n->name = sym;

        assert(0 < mbs_len && mbs_len <= MB_LEN_MAX);
        (void) memcpy(n->mbs, mbs, mbs_len);
        n->mbs_len = mbs_len;

        m = avl_find(&cmap_mbs, n, &where_mbs);
        s = avl_find(&cmap_sym, n, &where_sym);

        /*
         * If we found the symbol, this is a dup.
         */
        if (s != NULL) {
                if (nodups) {
                        warn(_("%s: duplicate character symbol"), sym);
                }
                free(n);
                return;
        }

        /*
         * If we found the mbs, the new one is an alias,
         * which we'll add _only_ to the symbol AVL.
         */
        if (m != NULL) {
                /* The new one is an alias of the original. */
                n->alias_of = m;
                avl_insert(&cmap_sym, n, where_sym);
```



```
                return;
        }

        avl_insert(&cmap_sym, n, where_sym);
        avl_insert(&cmap_mbs, n, where_mbs);
}

static void
add_charmap_impl_to(char *sym, char *mbs, int mbs_len, int nodups)
{
        charmap_t       srch = {0};
        charmap_t       *m;
        avl_index_t     where;

        assert(0 < mbs_len && mbs_len <= MB_LEN_MAX);

        srch.name = sym;

        m = avl_find(&cmap_sym, &srch, &where);
        if (m == NULL) {
                if (sflag == 0)
                        warn(_("%s: symbol not found"), sym);
                return;
        }
        if (m->alias_of != NULL) {
                m = m->alias_of;

                /* don't warn for dups with aliases */
                if (m->tombs_len != 0)
                        return;
        }

        if (m->tombs_len != 0) {
                if (nodups) {
                        warn(_("%s: duplicate encoding for"), sym);
                }
                return;
        }

        (void) memcpy(m->tombs, mbs, mbs_len);
        m->tombs_len = mbs_len;
}

void
add_charmap(char *sym, char *mbs)
{
        /* mbs[0] is the length */
        int mbs_len = *mbs++;
        assert(0 < mbs_len && mbs_len <= MB_LEN_MAX);
        add_charmap_impl(sym, mbs, mbs_len, 1);
}


void
add_charmap_range(char *ssym, char *esym, char *mbs)
{
        int     ls, le;
        int     si;
        int     sn, en;
        int     i;
        int     mbs_len;
        char    tmbs[MB_LEN_MAX+1];
        char    *mb_last;

        static const char *digits = "0123456789";
```

```
392         /* mbs[0] is the length */
393         mbs_len = *mbs++;
394         assert(0 < mbs_len && mbs_len <= MB_LEN_MAX);
395         (void) memcpy(tmbs, mbs, mbs_len);
396         mb_last = tmbs + mbs_len - 1;

398         ls = strlen(ssym);
399         le = strlen(esym);

401         if (((si = strcspn(ssym, digits)) == 0) || (si == ls) ||
402             (strncmp(ssym, esym, si) != 0) ||
403             (strspn(ssym + si, digits) != (ls - si)) ||
404             (strspn(esym + si, digits) != (le - si)) ||
405             ((sn = atoi(ssym + si)) > ((en = atoi(esym + si))))) {
406                 errf(_("malformed charmap range"));
407                 return;
408         }

410         ssym[si] = 0;
411         for (i = sn; i <= en; i++) {
412                 char *nn;
413                 (void) asprintf(&nn, "%s%0*u", ssym, ls - si, i);
414                 if (nn == NULL) {
415                         errf(_("out of memory"));
416                         return;
417                 }

419                 add_charmap_impl(nn, tmbs, mbs_len, 1);
420                 (*mb_last)++;
421         }
422         free(ssym);
423         free(esym);
424 }

426 void
427 add_charmap_char(char *name, int c)
428 {
429         char mbs[MB_LEN_MAX+1];

431         mbs[0] = c;
432         mbs[1] = '\0';
433         add_charmap_impl(name, mbs, 1, 0);
434 }

436 /*
437  * POSIX insists that certain entries be present, even when not in the
438  * orginal charmap file.
439  */
440 void
441 add_charmap_posix(void)
442 {
443         char    i;

445         for (i = 0; portable_chars[i].name; i++) {
446                 add_charmap_char(portable_chars[i].name, portable_chars[i].ch);
447         }
448 }

450 static charmap_t *
451 find_mbs(const char *mbs, int len)
452 {
453         charmap_t srch = {0};
454         charmap_t *cm;

456         while (len > 0) {
457                 (void) memcpy(srch.mbs, mbs, len);
```



```
458                 srch.mbs_len = len;
459                 cm = avl_find(&cmap_mbs, &srch, NULL);
460                 if (cm != NULL)
461                         break;
462                 len--;
463         }

465         return (cm);
466 }

468 /*
469  * Return true if this sequence matches the initial part
470  * of any sequence known in this charmap.
471  */
472 static boolean_t
473 find_mbs_partial(const char *mbs, int len)
474 {
475         charmap_t srch = {0};
476         charmap_t *cm;
477         avl_index_t where;

479         (void) memcpy(srch.mbs, mbs, len);
480         srch.mbs_len = len;
481         cm = avl_find(&cmap_mbs, &srch, &where);
482         if (cm != NULL) {
483                 /* full match - not expected, but OK */
484                 return (B_TRUE);
485         }
486         cm = avl_nearest(&cmap_mbs, where, AVL_AFTER);
487         if (cm != NULL && 0 == memcmp(cm->mbs, mbs, len))
488                 return (B_TRUE);

490         return (B_FALSE);
491 }

493 /*
494  * Do like iconv(3), but with charmaps.
495  */
496 size_t
497 cm_iconv(const char **iptr, size_t *ileft, char **optr, size_t *oleft)
498 {
499         charmap_t *cm;
500         int mbs_len;

502         /* Ignore state reset requests. */
503         if (iptr == NULL || *iptr == NULL)
504                 return (0);

506         if (*oleft < MB_LEN_MAX) {
507                 errno = E2BIG;
508                 return ((size_t)-1);
509         }

511         while (*ileft > 0 && *oleft >= MB_LEN_MAX) {
512                 mbs_len = MB_LEN_MAX;
513                 if (mbs_len > *ileft)
514                         mbs_len = *ileft;
515                 cm = find_mbs(*iptr, mbs_len);
516                 if (cm == NULL) {
517                         if (mbs_len < MB_LEN_MAX &&
518                             find_mbs_partial(*iptr, mbs_len)) {
519                                 /* incomplete sequence */
520                                 errno = EINVAL;
521                         } else {
522                                 errno = EILSEQ;
523                         }
```

```
524                         return ((size_t)-1);
525                 }
526                 assert(cm->mbs_len > 0);
527                 if (cm->tombs_len == 0) {
528                         if (sflag == 0 && cm->warned == 0) {
529                                 cm->warned = 1;
530                                 fprintf(stderr, gettext(
531                                     "To-map does not encode <%s>\n"),
532                                     cm->name);
533                         }
534                         if (cflag == 0) {
535                                 errno = EILSEQ;
536                                 return ((size_t)-1);
537                         }
538                         /* just skip this input seq. */
539                         *iptr  += cm->mbs_len;
540                         *ileft -= cm->mbs_len;
541                         continue;
542                 }

544                 *iptr  += cm->mbs_len;
545                 *ileft -= cm->mbs_len;
546                 (void) memcpy(*optr, cm->tombs, cm->tombs_len);
547                 *optr  += cm->tombs_len;
548                 *oleft -= cm->tombs_len;
549         }

551         return (0);
552 }
```

```
**********************************************************
    1534 Sat May 28 21:32:38 2011
new/usr/src/cmd/iconv/charmap.h
30 Need iconv
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy is of the CDDL is also available via the Internet
 * at http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

/*
 * CHARMAP file handling for iconv.
 */

/* Common header files. */
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <sys/types.h>
#include <libintl.h>

enum cmap_pass {
        CMAP_PASS_FROM,
        CMAP_PASS_TO };

extern int com_char;
extern int esc_char;
extern int mb_cur_max;
extern int mb_cur_min;
extern int last_kw;
extern int verbose;
extern int yydebug;
extern int lineno;
extern int debug;
extern int warnings;
extern int cflag;
extern int sflag;

int yyparse(void);
void yyerror(const char *);
void errf(const char *, ...);
void warn(const char *, ...);

void reset_scanner(const char *);
void scan_to_eol(void);

/* charmap.c - CHARMAP handling */
void init_charmap(void);
void add_charmap(char *, char *);
void add_charmap_posix(void);
void add_charmap_range(char *, char *, char *);

void charmap_init(char *to, char *fr);
size_t cm_iconv(const char **iptr, size_t *ileft, char **optr, size_t *oleft);
void charmap_dump(void);
```



```
#define _(x)    gettext(x)
#define INTERR  errf(_("internal fault (%s:%d)"), __FILE__, __LINE__)
```

```
**********************************************************
    5971 Sat May 28 21:32:39 2011
new/usr/src/cmd/iconv/iconv.c
30 Need iconv
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

/*
 * iconv(1) command.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <limits.h>
#include <iconv.h>
#include <libintl.h>
#include <langinfo.h>
#include <locale.h>
#include "charmap.h"

#include <assert.h>

const char *progname = "iconv";

char *from_cs;
char *to_cs;
int debug;
int cflag;      /* skip invalid characters */
int sflag;      /* silent */
int lflag;      /* list conversions */

void iconv_file(FILE *, const char *);
void list_codesets(void);

iconv_t ich;    /* iconv(3c) lib handle */
size_t (*pconv)(const char **iptr, size_t *ileft,
                char **optr, size_t *oleft);

size_t
lib_iconv(const char **iptr, size_t *ileft, char **optr, size_t *oleft)
{
        return (iconv(ich, iptr, ileft, optr, oleft));
}

void
usage(void)
{
        (void) fprintf(stderr, gettext(
            "usage: %s [-cs] [-f from-codeset] [-t to-codeset] "
            "[file ...]\n"), progname);
        (void) fprintf(stderr, gettext("\t%s -l\n"), progname);
```



```
        exit(1);
}

int
main(int argc, char **argv)
{
        FILE *fp;
        char *fslash, *tslash;
        int c;

        (void) setlocale(LC_ALL, "");

#if !defined(TEXT_DOMAIN)
#define TEXT_DOMAIN     "SYS_TEST"
#endif
        (void) textdomain(TEXT_DOMAIN);

        while ((c = getopt(argc, argv, "cdlsf:t:")) != EOF) {
                switch (c) {
                case 'c':
                        cflag++;
                        break;
                case 'd':
                        debug++;
                        break;
                case 'l':
                        lflag++;
                        break;
                case 's':
                        sflag++;
                        break;
                case 'f':
                        from_cs = optarg;
                        break;
                case 't':
                        to_cs = optarg;
                        break;
                case '?':
                        usage();
                }
        }

        if (lflag) {
                if (from_cs != NULL || to_cs != NULL || optind != argc)
                        usage();
                list_codesets();
                exit(0);
        }

        if (from_cs == NULL)
                from_cs = nl_langinfo(CODESET);
        if (to_cs == NULL)
                to_cs = nl_langinfo(CODESET);

        /*
         * If either "from" or "to" contains a slash,
         * then we're using charmaps.
         */
        fslash = strchr(from_cs, '/');
        tslash = strchr(to_cs, '/');
        if (fslash != NULL || tslash != NULL) {
                charmap_init(to_cs, from_cs);
                pconv = cm_iconv;
                if (debug)
                        charmap_dump();
        } else {
```

```
128                 ich = iconv_open(to_cs, from_cs);
129                 if (ich == ((iconv_t)-1)) {
130                         (void) fprintf(stderr, gettext("iconv_open failed\n"));
131                         exit(1);
132                 }
133                 pconv = lib_iconv;
134         }

136         if (optind == argc || optind == argc - 1 &&
137             0 == strcmp(argv[optind], "-")) {
138                 iconv_file(stdin, "stdin");
139                 exit(0);
140         }

142         for (; optind < argc; optind++) {
143                 fp = fopen(argv[optind], "r");
144                 if (fp == NULL) {
145                         perror(argv[optind]);
146                         exit(1);
147                 }
148                 iconv_file(fp, argv[optind]);
149                 (void) fclose(fp);
150         }
151         exit(0);
152 }

154 /*
155  * Conversion buffer sizes:
156  *
157  * The input buffer has room to prepend one mbs character if needed for
158  * handling a left-over at the end of a previous conversion buffer.
159  *
160  * Conversions may grow or shrink data, so using a larger output buffer
161  * to reduce the likelihood of leftover input buffer data in each pass.
162  */
163 #define IBUFSIZ (MB_LEN_MAX + BUFSIZ)
164 #define OBUFSIZ (2 * BUFSIZ)

166 void
167 iconv_file(FILE *fp, const char *fname)
168 {
169         static char ibuf[IBUFSIZ];
170         static char obuf[OBUFSIZ];
171         const char *iptr;
172         char *optr;
173         off64_t offset;
174         size_t ileft, oleft, ocnt;
175         int iconv_errno;
176         int nr, nw, rc;

178         offset = 0;
179         ileft = 0;
180         iptr = ibuf + MB_LEN_MAX;

182         while ((nr = fread(ibuf+MB_LEN_MAX, 1, BUFSIZ, fp)) > 0) {

184                 assert(iptr <= ibuf+MB_LEN_MAX);
185                 assert(ileft <= MB_LEN_MAX);
186                 ileft += nr;
187                 offset += nr;

189                 optr = obuf;
190                 oleft = OBUFSIZ;

192         iconv_again:
193                 rc = (*pconv)(&iptr, &ileft, &optr, &oleft);
```



```
194                 iconv_errno = errno;

196                 ocnt = OBUFSIZ - oleft;
197                 if (ocnt > 0) {
198                         nw = fwrite(obuf, 1, ocnt, stdout);
199                         if (nw != ocnt) {
200                                 perror("fwrite");
201                                 exit(1);
202                         }
203                 }
204                 optr = obuf;
205                 oleft = OBUFSIZ;

207                 if (rc == (size_t)-1) {
208                         switch (iconv_errno) {

210                         case E2BIG:     /* no room in output buffer */
211                                 goto iconv_again;

213                         case EINVAL:    /* incomplete sequence on input */
214                                 if (debug) {
215                                         (void) fprintf(stderr,
216                                 _("Incomplete sequence in %s at offset %lld\n"),
217                                                 fname, offset - ileft);
218                                 }
219                                 /*
220                                  * Copy the reminder to the space reserved
221                                  * at the start of the input buffer.
222                                  */
223                                 assert(ileft > 0);
224                                 if (ileft <= MB_LEN_MAX) {
225                                         char *p = ibuf+MB_LEN_MAX-ileft;
226                                         (void) memcpy(p, iptr, ileft);
227                                         iptr = p;
228                                         continue; /* read again */
229                                 }
230                                 /*
231                                  * Should not see ileft > MB_LEN_MAX,
232                                  * but if we do, handle as EILSEQ.
233                                  */
234                                 /* FALLTHROUGH */

236                         case EILSEQ:    /* invalid sequence on input */
237                                 if (!sflag) {
238                                         (void) fprintf(stderr,
239                                 _("Illegal sequence in %s at offset %lld\n"),
240                                                 fname, offset - ileft);
241                                         (void) fprintf(stderr,
242                                 _("bad seq: \\x%02x\\x%02x\\x%02x\n"),
243                                                 iptr[0] & 0xff,
244                                                 iptr[1] & 0xff,
245                                                 iptr[2] & 0xff);
246                                 }
247                                 assert(ileft > 0);
248                                 /* skip one */
249                                 iptr++;
250                                 ileft--;
251                                 assert(oleft > 0);
252                                 if (!cflag) {
253                                         *optr++ = '?';
254                                         oleft--;
255                                 }
256                                 goto iconv_again;

258                         default:
259                                 (void) fprintf(stderr,
```

```
260                         _("iconv error (%s) in file $s at offset %lld\n"),
261                                     strerror(errno), fname, offset - ileft);
262                                 perror("iconv");
263                                 break;
264                         }
265                 }

267                 /* normal iconv return */
268                 ileft = 0;
269                 iptr = ibuf + MB_LEN_MAX;
270         }

272         /*
273          * End of file
274          * Flush any shift encodings.
275          */
276         iptr = NULL;
277         ileft = 0;
278         optr = obuf;
279         oleft = OBUFSIZ;
280         (*pconv)(&iptr, &ileft, &optr, &oleft);
281         ocnt = OBUFSIZ - oleft;
282         if (ocnt > 0) {
283                 (void) fwrite(obuf, 1, ocnt, stdout);
284         }
285 }

287 /*
288  * scan the /usr/lib/iconv directory...
289  * A script for this seems appropriate.
290  */
291 void
292 list_codesets(void)
293 {
294         (void) system("/usr/lib/iconv/iconv_list");
295 }
```

```
**********************************************************
    1418 Sat May 28 21:32:40 2011
new/usr/src/cmd/iconv/iconv_list.sh
30 Need iconv
**********************************************************
```

```
#!/bin/ksh
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2011 Nexenta Systems, Inc.  All rights reserved.

# List all iconv(1) codesets

cd /usr/lib/iconv || exit 1

typeset -A all

/usr/bin/ls | while read f
do
    case "$f" in
    geniconvtbl.so)
        ;;
    *.so)
        IFS="%"
        set ${f%.so}
        all[$1]=" "
        all[$2]=" "
        ;;
    *.t)
        IFS="."
        set ${f%.t}
        all[$1]=" "
        all[$2]=" "
        ;;
    *)
        ;;
    esac
done

/usr/bin/ls geniconvtbl/binarytables |
while read f
do
    case "$f" in
    *.bt)
        IFS="%"
        set ${f%.bt}
        all[$1]=" "
        all[$2]=" "
        ;;
    *)
        ;;
    esac
done

# Only store aliases for names we've seen

IFS="   "
```



```
while read a c
do
    case "$a" in
    \#*)
        ;;
    *)
        if [ "$c" -a "${all[$c]}" ] ; then
                all[$c]="${all[$c]} $a"
        fi
        ;;
    esac
done < "alias"

cat <<EOF
The following are all supported code set names.  Conversions
between some fromcode-tocode pairs might not be available.
Some of these code set names have aliases, which are shown
after the canonical name.

EOF

for i in "${!all[@]}"
do
        echo "$i ${all[$i]}"
done
```

```
**********************************************************
    1939 Sat May 28 21:32:40 2011
new/usr/src/cmd/iconv/parser.y
30 Need iconv
**********************************************************
```

```
%{
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

/*
 * POSIX iconv charmap grammar.
 */

#include <wchar.h>
#include <stdio.h>
#include <limits.h>
#include "charmap.h"

%}
%union {
        char            *token;
        int             num;
        char            mbs[MB_LEN_MAX + 2]; /* NB: [0] is length! */
}

%token          T_CODE_SET
%token          T_MB_CUR_MAX
%token          T_MB_CUR_MIN
%token          T_COM_CHAR
%token          T_ESC_CHAR
%token          T_LT
%token          T_GT
%token          T_NL
%token          T_SEMI
%token          T_COMMA
%token          T_ELLIPSIS
%token          T_RPAREN
%token          T_LPAREN
%token          T_QUOTE
%token          T_NULL
%token          T_END
%token          T_CHARMAP
%token          T_WIDTH
%token          T_WIDTH_DEFAULT
%token  <mbs>           T_CHAR
%token  <token>         T_NAME
%token  <num>           T_NUMBER
%token  <token>         T_SYMBOL

%%

goal            : setting_list charmap
                | charmap
                ;
```



```
string          : T_QUOTE charlist T_QUOTE
                | T_QUOTE T_QUOTE
                ;

charlist        : charlist T_CHAR
                | T_CHAR
                ;

setting_list    : setting_list setting
                | setting
                ;

setting         : T_COM_CHAR T_CHAR T_NL
                {
                        com_char = $2[1];
                }
                | T_ESC_CHAR T_CHAR T_NL
                {
                        esc_char = $2[1];
                }
                | T_MB_CUR_MAX T_NUMBER T_NL
                {
                        mb_cur_max = $2;
                }
                | T_MB_CUR_MIN T_NUMBER T_NL
                {
                        mb_cur_min = $2;
                }
                | T_CODE_SET string T_NL
                {
                        /* ignore */
                }
                ;

charmap         : T_CHARMAP T_NL charmap_list T_END T_CHARMAP T_NL

charmap_list    : charmap_list charmap_entry
                | charmap_entry
                ;

charmap_entry   : T_SYMBOL T_CHAR
                {
                        add_charmap($1, $2);
                        scan_to_eol();
                }
                | T_SYMBOL T_ELLIPSIS T_SYMBOL T_CHAR
                {
                        add_charmap_range($1, $3, $4);
                        scan_to_eol();
                }
                | T_NL
                ;
```

```
**********************************************************
   11780 Sat May 28 21:32:41 2011
new/usr/src/cmd/iconv/scanner.c
30 Need iconv
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

/*
 * This file contains the "scanner", which tokenizes charmap files
 * for iconv for processing by the higher level grammar processor.
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <limits.h>
#include <string.h>
#include <widec.h>
#include <sys/types.h>
#include <assert.h>
#include "charmap.h"
#include "parser.tab.h"

int                     com_char = '#';
int                     esc_char = '\\';
int                     mb_cur_min = 1;
int                     mb_cur_max = 1;
int                     lineno = 1;
int                     warnings = 0;
static int              nextline;
static FILE             *input = stdin;
static const char       *filename = "<stdin>";
static int              instring = 0;
static int              escaped = 0;

/*
 * Token space ... grows on demand.
 */
static char *token = NULL;
static int tokidx;
static int toksz = 0;
static int hadtok = 0;

/*
 * The last keyword seen.  This is useful to trigger the special lexer rules
 * for "copy" and also collating symbols and elements.
 */
int     last_kw = 0;
static int      category = T_END;

static struct token {
        int id;
        const char *name;
```



```
} keywords[] = {
        { T_COM_CHAR,           "comment_char" },
        { T_ESC_CHAR,           "escape_char" },
        { T_END,                "END" },

        /*
         * These are keywords used in the charmap file.  Note that
         * Solaris orginally used angle brackets to wrap some of them,
         * but we removed that to simplify our parser.  The first of these
         * items are "global items."
         */
        { T_CHARMAP,            "CHARMAP" },
        { T_WIDTH,              "WIDTH" },
        { T_WIDTH_DEFAULT,      "WIDTH_DEFAULT" },

        { -1, NULL },
};

/*
 * These special words are only used in a charmap file, enclosed in <>.
 */
static struct token symwords[] = {
        { T_COM_CHAR,           "comment_char" },
        { T_ESC_CHAR,           "escape_char" },
        { T_CODE_SET,           "code_set_name" },
        { T_MB_CUR_MAX,         "mb_cur_max" },
        { T_MB_CUR_MIN,         "mb_cur_min" },
        { -1, NULL },
};

static int categories[] = {
        T_CHARMAP,
        0
};

void
reset_scanner(const char *fname)
{
        if (fname == NULL) {
                filename = "<stdin>";
                input = stdin;
        } else {
                if (input != stdin)
                        (void) fclose(input);
                if ((input = fopen(fname, "r")) == NULL) {
                        perror(fname);
                        exit(1);
                }
                filename = fname;
        }
        com_char = '#';
        esc_char = '\\';
        instring = 0;
        escaped = 0;
        lineno = 1;
        nextline = 1;
        tokidx = 0;
        last_kw = 0;
        category = T_END;
}

#define hex(x)  \
        (isdigit(x) ? (x - '0') : ((islower(x) ? (x - 'a') : (x - 'A')) + 10))
#define isodigit(x)     ((x >= '0') && (x <= '7'))

static int
```

```
128 scanc(void)
129 {
130         int     c;

132         c = getc(input);
133         lineno = nextline;
134         if (c == '\n') {
135                 nextline++;
136         }
137         return (c);
138 }

140 static void
141 unscanc(int c)
142 {
143         if (c == '\n') {
144                 nextline--;
145         }
146         if (ungetc(c, input) < 0) {
147                 yyerror(_("ungetc failed"));
148         }
149 }

151 static int
152 scan_hex_byte(void)
153 {
154         int     c1, c2;
155         int     v;

157         c1 = scanc();
158         if (!isxdigit(c1)) {
159                 yyerror(_("malformed hex digit"));
160                 return (0);
161         }
162         c2 = scanc();
163         if (!isxdigit(c2)) {
164                 yyerror(_("malformed hex digit"));
165                 return (0);
166         }
167         v = ((hex(c1) << 4) | hex(c2));
168         return (v);
169 }

171 static int
172 scan_dec_byte(void)
173 {
174         int     c1, c2, c3;
175         int     b;

177         c1 = scanc();
178         if (!isdigit(c1)) {
179                 yyerror(_("malformed decimal digit"));
180                 return (0);
181         }
182         b = c1 - '0';
183         c2 = scanc();
184         if (!isdigit(c2)) {
185                 yyerror(_("malformed decimal digit"));
186                 return (0);
187         }
188         b *= 10;
189         b += (c2 - '0');
190         c3 = scanc();
191         if (!isdigit(c3)) {
192                 unscanc(c3);
193         } else {
```



```
194                 b *= 10;
195                 b += (c3 - '0');
196         }
197         return (b);
198 }

200 static int
201 scan_oct_byte(void)
202 {
203         int c1, c2, c3;
204         int     b;

206         b = 0;

208         c1 = scanc();
209         if (!isodigit(c1)) {
210                 yyerror(_("malformed octal digit"));
211                 return (0);
212         }
213         b = c1 - '0';
214         c2 = scanc();
215         if (!isodigit(c2)) {
216                 yyerror(_("malformed octal digit"));
217                 return (0);
218         }
219         b *= 8;
220         b += (c2 - '0');
221         c3 = scanc();
222         if (!isodigit(c3)) {
223                 unscanc(c3);
224         } else {
225                 b *= 8;
226                 b += (c3 - '0');
227         }
228         return (b);
229 }

231 void
232 add_tok(int c)
233 {
234         if ((tokidx + 1) >= toksz) {
235                 toksz += 64;
236                 if ((token = realloc(token, toksz)) == NULL) {
237                         yyerror(_("out of memory"));
238                         tokidx = 0;
239                         toksz = 0;
240                         return;
241                 }
242         }

244         token[tokidx++] = (char)c;
245         token[tokidx] = 0;
246 }

248 static int
249 get_byte(void)
250 {
251         int     c;

253         if ((c = scanc()) != esc_char) {
254                 unscanc(c);
255                 return (EOF);
256         }
257         c = scanc();

259         switch (c) {
```

```
260         case 'd':
261         case 'D':
262                 return (scan_dec_byte());
263         case 'x':
264         case 'X':
265                 return (scan_hex_byte());
266         case '0':
267         case '1':
268         case '2':
269         case '3':
270         case '4':
271         case '5':
272         case '6':
273         case '7':
274                 /* put the character back so we can get it */
275                 unscanc(c);
276                 return (scan_oct_byte());
277         default:
278                 unscanc(c);
279                 unscanc(esc_char);
280                 return (EOF);
281         }
282 }

284 int
285 get_escaped(int c)
286 {
287         switch (c) {
288         case 'n':
289                 return ('\n');
290         case 'r':
291                 return ('\r');
292         case 't':
293                 return ('\t');
294         case 'f':
295                 return ('\f');
296         case 'v':
297                 return ('\v');
298         case 'b':
299                 return ('\b');
300         case 'a':
301                 return ('\a');
302         default:
303                 return (c);
304         }
305 }

307 int
308 get_wide(void)
309 {
310         /* NB: yylval.mbs[0] is the length */
311         char *mbs = &yylval.mbs[1];
312         int mbi = 0;
313         int c;

315         mbs[mbi] = 0;
316         if (mb_cur_max > MB_LEN_MAX) {
317                 yyerror(_("max multibyte character size too big"));
318                 return (T_NULL);
319         }
320         for (;;) {
321                 if ((c = get_byte()) == EOF)
322                         break;
323                 if (mbi == mb_cur_max) {
324                         unscanc(c);
325                         yyerror(_("length > mb_cur_max"));
```



```
326                         return (T_NULL);
327                 }
328                 mbs[mbi++] = c;
329                 mbs[mbi] = 0;
330         }

332         /* result in yylval.mbs */
333         mbs[-1] = mbi;
334         return (T_CHAR);
335 }

337 int
338 get_symbol(void)
339 {
340         int     c;

342         while ((c = scanc()) != EOF) {
343                 if (escaped) {
344                         escaped = 0;
345                         if (c == '\n')
346                                 continue;
347                         add_tok(get_escaped(c));
348                         continue;
349                 }
350                 if (c == esc_char) {
351                         escaped = 1;
352                         continue;
353                 }
354                 if (c == '\n') {        /* well that's strange! */
355                         yyerror(_("unterminated symbolic name"));
356                         continue;
357                 }
358                 if (c == '>') {         /* end of symbol */

360                         /*
361                          * This restarts the token from the beginning
362                          * the next time we scan a character.  (This
363                          * token is complete.)
364                          */

366                         if (token == NULL) {
367                                 yyerror(_("missing symbolic name"));
368                                 return (T_NULL);
369                         }
370                         tokidx = 0;

372                         /*
373                          * A few symbols are handled as keywords outside
374                          * of the normal categories.
375                          */
376                         if (category == T_END) {
377                                 int i;
378                                 for (i = 0; symwords[i].name != 0; i++) {
379                                         if (strcmp(token, symwords[i].name) ==
380                                             0) {
381                                                 last_kw = symwords[i].id;
382                                                 return (last_kw);
383                                         }
384                                 }
385                         }
386                         /* its an undefined symbol */
387                         yylval.token = strdup(token);
388                         token = NULL;
389                         toksz = 0;
390                         tokidx = 0;
391                         return (T_SYMBOL);
```

```
392                 }
393                 add_tok(c);
394         }

396         yyerror(_("unterminated symbolic name"));
397         return (EOF);
398 }


401 static int
402 consume_token(void)
403 {
404         int     len = tokidx;
405         int     i;

407         tokidx = 0;
408         if (token == NULL)
409                 return (T_NULL);

411         /*
412          * this one is special, because we don't want it to alter the
413          * last_kw field.
414          */
415         if (strcmp(token, "...") == 0) {
416                 return (T_ELLIPSIS);
417         }

419         /* search for reserved words first */
420         for (i = 0; keywords[i].name; i++) {
421                 int j;
422                 if (strcmp(keywords[i].name, token) != 0) {
423                         continue;
424                 }

426                 last_kw = keywords[i].id;

428                 /* clear the top level category if we're done with it */
429                 if (last_kw == T_END) {
430                         category = T_END;
431                 }

433                 /* set the top level category if we're changing */
434                 for (j = 0; categories[j]; j++) {
435                         if (categories[j] != last_kw)
436                                 continue;
437                         category = last_kw;
438                 }

440                 return (keywords[i].id);
441         }

443         /* maybe its a numeric constant? */
444         if (isdigit(*token) || (*token == '-' && isdigit(token[1]))) {
445                 char *eptr;
446                 yylval.num = strtol(token, &eptr, 10);
447                 if (*eptr != 0)
448                         yyerror(_("malformed number"));
449                 return (T_NUMBER);
450         }

452         /*
453          * A single lone character is treated as a character literal.
454          * To avoid duplication of effort, we stick in the charmap.
455          */
456         if (len == 1) {
457                 yylval.mbs[0] = 1; /* length */
```



```
458                 yylval.mbs[1] = token[0];
459                 yylval.mbs[2] = '\0';
460                 return (T_CHAR);
461         }

463         /* anything else is treated as a symbolic name */
464         yylval.token = strdup(token);
465         token = NULL;
466         toksz = 0;
467         tokidx = 0;
468         return (T_NAME);
469 }

471 void
472 scan_to_eol(void)
473 {
474         int     c;
475         while ((c = scanc()) != '\n') {
476                 if (c == EOF) {
477                         /* end of file without newline! */
478                         errf(_("missing newline"));
479                         return;
480                 }
481         }
482         assert(c == '\n');
483 }

485 int
486 yylex(void)
487 {
488         int             c;

490         while ((c = scanc()) != EOF) {

492                 /* special handling for quoted string */
493                 if (instring) {
494                         if (escaped) {
495                                 escaped = 0;

497                                 /* if newline, just eat and forget it */
498                                 if (c == '\n')
499                                         continue;

501                                 if (strchr("xXd01234567", c)) {
502                                         unscanc(c);
503                                         unscanc(esc_char);
504                                         return (get_wide());
505                                 }
506                                 yylval.mbs[0] = 1; /* length */
507                                 yylval.mbs[1] = get_escaped(c);
508                                 yylval.mbs[2] = '\0';
509                                 return (T_CHAR);
510                         }
511                         if (c == esc_char) {
512                                 escaped = 1;
513                                 continue;
514                         }
515                         switch (c) {
516                         case '<':
517                                 return (get_symbol());
518                         case '>':
519                                 /* oops! should generate syntax error  */
520                                 return (T_GT);
521                         case '"':
522                                 instring = 0;
523                                 return (T_QUOTE);
```

```
524                     default:
525                             yylval.mbs[0] = 1; /* length */
526                             yylval.mbs[1] = c;
527                             yylval.mbs[2] = '\0';
528                             return (T_CHAR);
529                     }
530             }

532             /* escaped characters first */
533             if (escaped) {
534                     escaped = 0;
535                     if (c == '\n') {
536                             /* eat the newline */
537                             continue;
538                     }
539                     hadtok = 1;
540                     if (tokidx) {
541                             /* an escape mid-token is nonsense */
542                             return (T_NULL);
543                     }

545                     /* numeric escapes are treated as wide characters */
546                     if (strchr("xXd01234567", c)) {
547                             unscanc(c);
548                             unscanc(esc_char);
549                             return (get_wide());
550                     }

552                     add_tok(get_escaped(c));
553                     continue;
554             }

556             /* if it is the escape charter itself note it */
557             if (c == esc_char) {
558                     escaped = 1;
559                     continue;
560             }

562             /* remove from the comment char to end of line */
563             if (c == com_char) {
564                     while (c != '\n') {
565                             if ((c = scanc()) == EOF) {
566                                     /* end of file without newline! */
567                                     return (EOF);
568                             }
569                     }
570                     assert(c == '\n');
571                     if (!hadtok) {
572                             /*
573                              * If there were no tokens on this line,
574                              * then just pretend it didn't exist at all.
575                              */
576                             continue;
577                     }
578                     hadtok = 0;
579                     return (T_NL);
580             }

582             if (strchr(" \t\n;()<>,\"", c) && (tokidx != 0)) {
583                     /*
584                      * These are all token delimiters.  If there
585                      * is a token already in progress, we need to
586                      * process it.
587                      */
588                     unscanc(c);
589                     return (consume_token());
```



```
590             }

592             switch (c) {
593             case '\n':
594                     if (!hadtok) {
595                             /*
596                              * If the line was completely devoid of tokens,
597                              * then just ignore it.
598                              */
599                             continue;
600                     }
601                     /* we're starting a new line, reset the token state */
602                     hadtok = 0;
603                     return (T_NL);
604             case ',':
605                     hadtok = 1;
606                     return (T_COMMA);
607             case ';':
608                     hadtok = 1;
609                     return (T_SEMI);
610             case '(':
611                     hadtok = 1;
612                     return (T_LPAREN);
613             case ')':
614                     hadtok = 1;
615                     return (T_RPAREN);
616             case '>':
617                     hadtok = 1;
618                     return (T_GT);
619             case '<':
620                     /* symbol start! */
621                     hadtok = 1;
622                     return (get_symbol());
623             case ' ':
624             case '\t':
625                     /* whitespace, just ignore it */
626                     continue;
627             case '"':
628                     hadtok = 1;
629                     instring = 1;
630                     return (T_QUOTE);
631             default:
632                     hadtok = 1;
633                     add_tok(c);
634                     continue;
635             }
636     }
637     return (EOF);
638 }

640 void
641 yyerror(const char *msg)
642 {
643     (void) fprintf(stderr, _("%s: %d: error: %s\n"),
644         filename, lineno, msg);
645     exit(1);
646 }

648 void
649 errf(const char *fmt, ...)
650 {
651     char    *msg;

653     va_list va;
654     va_start(va, fmt);
655     (void) vasprintf(&msg, fmt, va);
```

```
656         va_end(va);

658         (void) fprintf(stderr, _("%s: %d: error: %s\n"),
659             filename, lineno, msg);
660         free(msg);
661         exit(1);
662 }

664 void
665 warn(const char *fmt, ...)
666 {
667         char    *msg;

669         va_list va;
670         va_start(va, fmt);
671         (void) vasprintf(&msg, fmt, va);
672         va_end(va);

674         (void) fprintf(stderr, _("%s: %d: warning: %s\n"),
675             filename, lineno, msg);
676         free(msg);
677         warnings++;
678 }
```