```
**********************************************************
   23115 Fri May 22 11:19:39 2015
new/usr/src/cmd/make/bin/ar.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
______unchanged_portion_omitted_

/*
 * Static variables
 */

/*
 * File table of contents
 */
extern  timestruc_t&    read_archive(register Name target);
static  Boolean         open_archive(char *filename, register Ar *arp);
static  void            close_archive(register Ar *arp);
static  Boolean         read_archive_dir(register Ar *arp, Name library, char **
static  void            translate_entry(register Ar *arp, Name target, register
static  long            sgetl(char *);

/*
 *      read_archive(target)
 *
 *      Read the contents of an ar file.
 *
 *      Return value:
 *                              The time the member was created
 *
 *      Parameters:
 *              target          The member to find time for
 *
 *      Global variables used:
 *              empty_name      The Name ""
 */

int read_member_header (Ar_port *header, FILE *fd, char* filename);
int process_long_names_member (register Ar *arp, char **long_names_table, char *

timestruc_t&
read_archive(register Name target)
{
        register Property       member;
        wchar_t                 *slash;
        String_rec              true_member_name;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        register Name           true_member = NULL;
        Ar                      ar;
        char                    *long_names_table = NULL; /* Table of long
                                                             member names */

        member = get_prop(target->prop, member_prop);
        /*
         * Check if the member has directory component.
         * If so, remove the dir and see if we know the date.
         */
        if (member->body.member.member != NULL) {
                Wstring member_string(member->body.member.member);
                wchar_t * wcb = member_string.get_string();
                if((slash = (wchar_t *) wcsrchr(wcb, (int) slash_char)) != NULL)
                if((slash = (wchar_t *) wsrchr(wcb, (int) slash_char)) != NULL)
                        INIT_STRING_FROM_STACK(true_member_name, buffer);
                        append_string(member->body.member.library->string_mb,
                                      &true_member_name,
                                      FIND_LENGTH);
                        append_char((int) parenleft_char, &true_member_name);
```

```
                        append_string(slash + 1, &true_member_name, FIND_LENGTH)
                        append_char((int) parenright_char, &true_member_name);
                        true_member = GETNAME(true_member_name.buffer.start,
                                                      FIND_LENGTH);
                        if (true_member->stat.time != file_no_time) {
                                target->stat.time = true_member->stat.time;
                                return target->stat.time;
                        }
                }
        }
        if (open_archive(member->body.member.library->string_mb, &ar) == failed)
                if (errno == ENOENT) {
                        target->stat.stat_errno = ENOENT;
                        close_archive(&ar);
                        if (member->body.member.member == NULL) {
                                member->body.member.member = empty_name;
                        }
                        return target->stat.time = file_doesnt_exist;
                } else {
                        fatal(gettext("Can't access archive '%s': %s"),
                              member->body.member.library->string_mb,
                              errmsg(errno));
                }
        }
        if (target->stat.time == file_no_time) {
                if (read_archive_dir(&ar, member->body.member.library,
                                     &long_names_table)
                    == failed){
                        fatal(gettext("Can't access archive '%s': %s"),
                              member->body.member.library->string_mb,
                              errmsg(errno));
                }
        }
        if (member->body.member.entry != NULL) {
                translate_entry(&ar, target, member,&long_names_table);
        }
        close_archive(&ar);
        if (long_names_table) {
                retmem_mb(long_names_table);
        }
        if (true_member != NULL) {
                target->stat.time = true_member->stat.time;
        }
        if (target->stat.time == file_no_time) {
                target->stat.time = file_doesnt_exist;
        }
        return target->stat.time;
}
______unchanged_portion_omitted_

/*
 *      translate_entry(arp, target, member)
 *
 *      Finds the member for one lib.a((entry))
 *
 *      Parameters:
 *              arp             Pointer to ar file description block
 *              target          Target to find member name for
 *              member          Property to fill in with info
 *
 *      Global variables used:
 */
static void
translate_entry(register Ar *arp, Name target, register Property member, char **
{
        register int            len;
        register int            i;
        wchar_t                 *member_string;
        ar_port_word            *offs;
        int                     strtablen;
        char                    *syms;          /* string table */
        char                    *csym;          /* string table */
        ar_port_word            *offend;        /* end of offsets table */
        int                     date;
        register wchar_t        *ap;
        register char           *hp;
        int                     maxs;
        int                     offset;
        char            buffer[4];

        if (arp->sym_begin == 0L || arp->num_symbols == 0L) {
                fatal(gettext("Cannot find symbol ‘%s’ in archive ‘%s’"),
                      member->body.member.entry->string_mb,
                      member->body.member.library->string_mb);
        }

        if (fseek(arp->fd, arp->sym_begin, 0) != 0) {
                goto read_error;
        }
        member_string = ALLOC_WC((int) ((int) ar_member_name_len * 2));

        switch (arp->type) {
        case AR_5:
                if ((len = member->body.member.entry->hash.length) > 8) {
                        len = 8;
                }
                for (i = 0; i < arp->num_symbols; i++) {
                        if (fread((char *) &arp->ars_5,
                                  sizeof arp->ars_5,
                                  1,
                                  arp->fd) != 1) {
                                goto read_error;
                        }
                        if (IS_EQUALN(arp->ars_5.sym_name,
                                      member->body.member.entry->string_mb,
                                      len)) {
                                if ((fseek(arp->fd,
                                           sgetl(arp->ars_5.sym_ptr),
                                           0) != 0) ||
                                    (fread((char *) &arp->arf_5,
                                           sizeof arp->arf_5,
                                           1,
                                           arp->fd) != 1)) {
                                        goto read_error;
                                }
                                MBSTOWCS(wcs_buffer, arp->arf_5.arf_name);
                                (void) wcsncpy(member_string,
                                (void) wsncpy(member_string,
                                              wcs_buffer,
                                              wcslen(wcs_buffer));
                                              wslen(wcs_buffer));
                                member_string[sizeof(arp->arf_5.arf_name)] =
                                                                (int) nul_char;
                                member->body.member.member =
                                        GETNAME(member_string, FIND_LENGTH);
                                target->stat.time.tv_sec = sgetl(arp->arf_5.arf_
                                target->stat.time.tv_nsec = LONG_MAX;
                                return;
                        }
                }
                break;
        case AR_PORT:
```

```
                        offs = (ar_port_word *) alloca((int) (arp->num_symbols * AR_PORT
                        if (fread((char *) offs,
                                  AR_PORT_WORD,
                                  (int) arp->num_symbols,
                                  arp->fd) != arp->num_symbols) {
                                goto read_error;
                        }

                        for(i=0;i<arp->num_symbols;i++) {
                                *((int*)buffer)=offs[i];
                                offs[i]=(ar_port_word)sgetl(buffer);
                        }

                        strtablen=arp->sym_size-4-(int) (arp->num_symbols * AR_PORT_WORD
                        syms = (char *) alloca(strtablen);
                        if (fread(syms,
                                  sizeof (char),
                                  strtablen,
                                  arp->fd) != strtablen) {
                                goto read_error;
                        }
                        offend = &offs[arp->num_symbols];
                        while (offs < offend) {
                                maxs = strlen(member->body.member.entry->string_mb);
                                if(strlen(syms) > maxs)
                                        maxs = strlen(syms);
                                if (IS_EQUALN(syms,
                                              member->body.member.entry->string_mb,
                                              maxs)) {
                                        if (fseek(arp->fd,
                                                  (long) *offs,
                                                  0) != 0) {
                                                goto read_error;
                                        }
                                        if ((fread((char *) &arp->ar_port,
                                                   sizeof arp->ar_port,
                                                   1,
                                                   arp->fd) != 1) ||
                                            !IS_EQUALN(arp->ar_port.ar_fmag,
                                                       AR_PORT_END_MAGIC,
                                                       sizeof arp->ar_port.ar_fmag)) {
                                                goto read_error;
                                        }
                                        if (sscanf(arp->ar_port.ar_date,
                                                   "%ld",
                                                   &date) != 1) {
                                                fatal(gettext("Bad date field for member
                                                      arp->ar_port.ar_name,
                                                      target->string_mb);
                                        }
                            /* If it’s a long name, retrieve it from long name table */
                            if (arp->ar_port.ar_name[0] == ’/’) {
                                    sscanf(arp->ar_port.ar_name + 1,
                                           "%ld",
                                           &offset);
                                    len = ar_member_name_len;
                                    hp = *long_names_table + offset;
                            } else {
                                    len = sizeof arp->ar_port.ar_name;
                                    hp = arp->ar_port.ar_name;
                            }
                                        ap = member_string;
                                        while (*hp &&
                                               (*hp != (int) slash_char) &&
                                               (ap < &member_string[len])) {
                                                MBTOWC(ap, hp);
                                        ap++;
                                        hp++;
                                }
                                *ap = (int) nul_char;
                                member->body.member.member =
                                        GETNAME(member_string, FIND_LENGTH);
                                target->stat.time.tv_sec = date;
                                target->stat.time.tv_nsec = LONG_MAX;
                                return;
                        }
                        offs++;
                        while(*syms!='\0') syms++;
                        syms++;
                }
        }
        fatal(gettext("Cannot find symbol '%s' in archive '%s'"),
              member->body.member.entry->string_mb,
              member->body.member.library->string_mb);
        /*NOTREACHED*/

read_error:
        if (ferror(arp->fd)) {
                fatal(gettext("Read error in archive '%s': %s"),
                      member->body.member.library->string_mb,
                      errmsg(errno));
        } else {
                fatal(gettext("Read error in archive '%s': Premature EOF"),
                      member->body.member.library->string_mb);
        }
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   91165 Fri May 22 11:19:40 2015
new/usr/src/cmd/make/bin/doname.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
______unchanged_portion_omitted_
```

```
/*
 *      dynamic_dependencies(target)
 *
 *      Checks if any dependency contains a macro ref
 *      If so, it replaces the dependency with the expanded version.
 *      Here, "$@" gets translated to target->string. That is
 *      the current name on the left of the colon in the
 *      makefile.  Thus,
 *              xyz:    s.$@.c
 *      translates into
 *              xyz:    s.xyz.c
 *
 *      Also, "$(@F)" translates to the same thing without a preceeding
 *      directory path (if one exists).
 *      Note, to enter "$@" on a dependency line in a makefile
 *      "$$@" must be typed. This is because make expands
 *      macros in dependency lists upon reading them.
 *      dynamic_dependencies() also expands file wildcards.
 *      If there are any Shell meta characters in the name,
 *      search the directory, and replace the dependency
 *      with the set of files the pattern matches
 *
 *      Parameters:
 *              target          Target to sanitize dependencies for
 *
 *      Global variables used:
 *              c_at            The Name "@", used to set macro value
 *              debug_level     Should we trace actions?
 *              dot             The Name ".", used to read directory
 *              recursion_level Used for tracing
 */
void
dynamic_dependencies(Name target)
{
        wchar_t                 pattern[MAXPATHLEN];
        register wchar_t        *p;
        Property                line;
        register Dependency     dependency;
        register Dependency     *remove;
        String_rec              string;
        wchar_t                 buffer[MAXPATHLEN];
        register Boolean        set_at = false;
        register wchar_t        *start;
        Dependency              new_depe;
        register Boolean        reuse_cell;
        Dependency              first_member;
        Name                    directory;
        Name                    lib;
        Name                    member;
        Property                prop;
        Name                    true_target = target;
        wchar_t                 *library;

        if ((line = get_prop(target->prop, line_prop)) == NULL) {
                return;
        }
        /* If the target is constructed from a "::" target we consider that */
        if (target->has_target_prop) {
                true_target = get_prop(target->prop,
                                       target_prop)->body.target.target;
        }
        /* Scan all dependencies and process the ones that contain "$" chars */
        for (dependency = line->body.line.dependencies;
             dependency != NULL;
             dependency = dependency->next) {
                if (!dependency->name->dollar) {
                        continue;
                }
                target->has_depe_list_expanded = true;

                /* The make macro $@ is bound to the target name once per */
                /* invocation of dynamic_dependencies() */
                if (!set_at) {
                        (void) SETVAR(c_at, true_target, false);
                        set_at = true;
                }
                /* Expand this dependency string */
                INIT_STRING_FROM_STACK(string, buffer);
                expand_value(dependency->name, &string, false);
                /* Scan the expanded string. It could contain whitespace */
                /* which mean it expands to several dependencies */
                start = string.buffer.start;
                while (iswspace(*start)) {
                        start++;
                }
                /* Remove the cell (later) if the macro was empty */
                if (start[0] == (int) nul_char) {
                        dependency->name = NULL;
                }

/* azv 10/26/95 to fix bug BID_1170218 */
                if ((start[0] == (int) period_char) &&
                    (start[1] == (int) slash_char)) {
                        start += 2;
                }
/* azv */

                first_member = NULL;
                /* We use the original dependency cell for the first */
                /* dependency from the expansion */
                reuse_cell = true;
                /* We also have to deal with dependencies that expand to */
                /* lib.a(members) notation */
                for (p = start; *p != (int) nul_char; p++) {
                        if ((*p == (int) parenleft_char)) {
                                lib = GETNAME(start, p - start);
                                lib->is_member = true;
                                first_member = dependency;
                                start = p + 1;
                                while (iswspace(*start)) {
                                        start++;
                                }
                                break;
                        }
                }
                do {
                    /* First skip whitespace */
                        for (p = start; *p != (int) nul_char; p++) {
                                if ((*p == (int) nul_char) ||
                                    iswspace(*p) ||
                                    (*p == (int) parenright_char)) {
                                        break;
                                }
                        }
                        /* Enter dependency from expansion */
                        if (p != start) {
                                /* Create new dependency cell if */
                                /* this is not the first dependency */
                                /* picked from the expansion */
                                if (!reuse_cell) {
                                        new_depe = ALLOC(Dependency);
                                        new_depe->next = dependency->next;
                                        new_depe->automatic = false;
                                        new_depe->stale = false;
                                        new_depe->built = false;
                                        dependency->next = new_depe;
                                        dependency = new_depe;
                                }
                                reuse_cell = false;
                                /* Internalize the dependency name */
                                // tolik. Fix for bug 4110429: inconsistent expa
                                // include "//" and "/./"
                                //dependency->name = GETNAME(start, p - start);
                                dependency->name = normalize_name(start, p - sta
                                if ((debug_level > 0) &&
                                    (first_member == NULL)) {
                                        (void) printf(gettext("%*sDynamic depend
                                                      recursion_level,
                                                      "",
                                                      dependency->name->string_m
                                                      true_target->string_mb);
                                }
                                for (start = p; iswspace(*start); start++);
                                p = start;
                        }
                } while ((*p != (int) nul_char) &&
                         (*p != (int) parenright_char));
                /* If the expansion was of lib.a(members) format we now */
                /* enter the proper member cells */
                if (first_member != NULL) {
                        /* Scan the new dependencies and transform them from */
                        /* "foo" to "lib.a(foo)" */
                        for (; 1; first_member = first_member->next) {
                                /* Build "lib.a(foo)" name */
                                INIT_STRING_FROM_STACK(string, buffer);
                                APPEND_NAME(lib,
                                              &string,
                                              (int) lib->hash.length);
                                append_char((int) parenleft_char, &string);
                                APPEND_NAME(first_member->name,
                                              &string,
                                              FIND_LENGTH);
                                append_char((int) parenright_char, &string);
                                member = first_member->name;
                                /* Replace "foo" with "lib.a(foo)" */
                                first_member->name =
                                  GETNAME(string.buffer.start, FIND_LENGTH);
                                if (string.free_after_use) {
                                        retmem(string.buffer.start);
                                }
                                if (debug_level > 0) {
                                        (void) printf(gettext("%*sDynamic depend
                                                      recursion_level,
                                                      "",
                                                      first_member->name->
                                                      string_mb,
                                                      true_target->string_mb);
                                }
                                first_member->name->is_member = lib->is_member;
                                /* Add member property to member */
                                prop = maybe_append_prop(first_member->name,
```

```
                                                         member_prop);
                                prop->body.member.library = lib;
                                prop->body.member.entry = NULL;
                                prop->body.member.member = member;
                                if (first_member == dependency) {
                                        break;
                                }
                        }
                }
        }
        Wstring wcb;
        /* Then scan all the dependencies again. This time we want to expand */
        /* shell file wildcards */
        for (remove = &line->body.line.dependencies, dependency = *remove;
             dependency != NULL;
             dependency = *remove) {
                if (dependency->name == NULL) {
                        dependency = *remove = (*remove)->next;
                        continue;
                }
                /* If dependency name string contains shell wildcards */
                /* replace the name with the expansion */
                if (dependency->name->wildcard) {
                        wcb.init(dependency->name);
                        if ((start = (wchar_t *) wcschr(wcb.get_string(),
                        if ((start = (wchar_t *) wschr(wcb.get_string(),
                                           (int) parenleft_char)) != NULL) {
                                /* lib(*) type pattern */
                                library = buffer;
                                (void) wcsncpy(buffer,
                                (void) wsncpy(buffer,
                                              wcb.get_string(),
                                              start - wcb.get_string());
                                buffer[start-wcb.get_string()] =
                                  (int) nul_char;
                                (void) wcsncpy(pattern,
                                (void) wsncpy(pattern,
                                              start + 1,
(int) (dependency->name->hash.length-(start-wcb.get_string())-2));
                                pattern[dependency->name->hash.length -
                                        (start-wcb.get_string()) - 2] =
                                          (int) nul_char;
                        } else {
                                library = NULL;
                                (void) wcsncpy(pattern,
                                (void) wsncpy(pattern,
                                              wcb.get_string(),
                                              (int) dependency->name->hash.lengt
                                pattern[dependency->name->hash.length] =
                                  (int) nul_char;
                        }
                        start = (wchar_t *) wcsrchr(pattern, (int) slash_char);
                        start = (wchar_t *) wsrchr(pattern, (int) slash_char);
                        if (start == NULL) {
                                directory = dot;
                                p = pattern;
                        } else {
                                directory = GETNAME(pattern, start-pattern);
                                p = start+1;
                        }
                        /* The expansion is handled by the read_dir() routine*/
                        if (read_dir(directory, p, line, library)) {
                                *remove = (*remove)->next;
                        } else {
                                remove = &dependency->next;
                        }
                } else {
                        remove = &dependency->next;
                }
        }

        /* Then unbind $@ */
        (void) SETVAR(c_at, (Name) NULL, false);
}
______unchanged_portion_omitted_

/*
 *      sccs_get(target, command)
 *
 *      Figures out if it possible to sccs get a file
 *      and builds the command to do it if it is.
 *
 *      Return value:
 *                              Indicates if sccs get failed or not
 *
 *      Parameters:
 *              target          Target to get
 *              command         Where to deposit command to use
 *
 *      Global variables used:
 *              debug_level     Should we trace activities?
 *              recursion_level Used for tracing
 *              sccs_get_rule   The rule to used for sccs getting
 */
static Doname
sccs_get(register Name target, register Property *command)
{
        register int            result;
        char                    link[MAXPATHLEN];
        String_rec              string;
        wchar_t                 name[MAXPATHLEN];
        register wchar_t        *p;
        timestruc_t             sccs_time;
        register Property       line;
        int                     sym_link_depth = 0;

        /* For sccs, we need to chase symlinks. */
        while (target->stat.is_sym_link) {
                if (sym_link_depth++ > 90) {
                        fatal(gettext("Can't read symbolic link '%s': Number of
                              target->string_mb);
                }
                /* Read the value of the link. */
                result = readlink_vroot(target->string_mb,
                                        link,
                                        sizeof(link),
                                        NULL,
                                        VROOT_DEFAULT);
                if (result == -1) {
                        fatal(gettext("Can't read symbolic link '%s': %s"),
                              target->string_mb, errmsg(errno));
                }
                link[result] = 0;
                /* Use the value to build the proper filename. */
                INIT_STRING_FROM_STACK(string, name);

                Wstring wcb(target);
                if ((link[0] != slash_char) &&
                    ((p = (wchar_t *) wcsrchr(wcb.get_string(), slash_char)) !=
                    ((p = (wchar_t *) wsrchr(wcb.get_string(), slash_char)) != N
                        append_string(wcb.get_string(), &string, p - wcb.get_str
                }
```

```
                append_string(link, &string, result);
                /* Replace the old name with the translated name. */
                target = normalize_name(string.buffer.start, string.text.p - str
                (void) exists(target);
                if (string.free_after_use) {
                        retmem(string.buffer.start);
                }
        }

        /*
         * read_dir() also reads the ?/SCCS dir and saves information
         * about which files have SCSC/s. files.
         */
        if (target->stat.has_sccs == DONT_KNOW_SCCS) {
                read_directory_of_file(target);
        }
        switch (target->stat.has_sccs) {
        case DONT_KNOW_SCCS:
                /* We dont know by now there is no SCCS/s.* */
                target->stat.has_sccs = NO_SCCS;
        case NO_SCCS:
                /*
                 * If there is no SCCS/s.* but the plain file exists,
                 * we say things are OK.
                 */
                if (target->stat.time > file_doesnt_exist) {
                        return build_ok;
                }
                /* If we cant find the plain file, we give up. */
                return build_dont_know;
        case HAS_SCCS:
                /*
                 * Pay dirt. We now need to figure out if the plain file
                 * is out of date relative to the SCCS/s.* file.
                 */
                sccs_time = exists(get_prop(target->prop,
                                            sccs_prop)->body.sccs.file);
                break;
        }

        if ((!target->has_complained &&
            (sccs_time != file_doesnt_exist) &&
            (sccs_get_rule != NULL))) {
                /* only checking */
                if (command == NULL) {
                        return build_ok;
                }
                /*
                 * We provide a command line for the target. The line is a
                 * "sccs get" command from default.mk.
                 */
                line = maybe_append_prop(target, line_prop);
                *command = line;
                if (sccs_time > target->stat.time) {
                        /*
                         * And only if the plain file is out of date do we
                         * request execution of the command.
                         */
                        line->body.line.is_out_of_date = true;
                        if (debug_level > 0) {
                                (void) printf(gettext("%*sSccs getting %s becaus
                                              recursion_level,
                                              "",
                                              target->string_mb);
                        }
                }
                line->body.line.sccs_command = true;
                line->body.line.command_template = sccs_get_rule;
                if(!svr4 && (!allrules_read || posix)) {
                   if((target->prop) &&
                      (target->prop->body.sccs.file) &&
                      (target->prop->body.sccs.file->string_mb)) {
                      if((strlen(target->prop->body.sccs.file->string_mb) ==
                        strlen(target->string_mb) + 2) &&
                        (target->prop->body.sccs.file->string_mb[0] == 's') &&
                        (target->prop->body.sccs.file->string_mb[1] == '.')) {

                         line->body.line.command_template = get_posix_rule;
                      }
                   }
                }
                line->body.line.target = target;
                /*
                 * Also make sure the rule is build with $* and $<
                 * bound properly.
                 */
                line->body.line.star = NULL;
                line->body.line.less = NULL;
                line->body.line.percent = NULL;
                return build_ok;
        }
        return build_dont_know;
}

/*
 *      read_directory_of_file(file)
 *
 *      Reads the directory the specified file lives in.
 *
 *      Parameters:
 *              file            The file we need to read dir for
 *
 *      Global variables used:
 *              dot             The Name ".", used as the default dir
 */
void
read_directory_of_file(register Name file)
{

        Wstring file_string(file);
        wchar_t * wcb = file_string.get_string();
        wchar_t usr_include_buf[MAXPATHLEN];
        wchar_t usr_include_sys_buf[MAXPATHLEN];

        register Name           directory = dot;
        register wchar_t        *p = (wchar_t *) wcsrchr(wcb,
        register wchar_t        *p = (wchar_t *) wsrchr(wcb,
                                                        (int) slash_char);
        register int            length = p - wcb;
        static Name             usr_include;
        static Name             usr_include_sys;

        if (usr_include == NULL) {
                MBSTOWCS(usr_include_buf, "/usr/include");
                usr_include = GETNAME(usr_include_buf, FIND_LENGTH);
                MBSTOWCS(usr_include_sys_buf, "/usr/include/sys");
                usr_include_sys = GETNAME(usr_include_sys_buf, FIND_LENGTH);
        }

        /*
         * If the filename contains a "/" we have to extract the path
         * Else the path defaults to ".".
```

```
         */
        if (p != NULL) {
                /*
                 * Check some popular directories first to possibly
                 * save time. Compare string length first to gain speed.
                 */
                if ((usr_include->hash.length == length) &&
                    IS_WEQUALN(usr_include_buf,
                               wcb,
                               length)) {
                        directory = usr_include;
                } else if ((usr_include_sys->hash.length == length) &&
                           IS_WEQUALN(usr_include_sys_buf,
                                      wcb,
                                      length)) {
                        directory = usr_include_sys;
                } else {
                        directory = GETNAME(wcb, length);
                }
        }
        (void) read_dir(directory,
                        (wchar_t *) NULL,
                        (Property) NULL,
                        (wchar_t *) NULL);
}

/*
 *      add_pattern_conditionals(target)
 *
 *      Scan the list of conditionals defined for pattern targets and add any
 *      that match this target to its list of conditionals.
 *
 *      Parameters:
 *              target          The target we should add conditionals for
 *
 *      Global variables used:
 *              conditionals    The list of pattern conditionals
 */
static void
add_pattern_conditionals(register Name target)
{
        register Property       conditional;
        Property                new_prop;
        Property                *previous;
        Name_rec                dummy;
        wchar_t                 *pattern;
        wchar_t                 *percent;
        int                     length;

        Wstring wcb(target);
        Wstring wcb1;

        for (conditional = get_prop(conditionals->prop, conditional_prop);
             conditional != NULL;
             conditional = get_prop(conditional->next, conditional_prop)) {
                wcb1.init(conditional->body.conditional.target);
                pattern = wcb1.get_string();
                if (pattern[1] != 0) {
                        percent = (wchar_t *) wcschr(pattern, (int) percent_char
                        percent = (wchar_t *) wschr(pattern, (int) percent_char)
                        if (!wcb.equaln(pattern, percent-pattern) ||
                            !IS_WEQUAL(wcb.get_string(wcb.length()-wcslen(percen
                            !IS_WEQUAL(wcb.get_string(wcb.length()-wslen(percent
                                continue;
                        }
                }
                for (previous = &target->prop;
                     *previous != NULL;
                     previous = &(*previous)->next) {
                        if (((*previous)->type == conditional_prop) &&
                            ((*previous)->body.conditional.sequence >
                             conditional->body.conditional.sequence)) {
                                break;
                        }
                }
                if (*previous == NULL) {
                        new_prop = append_prop(target, conditional_prop);
                } else {
                        dummy.prop = NULL;
                        new_prop = append_prop(&dummy, conditional_prop);
                        new_prop->next = *previous;
                        *previous = new_prop;
                }
                target->conditional_cnt++;
                new_prop->body.conditional = conditional->body.conditional;
        }
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
    4095 Fri May 22 11:19:41 2015
new/usr/src/cmd/make/bin/dosys.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
/*
 * 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 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      dosys.cc
 *
 *      Execute one commandline
 */

/*
 * Included files
 */
#include <fcntl.h>              /* open() */
#include <mk/defs.h>
#include <mksh/dosys.h>         /* doshell(), doexec() */
#include <mksh/misc.h>          /* getmem() */
#include <sys/stat.h>           /* open() */
#include <unistd.h>             /* getpid() */

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static  int             filter_file;
static  char            *filter_file_name;

/*
 * File table of contents
 */
static  void            redirect_stderr(void);

/*
```

```
 *      dosys(command, ignore_error, call_make, silent_error, target)
 *
 *      Check if command string contains meta chars and dispatch to
 *      the proper routine for executing one command line.
 *
 *      Return value:
 *                              Indicates if the command execution failed
 *
 *      Parameters:
 *              command         The command to run
 *              ignore_error    Should make abort when an error is seen?
 *              call_make       Did command reference $(MAKE) ?
 *              silent_error    Should error messages be suppressed for pmake?
 *              target          Target we are building
 *
 *      Global variables used:
 *              do_not_exec_rule Is -n on?
 *              working_on_targets We started processing real targets
 */
Doname
dosys(register Name command, register Boolean ignore_error, register Boolean cal
{
        timestruc_t             before;
        register int            length = command->hash.length;
        Wstring                 wcb(command);
        register wchar_t        *p = wcb.get_string();
        register wchar_t        *q;
        Doname                  result;

        /* Strip spaces from head of command string */
        while (iswspace(*p)) {
                p++, length--;
        }
        if (*p == (int) nul_char) {
                return build_failed;
        }
        /* If we are faking it we just return */
        if (do_not_exec_rule &&
            working_on_targets &&
            !call_make &&
            !always_exec) {
                return build_ok;
        }
        /* no_action_was_taken is used to print special message */
        no_action_was_taken = false;

        /* Copy string to make it OK to write it. */
        q = ALLOC_WC(length + 1);
        (void) wcscpy(q, p);
        (void) wscpy(q, p);
        /* Write the state file iff this command uses make. */
        if (call_make && command_changed) {
                write_state_file(0, false);
        }
        make_state->stat.time = file_no_time;
        (void)exists(make_state);
        before = make_state->stat.time;
        /*
         * Run command directly if it contains no shell meta chars,
         * else run it using the shell.
         */
        if (await(ignore_error,
                  silent_error,
                  target,
                  wcb.get_string(),
                  command->meta ?
```

```
                        doshell(q, ignore_error,
                                stdout_file, stderr_file, 0) :
                        doexec(q, ignore_error,
                               stdout_file, stderr_file,
                               vroot_path, 0),
                      NULL,
                      -1
                      )) {
                    result = build_ok;
            } else {
                    result = build_failed;
            }
            retmem(q);

            if ((report_dependencies_level == 0) &&
                call_make) {
                    make_state->stat.time = file_no_time;
                    (void)exists(make_state);
                    if (before == make_state->stat.time) {
                            return result;
                    }
                    makefile_type = reading_statefile;
                    if (read_trace_level > 1) {
                            trace_reader = true;
                    }
                    temp_file_number++;
                    (void) read_simple_file(make_state,
                                            false,
                                            false,
                                            false,
                                            false,
                                            false,
                                            true);
                    trace_reader = false;
            }
            return result;
    }
```
______***unchanged_portion_omitted_***

****
   18370 Fri May 22 11:19:41 2015
new/usr/src/cmd/make/bin/files.cc
make: use the more modern wchar routines, not widec.h
****
______*unchanged_portion_omitted_*

```
/*
 *      vpath_exists(target)
 *
 *      Called if exists() discovers that there is a VPATH defined.
 *      This function stats the VPATH translation of the target.
 *
 *      Return value:
 *                              The time the target was created
 *
 *      Parameters:
 *              target          The target to check
 *
 *      Global variables used:
 *              vpath_name      The Name "VPATH", used to get macro value
 */
static timestruc_t&
vpath_exists(register Name target)
{
        wchar_t                 *vpath;
        wchar_t                 file_name[MAXPATHLEN];
        wchar_t                 *name_p;
        Name                    alias;

        /*
         * To avoid recursive search through VPATH when exists(alias) is called
         */
        vpath_defined = false;

        Wstring wcb(getvar(vpath_name));
        Wstring wcb1(target);

        vpath = wcb.get_string();

        while (*vpath != (int) nul_char) {
                name_p = file_name;
                while ((*vpath != (int) colon_char) &&
                       (*vpath != (int) nul_char)) {
                        *name_p++ = *vpath++;
                }
                *name_p++ = (int) slash_char;
                (void) wcscpy(name_p, wcb1.get_string());
                (void) wscpy(name_p, wcb1.get_string());
                alias = GETNAME(file_name, FIND_LENGTH);
                if (exists(alias) != file_doesnt_exist) {
                        target->stat.is_file = true;
                        target->stat.mode = alias->stat.mode;
                        target->stat.size = alias->stat.size;
                        target->stat.is_dir = alias->stat.is_dir;
                        target->stat.time = alias->stat.time;
                        maybe_append_prop(target, vpath_alias_prop)->
                                                body.vpath_alias.alias = alias;
                        target->has_vpath_alias_prop = true;
                        vpath_defined = true;
                        return alias->stat.time;
                }
                while ((*vpath != (int) nul_char) &&
                       ((*vpath == (int) colon_char) || iswspace(*vpath))) {
                        vpath++;
```

```
                }
        }
        /*
         * Restore vpath_defined
         */
        vpath_defined = true;
        return target->stat.time;
}

/*
 *      read_dir(dir, pattern, line, library)
 *
 *      Used to enter the contents of directories into makes namespace.
 *      Presence of a file is important when scanning for implicit rules.
 *      read_dir() is also used to expand wildcards in dependency lists.
 *
 *      Return value:
 *                              Non-0 if we found files to match the pattern
 *
 *      Parameters:
 *              dir             Path to the directory to read
 *              pattern         Pattern for that files should match or NULL
 *              line            When we scan using a pattern we enter files
 *                              we find as dependencies for this line
 *              library         If we scan for "lib.a(<wildcard-member>)"
 *
 *      Global variables used:
 *              debug_level     Should we trace the dir reading?
 *              dot             The Name ".", compared against
 *              sccs_dir_path   The path to the SCCS dir (from PROJECTDIR)
 *              vpath_defined   Was the variable VPATH defined in environment?
 *              vpath_name      The Name "VPATH", use to get macro value
 */
int
read_dir(Name dir, wchar_t *pattern, Property line, wchar_t *library)
{
        wchar_t                 file_name[MAXPATHLEN];
        wchar_t                 *file_name_p = file_name;
        Name                    file;
        wchar_t                 plain_file_name[MAXPATHLEN];
        wchar_t                 *plain_file_name_p;
        Name                    plain_file;
        wchar_t                 tmp_wcs_buffer[MAXPATHLEN];
        DIR                     *dir_fd;
        int                     m_local_dependency=0;
#define d_fileno d_ino
        register struct dirent  *dp;
        wchar_t                 *vpath = NULL;
        wchar_t                 *p;
        int                     result = 0;

        if(dir->hash.length >= MAXPATHLEN) {
                return 0;
        }

        Wstring wcb(dir);
        Wstring vps;

        /* A directory is only read once unless we need to expand wildcards. */
        if (pattern == NULL) {
                if (dir->has_read_dir) {
                        return 0;
                }
                dir->has_read_dir = true;
        }
        /* Check if VPATH is active and setup list if it is. */
        if (vpath_defined && (dir == dot)) {
                vps.init(getvar(vpath_name));
                vpath = vps.get_string();
        }

        /*
         * Prepare the string where we build the full name of the
         * files in the directory.
         */
        if ((dir->hash.length > 1) || (wcb.get_string()[0] != (int) period_char)
                (void) wcscpy(file_name, wcb.get_string());
                (void) wscpy(file_name, wcb.get_string());
                MBSTOWCS(wcs_buffer, "/");
                (void) wcscat(file_name, wcs_buffer);
                file_name_p = file_name + wcslen(file_name);
                (void) wscat(file_name, wcs_buffer);
                file_name_p = file_name + wslen(file_name);
        }

        /* Open the directory. */
vpath_loop:
        dir_fd = opendir(dir->string_mb);
        if (dir_fd == NULL) {
                return 0;
        }

        /* Read all the directory entries. */
        while ((dp = readdir(dir_fd)) != NULL) {
                /* We ignore "." and ".." */
                if ((dp->d_fileno == 0) ||
                    ((dp->d_name[0] == (int) period_char) &&
                     ((dp->d_name[1] == 0) ||
                      ((dp->d_name[1] == (int) period_char) &&
                       (dp->d_name[2] == 0))))) {
                        continue;
                }
                /*
                 * Build the full name of the file using whatever
                 * path supplied to the function.
                 */
                MBSTOWCS(tmp_wcs_buffer, dp->d_name);
                (void) wcscpy(file_name_p, tmp_wcs_buffer);
                (void) wscpy(file_name_p, tmp_wcs_buffer);
                file = enter_file_name(file_name, library);
                if ((pattern != NULL) && amatch(tmp_wcs_buffer, pattern)) {
                        /*
                         * If we are expanding a wildcard pattern, we
                         * enter the file as a dependency for the target.
                         */
                        if (debug_level > 0){
                                WCSTOMBS(mbs_buffer, pattern);
                                (void) printf(gettext("'%s: %s' due to %s expans
                                              line->body.line.target->string_mb,
                                              file->string_mb,
                                              mbs_buffer);
                        }
                        enter_dependency(line, file, false);
                        result++;
                } else {
                        /*
                         * If the file has an SCCS/s. file,
                         * we will detect that later on.
                         */
                        file->stat.has_sccs = NO_SCCS;
                /*
                 * If this is an s. file, we also enter it as if it
```

```
                 * existed in the plain directory.
                 */
                if ((dp->d_name[0] == 's') &&
                    (dp->d_name[1] == (int) period_char)) {

                        MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
                        plain_file_name_p = plain_file_name;
                        (void) wcscpy(plain_file_name_p, tmp_wcs_buffer);
                        (void) wscpy(plain_file_name_p, tmp_wcs_buffer);
                        plain_file = GETNAME(plain_file_name, FIND_LENGTH);
                        plain_file->stat.is_file = true;
                        plain_file->stat.has_sccs = HAS_SCCS;
                        /*
                         * Enter the s. file as a dependency for the
                         * plain file.
                         */
                        maybe_append_prop(plain_file, sccs_prop)->
                          body.sccs.file = file;
                        MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
                        if ((pattern != NULL) &&
                            amatch(tmp_wcs_buffer, pattern)) {
                                if (debug_level > 0) {
                                        WCSTOMBS(mbs_buffer, pattern);
                                        (void) printf(gettext("'%s: %s' due to %
                                                      line->body.line.target->
                                                      string_mb,
                                                      plain_file->string_mb,
                                                      mbs_buffer);
                                }
                                enter_dependency(line, plain_file, false);
                                result++;
                        }
                }
              }
        }
        (void) closedir(dir_fd);
        if ((vpath != NULL) && (*vpath != (int) nul_char)) {
                while ((*vpath != (int) nul_char) &&
                       (iswspace(*vpath) || (*vpath == (int) colon_char))) {
                        vpath++;
                }
                p = vpath;
                while ((*vpath != (int) colon_char) &&
                       (*vpath != (int) nul_char)) {
                        vpath++;
                }
                if (vpath > p) {
                        dir = GETNAME(p, vpath - p);
                        goto vpath_loop;
                }
        }
/*
 * look into SCCS directory only if it's not svr4. For svr4 dont do that.
 */

/*
 * Now read the SCCS directory.
 * Files in the SCSC directory are considered to be part of the set of
 * files in the plain directory. They are also entered in their own right.
 * Prepare the string where we build the true name of the SCCS files.
 */
        (void) wcsncpy(plain_file_name,
        (void) wsncpy(plain_file_name,
                      file_name,
                      file_name_p - file_name);
        plain_file_name[file_name_p - file_name] = 0;
        plain_file_name_p = plain_file_name + wcslen(plain_file_name);
        plain_file_name_p = plain_file_name + wslen(plain_file_name);

        if(!svr4) {

          if (sccs_dir_path != NULL) {
                wchar_t         tmp_wchar;
                wchar_t         path[MAXPATHLEN];
                char            mb_path[MAXPATHLEN];

                if (file_name_p - file_name > 0) {
                        tmp_wchar = *file_name_p;
                        *file_name_p = 0;
                        WCSTOMBS(mbs_buffer, file_name);
                        (void) sprintf(mb_path, "%s/%s/SCCS",
                                        sccs_dir_path,
                                        mbs_buffer);
                        *file_name_p = tmp_wchar;
                } else {
                        (void) sprintf(mb_path, "%s/SCCS", sccs_dir_path);
                }
                MBSTOWCS(path, mb_path);
                (void) wcscpy(file_name, path);
                (void) wscpy(file_name, path);
          } else {
                MBSTOWCS(wcs_buffer, "SCCS");
                (void) wcscpy(file_name_p, wcs_buffer);
                (void) wscpy(file_name_p, wcs_buffer);
          }
        } else {
                MBSTOWCS(wcs_buffer, ".");
                (void) wcscpy(file_name_p, wcs_buffer);
                (void) wscpy(file_name_p, wcs_buffer);
        }
        /* Internalize the constructed SCCS dir name. */
        (void) exists(dir = GETNAME(file_name, FIND_LENGTH));
        /* Just give up if the directory file doesnt exist. */
        if (!dir->stat.is_file) {
                return result;
        }
        /* Open the directory. */
        dir_fd = opendir(dir->string_mb);
        if (dir_fd == NULL) {
                return result;
        }
        MBSTOWCS(wcs_buffer, "/");
        (void) wcscat(file_name, wcs_buffer);
        file_name_p = file_name + wcslen(file_name);
        (void) wscat(file_name, wcs_buffer);
        file_name_p = file_name + wslen(file_name);

        while ((dp = readdir(dir_fd)) != NULL) {
                if ((dp->d_fileno == 0) ||
                    ((dp->d_name[0] == (int) period_char) &&
                     ((dp->d_name[1] == 0) ||
                      ((dp->d_name[1] == (int) period_char) &&
                       (dp->d_name[2] == 0))))) {
                        continue;
                }
                /* Construct and internalize the true name of the SCCS file. */
                MBSTOWCS(wcs_buffer, dp->d_name);
                (void) wcscpy(file_name_p, wcs_buffer);
                (void) wscpy(file_name_p, wcs_buffer);
                file = GETNAME(file_name, FIND_LENGTH);
                file->stat.is_file = true;
                file->stat.has_sccs = NO_SCCS;
```

```
                /*
                 * If this is an s. file, we also enter it as if it
                 * existed in the plain directory.
                 */
                if ((dp->d_name[0] == 's') &&
                    (dp->d_name[1] == (int) period_char)) {

                        MBSTOWCS(wcs_buffer, dp->d_name + 2);
                        (void) wcscpy(plain_file_name_p, wcs_buffer);
                        (void) wscpy(plain_file_name_p, wcs_buffer);
                        plain_file = GETNAME(plain_file_name, FIND_LENGTH);
                        plain_file->stat.is_file = true;
                        plain_file->stat.has_sccs = HAS_SCCS;
                                /* if sccs dependency is already set,skip */
                        if(plain_file->prop) {
                                Property sprop = get_prop(plain_file->prop,sccs_
                                if(sprop != NULL) {
                                        if (sprop->body.sccs.file) {
                                                goto try_pattern;
                                        }
                                }
                        }

                        /*
                         * Enter the s. file as a dependency for the
                         * plain file.
                         */
                        maybe_append_prop(plain_file, sccs_prop)->
                          body.sccs.file = file;
try_pattern:
                        MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
                        if ((pattern != NULL) &&
                            amatch(tmp_wcs_buffer, pattern)) {
                                if (debug_level > 0) {
                                        WCSTOMBS(mbs_buffer, pattern);
                                        (void) printf(gettext("'%s: %s' due to %
                                                      line->body.line.target->
                                                      string_mb,
                                                      plain_file->string_mb,
                                                      mbs_buffer);
                                }
                                enter_dependency(line, plain_file, false);
                                result++;
                        }
                }
        }
        (void) closedir(dir_fd);

        return result;
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
   43023 Fri May 22 11:19:42 2015
new/usr/src/cmd/make/bin/implicit.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
______unchanged_portion_omitted_

/*
 *      find_percent_rule(target, command, rechecking)
 *
 *      Tries to find a rule from the list of wildcard matched rules.
 *      It scans the list attempting to match the target.
 *      For each target match it checks if the corresponding source exists.
 *      If it does the match is returned.
 *      The percent_list is built at makefile read time.
 *      Each percent rule get one entry on the list.
 *
 *      Return value:
 *                              Indicates if the scan failed or not
 *
 *      Parameters:
 *              target          The target we need a rule for
 *              command         Pointer to slot where we stuff cmd, if found
 *              rechecking      true if we are rechecking target which depends
 *                              on conditional macro and keep_state is set
 *
 *      Global variables used:
 *              debug_level     Indicates how much tracing to do
 *              percent_list    List of all percent rules
 *              recursion_level Used for tracing
 *              empty_name
 */
Doname
find_percent_rule(register Name target, Property *command, Boolean rechecking)
{
        register Percent        pat_rule, pat_depe;
        register Name           depe_to_check;
        register Dependency     depe;
        register Property       line;
        String_rec              string;
        wchar_t                 string_buf[STRING_BUFFER_LENGTH];
        String_rec              percent;
        wchar_t                 percent_buf[STRING_BUFFER_LENGTH];
        Name                    true_target = target;
        Name                    less;
        Boolean                 nonpattern_less;
        Boolean                 dep_name_found = false;
        Doname                  result = build_dont_know;
        Percent                 rule_candidate = NULL;
        Boolean                 rule_maybe_ok;
        Boolean                 is_pattern;

        /* If the target is constructed for a "::" target we consider that */
        if (target->has_target_prop) {
                true_target = get_prop(target->prop,
                                       target_prop)->body.target.target;
        }
        if (target->has_long_member_name) {
                true_target = get_prop(target->prop,
                                       long_member_name_prop)->body.long_member_
        }
        if (debug_level > 1) {
                (void) printf(gettext("%*sLooking for %% rule for %s\n"),
                              recursion_level,
                              "",
                              true_target->string_mb);
```

```
        }
        for (pat_rule = percent_list;
             pat_rule != NULL;
             pat_rule = pat_rule->next) {
                /* Avoid infinite recursion when expanding patterns */
                if (pat_rule->being_expanded == true) {
                        continue;
                }

                /* Mark this pat_rule as "maybe ok". If no % rule is found
                   make will use this rule. The following algorithm is used:
                   1) make scans all pattern rules in order to find the rule
                      where ALL dependencies, including nonpattern ones, exist o
                      can be built (GNU behaviour). If such rule is found make
                      will apply it.
                   2) During this check make also remembers the first pattern ru
                      where all PATTERN dependencies can be build (no matter wha
                      happens with nonpattern dependencies).
                   3) If no rule satisfying 1) is found, make will apply the rul
                      remembered in 2) if there is one.
                */
                rule_maybe_ok = true;

                /* used to track first percent dependency */
                less = NULL;
                nonpattern_less = true;

                /* check whether pattern matches.
                   if it matches, percent string will contain matched percent pa
                if (!match_found_with_pattern(true_target, pat_rule, &percent, p
                        continue;
                }
                if (pat_rule->dependencies != NULL) {
                        for (pat_depe = pat_rule->dependencies;
                             pat_depe != NULL;
                             pat_depe = pat_depe->next) {
                                /* checking result for dependency */
                                result = build_dont_know;

                                dep_name_found = true;
                                if (pat_depe->name->percent) {
                                        is_pattern = true;
                                        /* build dependency name */
                                        INIT_STRING_FROM_STACK(string, string_bu
                                        construct_string_from_pattern(pat_depe,
                                        depe_to_check = getname_fn(string.buffer
                                                FIND_LENGTH,
                                                false,
                                                &dep_name_found
                                        );

                                        if ((less == NULL) || nonpattern_less) {
                                                less = depe_to_check;
                                                nonpattern_less = false;
                                        }
                                } else {
                                        /* nonpattern dependency */
                                        is_pattern = false;
                                        depe_to_check = pat_depe->name;
                                        if(depe_to_check->dollar) {
                                                INIT_STRING_FROM_STACK(string, s
                                                expand_value(depe_to_check, &str
                                                depe_to_check = getname_fn(strin
                                                        FIND_LENGTH,
                                                        false,
                                                        &dep_name_found
                                                );
                                        }
                                        if (less == NULL) {
                                                less = depe_to_check;
                                        }
                                }

                                if (depe_to_check == empty_name) {
                                                result = build_ok;
                                } else {
                                        if (debug_level > 1) {
                                                (void) printf(gettext("%*sTrying
                                                              recursion_level,
                                                              "",
                                                              depe_to_check->str
                                        }

                                        pat_rule->being_expanded = true;

                                        /* suppress message output */
                                        int save_debug_level = debug_level;
                                        debug_level = 0;

                                        /* check whether dependency can be built
                                        if (dependency_exists(depe_to_check,
                                              get_prop(target->prop,
                                                       line_prop)))
                                        {
                                                result = (Doname) depe_to_check-
                                        } else {
                                                if(actual_doname) {
                                                        result = doname(depe_to_
                                                } else {
                                                        result = target_can_be_b
                                                }
                                                if(!dep_name_found) {
                                                        if(result != build_ok &&
                                                                free_name(depe_t
                                                        } else {
                                                                store_name(depe_
                                                        }
                                                }
                                        }
                                        if(result != build_ok && is_pattern) {
                                                rule_maybe_ok = false;
                                        }

                                        /* restore debug_level */
                                        debug_level = save_debug_level;
                                }

                                if (pat_depe->name->percent) {
                                        if (string.free_after_use) {
                                                retmem(string.buffer.start);
                                        }
                                }
                                /* make can't figure out how to make this depend
                                if (result != build_ok && result != build_runnin
                                        pat_rule->being_expanded = false;
                                        break;
                                }
                        }
                } else {
                        result = build_ok;
                }
```

```
                /* this pattern rule is the needed one since all dependencies co
                if (result == build_ok || result == build_running) {
                        break;
                }

                /* Make does not know how to build some of dependencies from thi
                   But if all "pattern" dependencies can be built, we remember t
                   as a candidate for the case if no other pattern rule found.
                */
                if(rule_maybe_ok && rule_candidate == NULL) {
                        rule_candidate = pat_rule;
                }
        }

        /* if no pattern matching rule was found, use the remembered candidate
           or return build_dont_know if there is no candidate.
        */
        if (result != build_ok && result != build_running) {
                if(rule_candidate) {
                        pat_rule = rule_candidate;
                } else {
                        return build_dont_know;
                }
        }

        /* if we are performing only check whether dependency could be built wit
           return success */
        if (command == NULL) {
                if(pat_rule != NULL) {
                        pat_rule->being_expanded = false;
                }
                return result;
        }

        if (debug_level > 1) {
                (void) printf(gettext("%*sMatched %s:"),
                                      recursion_level,
                                      "",
                                      target->string_mb);

                for (pat_depe = pat_rule->dependencies;
                     pat_depe != NULL;
                     pat_depe = pat_depe->next) {
                        if (pat_depe->name->percent) {
                                INIT_STRING_FROM_STACK(string, string_buf);
                                construct_string_from_pattern(pat_depe, &percent
                                depe_to_check = GETNAME(string.buffer.start, FIN
                        } else {
                                depe_to_check = pat_depe->name;
                                if(depe_to_check->dollar) {
                                        INIT_STRING_FROM_STACK(string, string_bu
                                        expand_value(depe_to_check, &string, fal
                                        depe_to_check = GETNAME(string.buffer.st
                                }
                        }

                        if (depe_to_check != empty_name) {
                                (void) printf(" %s", depe_to_check->string_mb);
                        }
                }

                (void) printf(gettext(" from: %s:"),
                              pat_rule->name->string_mb);

                for (pat_depe = pat_rule->dependencies;
                     pat_depe != NULL;
                     pat_depe = pat_depe->next) {
                        (void) printf(" %s", pat_depe->name->string_mb);
                }

                (void) printf("\n");
        }

        if (true_target->colons == no_colon) {
                true_target->colons = one_colon;
        }

        /* create deppendency list and target group from matched pattern rule */
        create_target_group_and_dependencies_list(target, pat_rule, &percent);

        /* save command */
        line = get_prop(target->prop, line_prop);
        *command = line;

        /* free query chain if one exist */
        while(line->body.line.query != NULL) {
                Chain to_free = line->body.line.query;
                line->body.line.query = line->body.line.query->next;
                retmem_mb((char *) to_free);
        }

        if (line->body.line.dependencies != NULL) {
                /* build all collected dependencies */
                for (depe = line->body.line.dependencies;
                     depe != NULL;
                     depe = depe->next) {
                        actual_doname = true;
                        result = doname_check(depe->name, true, true, depe->auto

                        actual_doname = false;
                        if (result == build_failed) {
                                pat_rule->being_expanded = false;
                                return build_failed;
                        }
                        if (result == build_running) {
                                pat_rule->being_expanded = false;
                                return build_running;
                        }

                        if ((depe->name->stat.time > line->body.line.dependency_
                            (debug_level > 1)) {
                                (void) printf(gettext("%*sDate(%s)=%s Date-depen
                                              recursion_level,
                                              "",
                                              depe->name->string_mb,
                                              time_to_string(depe->name->stat.ti
                                              true_target->string_mb,
                                              time_to_string(line->body.line.dep
                        }

                        line->body.line.dependency_time =
                          MAX(line->body.line.dependency_time, depe->name->stat.

                        /* determine whether this dependency made target out of
                        Boolean out_of_date;
                        if (target->is_member || depe->name->is_member) {
                                out_of_date = (Boolean) OUT_OF_DATE_SEC(target->
                        } else {
                                out_of_date = (Boolean) OUT_OF_DATE(target->stat
                        }
                        if (build_unconditional || out_of_date) {
                                if(!rechecking) {
```

```
                                        line->body.line.is_out_of_date = true;
                                }
                                add_target_to_chain(depe->name, &(line->body.lin

                                if (debug_level > 0) {
                                        (void) printf(gettext("%*sBuilding %s us
                                                      recursion_level,
                                                      "",
                                                      true_target->string_mb,
                                                      pat_rule->name->string_mb)

                                        for (pat_depe = pat_rule->dependencies;
                                             pat_depe != NULL;
                                             pat_depe = pat_depe->next) {
                                                (void) printf(" %s", pat_depe->n
                                        }

                                        (void) printf(gettext(" because it is ou
                                                      depe->name->string_mb);
                                }
                        }
                }
        } else {
                if ((true_target->stat.time <= file_doesnt_exist) ||
                    (true_target->stat.time < line->body.line.dependency_time))
                        if(!rechecking) {
                                line->body.line.is_out_of_date = true;
                        }
                        if (debug_level > 0) {
                                (void) printf(gettext("%*sBuilding %s using patt
                                              recursion_level,
                                              "",
                                              true_target->string_mb,
                                              pat_rule->name->string_mb,
                                              (target->stat.time > file_doesnt_e
                                              gettext("because it is out of date
                                              gettext("because it does not exist
                        }
                }
        }

        /* enter explicit rule from percent rule */
        Name lmn_target = true_target;
        if (true_target->has_long_member_name) {
                lmn_target = get_prop(true_target->prop, long_member_name_prop)-
        }
        line->body.line.sccs_command = false;
        line->body.line.target = true_target;
        line->body.line.command_template = pat_rule->command_template;
        line->body.line.star = GETNAME(percent.buffer.start, FIND_LENGTH);
        line->body.line.less = less;

        if (lmn_target->parenleft) {
                Wstring lmn_string(lmn_target);

                wchar_t *left = (wchar_t *) wcschr(lmn_string.get_string(), (int
                wchar_t *right = (wchar_t *) wcschr(lmn_string.get_string(), (in
                wchar_t *left = (wchar_t *) wschr(lmn_string.get_string(), (int)
                wchar_t *right = (wchar_t *) wschr(lmn_string.get_string(), (int

                if ((left == NULL) || (right == NULL)) {
                        line->body.line.percent = NULL;
                } else {
                        line->body.line.percent = GETNAME(left + 1, right - left
                }
        } else {
                line->body.line.percent = NULL;
        }
        pat_rule->being_expanded = false;

        return result;
}
```
______***unchanged_portion_omitted_***

****
    4001 Fri May 22 11:19:42 2015
new/usr/src/cmd/make/bin/macro.cc
make: use the more modern wchar routines, not widec.h
****

______***unchanged_portion_omitted_***

```
/*
 *      setvar_envvar()
 *
 *      This function scans the list of environment variables that have
 *      dynamic values and sets them.
 *
 *      Parameters:
 *
 *      Global variables used:
 *              envvar          A list of environment vars with $ in value
 */
void
setvar_envvar(void)
{
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        int                     length;
        register        char    *mbs, *tmp_mbs_buffer = NULL;
        register        char    *env, *tmp_mbs_buffer2 = NULL;
        Envvar                  p;
        String_rec              value;

        for (p = envvar; p != NULL; p = p->next) {
                if (p->already_put
                    ) {
                        continue;
                }
                INIT_STRING_FROM_STACK(value, buffer);
                expand_value(p->value, &value, false);
                if ((length = wcslen(value.buffer.start)) >= MAXPATHLEN) {
                if ((length = wslen(value.buffer.start)) >= MAXPATHLEN) {
                        mbs = tmp_mbs_buffer = getmem((length + 1) * MB_LEN_MAX)
                        (void) wcstombs(mbs,
                                        value.buffer.start,
                                        (length + 1) * MB_LEN_MAX);
                } else {
                        mbs = mbs_buffer;
                        WCSTOMBS(mbs, value.buffer.start);
                }
                length = 2 + strlen(p->name->string_mb) + strlen(mbs);
                if (!p->already_put || length > (MAXPATHLEN * MB_LEN_MAX)) {
                        env = tmp_mbs_buffer2 = getmem(length);
                } else {
                        env = mbs_buffer2;
                }
                (void) sprintf(env,
                               "%s=%s",
                               p->name->string_mb,
                               mbs);
                if (!p->already_put) {
                        (void) putenv(env);
                        p->already_put = true;
                        if (p->env_string) {
                                retmem_mb(p->env_string);
                        }
                        p->env_string = env;
                        tmp_mbs_buffer2 = NULL; // We should not return this mem
                }
                if (tmp_mbs_buffer2) {
                        retmem_mb(tmp_mbs_buffer2);
```

```
                        tmp_mbs_buffer2 = NULL;
                }
                if (tmp_mbs_buffer) {
                        retmem_mb(tmp_mbs_buffer);
                        tmp_mbs_buffer = NULL;
                }
        }
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
   86753 Fri May 22 11:19:43 2015
new/usr/src/cmd/make/bin/main.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
______unchanged_portion_omitted_

/*
 *      read_files_and_state(argc, argv)
 *
 *      Read the makefiles we care about and the environment
 *      Also read the = style command line options
 *
 *      Parameters:
 *              argc            You know what this is
 *              argv            You know what this is
 *
 *      Static variables used:
 *              env_wins        make -e, determines if env vars are RO
 *              ignore_default_mk make -r, determines if make.rules is read
 *              not_auto_depen  dwight
 *
 *      Global variables used:
 *              default_target_to_build Set to first proper target from file
 *              do_not_exec_rule Set to false when makfile is made
 *              dot             The Name ".", used to read current dir
 *              empty_name      The Name "", use as macro value
 *              keep_state      Set if KEEP_STATE is in environment
 *              make_state      The Name ".make.state", used to read file
 *              makefile_type   Set to type of file being read
 *              makeflags       The Name "MAKEFLAGS", used to set macro value
 *              not_auto        dwight
 *              read_trace_level Checked to se if the reader should trace
 *              report_dependencies If -P is on we do not read .make.state
 *              trace_reader    Set if reader should trace
 *              virtual_root    The Name "VIRTUAL_ROOT", used to check value
 */
static void
read_files_and_state(int argc, char **argv)
{
        wchar_t                 buffer[1000];
        wchar_t                 buffer_posix[1000];
        register char           ch;
        register char           *cp;
        Property                def_make_macro = NULL;
        Name                    def_make_name;
        Name                    default_makefile;
        String_rec              dest;
        wchar_t                 destbuffer[STRING_BUFFER_LENGTH];
        register int            i;
        register int            j;
        Name                    keep_state_name;
        int                     length;
        Name                    Makefile;
        register Property       macro;
        struct stat             make_state_stat;
        Name                    makefile_name;
        register int            makefile_next = 0;
        register Boolean        makefile_read = false;
        String_rec              makeflags_string;
        String_rec              makeflags_string_posix;
        String_rec *            makeflags_string_current;
        Name                    makeflags_value_saved;
        register Name           name;
        Name                    new_make_value;
        Boolean                 save_do_not_exec_rule;
```

```
        Name                    sdotMakefile;
        Name                    sdotmakefile_name;
        static wchar_t          state_file_str;
        static char             state_file_str_mb[MAXPATHLEN];
        static struct _Name     state_filename;
        Boolean                 temp;
        char                    tmp_char;
        wchar_t                 *tmp_wcs_buffer;
        register Name           value;
        ASCII_Dyn_Array         makeflags_and_macro;
        Boolean                 is_xpg4;

/*
 *      Remember current mode. It may be changed after reading makefile
 *      and we will have to correct MAKEFLAGS variable.
 */
        is_xpg4 = posix;

        MBSTOWCS(wcs_buffer, "KEEP_STATE");
        keep_state_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, "Makefile");
        Makefile = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, "makefile");
        makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, "s.makefile");
        sdotmakefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, "s.Makefile");
        sdotMakefile = GETNAME(wcs_buffer, FIND_LENGTH);

/*
 *      initialize global dependency entry for .NOT_AUTO
 */
        not_auto_depen->next = NULL;
        not_auto_depen->name = not_auto;
        not_auto_depen->automatic = not_auto_depen->stale = false;

/*
 *      Read internal definitions and rules.
 */
        if (read_trace_level > 1) {
                trace_reader = true;
        }
        if (!ignore_default_mk) {
                if (svr4) {
                        MBSTOWCS(wcs_buffer, "svr4.make.rules");
                        default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
                } else {
                        MBSTOWCS(wcs_buffer, "make.rules");
                        default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
                }
                default_makefile->stat.is_file = true;

                (void) read_makefile(default_makefile,
                                     true,
                                     false,
                                     true);
        }

        /*
         * If the user did not redefine the MAKE macro in the
         * default makefile (make.rules), then we’d like to
         * change the macro value of MAKE to be some form
         * of argv[0] for recursive MAKE builds.
         */
        MBSTOWCS(wcs_buffer, "MAKE");
        def_make_name = GETNAME(wcs_buffer, wcslen(wcs_buffer));
        def_make_name = GETNAME(wcs_buffer, wslen(wcs_buffer));
        def_make_macro = get_prop(def_make_name->prop, macro_prop);
        if ((def_make_macro != NULL) &&
            (IS_EQUAL(def_make_macro->body.macro.value->string_mb,
                      "make"))) {
                MBSTOWCS(wcs_buffer, argv_zero_string);
                new_make_value = GETNAME(wcs_buffer, wcslen(wcs_buffer));
                new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
                (void) SETVAR(def_make_name,
                              new_make_value,
                              false);
        }

        default_target_to_build = NULL;
        trace_reader = false;

/*
 *      Read environment args. Let file args which follow override unless
 *      -e option seen. If -e option is not mentioned.
 */
        read_environment(env_wins);
        if (getvar(virtual_root)->hash.length == 0) {
                maybe_append_prop(virtual_root, macro_prop)
                  ->body.macro.exported = true;
                MBSTOWCS(wcs_buffer, "/");
                (void) SETVAR(virtual_root,
                              GETNAME(wcs_buffer, FIND_LENGTH),
                              false);
        }

/*
 * We now scan mf_argv and argv to see if we need to set
 * any of the DMake-added options/variables in MAKEFLAGS.
 */

        makeflags_and_macro.start = 0;
        makeflags_and_macro.size = 0;
        enter_argv_values(mf_argc, mf_argv, &makeflags_and_macro);
        enter_argv_values(argc, argv, &makeflags_and_macro);

/*
 *      Set MFLAGS and MAKEFLAGS
 *
 *      Before reading makefile we do not know exactly which mode
 *      (posix or not) is used. So prepare two MAKEFLAGS strings
 *      for both posix and solaris modes because they are different.
 */
        INIT_STRING_FROM_STACK(makeflags_string, buffer);
        INIT_STRING_FROM_STACK(makeflags_string_posix, buffer_posix);
        append_char((int) hyphen_char, &makeflags_string);
        append_char((int) hyphen_char, &makeflags_string_posix);

        switch (read_trace_level) {
        case 2:
                append_char('D', &makeflags_string);
                append_char('D', &makeflags_string_posix);
        case 1:
                append_char('D', &makeflags_string);
                append_char('D', &makeflags_string_posix);
        }
        switch (debug_level) {
        case 2:
                append_char('d', &makeflags_string);
                append_char('d', &makeflags_string_posix);
        case 1:
                append_char('d', &makeflags_string);
```

```
                append_char('d', &makeflags_string_posix);
        }
        if (env_wins) {
                append_char('e', &makeflags_string);
                append_char('e', &makeflags_string_posix);
        }
        if (ignore_errors_all) {
                append_char('i', &makeflags_string);
                append_char('i', &makeflags_string_posix);
        }
        if (continue_after_error) {
                if (stop_after_error_ever_seen) {
                        append_char('S', &makeflags_string_posix);
                        append_char((int) space_char, &makeflags_string_posix);
                        append_char((int) hyphen_char, &makeflags_string_posix);
                }
                append_char('k', &makeflags_string);
                append_char('k', &makeflags_string_posix);
        } else {
                if (stop_after_error_ever_seen
                    && continue_after_error_ever_seen) {
                        append_char('k', &makeflags_string_posix);
                        append_char((int) space_char, &makeflags_string_posix);
                        append_char((int) hyphen_char, &makeflags_string_posix);
                        append_char('S', &makeflags_string_posix);
                }
        }
        if (do_not_exec_rule) {
                append_char('n', &makeflags_string);
                append_char('n', &makeflags_string_posix);
        }
        switch (report_dependencies_level) {
        case 4:
                append_char('P', &makeflags_string);
                append_char('P', &makeflags_string_posix);
        case 3:
                append_char('P', &makeflags_string);
                append_char('P', &makeflags_string_posix);
        case 2:
                append_char('P', &makeflags_string);
                append_char('P', &makeflags_string_posix);
        case 1:
                append_char('P', &makeflags_string);
                append_char('P', &makeflags_string_posix);
        }
        if (trace_status) {
                append_char('p', &makeflags_string);
                append_char('p', &makeflags_string_posix);
        }
        if (quest) {
                append_char('q', &makeflags_string);
                append_char('q', &makeflags_string_posix);
        }
        if (silent_all) {
                append_char('s', &makeflags_string);
                append_char('s', &makeflags_string_posix);
        }
        if (touch) {
                append_char('t', &makeflags_string);
                append_char('t', &makeflags_string_posix);
        }
        if (build_unconditional) {
                append_char('u', &makeflags_string);
                append_char('u', &makeflags_string_posix);
        }
        if (report_cwd) {
                append_char('w', &makeflags_string);
                append_char('w', &makeflags_string_posix);
        }
        /* -c dmake_rcfile */
        if (dmake_rcfile_specified) {
                MBSTOWCS(wcs_buffer, "DMAKE_RCFILE");
                dmake_rcfile = GETNAME(wcs_buffer, FIND_LENGTH);
                append_makeflags_string(dmake_rcfile, &makeflags_string);
                append_makeflags_string(dmake_rcfile, &makeflags_string_posix);
        }
        /* -g dmake_group */
        if (dmake_group_specified) {
                MBSTOWCS(wcs_buffer, "DMAKE_GROUP");
                dmake_group = GETNAME(wcs_buffer, FIND_LENGTH);
                append_makeflags_string(dmake_group, &makeflags_string);
                append_makeflags_string(dmake_group, &makeflags_string_posix);
        }
        /* -j dmake_max_jobs */
        if (dmake_max_jobs_specified) {
                MBSTOWCS(wcs_buffer, "DMAKE_MAX_JOBS");
                dmake_max_jobs = GETNAME(wcs_buffer, FIND_LENGTH);
                append_makeflags_string(dmake_max_jobs, &makeflags_string);
                append_makeflags_string(dmake_max_jobs, &makeflags_string_posix)
        }
        /* -m dmake_mode */
        if (dmake_mode_specified) {
                MBSTOWCS(wcs_buffer, "DMAKE_MODE");
                dmake_mode = GETNAME(wcs_buffer, FIND_LENGTH);
                append_makeflags_string(dmake_mode, &makeflags_string);
                append_makeflags_string(dmake_mode, &makeflags_string_posix);
        }
        /* -x dmake_compat_mode */
//      if (dmake_compat_mode_specified) {
//              MBSTOWCS(wcs_buffer, "SUN_MAKE_COMPAT_MODE");
//              dmake_compat_mode = GETNAME(wcs_buffer, FIND_LENGTH);
//              append_makeflags_string(dmake_compat_mode, &makeflags_string);
//              append_makeflags_string(dmake_compat_mode, &makeflags_string_pos
//      }
        /* -x dmake_output_mode */
        if (dmake_output_mode_specified) {
                MBSTOWCS(wcs_buffer, "DMAKE_OUTPUT_MODE");
                dmake_output_mode = GETNAME(wcs_buffer, FIND_LENGTH);
                append_makeflags_string(dmake_output_mode, &makeflags_string);
                append_makeflags_string(dmake_output_mode, &makeflags_string_pos
        }
        /* -o dmake_odir */
        if (dmake_odir_specified) {
                MBSTOWCS(wcs_buffer, "DMAKE_ODIR");
                dmake_odir = GETNAME(wcs_buffer, FIND_LENGTH);
                append_makeflags_string(dmake_odir, &makeflags_string);
                append_makeflags_string(dmake_odir, &makeflags_string_posix);
        }
        /* -M pmake_machinesfile */
        if (pmake_machinesfile_specified) {
                MBSTOWCS(wcs_buffer, "PMAKE_MACHINESFILE");
                pmake_machinesfile = GETNAME(wcs_buffer, FIND_LENGTH);
                append_makeflags_string(pmake_machinesfile, &makeflags_string);
                append_makeflags_string(pmake_machinesfile, &makeflags_string_po
        }
        /* -R */
        if (pmake_cap_r_specified) {
                append_char((int) space_char, &makeflags_string);
                append_char((int) hyphen_char, &makeflags_string);
                append_char('R', &makeflags_string);
                append_char((int) space_char, &makeflags_string_posix);
                append_char((int) hyphen_char, &makeflags_string_posix);
```

```
                append_char('R', &makeflags_string_posix);
        }

/*
 *      Make sure MAKEFLAGS is exported
 */
        maybe_append_prop(makeflags, macro_prop)->
          body.macro.exported = true;

        if (makeflags_string.buffer.start[1] != (int) nul_char) {
                if (makeflags_string.buffer.start[1] != (int) space_char) {
                        MBSTOWCS(wcs_buffer, "MFLAGS");
                        (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
                                      GETNAME(makeflags_string.buffer.start,
                                              FIND_LENGTH),
                                      false);
                } else {
                        MBSTOWCS(wcs_buffer, "MFLAGS");
                        (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
                                      GETNAME(makeflags_string.buffer.start + 2,
                                              FIND_LENGTH),
                                      false);
                }
        }

/*
 *      Add command line macro to POSIX makeflags_string
 */
        if (makeflags_and_macro.start) {
                tmp_char = (char) space_char;
                cp = makeflags_and_macro.start;
                do {
                        append_char(tmp_char, &makeflags_string_posix);
                } while ( tmp_char = *cp++ );
                retmem_mb(makeflags_and_macro.start);
        }

/*
 *      Now set the value of MAKEFLAGS macro in accordance
 *      with current mode.
 */
        macro = maybe_append_prop(makeflags, macro_prop);
        temp = (Boolean) macro->body.macro.read_only;
        macro->body.macro.read_only = false;
        if(posix || gnu_style) {
                makeflags_string_current = &makeflags_string_posix;
        } else {
                makeflags_string_current = &makeflags_string;
        }
        if (makeflags_string_current->buffer.start[1] == (int) nul_char) {
                makeflags_value_saved =
                        GETNAME( makeflags_string_current->buffer.start + 1
                               , FIND_LENGTH
                               );
        } else {
                if (makeflags_string_current->buffer.start[1] != (int) space_cha
                        makeflags_value_saved =
                                GETNAME( makeflags_string_current->buffer.start
                                       , FIND_LENGTH
                                       );
                } else {
                        makeflags_value_saved =
                                GETNAME( makeflags_string_current->buffer.start
                                       , FIND_LENGTH
                                       );
                }
        }
        (void) SETVAR( makeflags
                     , makeflags_value_saved
                     , false
                     );
        macro->body.macro.read_only = temp;

/*
 *      Read command line "-f" arguments and ignore -c, g, j, K, M, m, O and o a
 */
        save_do_not_exec_rule = do_not_exec_rule;
        do_not_exec_rule = false;
        if (read_trace_level > 0) {
                trace_reader = true;
        }

        for (i = 1; i < argc; i++) {
                if (argv[i] &&
                    (argv[i][0] == (int) hyphen_char) &&
                    (argv[i][1] == 'f') &&
                    (argv[i][2] == (int) nul_char)) {
                        argv[i] = NULL;         /* Remove -f */
                        if (i >= argc - 1) {
                                fatal(gettext("No filename argument after -f fla
                        }
                        MBSTOWCS(wcs_buffer, argv[++i]);
                        primary_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
                        (void) read_makefile(primary_makefile, true, true, true)
                        argv[i] = NULL;         /* Remove filename */
                        makefile_read = true;
                } else if (argv[i] &&
                           (argv[i][0] == (int) hyphen_char) &&
                           (argv[i][1] == 'c' ||
                            argv[i][1] == 'g' ||
                            argv[i][1] == 'j' ||
                            argv[i][1] == 'K' ||
                            argv[i][1] == 'M' ||
                            argv[i][1] == 'm' ||
                            argv[i][1] == 'O' ||
                            argv[i][1] == 'o') &&
                           (argv[i][2] == (int) nul_char)) {
                        argv[i] = NULL;
                        argv[++i] = NULL;
                }
        }

/*
 *      If no command line "-f" args then look for "makefile", and then for
 *      "Makefile" if "makefile" isn't found.
 */
        if (!makefile_read) {
                (void) read_dir(dot,
                                (wchar_t *) NULL,
                                (Property) NULL,
                                (wchar_t *) NULL);
            if (!posix) {
                if (makefile_name->stat.is_file) {
                        if (Makefile->stat.is_file) {
                                warning(gettext("Both 'makefile' and 'Makefile'
                        }
                        primary_makefile = makefile_name;
                        makefile_read = read_makefile(makefile_name,
                                                      false,
                                                      false,
                                                      true);
                }
```

```
                if (!makefile_read &&
                    Makefile->stat.is_file) {
                        primary_makefile = Makefile;
                        makefile_read = read_makefile(Makefile,
                                                      false,
                                                      false,
                                                      true);
                }
            } else {

                enum sccs_stat save_m_has_sccs = NO_SCCS;
                enum sccs_stat save_M_has_sccs = NO_SCCS;

                if (makefile_name->stat.is_file) {
                        if (Makefile->stat.is_file) {
                                warning(gettext("Both 'makefile' and 'Makefile'
                        }
                }
                if (makefile_name->stat.is_file) {
                        if (makefile_name->stat.has_sccs == NO_SCCS) {
                           primary_makefile = makefile_name;
                           makefile_read = read_makefile(makefile_name,
                                                      false,
                                                      false,
                                                      true);
                        } else {
                          save_m_has_sccs = makefile_name->stat.has_sccs;
                          makefile_name->stat.has_sccs = NO_SCCS;
                          primary_makefile = makefile_name;
                          makefile_read = read_makefile(makefile_name,
                                                      false,
                                                      false,
                                                      true);
                        }
                }
                if (!makefile_read &&
                    Makefile->stat.is_file) {
                        if (Makefile->stat.has_sccs == NO_SCCS) {
                           primary_makefile = Makefile;
                           makefile_read = read_makefile(Makefile,
                                                      false,
                                                      false,
                                                      true);
                        } else {
                          save_M_has_sccs = Makefile->stat.has_sccs;
                          Makefile->stat.has_sccs = NO_SCCS;
                          primary_makefile = Makefile;
                          makefile_read = read_makefile(Makefile,
                                                      false,
                                                      false,
                                                      true);
                        }
                }
                if (!makefile_read &&
                        makefile_name->stat.is_file) {
                           makefile_name->stat.has_sccs = save_m_has_sccs;
                           primary_makefile = makefile_name;
                           makefile_read = read_makefile(makefile_name,
                                                      false,
                                                      false,
                                                      true);
                }
                if (!makefile_read &&
                    Makefile->stat.is_file) {
                           Makefile->stat.has_sccs = save_M_has_sccs;
                           primary_makefile = Makefile;
                           makefile_read = read_makefile(Makefile,
                                                      false,
                                                      false,
                                                      true);
                }
            }
        }
        do_not_exec_rule = save_do_not_exec_rule;
        allrules_read = makefile_read;
        trace_reader = false;

/*
 *      Now get current value of MAKEFLAGS and compare it with
 *      the saved value we set before reading makefile.
 *      If they are different then MAKEFLAGS is subsequently set by
 *      makefile, just leave it there. Otherwise, if make mode
 *      is changed by using .POSIX target in makefile we need
 *      to correct MAKEFLAGS value.
 */
        Name mf_val = getvar(makeflags);
        if( (posix != is_xpg4)
         && (!strcmp(mf_val->string_mb, makeflags_value_saved->string_mb)))
        {
                if (makeflags_string_posix.buffer.start[1] == (int) nul_char) {
                        (void) SETVAR(makeflags,
                                      GETNAME(makeflags_string_posix.buffer.star
                                              FIND_LENGTH),
                                      false);
                } else {
                        if (makeflags_string_posix.buffer.start[1] != (int) spac
                                (void) SETVAR(makeflags,
                                              GETNAME(makeflags_string_posix.buf
                                                      FIND_LENGTH),
                                              false);
                        } else {
                                (void) SETVAR(makeflags,
                                              GETNAME(makeflags_string_posix.buf
                                                      FIND_LENGTH),
                                              false);
                        }
                }
        }

        if (makeflags_string.free_after_use) {
                retmem(makeflags_string.buffer.start);
        }
        if (makeflags_string_posix.free_after_use) {
                retmem(makeflags_string_posix.buffer.start);
        }
        makeflags_string.buffer.start = NULL;
        makeflags_string_posix.buffer.start = NULL;

        if (posix) {
                /*
                 * If the user did not redefine the ARFLAGS macro in the
                 * default makefile (make.rules), then we'd like to
                 * change the macro value of ARFLAGS to be in accordance
                 * with "POSIX" requirements.
                 */
                MBSTOWCS(wcs_buffer, "ARFLAGS");
                name = GETNAME(wcs_buffer, wcslen(wcs_buffer));
                name = GETNAME(wcs_buffer, wslen(wcs_buffer));
                macro = get_prop(name->prop, macro_prop);
                if ((macro != NULL) && /* Maybe (macro == NULL) || ? */
                    (IS_EQUAL(macro->body.macro.value->string_mb,
                              "rv"))) {
```

```
                        MBSTOWCS(wcs_buffer, "-rv");
                        value = GETNAME(wcs_buffer, wcslen(wcs_buffer));
                        value = GETNAME(wcs_buffer, wslen(wcs_buffer));
                        (void) SETVAR(name,
                                      value,
                                      false);
                }
        }

        if (!posix && !svr4) {
                set_sgs_support();
        }


/*
 *      Make sure KEEP_STATE is in the environment if KEEP_STATE is on.
 */
        macro = get_prop(keep_state_name->prop, macro_prop);
        if ((macro != NULL) &&
            macro->body.macro.exported) {
                keep_state = true;
        }
        if (keep_state) {
                if (macro == NULL) {
                        macro = maybe_append_prop(keep_state_name,
                                                  macro_prop);
                }
                macro->body.macro.exported = true;
                (void) SETVAR(keep_state_name,
                              empty_name,
                              false);

                /*
                 *      Read state file
                 */

                /* Before we read state, let's make sure we have
                ** right state file.
                */
                /* just in case macro references are used in make_state file
                ** name, we better expand them at this stage using expand_value.
                */
                INIT_STRING_FROM_STACK(dest, destbuffer);
                expand_value(make_state, &dest, false);

                make_state = GETNAME(dest.buffer.start, FIND_LENGTH);

                if(!stat(make_state->string_mb, &make_state_stat)) {
                   if(!(make_state_stat.st_mode & S_IFREG) ) {
                        /* copy the make_state structure to the other
                        ** and then let make_state point to the new
                        ** one.
                        */
                      memcpy(&state_filename, make_state,sizeof(state_filename))
                      state_filename.string_mb = state_file_str_mb;
                /* Just a kludge to avoid two slashes back to back */
                      if((make_state->hash.length == 1)&&
                                (make_state->string_mb[0] == '/')) {
                         make_state->hash.length = 0;
                         make_state->string_mb[0] = '\0';
                      }
                      sprintf(state_file_str_mb,"%s%s",
                       make_state->string_mb,"/.make.state");
                      make_state = &state_filename;
                        /* adjust the length to reflect the appended string */
                      make_state->hash.length += 12;
                   }
                } else { /* the file doesn’t exist or no permission */
                   char tmp_path[MAXPATHLEN];
                   char *slashp;

                   if (slashp = strrchr(make_state->string_mb, ’/’)) {
                      strncpy(tmp_path, make_state->string_mb,
                                (slashp - make_state->string_mb));
                        tmp_path[slashp - make_state->string_mb]=0;
                      if(strlen(tmp_path)) {
                        if(stat(tmp_path, &make_state_stat)) {
                          warning(gettext("directory %s for .KEEP_STATE_FILE doe
                        }
                        if (access(tmp_path, F_OK) != 0) {
                          warning(gettext("can’t access dir %s"),tmp_path);
                        }
                      }
                   }
                }
                if (report_dependencies_level != 1) {
                        Makefile_type   makefile_type_temp = makefile_type;
                        makefile_type = reading_statefile;
                        if (read_trace_level > 1) {
                                trace_reader = true;
                        }
                        (void) read_simple_file(make_state,
                                                false,
                                                false,
                                                false,
                                                false,
                                                false,
                                                true);
                        trace_reader = false;
                        makefile_type = makefile_type_temp;
                }
        }
}

/*
 * Scan the argv for options and "=" type args and make them readonly.
 */
static void
enter_argv_values(int argc, char *argv[], ASCII_Dyn_Array *makeflags_and_macro)
{
        register char           *cp;
        register int            i;
        int                     length;
        register Name           name;
        int                     opt_separator = argc;
        char                    tmp_char;
        wchar_t                 *tmp_wcs_buffer;
        register Name           value;
        Boolean                 append = false;
        Property                macro;
        struct stat             statbuf;


        /* Read argv options and "=" type args and make them readonly. */
        makefile_type = reading_nothing;
        for (i = 1; i < argc; ++i) {
                append = false;
                if (argv[i] == NULL) {
                        continue;
                } else if (((argv[i][0] == ’-’) && (argv[i][1] == ’-’)) ||
                           ((argv[i][0] == (int) ’ ’) &&
                            (argv[i][1] == (int) ’-’) &&
```

```
                            (argv[i][2] == (int) ’ ’) &&
                            (argv[i][3] == (int) ’-’))) {
                        argv[i] = NULL;
                        opt_separator = i;
                        continue;
                } else if ((i < opt_separator) && (argv[i][0] == (int) hyphen_ch
                        switch (parse_command_option(argv[i][1])) {
                        case 1: /* -f seen */
                                ++i;
                                continue;
                        case 2: /* -c seen */
                                if (argv[i+1] == NULL) {
                                        fatal(gettext("No dmake rcfile argument
                                }
                                MBSTOWCS(wcs_buffer, "DMAKE_RCFILE");
                                name = GETNAME(wcs_buffer, FIND_LENGTH);
                                break;
                        case 4: /* -g seen */
                                if (argv[i+1] == NULL) {
                                        fatal(gettext("No dmake group argument a
                                }
                                MBSTOWCS(wcs_buffer, "DMAKE_GROUP");
                                name = GETNAME(wcs_buffer, FIND_LENGTH);
                                break;
                        case 8: /* -j seen */
                                if (argv[i+1] == NULL) {
                                        fatal(gettext("No dmake max jobs argumen
                                }
                                MBSTOWCS(wcs_buffer, "DMAKE_MAX_JOBS");
                                name = GETNAME(wcs_buffer, FIND_LENGTH);
                                break;
                        case 16: /* -M seen */
                                if (argv[i+1] == NULL) {
                                        fatal(gettext("No pmake machinesfile arg
                                }
                                MBSTOWCS(wcs_buffer, "PMAKE_MACHINESFILE");
                                name = GETNAME(wcs_buffer, FIND_LENGTH);
                                break;
                        case 32: /* -m seen */
                                if (argv[i+1] == NULL) {
                                        fatal(gettext("No dmake mode argument af
                                }
                                MBSTOWCS(wcs_buffer, "DMAKE_MODE");
                                name = GETNAME(wcs_buffer, FIND_LENGTH);
                                break;
                        case 256: /* -K seen */
                                if (argv[i+1] == NULL) {
                                        fatal(gettext("No makestate filename arg
                                }
                                MBSTOWCS(wcs_buffer, argv[i+1]);
                                make_state = GETNAME(wcs_buffer, FIND_LENGTH);
                                keep_state = true;
                                argv[i] = NULL;
                                argv[i+1] = NULL;
                                continue;
                        case 512:       /* -o seen */
                                if (argv[i+1] == NULL) {
                                        fatal(gettext("No dmake output dir argum
                                }
                                MBSTOWCS(wcs_buffer, "DMAKE_ODIR");
                                name = GETNAME(wcs_buffer, FIND_LENGTH);
                                break;
                        case 1024: /* -x seen */
                                if (argv[i+1] == NULL) {
                                        fatal(gettext("No argument after -x flag
                                }
                                length = strlen( "SUN_MAKE_COMPAT_MODE=");
                                if (strncmp(argv[i+1], "SUN_MAKE_COMPAT_MODE=",
                                        argv[i+1] = &argv[i+1][length];
                                        MBSTOWCS(wcs_buffer, "SUN_MAKE_COMPAT_MO
                                        name = GETNAME(wcs_buffer, FIND_LENGTH);
                                        dmake_compat_mode_specified = dmake_add_
                                        break;
                                }
                                length = strlen( "DMAKE_OUTPUT_MODE=");
                                if (strncmp(argv[i+1], "DMAKE_OUTPUT_MODE=", len
                                        argv[i+1] = &argv[i+1][length];
                                        MBSTOWCS(wcs_buffer, "DMAKE_OUTPUT_MODE"
                                        name = GETNAME(wcs_buffer, FIND_LENGTH);
                                        dmake_output_mode_specified = dmake_add_
                                } else {
                                        warning(gettext("Unknown argument ‘%s’ a
                                              argv[i+1]);
                                        argv[i] = argv[i + 1] = NULL;
                                        continue;
                                }
                                break;
                        default: /* Shouldn’t reach here */
                                argv[i] = NULL;
                                continue;
                        }
                        argv[i] = NULL;
                        if (i == (argc - 1)) {
                                break;
                        }
                        if ((length = strlen(argv[i+1])) >= MAXPATHLEN) {
                                tmp_wcs_buffer = ALLOC_WC(length + 1);
                                (void) mbstowcs(tmp_wcs_buffer, argv[i+1], lengt
                                value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
                                retmem(tmp_wcs_buffer);
                        } else {
                                MBSTOWCS(wcs_buffer, argv[i+1]);
                                value = GETNAME(wcs_buffer, FIND_LENGTH);
                        }
                        argv[i+1] = NULL;
                } else if ((cp = strchr(argv[i], (int) equal_char)) != NULL) {
/*
 * Combine all macro in dynamic array
 */
                        if(*(cp-1) == (int) plus_char)
                        {
                                if(isspace(*(cp-2))) {
                                        append = true;
                                        cp--;
                                }
                        }
                        if(!append)
                                append_or_replace_macro_in_dyn_array(makeflags_a

                        while (isspace(*(cp-1))) {
                                cp--;
                        }
                        tmp_char = *cp;
                        *cp = (int) nul_char;
                        MBSTOWCS(wcs_buffer, argv[i]);
                        *cp = tmp_char;
                        name = GETNAME(wcs_buffer, wcslen(wcs_buffer));
                        name = GETNAME(wcs_buffer, wslen(wcs_buffer));
                        while (*cp != (int) equal_char) {
                                cp++;
                        }
                        cp++;
```

```
                        while (isspace(*cp) && (*cp != (int) nul_char)) {
                                cp++;
                        }
                        if ((length = strlen(cp)) >= MAXPATHLEN) {
                                tmp_wcs_buffer = ALLOC_WC(length + 1);
                                (void) mbstowcs(tmp_wcs_buffer, cp, length + 1);
                                value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
                                retmem(tmp_wcs_buffer);
                        } else {
                                MBSTOWCS(wcs_buffer, cp);
                                value = GETNAME(wcs_buffer, FIND_LENGTH);
                        }
                        argv[i] = NULL;
                } else {
                        /* Illegal MAKEFLAGS argument */
                        continue;
                }
                if(append) {
                        setvar_append(name, value);
                        append = false;
                } else {
                        macro = maybe_append_prop(name, macro_prop);
                        macro->body.macro.exported = true;
                        SETVAR(name, value, false)->body.macro.read_only = true;
                }
        }
}
```

______***unchanged_portion_omitted_***

```
/*
 *      read_environment(read_only)
 *
 *      This routine reads the process environment when make starts and enters
 *      it as make macros. The environment variable SHELL is ignored.
 *
 *      Parameters:
 *              read_only       Should we make env vars read only?
 *
 *      Global variables used:
 *              report_pwd      Set if this make was started by other make
 */
static void
read_environment(Boolean read_only)
{
        register char           **environment;
        int                     length;
        wchar_t                 *tmp_wcs_buffer;
        Boolean                 alloced_tmp_wcs_buffer = false;
        register wchar_t        *name;
        register wchar_t        *value;
        register Name           macro;
        Property                val;
        Boolean                 read_only_saved;

        reading_environment = true;
        environment = environ;
        for (; *environment; environment++) {
                read_only_saved = read_only;
                if ((length = strlen(*environment)) >= MAXPATHLEN) {
                        tmp_wcs_buffer = ALLOC_WC(length + 1);
                        alloced_tmp_wcs_buffer = true;
                        (void) mbstowcs(tmp_wcs_buffer, *environment, length + 1
                        name = tmp_wcs_buffer;
                } else {
                        MBSTOWCS(wcs_buffer, *environment);
                        name = wcs_buffer;
                }
                value = (wchar_t *) wcschr(name, (int) equal_char);
                value = (wchar_t *) wschr(name, (int) equal_char);

                /*
                 * Looks like there's a bug in the system, but sometimes
                 * you can get blank lines in *environment.
                 */
                if (!value) {
                        continue;
                }
                MBSTOWCS(wcs_buffer2, "SHELL=");
                if (IS_WEQUALN(name, wcs_buffer2, wcslen(wcs_buffer2))) {
                if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
                        continue;
                }
                MBSTOWCS(wcs_buffer2, "MAKEFLAGS=");
                if (IS_WEQUALN(name, wcs_buffer2, wcslen(wcs_buffer2))) {
                if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
                        report_pwd = true;
                        /*
                         * In POSIX mode we do not want MAKEFLAGS to be readonly
                         * If the MAKEFLAGS macro is subsequently set by the mak
                         * it replaces the MAKEFLAGS variable currently found in
                         * environment.
                         * See Assertion 50 in section 6.2.5.3 of standard P1003
                         */
                        if(posix) {
                                read_only_saved = false;
                        }
                }

                /*
                 * We ignore SUNPRO_DEPENDENCIES. This environment variable is
                 * set by make and read by cpp which then writes info to
                 * .make.dependency.xxx.  When make is invoked by another make
                 * (recursive make), we don't want to read this because then
                 * the child make will end up writing to the parent
                 * directory's .make.state and clobbering them.
                 */
                MBSTOWCS(wcs_buffer2, "SUNPRO_DEPENDENCIES");
                if (IS_WEQUALN(name, wcs_buffer2, wcslen(wcs_buffer2))) {
                if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
                        continue;
                }

                macro = GETNAME(name, value - name);
                maybe_append_prop(macro, macro_prop)->body.macro.exported =
                  true;
                if ((value == NULL) || ((value + 1)[0] == (int) nul_char)) {
                        val = setvar_daemon(macro,
                                            (Name) NULL,
                                            false, no_daemon, false, debug_level
                } else {
                        val = setvar_daemon(macro,
                                            GETNAME(value + 1, FIND_LENGTH),
                                            false, no_daemon, false, debug_level
                }
                val->body.macro.read_only = read_only_saved;
                if (alloced_tmp_wcs_buffer) {
                        retmem(tmp_wcs_buffer);
                        alloced_tmp_wcs_buffer = false;
                }
        }
        reading_environment = false;
}
```
______***unchanged_portion_omitted_***

```
/*
 *      make_targets(argc, argv, parallel_flag)
 *
 *      Call doname on the specified targets
 *
 *      Parameters:
 *              argc            You know what this is
 *              argv            You know what this is
 *              parallel_flag   True if building in parallel
 *
 *      Global variables used:
 *              build_failed_seen Used to generated message after failed -k
 *              commands_done   Used to generate message "Up to date"
 *              default_target_to_build First proper target in makefile
 *              init            The Name ".INIT", use to run command
 *              parallel        Global parallel building flag
 *              quest           make -q, suppresses messages
 *              recursion_level Initialized, used for tracing
 *              report_dependencies make -P, regroves whole process
 */
static void
make_targets(int argc, char **argv, Boolean parallel_flag)
{
        int                     i;
        char                    *cp;
        Doname                  result;
        register Boolean        target_to_make_found = false;

        (void) doname(init, true, true);
        recursion_level = 1;
        parallel = parallel_flag;
/*
 *      make remaining args
 */
/*
        if ((report_dependencies_level == 0) && parallel) {
 */
        if (parallel) {
                /*
                 * If building targets in parallel, start all of the
                 * remaining args to build in parallel.
                 */
                for (i = 1; i < argc; i++) {
                        if ((cp = argv[i]) != NULL) {
                                commands_done = false;
                                if ((cp[0] == (int) period_char) &&
                                    (cp[1] == (int) slash_char)) {
                                        cp += 2;
                                }
                                 if((cp[0] == (int) ' ') &&
                                    (cp[1] == (int) '-') &&
                                    (cp[2] == (int) ' ') &&
                                    (cp[3] == (int) '-')) {
                                    argv[i] = NULL;
                                        continue;
                                }
                                MBSTOWCS(wcs_buffer, cp);
                                //default_target_to_build = GETNAME(wcs_buffer,
                                //                                  FIND_LENGTH);
                                default_target_to_build = normalize_name(wcs_buf
                                                                  wcslen(wcs_buf
                                                                  wslen(wcs_buff
                                if (default_target_to_build == wait_name) {
                                        if (parallel_process_cnt > 0) {
                                                finish_running();
                                        }
                                        continue;
                                }
                                top_level_target = get_wstring(default_target_to
                                /*
                                 * If we can't execute the current target in
                                 * parallel, hold off the target processing
                                 * to preserve the order of the targets as they
                                 * in command line.
                                 */
                                if (!parallel_ok(default_target_to_build, false)
                                                && parallel_process_cnt > 0) {
                                        finish_running();
                                }
                                result = doname_check(default_target_to_build,
                                                      true,
                                                      false,
                                                      false);
                                gather_recursive_deps();
                                if (/* !commands_done && */
                                    (result == build_ok) &&
                                    !quest &&
                                    (report_dependencies_level == 0) /*  &&
                                    (exists(default_target_to_build) > file_does
                                        if (posix) {
                                                if (!commands_done) {
                                                        (void) printf(gettext("'
                                                                      default_ta
                                                } else {
                                                        if (no_action_was_taken)
                                                                (void) printf(ge
                                                                              de
                                                        }
                                                }
                                        } else {
                                                default_target_to_build->stat.ti
                                                if (!commands_done &&
                                                    (exists(default_target_to_bu
                                                        (void) printf(gettext("'
                                                                      default_ta
                                                }
                                        }
                                }
                        }
                }
                /* Now wait for all of the targets to finish running */
                finish_running();
                //              setjmp(jmpbuffer);

        }
        for (i = 1; i < argc; i++) {
                if ((cp = argv[i]) != NULL) {
                        target_to_make_found = true;
                        if ((cp[0] == (int) period_char) &&
                            (cp[1] == (int) slash_char)) {
                                cp += 2;
                        }
                                 if((cp[0] == (int) ' ') &&
                                    (cp[1] == (int) '-') &&
                                    (cp[2] == (int) ' ') &&
                                    (cp[3] == (int) '-')) {
                                    argv[i] = NULL;
                                        continue;
                                }
                        MBSTOWCS(wcs_buffer, cp);
                        default_target_to_build = normalize_name(wcs_buffer, wcs
```

```
                        default_target_to_build = normalize_name(wcs_buffer, wsl
                        top_level_target = get_wstring(default_target_to_build->
                        report_recursion(default_target_to_build);
                        commands_done = false;
                        if (parallel) {
                                result = (Doname) default_target_to_build->state
                        } else {
                                result = doname_check(default_target_to_build,
                                                      true,
                                                      false,
                                                      false);
                        }
                        gather_recursive_deps();
                        if (build_failed_seen) {
                                build_failed_ever_seen = true;
                                warning(gettext("Target '%s' not remade because
                                        default_target_to_build->string_mb);
                        }
                        build_failed_seen = false;
                        if (report_dependencies_level > 0) {
                                print_dependencies(default_target_to_build,
                                                   get_prop(default_target_to_bu
                                                            line_prop));
                        }
                        default_target_to_build->stat.time =
                          file_no_time;
                        if (default_target_to_build->colon_splits > 0) {
                                default_target_to_build->state =
                                  build_dont_know;
                        }
                        if (!parallel &&
                            /* !commands_done && */
                            (result == build_ok) &&
                            !quest &&
                            (report_dependencies_level == 0) /*  &&
                            (exists(default_target_to_build) > file_doesnt_exist
                                if (posix) {
                                        if (!commands_done) {
                                                (void) printf(gettext("'%s' is u
                                                              default_target_to_
                                        } else {
                                                if (no_action_was_taken) {
                                                        (void) printf(gettext("'
                                                                      default_ta
                                                }
                                        }
                                } else {
                                        if (!commands_done &&
                                            (exists(default_target_to_build) > f
                                                (void) printf(gettext("'%s' is u
                                                              default_target_to_
                                        }
                                }
                        }
                }
        }

/*
 *      If no file arguments have been encountered,
 *      make the first name encountered that doesnt start with a dot
 */
        if (!target_to_make_found) {
                if (default_target_to_build == NULL) {
                        fatal(gettext("No arguments to build"));
                }
                commands_done = false;
                top_level_target = get_wstring(default_target_to_build->string_m
                report_recursion(default_target_to_build);


                if (getenv("SPRO_EXPAND_ERRORS")){
                        (void) printf("::(%s)\n",
                                      default_target_to_build->string_mb);
                }


                result = doname_parallel(default_target_to_build, true, false);
                gather_recursive_deps();
                if (build_failed_seen) {
                        build_failed_ever_seen = true;
                        warning(gettext("Target ‘%s’ not remade because of error
                                default_target_to_build->string_mb);
                }
                build_failed_seen = false;
                if (report_dependencies_level > 0) {
                        print_dependencies(default_target_to_build,
                                           get_prop(default_target_to_build->
                                                    prop,
                                                    line_prop));
                }
                default_target_to_build->stat.time = file_no_time;
                if (default_target_to_build->colon_splits > 0) {
                        default_target_to_build->state = build_dont_know;
                }
                if (/* !commands_done && */
                    (result == build_ok) &&
                    !quest &&
                    (report_dependencies_level == 0) /*  &&
                    (exists(default_target_to_build) > file_doesnt_exist)  */) {
                        if (posix) {
                                if (!commands_done) {
                                        (void) printf(gettext("‘%s’ is updated.\
                                                      default_target_to_build->s
                                } else {
                                        if (no_action_was_taken) {
                                                (void) printf(gettext("‘%s’: no
                                                              default_target_to_
                                        }
                                }
                        } else {
                                if (!commands_done &&
                                    (exists(default_target_to_build) > file_does
                                        (void) printf(gettext("‘%s’ is up to dat
                                                      default_target_to_build->s
                                }
                        }
                }
        }
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   19334 Fri May 22 11:19:43 2015
new/usr/src/cmd/make/bin/misc.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
______unchanged_portion_omitted_

void
dump_target_list(void)
{
        Name_set::iterator      p, e;
        Wstring str;

        for (p = hashtab.begin(), e = hashtab.end(); p != e; p++) {
                        str.init(p);
                        wchar_t * wcb = str.get_string();
                        if ((p->colons != no_colon) &&
                            ((wcb[0] != (int) period_char) ||
                             ((wcb[0] == (int) period_char) &&
                              (wcschr(wcb, (int) slash_char))))) {
                              (wschr(wcb, (int) slash_char))))) {
                                print_target_n_deps(p);
                        }
        }
}
______unchanged_portion_omitted_
```

```
**********************************************************
    8765 Fri May 22 11:19:44 2015
new/usr/src/cmd/make/bin/nse_printdep.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
______unchanged_portion_omitted_
```

```
/*
 *  If target is recursive,  print the following to standard out:
 *      .RECURSIVE subdir targ Makefile
 */
static void
print_rec_info(Name target)
{
        Recursive_make  rp;
        wchar_t         *colon;

        report_recursive_init();

        rp = find_recursive_target(target);

        if (rp) {
                /*
                 * if found,  print starting with the space after the ':'
                 */
                colon = (wchar_t *) wcschr(rp->oldline, (int) colon_char);
                colon = (wchar_t *) wschr(rp->oldline, (int) colon_char);
                (void) printf("%s", colon + 1);
        }
}
______unchanged_portion_omitted_
```

```
**********************************************************
   11055 Fri May 22 11:19:44 2015
new/usr/src/cmd/make/bin/pmake.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
```

```
/*
 * 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 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */


/*
 * Included files
 */
#include <arpa/inet.h>
#include <mk/defs.h>
#include <mksh/misc.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>
#include <rpc/rpc.h>            /* host2netname(), netname2host() */
#include <libintl.h>

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */

/*
 * File table of contents
 */
static int              get_max(wchar_t **ms_address, wchar_t *hostname);
static Boolean          pskip_comment(wchar_t **cp_address);
static void             pskip_till_next_word(wchar_t **cp);
static Boolean          pskip_white_space(wchar_t **cp_address);
```

```
/*
 *      read_make_machines(Name make_machines_name)
 *
 *      For backwards compatibility w/ PMake 1.x, when DMake 2.x is
 *      being run in parallel mode, DMake should parse the PMake startup
 *      file $(HOME)/.make.machines to get the PMake max jobs.
 *
 *      Return value:
 *              int of PMake max jobs
 *
 *      Parameters:
 *              make_machines_name      Name of .make.machines file
 *
 */
int
read_make_machines(Name make_machines_name)
{
        wchar_t                 c;
        Boolean                 default_make_machines;
        struct hostent          *hp;
        wchar_t                 local_host[MAX_HOSTNAMELEN + 1];
        char                    local_host_mb[MAX_HOSTNAMELEN + 1] = "";
        int                     local_host_wslen;
        wchar_t                 full_host[MAXNETNAMELEN + 1];
        int                     full_host_wslen = 0;
        char                    *homedir;
        Name                    MAKE_MACHINES;
        struct stat             make_machines_buf;
        FILE                    *make_machines_file;
        wchar_t                 *make_machines_list = NULL;
        char                    *make_machines_list_mb = NULL;
        wchar_t                 make_machines_path[MAXPATHLEN];
        char                    mb_make_machines_path[MAXPATHLEN];
        wchar_t                 *mp;
        wchar_t                 *ms;
        int                     pmake_max_jobs = 0;
        struct utsname          uts_info;


        MBSTOWCS(wcs_buffer, "MAKE_MACHINES");
        MAKE_MACHINES = GETNAME(wcs_buffer, FIND_LENGTH);
        /* Did the user specify a .make.machines file on the command line? */
        default_make_machines = false;
        if (make_machines_name == NULL) {
                /* Try reading the default .make.machines file, in $(HOME). */
                homedir = getenv("HOME");
                if ((homedir != NULL) && (strlen(homedir) < (sizeof(mb_make_mach
                        sprintf(mb_make_machines_path,
                         "%s/.make.machines", homedir);
                        MBSTOWCS(make_machines_path, mb_make_machines_path);
                        make_machines_name = GETNAME(make_machines_path, FIND_LE
                        default_make_machines = true;
                }
                if (make_machines_name == NULL) {
                        /*
                         * No $(HOME)/.make.machines file.
                         * Return 0 for PMake max jobs.
                         */
                        return(0);
                }
        }
/*
        make_machines_list_mb = getenv(MAKE_MACHINES->string_mb);
 */
        /* Open the .make.machines file. */
        if ((make_machines_file = fopen(make_machines_name->string_mb, "r")) ==
                if (!default_make_machines) {
                        /* Error opening .make.machines file. */
                        fatal(gettext("Open of %s failed: %s"),
                              make_machines_name->string_mb,
                              errmsg(errno));
                } else {
                        /*
                         * No $(HOME)/.make.machines file.
                         * Return 0 for PMake max jobs.
                         */
                        return(0);
                }
        /* Stat the .make.machines file to get the size of the file.  */
        } else if (fstat(fileno(make_machines_file), &make_machines_buf) < 0) {
                /* Error stat'ing .make.machines file. */
                fatal(gettext("Stat of %s failed: %s"),
                      make_machines_name->string_mb,
                      errmsg(errno));
        } else {
                /* Allocate memory for "MAKE_MACHINES=<contents of .m.m>" */
                make_machines_list_mb =
                  (char *) getmem((int) (strlen(MAKE_MACHINES->string_mb) +
                                         2 +
                                         make_machines_buf.st_size));
                sprintf(make_machines_list_mb,
                        "%s=",
                        MAKE_MACHINES->string_mb);
                /* Read in the .make.machines file. */
                if (fread(make_machines_list_mb + strlen(MAKE_MACHINES->string_m
                          sizeof(char),
                          (int) make_machines_buf.st_size,
                          make_machines_file) != make_machines_buf.st_size) {
                        /*
                         * Error reading .make.machines file.
                         * Return 0 for PMake max jobs.
                         */
                        warning(gettext("Unable to read %s"),
                                make_machines_name->string_mb);
                        (void) fclose(make_machines_file);
                        retmem_mb((caddr_t) make_machines_list_mb);
                        return(0);
                } else {
                        (void) fclose(make_machines_file);
                        /* putenv "MAKE_MACHINES=<contents of .m.m>" */
                        *(make_machines_list_mb +
                          strlen(MAKE_MACHINES->string_mb) +
                          1 +
                          make_machines_buf.st_size) = (int) nul_char;
                        if (putenv(make_machines_list_mb) != 0) {
                                warning(gettext("Couldn't put contents of %s in
                                        make_machines_name->string_mb);
                        } else {
                                make_machines_list_mb += strlen(MAKE_MACHINES->s
                                make_machines_list = ALLOC_WC(strlen(make_machin
                                (void) mbstowcs(make_machines_list,
                                                make_machines_list_mb,
                                                (strlen(make_machines_list_mb) +
                        }
                }
        }

        uname(&uts_info);
        strcpy(local_host_mb, &uts_info.nodename[0]);
        MBSTOWCS(local_host, local_host_mb);
        local_host_wslen = wcslen(local_host);
```

```
        local_host_wslen = wslen(local_host);

        // There is no getdomainname() function on Solaris.
        // And netname2host() function does not work on Linux.
        // So we have to use different APIs.
        if (host2netname(mbs_buffer, NULL, NULL) &&
            netname2host(mbs_buffer, mbs_buffer2, MAXNETNAMELEN+1)) {
                MBSTOWCS(full_host, mbs_buffer2);
                full_host_wslen = wcslen(full_host);
                full_host_wslen = wslen(full_host);
        }

        for (ms = make_machines_list;
             (ms) && (*ms );
             ) {
                /*
                 * Skip white space and comments till you reach
                 * a machine name.
                 */
                pskip_till_next_word(&ms);

                /*
                 * If we haven't reached the end of file, process the
                 * machine name.
                 */
                if (*ms) {
                        /*
                         * If invalid machine name decrement counter
                         * and skip line.
                         */
                        mp = ms;
                        SKIPWORD(ms);
                        c = *ms;
                        *ms++ = '\0'; /* Append null to machine name. */
                        /*
                         * If this was the beginning of a comment
                         * (we overwrote a # sign) and it's not
                         * end of line yet, shift the # sign.
                         */
                        if ((c == '#') && (*ms != '\n') && (*ms)) {
                                *ms = '#';
                        }
                        WCSTOMBS(mbs_buffer, mp);
                        /*
                         * Print "Ignoring unknown host" if:
                         * 1) hostname is longer than MAX_HOSTNAMELEN, or
                         * 2) hostname is unknown
                         */
                        if ((wcslen(mp) > MAX_HOSTNAMELEN) ||
                        if ((wslen(mp) > MAX_HOSTNAMELEN) ||
                            ((hp = gethostbyname(mbs_buffer)) == NULL)) {
                                warning(gettext("Ignoring unknown host %s"),
                                        mbs_buffer);
                                SKIPTOEND(ms);
                                /* Increment ptr if not end of file. */
                                if (*ms) {
                                        ms++;
                                }
                        } else {
                                /* Compare current hostname with local_host. */
                                if (wcslen(mp) == local_host_wslen &&
                                if (wslen(mp) == local_host_wslen &&
                                    IS_WEQUALN(mp, local_host, local_host_wslen)
                                        /*
                                         * Bingo, local_host is in .make.machine
                                         * Continue reading.
```

```
                                                 */
                                                pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
                                        /* Compare current hostname with full_host. */
                                        } else if (wcslen(mp) == full_host_wslen &&
                                        } else if (wslen(mp) == full_host_wslen &&
                                                   IS_WEQUALN(mp, full_host, full_host_w
                                                /*
                                                 * Bingo, full_host is in .make.machines
                                                 * Continue reading.
                                                 */
                                                pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
                                        } else {
                                                if (c != '\n') {
                                                    SKIPTOEND(ms);
                                                    if (*ms) {
                                                        ms++;
                                                    }
                                                }
                                                continue;
                                        }
                                        /* If we get here, local_host is in .make.machin
                                        if (c != '\n')  {
                                                /* Now look for keyword 'max'. */
                                                MBSTOWCS(wcs_buffer, "max");
                                                SKIPSPACE(ms);
                                                while ((*ms != '\n') && (*ms)) {
                                                        if (*ms == '#') {
                                                                pskip_comment(&ms);
                                                        } else if (IS_WEQUALN(ms, wcs_bu
                                                                /* Skip "max". */
                                                                ms += 3;
                                                                pmake_max_jobs = get_max
                                                                SKIPSPACE(ms);
                                                        } else {
                                                                warning(gettext("unknown
                                                                SKIPTOEND(ms);
                                                                break;
                                                        }
                                                }
                                        }
                                        break; /* out of outermost for() loop. */
                                }
                        }
                }
        retmem(make_machines_list);
        return(pmake_max_jobs);
}
```

______***unchanged_portion_omitted_***

```
***********************************************************
   56715 Fri May 22 11:19:45 2015
new/usr/src/cmd/make/bin/read.cc
make: use the more modern wchar routines, not widec.h
***********************************************************
/*
 * 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 2006 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      read.c
 *
 *      This file contains the makefile reader.
 */

/*
 * Included files
 */
#include <alloca.h>             /* alloca() */
#include <errno.h>              /* errno */
#include <fcntl.h>              /* fcntl() */
#include <mk/defs.h>
#include <mksh/macro.h>         /* expand_value(), expand_macro() */
#include <mksh/misc.h>          /* getmem() */
#include <mksh/read.h>          /* get_next_block_fn() */
#include <sys/uio.h>            /* read() */
#include <unistd.h>             /* read(), unlink() */
#include <libintl.h>


/*
 * typedefs & structs
 */

/*
 * Static variables
 */

static int line_started_with_space=0; // Used to diagnose spaces instead of tabs

/*
 * File table of contents
 */
static  void            parse_makefile(register Name true_makefile_name, registe
static  Source          push_macro_value(register Source bp, register wchar_t *b
```

```
extern  void            enter_target_groups_and_dependencies(Name_vector target,
extern  Name            normalize_name(register wchar_t *name_string, register i

/*
 *      read_simple_file(makefile_name, chase_path, doname_it,
 *               complain, must_exist, report_file, lock_makefile)
 *
 *      Make the makefile and setup to read it. Actually read it if it is stdio
 *
 *      Return value:
 *                              false if the read failed
 *
 *      Parameters:
 *              makefile_name   Name of the file to read
 *              chase_path      Use the makefile path when opening file
 *              doname_it       Call doname() to build the file first
 *              complain        Print message if doname/open fails
 *              must_exist      Generate fatal if file is missing
 *              report_file     Report file when running -P
 *              lock_makefile   Lock the makefile when reading
 *
 *      Static variables used:
 *
 *      Global variables used:
 *              do_not_exec_rule Is -n on?
 *              file_being_read Set to the name of the new file
 *              line_number     The number of the current makefile line
 *              makefiles_used  A list of all makefiles used, appended to
 */


Boolean
read_simple_file(register Name makefile_name, register Boolean chase_path, regis
{
        static short            max_include_depth;
        register Property       makefile = maybe_append_prop(makefile_name,
                                                             makefile_prop);
        Boolean                 forget_after_parse = false;
        static pathpt           makefile_path;
        register int            n;
        char                    *path;
        register Source         source = ALLOC(Source);
        Property                orig_makefile = makefile;
        Dependency              *dpp;
        Dependency              dp;
        register int            length;
        wchar_t                 *previous_file_being_read = file_being_read;
        int                     previous_line_number = line_number;
        wchar_t                 previous_current_makefile[MAXPATHLEN];
        Makefile_type           save_makefile_type;
        Name                    normalized_makefile_name;
        register wchar_t        *string_start;
        register wchar_t        *string_end;



        wchar_t * wcb = get_wstring(makefile_name->string_mb);

        if (max_include_depth++ >= 40) {
                fatal(gettext("Too many nested include statements"));
        }
        if (makefile->body.makefile.contents != NULL) {
                retmem(makefile->body.makefile.contents);
        }
        source->inp_buf =
          source->inp_buf_ptr =
            source->inp_buf_end = NULL;
        source->error_converting = false;
        makefile->body.makefile.contents = NULL;
        makefile->body.makefile.size = 0;
        if ((makefile_name->hash.length != 1) ||
            (wcb[0] != (int) hyphen_char)) {
                if ((makefile->body.makefile.contents == NULL) &&
                    (doname_it)) {
                        if (makefile_path == NULL) {
                                char *pfx = make_install_prefix();
                                char *path;

                                add_dir_to_path(".",
                                                &makefile_path,
                                                -1);

                                // As regularly installed
                                asprintf(&path, "%s/../share/lib/make", pfx);
                                add_dir_to_path(path, &makefile_path, -1);
                                free(path);

                                // Tools build
                                asprintf(&path, "%s/../../share/", pfx);
                                add_dir_to_path(path, &makefile_path, -1);
                                free(path);

                                add_dir_to_path("/usr/share/lib/make",
                                                &makefile_path,
                                                -1);
                                add_dir_to_path("/etc/default",
                                                &makefile_path,
                                                -1);

                                free(pfx);
                        }
                        save_makefile_type = makefile_type;
                        makefile_type = reading_nothing;
                        if (doname(makefile_name, true, false) == build_dont_kno
                                /* Try normalized filename */
                                string_start=get_wstring(makefile_name->string_m
                                for (string_end=string_start+1; *string_end != L
                                normalized_makefile_name=normalize_name(string_s
                                if ((strcmp(makefile_name->string_mb, normalized
                                        (doname(normalized_makefile_name, true,
                                        n = access_vroot(makefile_name->string_m
                                                 4,
                                                 chase_path ?
                                                 makefile_path : NULL,
                                                 VROOT_DEFAULT);
                                        if (n == 0) {
                                                get_vroot_path((char **) NULL,
                                                       &path,
                                                       (char **) NULL);
                                                if ((path[0] == (int) period_cha
                                                    (path[1] == (int) slash_char
                                                        path += 2;
                                                }
                                                MBSTOWCS(wcs_buffer, path);
                                                makefile_name = GETNAME(wcs_buff
                                                                FIND_LENGTH);
                                        }
                                }
                                retmem(string_start);
                                /*
                                 * Commented out: retmem_mb(normalized_makefile_
                                 * We have to return this memory, but it seems t
```

```
                                 * in dmake or in Sun C++ 5.7 compiler (it works
                                 * is compiled using Sun C++ 5.6).
                                 */
                                // retmem_mb(normalized_makefile_name->string_mb
                        }
                        makefile_type = save_makefile_type;
                }
                source->string.free_after_use = false;
                source->previous = NULL;
                source->already_expanded = false;
                /* Lock the file for read, but not when -n. */
                if (lock_makefile &&
                    !do_not_exec_rule) {

                         make_state_lockfile = getmem(strlen(make_state->string_
                         (void) sprintf(make_state_lockfile,
                                                "%s.lock",
                                                make_state->string_mb);
                        (void) file_lock(make_state->string_mb,
                                         make_state_lockfile,
                                         (int *) &make_state_locked,
                                         0);
                        if(!make_state_locked) {
                                printf("-- NO LOCKING for read\n");
                                retmem_mb(make_state_lockfile);
                                make_state_lockfile = 0;
                                return failed;
                        }
                }
                if (makefile->body.makefile.contents == NULL) {
                        save_makefile_type = makefile_type;
                        makefile_type = reading_nothing;
                        if ((doname_it) &&
                            (doname(makefile_name, true, false) == build_failed)
                                if (complain) {
                                        (void) fprintf(stderr,
                                                       gettext("%s: Couldn't mak
                                                       getprogname(),
                                                       makefile_name->string_mb)
                                }
                                max_include_depth--;
                                makefile_type = save_makefile_type;
                                return failed;
                        }
                        makefile_type = save_makefile_type;
                        //
                        // Before calling exists() make sure that we have the ri
                        //
                        makefile_name->stat.time = file_no_time;

                        if (exists(makefile_name) == file_doesnt_exist) {
                                if (complain ||
                                    (makefile_name->stat.stat_errno != ENOENT))
                                        if (must_exist) {
                                                fatal(gettext("Can't find '%s':
                                                      makefile_name->string_mb,
                                                      errmsg(makefile_name->
                                                             stat.stat_errno));
                                        } else {
                                                warning(gettext("Can't find '%s'
                                                        makefile_name->string_mb
                                                        errmsg(makefile_name->
                                                               stat.stat_errno))
                                        }
                                }
                                max_include_depth--;
                                if(make_state_locked && (make_state_lockfile !=
                                        (void) unlink(make_state_lockfile);
                                        retmem_mb(make_state_lockfile);
                                        make_state_lockfile = NULL;
                                        make_state_locked = false;
                                }
                                retmem(wcb);
                                retmem_mb((char *)source);
                                return failed;
                        }
                        /*
                         * These values are the size and bytes of
                         * the MULTI-BYTE makefile.
                         */
                        orig_makefile->body.makefile.size =
                          makefile->body.makefile.size =
                            source->bytes_left_in_file =
                              makefile_name->stat.size;
                        if (report_file) {
                                for (dpp = &makefiles_used;
                                     *dpp != NULL;
                                     dpp = &(*dpp)->next);
                                dp = ALLOC(Dependency);
                                dp->next = NULL;
                                dp->name = makefile_name;
                                dp->automatic = false;
                                dp->stale = false;
                                dp->built = false;
                                *dpp = dp;
                        }
                        source->fd = open_vroot(makefile_name->string_mb,
                                                O_RDONLY,
                                                0,
                                                NULL,
                                                VROOT_DEFAULT);
                        if (source->fd < 0) {
                                if (complain || (errno != ENOENT)) {
                                        if (must_exist) {
                                                fatal(gettext("Can't open '%s':
                                                      makefile_name->string_mb,
                                                      errmsg(errno));
                                        } else {
                                                warning(gettext("Can't open '%s'
                                                        makefile_name->string_mb
                                                        errmsg(errno));
                                        }
                                }
                                max_include_depth--;
                                return failed;
                        }
                        (void) fcntl(source->fd, F_SETFD, 1);
                        orig_makefile->body.makefile.contents =
                          makefile->body.makefile.contents =
                            source->string.text.p =
                              source->string.buffer.start =
                                ALLOC_WC((int) (makefile_name->stat.size + 2));
                        if (makefile_type == reading_cpp_file) {
                                forget_after_parse = true;
                        }
                        source->string.text.end = source->string.text.p;
                        source->string.buffer.end =
                          source->string.text.p + makefile_name->stat.size;
                } else {
                        /* Do we ever reach here? */
                        source->fd = -1;
                        source->string.text.p =
```

```
                                  source->string.buffer.start =
                                    makefile->body.makefile.contents;
                                source->string.text.end =
                                  source->string.buffer.end =
                                    source->string.text.p + makefile->body.makefile.size
                                source->bytes_left_in_file =
                                  makefile->body.makefile.size;
                        }
                        file_being_read = wcb;
                } else {
                        char            *stdin_text_p;
                        char            *stdin_text_end;
                        char            *stdin_buffer_start;
                        char            *stdin_buffer_end;
                        char            *p_mb;
                        int             num_mb_chars;
                        size_t          num_wc_chars;

                        MBSTOWCS(wcs_buffer, "Standard in");
                        makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
                        /*
                         * Memory to read standard in, then convert it
                         * to wide char strings.
                         */
                        stdin_buffer_start =
                          stdin_text_p = getmem(length = 1024);
                        stdin_buffer_end = stdin_text_p + length;
                        MBSTOWCS(wcs_buffer, "standard input");
                        file_being_read = (wchar_t *) wcsdup(wcs_buffer);
                        file_being_read = (wchar_t *) wsdup(wcs_buffer);
                        line_number = 0;
                        while ((n = read(fileno(stdin),
                                         stdin_text_p,
                                         length)) > 0) {
                                length -= n;
                                stdin_text_p += n;
                                if (length == 0) {
                                        p_mb = getmem(length = 1024 +
                                                      (stdin_buffer_end -
                                                       stdin_buffer_start));
                                        (void) strncpy(p_mb,
                                                       stdin_buffer_start,
                                                       (stdin_buffer_end -
                                                        stdin_buffer_start));
                                        retmem_mb(stdin_buffer_start);
                                        stdin_text_p = p_mb +
                                          (stdin_buffer_end - stdin_buffer_start);
                                        stdin_buffer_start = p_mb;
                                        stdin_buffer_end =
                                          stdin_buffer_start + length;
                                        length = 1024;
                                }
                        }
                        if (n < 0) {
                                fatal(gettext("Error reading standard input: %s"),
                                      errmsg(errno));
                        }
                        stdin_text_p = stdin_buffer_start;
                        stdin_text_end = stdin_buffer_end - length;
                        num_mb_chars = stdin_text_end - stdin_text_p;

                        /*
                         * Now, convert the sequence of multibyte chars into
                         * a sequence of corresponding wide character codes.
                         */
                        source->string.free_after_use = false;
                source->previous = NULL;
                source->bytes_left_in_file = 0;
                source->fd = -1;
                source->already_expanded = false;
                source->string.buffer.start =
                  source->string.text.p = ALLOC_WC(num_mb_chars + 1);
                source->string.buffer.end =
                    source->string.text.p + num_mb_chars;
                num_wc_chars = mbstowcs(source->string.text.p,
                                        stdin_text_p,
                                        num_mb_chars);
                if ((int) num_wc_chars >= 0) {
                        source->string.text.end =
                          source->string.text.p + num_wc_chars;
                }
                (void) retmem_mb(stdin_text_p);
        }
        line_number = 1;
        if (trace_reader) {
                (void) printf(gettext(">>>>>>>>>>>>>>>> Reading makefile %s\n"),
                              makefile_name->string_mb);
        }
        parse_makefile(makefile_name, source);
        if (trace_reader) {
                (void) printf(gettext(">>>>>>>>>>>>>>>> End of makefile %s\n"),
                              makefile_name->string_mb);
        }
        if(file_being_read) {
                retmem(file_being_read);
        }
        file_being_read = previous_file_being_read;
        line_number = previous_line_number;
        makefile_type = reading_nothing;
        max_include_depth--;
        if (make_state_locked) {
                /* Unlock .make.state. */
                unlink(make_state_lockfile);
                make_state_locked = false;
                retmem_mb(make_state_lockfile);
        }
        if (forget_after_parse) {
                retmem(makefile->body.makefile.contents);
                makefile->body.makefile.contents = NULL;
        }
        retmem_mb((char *)source);
        return succeeded;
}

/*
 *      parse_makefile(true_makefile_name, source)
 *
 *      Strings are read from Sources.
 *      When macros are found, their values are represented by a
 *      Source that is pushed on a stack. At end of string
 *      (that is returned from GET_CHAR() as 0), the block is popped.
 *
 *      Parameters:
 *              true_makefile_name      The name of makefile we are parsing
 *              source                  The source block to read from
 *
 *      Global variables used:
 *              do_not_exec_rule Is -n on?
 *              line_number     The number of the current makefile line
 *              makefile_type   What kind of makefile are we reading?
 *              empty_name      The Name ""
 */
```

```
static void
parse_makefile(register Name true_makefile_name, register Source source)
{
/*
        char                    mb_buffer[MB_LEN_MAX];
 */
        register wchar_t        *source_p;
        register wchar_t        *source_end;
        register wchar_t        *string_start;
        wchar_t                 *string_end;
        register Boolean        macro_seen_in_string;
        Boolean                 append;
        String_rec              name_string;
        wchar_t                 name_buffer[STRING_BUFFER_LENGTH];
        register int            distance;
        register int            paren_count;
        int                     brace_count;
        int                     char_number;
        Cmd_line                command;
        Cmd_line                command_tail;
        Name                    macro_value;

        Name_vector_rec         target;
        Name_vector_rec         depes;
        Name_vector_rec         extra_name_vector;
        Name_vector             current_names;
        Name_vector             extra_names = &extra_name_vector;
        Name_vector             nvp;
        Boolean                 target_group_seen;

        register Reader_state   state;
        register Reader_state   on_eoln_state;
        register Separator      separator;

        wchar_t                 buffer[4 * STRING_BUFFER_LENGTH];
        Source                  extrap;

        Boolean                 save_do_not_exec_rule = do_not_exec_rule;
        Name                    makefile_name;

        static Name             sh_name;
        static Name             shell_name;
        int                     i;

        static wchar_t          include_space[10];
        static wchar_t          include_tab[10];
        int                     tmp_bytes_left_in_string;
        Boolean                 tmp_maybe_include = false;
        int                     emptycount = 0;
        Boolean                 first_target;

        String_rec              include_name;
        wchar_t                 include_buffer[STRING_BUFFER_LENGTH];

        target.next = depes.next = NULL;
        /* Move some values from their struct to register declared locals */
        CACHE_SOURCE(0);

 start_new_line:
        /*
         * Read whitespace on old line. Leave pointer on first char on
         * next line.
         */
        first_target = true;
        on_eoln_state = exit_state;
/*
```

```
        for (WCTOMB(mb_buffer, GET_CHAR());
             1;
             source_p++, WCTOMB(mb_buffer, GET_CHAR()))
                switch (mb_buffer[0]) {
 */
        for (char_number=0; 1; source_p++,char_number++) switch (GET_CHAR()) {
        case nul_char:
                /* End of this string. Pop it and return to the previous one */
                GET_NEXT_BLOCK(source);
                source_p--;
                if (source == NULL) {
                        GOTO_STATE(on_eoln_state);
                }
                break;
        case newline_char:
        end_of_line:
                source_p++;
                if (source->fd >= 0) {
                        line_number++;
                }
                switch (GET_CHAR()) {
                case nul_char:
                        GET_NEXT_BLOCK(source);
                        if (source == NULL) {
                                GOTO_STATE(on_eoln_state);
                        }
                        /* Go back to the top of this loop */
                        goto start_new_line;
                case newline_char:
                case numbersign_char:
                case dollar_char:
                case space_char:
                case tab_char:
                        /*
                         * Go back to the top of this loop since the
                         * new line does not start with a regular char.
                         */
                        goto start_new_line;
                default:
                        /* We found the first proper char on the new line */
                        goto start_new_line_no_skip;
                }
        case space_char:
                if (char_number == 0)
                        line_started_with_space=line_number;
        case tab_char:
                /* Whitespace. Just keep going in this loop */
                break;
        case numbersign_char:
                /* Comment. Skip over it */
                for (; 1; source_p++) {
                        switch (GET_CHAR()) {
                        case nul_char:
                                GET_NEXT_BLOCK_NOCHK(source);
                                if (source == NULL) {
                                        GOTO_STATE(on_eoln_state);
                                }
                                if (source->error_converting) {
                                // Illegal byte sequence - skip its first byte
                                        source->inp_buf_ptr++;
                                }
                                source_p--;
                                break;
                        case backslash_char:
                                /* Comments can be continued */
                                if (*++source_p == (int) nul_char) {
```

```
                                        GET_NEXT_BLOCK_NOCHK(source);
                                        if (source == NULL) {
                                                GOTO_STATE(on_eoln_state);
                                        }
                                        if (source->error_converting) {
                                        // Illegal byte sequence - skip its firs
                                                source->inp_buf_ptr++;
                                                source_p--;
                                                break;
                                        }
                                }
                                if(*source_p == (int) newline_char) {
                                        if (source->fd >= 0) {
                                                line_number++;
                                        }
                                }
                                break;
                        case newline_char:
                                /*
                                 * After we skip the comment we go to
                                 * the end of line handler since end of
                                 * line terminates comments.
                                 */
                                goto end_of_line;
                        }
                }
        case dollar_char:
                /* Macro reference */
                if (source->already_expanded) {
                        /*
                         * If we are reading from the expansion of a
                         * macro we already expanded everything enough.
                         */
                        goto start_new_line_no_skip;
                }
                /*
                 * Expand the value and push the Source on the stack of
                 * things being read.
                 */
                source_p++;
                UNCACHE_SOURCE();
                {
                        Source t = (Source) alloca((int) sizeof (Source_rec));
                        source = push_macro_value(t,
                                                  buffer,
                                                  sizeof buffer,
                                                  source);
                }
                CACHE_SOURCE(1);
                break;
        default:
                /* We found the first proper char on the new line */
                goto start_new_line_no_skip;
        }

        /*
         * We found the first normal char (one that starts an identifier)
         * on the newline.
         */
start_new_line_no_skip:
        /* Inspect that first char to see if it maybe is special anyway */
        switch (GET_CHAR()) {
        case nul_char:
                GET_NEXT_BLOCK(source);
                if (source == NULL) {
                        GOTO_STATE(on_eoln_state);
                }
                goto start_new_line_no_skip;
        case newline_char:
                /* Just in case */
                goto start_new_line;
        case exclam_char:
                /* Evaluate the line before it is read */
                string_start = source_p + 1;
                macro_seen_in_string = false;
                /* Stuff the line in a string so we can eval it. */
                for (; 1; source_p++) {
                        switch (GET_CHAR()) {
                        case newline_char:
                                goto eoln_1;
                        case nul_char:
                                if (source->fd > 0) {
                                        if (!macro_seen_in_string) {
                                                macro_seen_in_string = true;
                                                INIT_STRING_FROM_STACK(
                                                      name_string, name_buffer);
                                        }
                                        append_string(string_start,
                                                      &name_string,
                                                      source_p - string_start);
                                        GET_NEXT_BLOCK(source);
                                        string_start = source_p;
                                        source_p--;
                                        break;
                                }
                        eoln_1:
                                if (!macro_seen_in_string) {
                                        INIT_STRING_FROM_STACK(name_string,
                                                               name_buffer);
                                }
                                append_string(string_start,
                                              &name_string,
                                              source_p - string_start);
                                extrap = (Source)
                                  alloca((int) sizeof (Source_rec));
                                extrap->string.buffer.start = NULL;
                                extrap->inp_buf =
                                  extrap->inp_buf_ptr =
                                    extrap->inp_buf_end = NULL;
                                extrap->error_converting = false;
                                if (*source_p == (int) nul_char) {
                                        source_p++;
                                }
                                /* Eval the macro */
                                expand_value(GETNAME(name_string.buffer.start,
                                                     FIND_LENGTH),
                                             &extrap->string,
                                             false);
                                if (name_string.free_after_use) {
                                        retmem(name_string.buffer.start);
                                }
                                UNCACHE_SOURCE();
                                extrap->string.text.p =
                                  extrap->string.buffer.start;
                                extrap->fd = -1;
                                /* And push the value */
                                extrap->previous = source;
                                source = extrap;
                                CACHE_SOURCE(0);
                                goto line_evald;
                        }
                }
```

```
        default:
                goto line_evald;
        }

        /* We now have a line we can start reading */
line_evald:
        if (source == NULL) {
                GOTO_STATE(exit_state);
        }
        /* Check if this is an include command */
        if ((makefile_type == reading_makefile) &&
            !source->already_expanded) {
            if (include_space[0] == (int) nul_char) {
                MBSTOWCS(include_space, "include ");
                MBSTOWCS(include_tab, "include\t");
            }
            if ((IS_WEQUALN(source_p, include_space, 8)) ||
                (IS_WEQUALN(source_p, include_tab, 8))) {
                source_p += 7;
                if (iswspace(*source_p)) {
                        Makefile_type save_makefile_type;
                        wchar_t         *name_start;
                        int             name_length;

                        /*
                         * Yes, this is an include.
                         * Skip spaces to get to the filename.
                         */
                        while (iswspace(*source_p) ||
                               (*source_p == (int) nul_char)) {
                                switch (GET_CHAR()) {
                                case nul_char:
                                        GET_NEXT_BLOCK(source);
                                        if (source == NULL) {
                                                GOTO_STATE(on_eoln_state);
                                        }
                                        break;

                                default:
                                        source_p++;
                                        break;
                                }
                        }

                        string_start = source_p;
                        /* Find the end of the filename */
                        macro_seen_in_string = false;
                        while (!iswspace(*source_p) ||
                               (*source_p == (int) nul_char)) {
                                switch (GET_CHAR()) {
                                case nul_char:
                                        if (!macro_seen_in_string) {
                                                INIT_STRING_FROM_STACK(name_stri
                                                                       name_buff
                                        }
                                        append_string(string_start,
                                                      &name_string,
                                                      source_p - string_start);
                                        macro_seen_in_string = true;
                                        GET_NEXT_BLOCK(source);
                                        string_start = source_p;
                                        if (source == NULL) {
                                                GOTO_STATE(on_eoln_state);
                                        }
                                        break;
                                default:
                                        source_p++;
                                        break;
                                }
                        }

                        source->string.text.p = source_p;
                        if (macro_seen_in_string) {
                                append_string(string_start,
                                              &name_string,
                                              source_p - string_start);
                                name_start = name_string.buffer.start;
                                name_length = name_string.text.p - name_start;
                        } else {
                                name_start = string_start;
                                name_length = source_p - string_start;
                        }

                        /* Strip "./" from the head of the name */
                        if ((name_start[0] == (int) period_char) &&
                           (name_start[1] == (int) slash_char)) {
                                name_start += 2;
                                name_length -= 2;
                        }
                        /* if include file name is surrounded by double quotes *
                        if ((name_start[0] == (int) doublequote_char) &&
                            (name_start[name_length - 1] == (int) doublequote_ch
                                name_start += 1;
                                name_length -= 2;

                                /* if name does not begin with a slash char */
                                if (name_start[0] != (int) slash_char) {
                                        if ((name_start[0] == (int) period_char)
                                            (name_start[1] == (int) slash_char))
                                                name_start += 2;
                                                name_length -= 2;
                                        }

                                        INIT_STRING_FROM_STACK(include_name, inc
                                        APPEND_NAME(true_makefile_name,
                                                      &include_name,
                                                      true_makefile_name->hash.l

                                        wchar_t *slash = wcsrchr(include_name.bu
                                        wchar_t *slash = wsrchr(include_name.buf
                                        if (slash != NULL) {
                                                include_name.text.p = slash + 1;
                                                append_string(name_start,
                                                              &include_name,
                                                              name_length);

                                                name_start = include_name.buffer
                                                name_length = include_name.text.
                                        }
                                }
                        }

                        /* Even when we run -n we want to create makefiles */
                        do_not_exec_rule = false;
                        makefile_name = GETNAME(name_start, name_length);
                        if (makefile_name->dollar) {
                                String_rec      destination;
                                wchar_t         buffer[STRING_BUFFER_LENGTH];
                                wchar_t         *p;
                                wchar_t         *q;
```

```
                                INIT_STRING_FROM_STACK(destination, buffer);
                                expand_value(makefile_name,
                                             &destination,
                                             false);
                                for (p = destination.buffer.start;
                                     (*p != (int) nul_char) && iswspace(*p);
                                     p++);
                                for (q = p;
                                     (*q != (int) nul_char) && !iswspace(*q);
                                     q++);
                                makefile_name = GETNAME(p, q-p);
                                if (destination.free_after_use) {
                                        retmem(destination.buffer.start);
                                }
                        }
                        source_p++;
                        UNCACHE_SOURCE();
                        /* Read the file */
                        save_makefile_type = makefile_type;
                        if (read_simple_file(makefile_name,
                                             true,
                                             true,
                                             true,
                                             false,
                                             true,
                                             false) == failed) {
                                fatal_reader(gettext("Read of include file `%s'
                                             makefile_name->string_mb);
                        }
                        makefile_type = save_makefile_type;
                        do_not_exec_rule = save_do_not_exec_rule;
                        CACHE_SOURCE(0);
                        goto start_new_line;
                } else {
                        source_p -= 7;
                }
            } else {
                /* Check if the word include was split across 8K boundary. */

                tmp_bytes_left_in_string = source->string.text.end - source_p;
                if (tmp_bytes_left_in_string < 8) {
                        tmp_maybe_include = false;
                        if (IS_WEQUALN(source_p,
                                       include_space,
                                       tmp_bytes_left_in_string)) {
                                tmp_maybe_include = true;
                        }
                        if (tmp_maybe_include) {
                                GET_NEXT_BLOCK(source);
                                tmp_maybe_include = false;
                                goto line_evald;
                        }
                }
            }
        }

        /* Reset the status in preparation for the new line */
        for (nvp = &target; nvp != NULL; nvp = nvp->next) {
                nvp->used = 0;
        }
        for (nvp = &depes; nvp != NULL; nvp = nvp->next) {
                nvp->used = 0;
        }
        target_group_seen = false;
        command = command_tail = NULL;
        macro_value = NULL;
        append = false;
        current_names = &target;
        SET_STATE(scan_name_state);
        on_eoln_state = illegal_eoln_state;
        separator = none_seen;

        /* The state machine starts here */
 enter_state:
        while (1) switch (state) {

/****************************************************************
 *      Scan name state
 */
case scan_name_state:
        /* Scan an identifier. We skip over chars until we find a break char */
        /* First skip white space. */
        for (; 1; source_p++) switch (GET_CHAR()) {
        case nul_char:
                GET_NEXT_BLOCK(source);
                source_p--;
                if (source == NULL) {
                        GOTO_STATE(on_eoln_state);
                }
                break;
        case newline_char:
                /* We found the end of the line. */
                /* Do postprocessing or return error */
                source_p++;
                if (source->fd >= 0) {
                        line_number++;
                }
                GOTO_STATE(on_eoln_state);
        case backslash_char:
                /* Continuation */
                if (*++source_p == (int) nul_char) {
                        GET_NEXT_BLOCK(source);
                        if (source == NULL) {
                                GOTO_STATE(on_eoln_state);
                        }
                }
                if (*source_p == (int) newline_char) {
                        if (source->fd >= 0) {
                                line_number++;
                        }
                } else {
                        source_p--;
                }
                break;
        case tab_char:
        case space_char:
                /* Whitespace is skipped */
                break;
        case numbersign_char:
                /* Comment. Skip over it */
                for (; 1; source_p++) {
                        switch (GET_CHAR()) {
                        case nul_char:
                                GET_NEXT_BLOCK_NOCHK(source);
                                if (source == NULL) {
                                        GOTO_STATE(on_eoln_state);
                                }
                                if (source->error_converting) {
                                // Illegal byte sequence - skip its first byte
                                        source->inp_buf_ptr++;
                                }
                                source_p--;
```

```
                                break;
                        case backslash_char:
                                if (*++source_p == (int) nul_char) {
                                        GET_NEXT_BLOCK_NOCHK(source);
                                        if (source == NULL) {
                                                GOTO_STATE(on_eoln_state);
                                        }
                                        if (source->error_converting) {
                                        // Illegal byte sequence - skip its firs
                                                source->inp_buf_ptr++;
                                                source_p--;
                                                break;
                                        }
                                }
                                if(*source_p == (int) newline_char) {
                                        if (source->fd >= 0) {
                                                line_number++;
                                        }
                                }
                                break;
                        case newline_char:
                                source_p++;
                                if (source->fd >= 0) {
                                        line_number++;
                                }
                                GOTO_STATE(on_eoln_state);
                        }
                }
        case dollar_char:
                /* Macro reference. Expand and push value */
                if (source->already_expanded) {
                        goto scan_name;
                }
                source_p++;
                UNCACHE_SOURCE();
                {
                        Source t = (Source) alloca((int) sizeof (Source_rec));
                        source = push_macro_value(t,
                                                 buffer,
                                                 sizeof buffer,
                                                 source);
                }
                CACHE_SOURCE(1);
                break;
        default:
                /* End of white space */
                goto scan_name;
        }

        /* First proper identifier character */
scan_name:

        string_start = source_p;
        paren_count = brace_count = 0;
        macro_seen_in_string = false;
        resume_name_scan:
        for (; 1; source_p++) {
                switch (GET_CHAR()) {
                case nul_char:
                        /* Save what we have seen so far of the identifier */
                        if (source_p != string_start) {
                                if (!macro_seen_in_string) {
                                        INIT_STRING_FROM_STACK(name_string,
                                                              name_buffer);
                                }
                                append_string(string_start,
                                              &name_string,
                                              source_p - string_start);
                                macro_seen_in_string = true;
                        }
                        /* Get more text to read */
                        GET_NEXT_BLOCK(source);
                        string_start = source_p;
                        source_p--;
                        if (source == NULL) {
                                GOTO_STATE(on_eoln_state);
                        }
                        break;
                case newline_char:
                        if (paren_count > 0) {
                                fatal_reader(gettext("Unmatched ‘(’ on line"));
                        }
                        if (brace_count > 0) {
                                fatal_reader(gettext("Unmatched ‘{’ on line"));
                        }
                        source_p++;
                        /* Enter name */
                        current_names = enter_name(&name_string,
                                                   macro_seen_in_string,
                                                   string_start,
                                                   source_p - 1,
                                                   current_names,
                                                   &extra_names,
                                                   &target_group_seen);
                        first_target = false;
                        if (extra_names == NULL) {
                                extra_names = (Name_vector)
                                  alloca((int) sizeof (Name_vector_rec));
                        }
                        /* Do postprocessing or return error */
                        if (source->fd >= 0) {
                                line_number++;
                        }
                        GOTO_STATE(on_eoln_state);
                case backslash_char:
                        /* Check if this is a quoting backslash */
                        if (!macro_seen_in_string) {
                                INIT_STRING_FROM_STACK(name_string,
                                                       name_buffer);
                                macro_seen_in_string = true;
                        }
                        append_string(string_start,
                                      &name_string,
                                      source_p - string_start);
                        if (*++source_p == (int) nul_char) {
                                GET_NEXT_BLOCK(source);
                                if (source == NULL) {
                                        GOTO_STATE(on_eoln_state);
                                }
                        }
                        if (*source_p == (int) newline_char) {
                                if (source->fd >= 0) {
                                        line_number++;
                                }
                                *source_p = (int) space_char;
                                string_start = source_p;
                                goto resume_name_scan;
                        } else {
                                string_start = source_p;
                                break;
                        }
                        break;
```

```
                case numbersign_char:
                        if (paren_count + brace_count > 0) {
                                break;
                        }
                        fatal_reader(gettext("Unexpected comment seen"));
                case dollar_char:
                        if (source->already_expanded) {
                                break;
                        }
                        /* Save the identifier so far */
                        if (source_p != string_start) {
                                if (!macro_seen_in_string) {
                                        INIT_STRING_FROM_STACK(name_string,
                                                               name_buffer);
                                }
                                append_string(string_start,
                                              &name_string,
                                              source_p - string_start);
                                macro_seen_in_string = true;
                        }
                        /* Eval and push the macro */
                        source_p++;
                        UNCACHE_SOURCE();
                        {
                                Source t =
                                  (Source) alloca((int) sizeof (Source_rec));
                                source = push_macro_value(t,
                                                          buffer,
                                                          sizeof buffer,
                                                          source);
                        }
                        CACHE_SOURCE(1);
                        string_start = source_p + 1;
                        break;
                case parenleft_char:
                        paren_count++;
                        break;
                case parenright_char:
                        if (--paren_count < 0) {
                                fatal_reader(gettext("Unmatched ‘)’ on line"));
                        }
                        break;
                case braceleft_char:
                        brace_count++;
                        break;
                case braceright_char:
                        if (--brace_count < 0) {
                                fatal_reader(gettext("Unmatched ‘}’ on line"));
                        }
                        break;
                case ampersand_char:
                case greater_char:
                case bar_char:
                        if (paren_count + brace_count == 0) {
                                source_p++;
                        }
                        /* Fall into */
                case tab_char:
                case space_char:
                        if (paren_count + brace_count > 0) {
                                break;
                        }
                        current_names = enter_name(&name_string,
                                                   macro_seen_in_string,
                                                   string_start,
                                                   source_p,
                                                   current_names,
                                                   &extra_names,
                                                   &target_group_seen);
                        first_target = false;
                        if (extra_names == NULL) {
                                extra_names = (Name_vector)
                                  alloca((int) sizeof (Name_vector_rec));
                        }
                        goto enter_state;
                case colon_char:
                        if (paren_count + brace_count > 0) {
                                break;
                        }
                        if (separator == conditional_seen) {
                                break;
                        }
/** POSIX **/
#if 0
                        if(posix) {
                          emptycount = 0;
                        }
#endif
/** END POSIX **/
                        /* End of the target list. We now start reading */
                        /* dependencies or a conditional assignment */
                        if (separator != none_seen) {
                                fatal_reader(gettext("Extra ‘:’, ‘::’, or ‘:=’ o
                        }
                        /* Enter the last target */
                        if ((string_start != source_p) ||
                            macro_seen_in_string) {
                                current_names =
                                  enter_name(&name_string,
                                             macro_seen_in_string,
                                             string_start,
                                             source_p,
                                             current_names,
                                             &extra_names,
                                             &target_group_seen);
                                first_target = false;
                                if (extra_names == NULL) {
                                        extra_names = (Name_vector)
                                          alloca((int)
                                                 sizeof (Name_vector_rec));
                                }
                        }
                        /* Check if it is ":" "::" or ":=" */
                scan_colon_label:
                        switch (*++source_p) {
                        case nul_char:
                                GET_NEXT_BLOCK(source);
                                source_p--;
                                if (source == NULL) {
                                        GOTO_STATE(enter_dependencies_state);
                                }
                                goto scan_colon_label;
                        case equal_char:
                                if(svr4) {
                                  fatal_reader(gettext("syntax error"));
                                }
                                separator = conditional_seen;
                                source_p++;
                                current_names = &depes;
                                GOTO_STATE(scan_name_state);
                        case colon_char:
                                separator = two_colon;
```

```
                                source_p++;
                                break;
                        default:
                                separator = one_colon;
                        }
                        current_names = &depes;
                        on_eoln_state = enter_dependencies_state;
                        GOTO_STATE(scan_name_state);
                case semicolon_char:
                        if (paren_count + brace_count > 0) {
                                break;
                        }
                        /* End of reading names. Start reading the rule */
                        if ((separator != one_colon) &&
                            (separator != two_colon)) {
                                fatal_reader(gettext("Unexpected command seen"))
                        }
                        /* Enter the last dependency */
                        if ((string_start != source_p) ||
                            macro_seen_in_string) {
                                current_names =
                                  enter_name(&name_string,
                                             macro_seen_in_string,
                                             string_start,
                                             source_p,
                                             current_names,
                                             &extra_names,
                                             &target_group_seen);
                                first_target = false;
                                if (extra_names == NULL) {
                                        extra_names = (Name_vector)
                                          alloca((int)
                                                 sizeof (Name_vector_rec));
                                }
                        }
                        source_p++;
                        /* Make sure to enter a rule even if the is */
                        /* no text here */
                        command = command_tail = ALLOC(Cmd_line);
                        command->next = NULL;
                        command->command_line = empty_name;
                        command->make_refd = false;
                        command->ignore_command_dependency = false;
                        command->assign = false;
                        command->ignore_error = false;
                        command->silent = false;

                        GOTO_STATE(scan_command_state);
                case plus_char:
                        /*
                        ** following code drops the target separator plus char i
                        ** a line.
                        */
                        if(first_target && !macro_seen_in_string &&
                                        source_p == string_start) {
                                for (; 1; source_p++)
                                switch (GET_CHAR()) {
                                case nul_char:
                                        if (source_p != string_start) {
                                                if (!macro_seen_in_string) {
                                                        INIT_STRING_FROM_STACK(n
                                                                               n
                                                }
                                                append_string(string_start,
                                                              &name_string,
                                                              source_p - string_
                                                macro_seen_in_string = true;
                                        }
                                        GET_NEXT_BLOCK(source);
                                        string_start = source_p;
                                        source_p--;
                                        if (source == NULL) {
                                                GOTO_STATE(on_eoln_state);
                                        }
                                        break;
                                case plus_char:
                                        source_p++;
                                        while (*source_p == (int) nul_char) {
                                                if (source_p != string_start) {
                                                        if (!macro_seen_in_strin
                                                                INIT_STRING_FROM
                                                                               n
                                                        }
                                                        append_string(string_sta
                                                                      &name_stri
                                                                      source_p -
                                                        macro_seen_in_string = t
                                                }
                                                GET_NEXT_BLOCK(source);
                                                string_start = source_p;
                                                if (source == NULL) {
                                                        GOTO_STATE(on_eoln_state
                                                }
                                        }
                                        if (*source_p == (int) tab_char ||
                                                        *source_p == (int) space
                                                macro_seen_in_string = false;
                                                string_start = source_p + 1;
                                        } else {
                                                goto resume_name_scan;
                                        }
                                        break;
                                case tab_char:
                                case space_char:
                                        string_start = source_p + 1;
                                        break;
                                default:
                                        goto resume_name_scan;
                                }
                        }
                        if (paren_count + brace_count > 0) {
                                break;
                        }
                        /* We found "+=" construct */
                        if (source_p != string_start) {
                                /* "+" is not a break char. */
                                /* Ignore it if it is part of an identifier */
                                source_p++;
                                goto resume_name_scan;
                        }
                        /* Make sure the "+" is followed by a "=" */
                scan_append:
                        switch (*++source_p) {
                        case nul_char:
                                if (!macro_seen_in_string) {
                                        INIT_STRING_FROM_STACK(name_string,
                                                               name_buffer);
                                }
                                append_string(string_start,
                                              &name_string,
                                              source_p - string_start);
                                GET_NEXT_BLOCK(source);
```

```
                                source_p--;
                                string_start = source_p;
                                if (source == NULL) {
                                        GOTO_STATE(illegal_eoln_state);
                                }
                                goto scan_append;
                        case equal_char:
                                if(!svr4) {
                                  append = true;
                                } else {
                                  fatal_reader(gettext("Must be a separator on r
                                }
                                break;
                        default:
                                /* The "+" just starts a regular name. */
                                /* Start reading that name */
                                goto resume_name_scan;
                        }
                        /* Fall into */
                case equal_char:
                        if (paren_count + brace_count > 0) {
                                break;
                        }
                        /* We found macro assignment. */
                        /* Check if it is legal and if it is appending */
                        switch (separator) {
                        case none_seen:
                                separator = equal_seen;
                                on_eoln_state = enter_equal_state;
                                break;
                        case conditional_seen:
                                on_eoln_state = enter_conditional_state;
                                break;
                        default:
                                /* Reader must special check for "MACRO:sh=" */
                                /* notation */
                                if (sh_name == NULL) {
                                        MBSTOWCS(wcs_buffer, "sh");
                                        sh_name = GETNAME(wcs_buffer, FIND_LENGT
                                        MBSTOWCS(wcs_buffer, "shell");
                                        shell_name = GETNAME(wcs_buffer, FIND_LE
                                }

                                if (!macro_seen_in_string) {
                                        INIT_STRING_FROM_STACK(name_string,
                                                       name_buffer);
                                }
                                append_string(string_start,
                                              &name_string,
                                              source_p - string_start
                                );

                                if ( (((target.used == 1) &&
                                     (depes.used == 1) &&
                                     (depes.names[0] == sh_name)) ||
                                    ((target.used == 1) &&
                                     (depes.used == 0) &&
                                     (separator == one_colon) &&
                                     (GETNAME(name_string.buffer.start,FIND_LENG
                                    (!svr4)) {
                                        String_rec      macro_name;
                                        wchar_t         buffer[100];

                                        INIT_STRING_FROM_STACK(macro_name,
                                                               buffer);
                                        APPEND_NAME(target.names[0],
                                                      &macro_name,
                                                      FIND_LENGTH);
                                        append_char((int) colon_char,
                                                    &macro_name);
                                        APPEND_NAME(sh_name,
                                                      &macro_name,
                                                      FIND_LENGTH);
                                        target.names[0] =
                                          GETNAME(macro_name.buffer.start,
                                                  FIND_LENGTH);
                                        separator = equal_seen;
                                        on_eoln_state = enter_equal_state;
                                        break;
                                } else if ( (((target.used == 1) &&
                                            (depes.used == 1) &&
                                            (depes.names[0] == shell_name)) ||
                                           ((target.used == 1) &&
                                            (depes.used == 0) &&
                                            (separator == one_colon) &&
                                            (GETNAME(name_string.buffer.start,FI
                                           (!svr4)) {
                                        String_rec      macro_name;
                                        wchar_t         buffer[100];

                                        INIT_STRING_FROM_STACK(macro_name,
                                                               buffer);
                                        APPEND_NAME(target.names[0],
                                                      &macro_name,
                                                      FIND_LENGTH);
                                        append_char((int) colon_char,
                                                    &macro_name);
                                        APPEND_NAME(shell_name,
                                                      &macro_name,
                                                      FIND_LENGTH);
                                        target.names[0] =
                                          GETNAME(macro_name.buffer.start,
                                                  FIND_LENGTH);
                                        separator = equal_seen;
                                        on_eoln_state = enter_equal_state;
                                        break;
                                }
                                if(svr4) {
                                  fatal_reader(gettext("syntax error"));
                                }
                                else {
                                  fatal_reader(gettext("Macro assignment on depe
                                }
                        }
                        if (append) {
                                source_p--;
                        }
                        /* Enter the macro name */
                        if ((string_start != source_p) ||
                            macro_seen_in_string) {
                                current_names =
                                  enter_name(&name_string,
                                             macro_seen_in_string,
                                             string_start,
                                             source_p,
                                             current_names,
                                             &extra_names,
                                             &target_group_seen);
                                first_target = false;
                                if (extra_names == NULL) {
                                        extra_names = (Name_vector)
                                          alloca((int)
```

```
                                                 sizeof (Name_vector_rec));
                                }
                        }
                        if (append) {
                                source_p++;
                        }
                        macro_value = NULL;
                        source_p++;
                        distance = 0;
                        /* Skip whitespace to the start of the value */
                        macro_seen_in_string = false;
                        for (; 1; source_p++) {
                                switch (GET_CHAR()) {
                                case nul_char:
                                        GET_NEXT_BLOCK(source);
                                        source_p--;
                                        if (source == NULL) {
                                                GOTO_STATE(on_eoln_state);
                                        }
                                        break;
                                case backslash_char:
                                        if (*++source_p == (int) nul_char) {
                                                GET_NEXT_BLOCK(source);
                                                if (source == NULL) {
                                                        GOTO_STATE(on_eoln_state
                                                }
                                        }
                                        if (*source_p != (int) newline_char) {
                                                if (!macro_seen_in_string) {
                                                        macro_seen_in_string =
                                                          true;
                                                        INIT_STRING_FROM_STACK(n
                                                                               n
                                                }
                                                append_char((int)
                                                            backslash_char,
                                                            &name_string);
                                                append_char(*source_p,
                                                            &name_string);
                                                string_start = source_p+1;
                                                goto macro_value_start;
                                        } else {
                                            if (source->fd >= 0) {
                                                line_number++;
                                            }
                                        }
                                        break;
                                case newline_char:
                                case numbersign_char:
                                        string_start = source_p;
                                        goto macro_value_end;
                                case tab_char:
                                case space_char:
                                        break;
                                default:
                                        string_start = source_p;
                                        goto macro_value_start;
                                }
                        }
                macro_value_start:
                        /* Find the end of the value */
                        for (; 1; source_p++) {
                                if (distance != 0) {
                                        *source_p = *(source_p + distance);
                                }
                                switch (GET_CHAR()) {
```

```
                                case nul_char:
                                        if (!macro_seen_in_string) {
                                                macro_seen_in_string = true;
                                                INIT_STRING_FROM_STACK(name_stri
                                                                       name_buff
                                        }
                                        append_string(string_start,
                                                      &name_string,
                                                      source_p - string_start);
                                        GET_NEXT_BLOCK(source);
                                        string_start = source_p;
                                        source_p--;
                                        if (source == NULL) {
                                                GOTO_STATE(on_eoln_state);
                                        }
                                        break;
                                case backslash_char:
                                        source_p++;
                                        if (distance != 0) {
                                                *source_p =
                                                  *(source_p + distance);
                                        }
                                        if (*source_p == (int) nul_char) {
                                                if (!macro_seen_in_string) {
                                                        macro_seen_in_string =
                                                          true;
                                                        INIT_STRING_FROM_STACK(n
                                                                               n
                                                }

/*  BID_1225561 */
                                                *(source_p - 1) = (int) space_ch
                                                append_string(string_start,
                                                              &name_string,
                                                              source_p -
                                                              string_start - 1);
                                                GET_NEXT_BLOCK(source);
                                                string_start = source_p;
                                                if (source == NULL) {
                                                        GOTO_STATE(on_eoln_state
                                                }
                                                if (distance != 0) {
                                                        *source_p =
                                                          *(source_p +
                                                            distance);
                                                }
                                                if (*source_p == (int) newline_c
                                                        append_char((int) space_
                                                } else {
                                                        append_char((int) backsl
                                                }
/****************/
                                        }
                                        if (*source_p == (int) newline_char) {
                                                source_p--;
                                                line_number++;
                                                distance++;
                                                *source_p = (int) space_char;
                                                while ((*(source_p +
                                                          distance + 1) ==
                                                        (int) tab_char) ||
                                                       (*(source_p +
                                                          distance + 1) ==
                                                        (int) space_char)) {
                                                        distance++;
                                                }
```

```
                                        }
                                        break;
                                case newline_char:
                                case numbersign_char:
                                        goto macro_value_end;
                                }
                        }
                macro_value_end:
                        /* Complete the value in the string */
                        if (!macro_seen_in_string) {
                                macro_seen_in_string = true;
                                INIT_STRING_FROM_STACK(name_string,
                                                       name_buffer);
                        }
                        append_string(string_start,
                                      &name_string,
                                      source_p - string_start);
                        if (name_string.buffer.start != name_string.text.p) {
                                        macro_value =
                                          GETNAME(name_string.buffer.start,
                                                  FIND_LENGTH);
                                }
                        if (name_string.free_after_use) {
                                retmem(name_string.buffer.start);
                        }
                        for (; distance > 0; distance--) {
                                *source_p++ = (int) space_char;
                        }
                        GOTO_STATE(on_eoln_state);
                }
        }

/****************************************************************
 *      enter dependencies state
 */
 case enter_dependencies_state:
 enter_dependencies_label:
/* Expects pointer on first non whitespace char after last dependency. (On */
/* next line.) We end up here after having read a "targets : dependencies" */
/* line. The state checks if there is a rule to read and if so dispatches */
/* to scan_command_state scan_command_state reads one rule line and the */
/* returns here */

        /* First check if the first char on the next line is special */
        switch (GET_CHAR()) {
        case nul_char:
                GET_NEXT_BLOCK(source);
                if (source == NULL) {
                        break;
                }
                goto enter_dependencies_label;
        case exclam_char:
                /* The line should be evaluate before it is read */
                macro_seen_in_string = false;
                string_start = source_p + 1;
                for (; 1; source_p++) {
                        switch (GET_CHAR()) {
                        case newline_char:
                                goto eoln_2;
                        case nul_char:
                                if (source->fd > 0) {
                                        if (!macro_seen_in_string) {
                                                macro_seen_in_string = true;
                                                INIT_STRING_FROM_STACK(name_stri
                                                                       name_buff
                                        }
                                        append_string(string_start,
                                                      &name_string,
                                                      source_p - string_start);
                                        GET_NEXT_BLOCK(source);
                                        string_start = source_p;
                                        source_p--;
                                        break;
                                }
                        eoln_2:
                                if (!macro_seen_in_string) {
                                        INIT_STRING_FROM_STACK(name_string,
                                                               name_buffer);
                                }
                                append_string(string_start,
                                              &name_string,
                                              source_p - string_start);
                                extrap = (Source)
                                  alloca((int) sizeof (Source_rec));
                                extrap->string.buffer.start = NULL;
                                extrap->inp_buf =
                                  extrap->inp_buf_ptr =
                                    extrap->inp_buf_end = NULL;
                                extrap->error_converting = false;
                                expand_value(GETNAME(name_string.buffer.start,
                                                     FIND_LENGTH),
                                             &extrap->string,
                                             false);
                                if (name_string.free_after_use) {
                                        retmem(name_string.buffer.start);
                                }
                                UNCACHE_SOURCE();
                                extrap->string.text.p =
                                  extrap->string.buffer.start;
                                extrap->fd = -1;
                                extrap->previous = source;
                                source = extrap;
                                CACHE_SOURCE(0);
                                goto enter_dependencies_label;
                        }
                }
        case dollar_char:
                if (source->already_expanded) {
                        break;
                }
                source_p++;
                UNCACHE_SOURCE();
                {
                        Source t = (Source) alloca((int) sizeof (Source_rec));
                        source = push_macro_value(t,
                                                  buffer,
                                                  sizeof buffer,
                                                  source);
                }
                CACHE_SOURCE(0);
                goto enter_dependencies_label;
        case numbersign_char:
                if (makefile_type != reading_makefile) {
                        source_p++;
                        GOTO_STATE(scan_command_state);
                }
                for (; 1; source_p++) {
                        switch (GET_CHAR()) {
                        case nul_char:
                                GET_NEXT_BLOCK_NOCHK(source);
                                if (source == NULL) {
                                        GOTO_STATE(on_eoln_state);
```

```
                                }
                                if (source->error_converting) {
                                // Illegal byte sequence - skip its first byte
                                        source->inp_buf_ptr++;
                                }
                                source_p--;
                                break;
                        case backslash_char:
                                if (*++source_p == (int) nul_char) {
                                        GET_NEXT_BLOCK_NOCHK(source);
                                        if (source == NULL) {
                                                GOTO_STATE(on_eoln_state);
                                        }
                                        if (source->error_converting) {
                                        // Illegal byte sequence - skip its firs
                                                source->inp_buf_ptr++;
                                                source_p--;
                                                break;
                                        }
                                }
                                if(*source_p == (int) newline_char) {
                                        if (source->fd >= 0) {
                                                line_number++;
                                        }
                                }
                                break;
                        case newline_char:
                                source_p++;
                                if (source->fd >= 0) {
                                        line_number++;
                                }
                                goto enter_dependencies_label;
                        }
                }

        case tab_char:
                GOTO_STATE(scan_command_state);
        }

        /* We read all the command lines for the target/dependency line. */
        /* Enter the stuff */
        enter_target_groups_and_dependencies( &target, &depes, command,
                                             separator, target_group_seen);

        goto start_new_line;

/****************************************************************
 *      scan command state
 */
case scan_command_state:
        /* We need to read one rule line. Do that and return to */
        /* the enter dependencies state */
        string_start = source_p;
        macro_seen_in_string = false;
        for (; 1; source_p++) {
                switch (GET_CHAR()) {
                case backslash_char:
                        if (!macro_seen_in_string) {
                                INIT_STRING_FROM_STACK(name_string,
                                                       name_buffer);
                        }
                        append_string(string_start,
                                      &name_string,
                                      source_p - string_start);
                        macro_seen_in_string = true;
                        if (*++source_p == (int) nul_char) {
                                GET_NEXT_BLOCK(source);
                                if (source == NULL) {
                                        string_start = source_p;
                                        goto command_newline;
                                }
                        }
                        append_char((int) backslash_char, &name_string);
                        append_char(*source_p, &name_string);
                        if (*source_p == (int) newline_char) {
                                if (source->fd >= 0) {
                                        line_number++;
                                }
                                if (*++source_p == (int) nul_char) {
                                        GET_NEXT_BLOCK(source);
                                        if (source == NULL) {
                                                string_start = source_p;
                                                goto command_newline;
                                        }
                                }
                                if (*source_p == (int) tab_char) {
                                        source_p++;
                                }
                        } else {
                                if (*++source_p == (int) nul_char) {
                                        GET_NEXT_BLOCK(source);
                                        if (source == NULL) {
                                                string_start = source_p;
                                                goto command_newline;
                                        }
                                }
                        }
                        string_start = source_p;
                        if ((*source_p == (int) newline_char) ||
                            (*source_p == (int) backslash_char) ||
                            (*source_p == (int) nul_char)) {
                                source_p--;
                        }
                        break;
                case newline_char:
                command_newline:
                        if ((string_start != source_p) ||
                            macro_seen_in_string) {
                                if (macro_seen_in_string) {
                                        append_string(string_start,
                                                      &name_string,
                                                      source_p - string_start);
                                        string_start =
                                          name_string.buffer.start;
                                        string_end = name_string.text.p;
                                } else {
                                        string_end = source_p;
                                }
                                while ((*string_start != (int) newline_char) &&
                                       iswspace(*string_start)){
                                        string_start++;
                                }
                                if ((string_end > string_start) ||
                                    (makefile_type == reading_statefile)) {
                                        if (command_tail == NULL) {
                                                command =
                                                  command_tail =
                                                    ALLOC(Cmd_line);
                                        } else {
                                                command_tail->next =
                                                  ALLOC(Cmd_line);
                                                command_tail =
```

```
                                                          command_tail->next;
                                                }
                                                command_tail->next = NULL;
                                                command_tail->make_refd = false;
                                                command_tail->ignore_command_dependency
                                                command_tail->assign = false;
                                                command_tail->ignore_error = false;
                                                command_tail->silent = false;
                                                command_tail->command_line =
                                                  GETNAME(string_start,
                                                          string_end - string_start);
                                                if (macro_seen_in_string &&
                                                    name_string.free_after_use) {
                                                        retmem(name_string.
                                                               buffer.start);
                                                }
                                        }
                                }
                                do {
                                        if ((source != NULL) && (source->fd >= 0)) {
                                                line_number++;
                                        }
                                        if ((source != NULL) &&
                                            (*++source_p == (int) nul_char)) {
                                                GET_NEXT_BLOCK(source);
                                                if (source == NULL) {
                                                        GOTO_STATE(on_eoln_state);
                                                }
                                        }
                                } while (*source_p == (int) newline_char);

                                GOTO_STATE(enter_dependencies_state);
                case nul_char:
                                if (!macro_seen_in_string) {
                                        INIT_STRING_FROM_STACK(name_string,
                                                               name_buffer);
                                }
                                append_string(string_start,
                                              &name_string,
                                              source_p - string_start);
                                macro_seen_in_string = true;
                                GET_NEXT_BLOCK(source);
                                string_start = source_p;
                                source_p--;
                                if (source == NULL) {
                                        GOTO_STATE(enter_dependencies_state);
                                }
                                break;
                }
        }

/****************************************************************
 *      enter equal state
 */
case enter_equal_state:
        if (target.used != 1) {
                GOTO_STATE(poorly_formed_macro_state);
        }
        enter_equal(target.names[0], macro_value, append);
        goto start_new_line;

/****************************************************************
 *      enter conditional state
 */
case enter_conditional_state:
        if (depes.used != 1) {
                GOTO_STATE(poorly_formed_macro_state);
        }
        for (nvp = &target; nvp != NULL; nvp = nvp->next) {
                for (i = 0; i < nvp->used; i++) {
                        enter_conditional(nvp->names[i],
                                          depes.names[0],
                                          macro_value,
                                          append);
                }
        }
        goto start_new_line;

/****************************************************************
 *      Error states
 */
case illegal_bytes_state:
        fatal_reader(gettext("Invalid byte sequence"));
case illegal_eoln_state:
        if (line_number > 1) {
                if (line_started_with_space == (line_number - 1)) {
                        line_number--;
                        fatal_reader(gettext("Unexpected end of line seen\n\t***
                }
        }
        fatal_reader(gettext("Unexpected end of line seen"));
case poorly_formed_macro_state:
        fatal_reader(gettext("Badly formed macro assignment"));
case exit_state:
        return;
default:
        fatal_reader(gettext("Internal error. Unknown reader state"));
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   51209 Fri May 22 11:19:46 2015
new/usr/src/cmd/make/bin/read2.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
/*
 * 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 2005 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      read.c
 *
 *      This file contains the makefile reader.
 */

/*
 * Included files
 */
#include <mk/defs.h>
#include <mksh/dosys.h>         /* sh_command2string() */
#include <mksh/macro.h>         /* expand_value() */
#include <mksh/misc.h>          /* retmem() */
#include <stdarg.h>             /* va_list, va_start(), va_end() */
#include <libintl.h>

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static  Boolean         built_last_make_run_seen;

/*
 * File table of contents
 */
static  Name_vector     enter_member_name(register wchar_t *lib_start, register
extern  Name            normalize_name(register wchar_t *name_string, register i
static  void            read_suffixes_list(register Name_vector depes);
static  void            make_relative(wchar_t *to, wchar_t *result);
```

```
static  void            print_rule(register Cmd_line command);
static  void            sh_transform(Name *name, Name *value);


/*
 *      enter_name(string, tail_present, string_start, string_end,
 *            current_names, extra_names, target_group_seen)
 *
 *      Take one string and enter it as a name. The string is passed in
 *      two parts. A make string and possibly a C string to append to it.
 *      The result is stuffed in the vector current_names.
 *      extra_names points to a vector that is used if current_names overflows.
 *      This is allocad in the calling routine.
 *      Here we handle the "lib.a[members]" notation.
 *
 *      Return value:
 *                              The name vector that was used
 *
 *      Parameters:
 *              tail_present    Indicates if both C and make string was passed
 *              string_start    C string
 *              string_end      Pointer to char after last in C string
 *              string          make style string with head of name
 *              current_names   Vector to deposit the name in
 *              extra_names     Where to get next name vector if we run out
 *              target_group_seen Pointer to boolean that is set if "+" is seen
 *
 *      Global variables used:
 *              makefile_type   When we read a report file we normalize paths
 *              plus            Points to the Name "+"
 */

Name_vector
enter_name(String string, Boolean tail_present, register wchar_t *string_start,
{
        Name                    name;
        register wchar_t        *cp;
        wchar_t                 ch;

        /* If we were passed a separate tail of the name we append it to the */
        /* make string with the rest of it */
        if (tail_present) {
                append_string(string_start, string, string_end - string_start);
                string_start = string->buffer.start;
                string_end = string->text.p;
        }
        ch = *string_end;
        *string_end = (int) nul_char;
        /*
         * Check if there are any ( or [ that are not prefixed with $.
         * If there are, we have to deal with the lib.a(members) format.
         */
        for (cp = (wchar_t *) wcschr(string_start, (int) parenleft_char);
        for (cp = (wchar_t *) wschr(string_start, (int) parenleft_char);
             cp != NULL;
             cp = (wchar_t *) wcschr(cp + 1, (int) parenleft_char)) {
             cp = (wchar_t *) wschr(cp + 1, (int) parenleft_char)) {
                if (*(cp - 1) != (int) dollar_char) {
                        *string_end = ch;
                        return enter_member_name(string_start,
                                                 cp,
                                                 string_end,
                                                 current_names,
                                                 extra_names);
                }
        }
        *string_end = ch;

        if (makefile_type == reading_cpp_file) {
                /* Remove extra ../ constructs if we are reading from a report f
                name = normalize_name(string_start, string_end - string_start);
        } else {
                /*
                 * /tolik, fix bug 1197477/
                 * Normalize every target name before entering.
                 * ..//obj/a.o and ../obj//a.o are not two different targets.
                 * There is only one target ../obj/a.o
                 */
                /*name = GETNAME(string_start, string_end - string_start);*/
                name = normalize_name(string_start, string_end - string_start);
        }

        /* Internalize the name. Detect the name "+" (target group here) */
if(current_names->used != 0 && current_names->names[current_names->used-1] == pl
        if(name == plus) {
                return current_names;
        }
}
```

______unchanged_portion_omitted_

```
/*
 *      enter_member_name(lib_start, member_start, string_end,
 *                current_names, extra_names)
 *
 *      A string has been found to contain member names.
 *      (The "lib.a[members]" and "lib.a(members)" notation)
 *      Handle it pretty much as enter_name() does for simple names.
 *
 *      Return value:
 *                              The name vector that was used
 *
 *      Parameters:
 *              lib_start       Points to the of start of "lib.a(member.o)"
 *              member_start    Points to "member.o" from above string.
 *              string_end      Points to char after last of above string.
 *              current_names   Vector to deposit the name in
 *              extra_names     Where to get next name vector if we run out
 *
 *      Global variables used:
 */
static Name_vector
enter_member_name(register wchar_t *lib_start, register wchar_t *member_start, r
{
        register Boolean        entry = false;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        Name                    lib;
        Name                    member;
        Name                    name;
        Property                prop;
        wchar_t                 *memberp;
        wchar_t                 *q;
        register int            paren_count;
        register Boolean        has_dollar;
        register wchar_t        *cq;
        Name                    long_member_name = NULL;

        /* Internalize the name of the library */
        lib = GETNAME(lib_start, member_start - lib_start);
        lib->is_member = true;
        member_start++;
        if (*member_start == (int) parenleft_char) {
                /* This is really the "lib.a((entries))" format */
```

```
                entry = true;
                member_start++;
        }
        /* Move the library name to the buffer where we intend to build the */
        /* "lib.a(member)" for each member */
        (void) wcsncpy(buffer, lib_start, member_start - lib_start);
        (void) wsncpy(buffer, lib_start, member_start - lib_start);
        memberp = buffer + (member_start-lib_start);
        while (1) {
                long_member_name = NULL;
                /* Skip leading spaces */
                for (;
                     (member_start < string_end) && iswspace(*member_start);
                     member_start++);
                /* Find the end of the member name. Allow nested (). Detect $*/
                for (cq = memberp, has_dollar = false, paren_count = 0;
                     (member_start < string_end) &&
                     ((*member_start != (int) parenright_char) ||
                      (paren_count > 0)) &&
                     !iswspace(*member_start);
                     *cq++ = *member_start++) {
                        switch (*member_start) {
                        case parenleft_char:
                                paren_count++;
                                break;
                        case parenright_char:
                                paren_count--;
                                break;
                        case dollar_char:
                                has_dollar = true;
                        }
                }
                /* Internalize the member name */
                member = GETNAME(memberp, cq - memberp);
                *cq = 0;
                if ((q = (wchar_t *) wcsrchr(memberp, (int) slash_char)) == NULL
                if ((q = (wchar_t *) wsrchr(memberp, (int) slash_char)) == NULL)
                        q = memberp;
                }
                if ((cq - q > (int) ar_member_name_len) &&
                    !has_dollar) {
                        *cq++ = (int) parenright_char;
                        if (entry) {
                                *cq++ = (int) parenright_char;
                        }
                        long_member_name = GETNAME(buffer, cq - buffer);
                        cq = q + (int) ar_member_name_len;
                }
                *cq++ = (int) parenright_char;
                if (entry) {
                        *cq++ = (int) parenright_char;
                }
                /* Internalize the "lib.a(member)" notation for this member */
                name = GETNAME(buffer, cq - buffer);
                name->is_member = lib->is_member;
                if (long_member_name != NULL) {
                        prop = append_prop(name, long_member_name_prop);
                        name->has_long_member_name = true;
                        prop->body.long_member_name.member_name =
                          long_member_name;
                }
                /* And add the member prop */
                prop = append_prop(name, member_prop);
                prop->body.member.library = lib;
                if (entry) {
                        /* "lib.a((entry))" notation */
                        prop->body.member.entry = member;
                        prop->body.member.member = NULL;
                } else {
                        /* "lib.a(member)" Notation */
                        prop->body.member.entry = NULL;
                        prop->body.member.member = member;
                }
                /* Handle overflow of current_names */
                if (current_names->used == VSIZEOF(current_names->names)) {
                        if (current_names->next != NULL) {
                                current_names = current_names->next;
                        } else {
                                if (*extra_names == NULL) {
                                        current_names =
                                          current_names->next =
                                            ALLOC(Name_vector);
                                } else {
                                        current_names =
                                          current_names->next =
                                            *extra_names;
                                        *extra_names = NULL;
                                }
                                current_names->used = 0;
                                current_names->next = NULL;
                        }
                }
                current_names->target_group[current_names->used] = NULL;
                current_names->names[current_names->used++] = name;
                while (iswspace(*member_start)) {
                        member_start++;
                }
                /* Check if there are more members */
                if ((*member_start == (int) parenright_char) ||
                    (member_start >= string_end)) {
                        return current_names;
                }
        }
        /* NOTREACHED */
}

/*
 *      normalize_name(name_string, length)
 *
 *      Take a namestring and remove redundant ../, // and ./ constructs
 *
 *      Return value:
 *                              The normalized name
 *
 *      Parameters:
 *              name_string     Path string to normalize
 *              length          Length of that string
 *
 *      Global variables used:
 *              dot             The Name ".", compared against
 *              dotdot          The Name "..", compared against
 */
Name
normalize_name(register wchar_t *name_string, register int length)
{
        static Name             dotdot;
        register wchar_t        *string = ALLOC_WC(length + 1);
        register wchar_t        *string2;
        register wchar_t        *cdp;
        wchar_t                 *current_component;
        Name                    name;
        register int            count;
```

```

        if (dotdot == NULL) {
                MBSTOWCS(wcs_buffer, "..");
                dotdot = GETNAME(wcs_buffer, FIND_LENGTH);
        }

        /*
         * Copy string removing ./ and //.
         * First strip leading ./
         */
        while ((length > 1) &&
               (name_string[0] == (int) period_char) &&
               (name_string[1] == (int) slash_char)) {
                name_string += 2;
                length -= 2;
                while ((length > 0) && (name_string[0] == (int) slash_char)) {
                        name_string++;
                        length--;
                }
        }
        /* Then copy the rest of the string removing /./ & // */
        cdp = string;
        while (length > 0) {
                if (((length > 2) &&
                     (name_string[0] == (int) slash_char) &&
                     (name_string[1] == (int) period_char) &&
                     (name_string[2] == (int) slash_char)) ||
                    ((length == 2) &&
                     (name_string[0] == (int) slash_char) &&
                     (name_string[1] == (int) period_char))) {
                        name_string += 2;
                        length -= 2;
                        continue;
                }
                if ((length > 1) &&
                    (name_string[0] == (int) slash_char) &&
                    (name_string[1] == (int) slash_char)) {
                        name_string++;
                        length--;
                        continue;
                }
                *cdp++ = *name_string++;
                length--;
        }
        *cdp = (int) nul_char;
        /*
         * Now scan for <name>/../ and remove such combinations iff <name>
         * is not another ..
         * Each time something is removed, the whole process is restarted.
         */
removed_one:
        name_string = string;
        string2 = name_string;          /*save for free*/
        current_component =
          cdp =
            string =
              ALLOC_WC((length = wcslen(name_string)) + 1);
              ALLOC_WC((length = wslen(name_string)) + 1);
        while (length > 0) {
                if (((length > 3) &&
                     (name_string[0] == (int) slash_char) &&
                     (name_string[1] == (int) period_char) &&
                     (name_string[2] == (int) period_char) &&
                     (name_string[3] == (int) slash_char)) ||
                    ((length == 3) &&
                     (name_string[0] == (int) slash_char) &&
                     (name_string[1] == (int) period_char) &&
                     (name_string[2] == (int) period_char))) {
                        /* Positioned on the / that starts a /.. sequence */
                        if (((count = cdp - current_component) != 0) &&
                            (exists(name = GETNAME(string, cdp - string)) > file
                            (!name->stat.is_sym_link)) {
                                name = GETNAME(current_component, count);
                                if(name != dotdot) {
                                        cdp = current_component;
                                        name_string += 3;
                                        length -= 3;
                                        if (length > 0) {
                                                name_string++;  /* skip slash */
                                                length--;
                                                while (length > 0) {
                                                        *cdp++ = *name_string++;
                                                        length--;
                                                }
                                        }
                                        *cdp = (int) nul_char;
                                        retmem(string2);
                                        goto removed_one;
                                }
                        }
                }
                if ((*cdp++ = *name_string++) == (int) slash_char) {
                        current_component = cdp;
                }
                length--;
        }
        *cdp = (int) nul_char;
        if (string[0] == (int) nul_char) {
                name = dot;
        } else {
                name = GETNAME(string, FIND_LENGTH);
        }
        retmem(string);
        retmem(string2);
        return name;
}
```

______***unchanged_portion_omitted_***

```
/*
 *      enter_dependencies(target, target_group, depes, command, separator)
 *
 *      Take one target and a list of dependencies and process the whole thing.
 *      The target might be special in some sense in which case that is handled
 *
 *      Parameters:
 *              target          The target we want to enter
 *              target_group    Non-NULL if target is part of a group this time
 *              depes           A list of dependencies for the target
 *              command         The command the target should be entered with
 *              separator       Indicates if this is a ":" or a "::" rule
 *
 *      Static variables used:
 *              built_last_make_run_seen If the previous target was
 *                                      .BUILT_LAST_MAKE_RUN we say to rewrite
 *                                      the state file later on
 *
 *      Global variables used:
 *              command_changed Set to indicate if .make.state needs rewriting
 *              default_target_to_build Set to the target if reading makefile
 *                                      and this is the first regular target
 *              force           The Name " FORCE", used with "::" targets
 *              makefile_type   We do different things for makefile vs. report
```

```
 *              not_auto        The Name ".NOT_AUTO", compared against
 *              recursive_name  The Name ".RECURSIVE", compared against
 *              temp_file_number Used to figure out when to clear stale
 *                                      automatic dependencies
 *              trace_reader    Indicates that we should echo stuff we read
 */
void
enter_dependencies(register Name target, Chain target_group, register Name_vecto
{
        register int            i;
        register Property       line;
        Name                    name;
        Name                    directory;
        wchar_t                 *namep;
        char                    *mb_namep;
        Dependency              dp;
        Dependency              *dpp;
        Property                line2;
        wchar_t                 relative[MAXPATHLEN];
        register int            recursive_state;
        Boolean                 register_as_auto;
        Boolean                 not_auto_found;
        char                    *slash;
        Wstring                 depstr;

        /* Check if this is a .RECURSIVE line */
        if ((depes->used >= 3) &&
            (depes->names[0] == recursive_name)) {
                target->has_recursive_dependency = true;
                depes->names[0] = NULL;
                recursive_state = 0;
                dp = NULL;
                dpp = &dp;
                /* Read the dependencies. They are "<directory> <target-made>*/
                /* <makefile>*" */
                for (; depes != NULL; depes = depes->next) {
                        for (i = 0; i < depes->used; i++) {
                                if (depes->names[i] != NULL) {
                                        switch (recursive_state++) {
                                        case 0: /* Directory */
                                        {
                                                depstr.init(depes->names[i]);
                                                make_relative(depstr.get_string(
                                                              relative);
                                                directory =
                                                  GETNAME(relative,
                                                          FIND_LENGTH);
                                        }
                                                break;
                                        case 1: /* Target */
                                                name = depes->names[i];
                                                break;
                                        default:        /* Makefiles */
                                                *dpp = ALLOC(Dependency);
                                                (*dpp)->next = NULL;
                                                (*dpp)->name = depes->names[i];
                                                (*dpp)->automatic = false;
                                                (*dpp)->stale = false;
                                                (*dpp)->built = false;
                                                dpp = &((*dpp)->next);
                                                break;
                                        }
                                }
                        }
                }
                /* Check if this recursion already has been reported else */
                /* enter the recursive prop for the target */
                /* The has_built flag is used to tell if this .RECURSIVE */
                /* was discovered from this run (read from a tmp file) */
                /* or was from discovered from the original .make.state */
                /* file */
                for (line = get_prop(target->prop, recursive_prop);
                     line != NULL;
                     line = get_prop(line->next, recursive_prop)) {
                        if ((line->body.recursive.directory == directory) &&
                            (line->body.recursive.target == name)) {
                                line->body.recursive.makefiles = dp;
                                line->body.recursive.has_built =
                                  (Boolean)
                                    (makefile_type == reading_cpp_file);
                                return;
                        }
                }
                line2 = append_prop(target, recursive_prop);
                line2->body.recursive.directory = directory;
                line2->body.recursive.target = name;
                line2->body.recursive.makefiles = dp;
                line2->body.recursive.has_built =
                    (Boolean) (makefile_type == reading_cpp_file);
                line2->body.recursive.in_depinfo = false;
                return;
        }
        /* If this is the first target that doesnt start with a "." in the */
        /* makefile we remember that */
        Wstring tstr(target);
        wchar_t * wcb = tstr.get_string();
        if ((makefile_type == reading_makefile) &&
            (default_target_to_build == NULL) &&
            ((wcb[0] != (int) period_char) ||
             wcschr(wcb, (int) slash_char))) {
             wschr(wcb, (int) slash_char))) {

/* BID 1181577: $(EMPTY_MACRO) + $(EMPTY_MACRO):
** The target with empty name cannot be default_target_to_build
*/
                if (target->hash.length != 0)
                        default_target_to_build = target;
        }
        /* Check if the line is ":" or "::" */
        if (makefile_type == reading_makefile) {
                if (target->colons == no_colon) {
                        target->colons = separator;
                } else {
                        if (target->colons != separator) {
                                fatal_reader(gettext(":/:: conflict for target ‘
                                             target->string_mb);
                        }
                }
                if (target->colons == two_colon) {
                        if (depes->used == 0) {
                                /* If this is a "::" type line with no */
                                /* dependencies we add one "FRC" type */
                                /* dependency for free */
                                depes->used = 1; /* Force :: targets with no
                                                  * depes to always run */
                                depes->names[0] = force;
                        }
                        /* Do not delete "::" type targets when interrupted */
                        target->stat.is_precious = true;
                        /*
                         * Build a synthetic target "<number>%target"
                         * for "target".
```

```
                         */
                        mb_namep = getmem((int) (strlen(target->string_mb) + 10)
                        namep = ALLOC_WC((int) (target->hash.length + 10));
                        slash = strrchr(target->string_mb, (int) slash_char);
                        if (slash == NULL) {
                                (void) sprintf(mb_namep,
                                                "%d@%s",
                                                target->colon_splits++,
                                                target->string_mb);
                        } else {
                                *slash = 0;
                                (void) sprintf(mb_namep,
                                                "%s/%d@%s",
                                                target->string_mb,
                                                target->colon_splits++,
                                                slash + 1);
                                *slash = (int) slash_char;
                        }
                        MBSTOWCS(namep, mb_namep);
                        retmem_mb(mb_namep);
                        name = GETNAME(namep, FIND_LENGTH);
                        retmem(namep);
                        if (trace_reader) {
                                (void) printf("%s:\t", target->string_mb);
                        }
                        /* Make "target" depend on "<number>%target */
                        line2 = maybe_append_prop(target, line_prop);
                        enter_dependency(line2, name, true);
                        line2->body.line.target = target;
                        /* Put a prop on "<number>%target that makes */
                        /* appear as "target" */
                        /* when it is processed */
                        maybe_append_prop(name, target_prop)->
                          body.target.target = target;
                        target->is_double_colon_parent = true;
                        name->is_double_colon = true;
                        name->has_target_prop = true;
                        if (trace_reader) {
                                (void) printf("\n");
                        }
                        (target = name)->stat.is_file = true;
                }
        }
        /* This really is a regular dependency line. Just enter it */
        line = maybe_append_prop(target, line_prop);
        line->body.line.target = target;
        /* Depending on what kind of makefile we are reading we have to */
        /* treat things differently */
        switch (makefile_type) {
        case reading_makefile:
                /* Reading regular makefile. Just notice whether this */
                /* redefines the rule for the  target */
                if (command != NULL) {
                        if (line->body.line.command_template != NULL) {
                                line->body.line.command_template_redefined =
                                  true;
                                if ((wcb[0] == (int) period_char) &&
                                    !wcschr(wcb, (int) slash_char)) {
                                    !wschr(wcb, (int) slash_char)) {
                                        line->body.line.command_template =
                                          command;
                                }
                        } else {
                                line->body.line.command_template = command;
                        }
                } else {
                        if ((wcb[0] == (int) period_char) &&
                            !wcschr(wcb, (int) slash_char)) {
                            !wschr(wcb, (int) slash_char)) {
                                line->body.line.command_template = command;
                        }
                }
                break;
        case rereading_statefile:
                /* Rereading the statefile. We only enter thing that changed */
                /* since the previous time we read it */
                if (!built_last_make_run_seen) {
                        for (Cmd_line next, cmd = command; cmd != NULL; cmd = ne
                                next = cmd->next;
                                free(cmd);
                        }
                        return;
                }
                built_last_make_run_seen = false;
                command_changed = true;
                target->ran_command = true;
        case reading_statefile:
                /* Reading the statefile for the first time. Enter the rules */
                /* as "Commands used" not "templates to use" */
                if (command != NULL) {
                        for (Cmd_line next, cmd = line->body.line.command_used;
                             cmd != NULL; cmd = next) {
                                next = cmd->next;
                                free(cmd);
                        }
                        line->body.line.command_used = command;
                }
        case reading_cpp_file:
                /* Reading report file from programs that reports */
                /* dependencies. If this is the first time the target is */
                /* read from this reportfile we clear all old */
                /* automatic depes */
                if (target->temp_file_number == temp_file_number) {
                        break;
                }
                target->temp_file_number = temp_file_number;
                command_changed = true;
                if (line != NULL) {
                        for (dp = line->body.line.dependencies;
                             dp != NULL;
                             dp = dp->next) {
                                if (dp->automatic) {
                                        dp->stale = true;
                                }
                        }
                }
                break;
        default:
                fatal_reader(gettext("Internal error. Unknown makefile type %d")
                             makefile_type);
        }
        /* A target may only be involved in one target group */
        if (line->body.line.target_group != NULL) {
                if (target_group != NULL) {
                        fatal_reader(gettext("Too many target groups for target
                                     target->string_mb);
                }
        } else {
                line->body.line.target_group = target_group;
        }

        if (trace_reader) {
```

```
                (void) printf("%s:\t", target->string_mb);
        }
        /* Enter the dependencies */
        register_as_auto = BOOLEAN(makefile_type != reading_makefile);
        not_auto_found = false;
        for (;
             (depes != NULL) && !not_auto_found;
             depes = depes->next) {
                for (i = 0; i < depes->used; i++) {
                        /* the dependency .NOT_AUTO signals beginning of
                         * explicit dependancies which were put at end of
                         * list in .make.state file - we stop entering
                         * dependencies at this point
                         */
                        if (depes->names[i] == not_auto) {
                                not_auto_found = true;
                                break;
                        }
                        enter_dependency(line,
                                         depes->names[i],
                                         register_as_auto);
                }
        }
        if (trace_reader) {
                (void) printf("\n");
                print_rule(command);
        }
}
______unchanged_portion_omitted_

/*
 *      enter_percent(target, depes, command)
 *
 *      Enter "x%y : a%b" type lines
 *      % patterns are stored in four parts head and tail for target and source
 *
 *      Parameters:
 *              target          Left hand side of pattern
 *              depes           The dependency list with the rh pattern
 *              command         The command for the pattern
 *
 *      Global variables used:
 *              empty_name      The Name "", compared against
 *              percent_list    The list of all percent rules, added to
 *              trace_reader    Indicates that we should echo stuff we read
 */
Percent
enter_percent(register Name target, Chain target_group, register Name_vector dep
{
        register Percent        result = ALLOC(Percent);
        register Percent        depe;
        register Percent        *depe_tail = &result->dependencies;
        register Percent        *insert;
        register wchar_t        *cp, *cp1;
        Name_vector             nvp;
        int                     i;
        int                     pattern;

        result->next = NULL;
        result->patterns = NULL;
        result->patterns_total = 0;
        result->command_template = command;
        result->being_expanded = false;
        result->name = target;
        result->dependencies = NULL;
        result->target_group = target_group;
        /* get patterns count */
        Wstring wcb(target);
        cp = wcb.get_string();
        while (true) {
                cp = (wchar_t *) wcschr(cp, (int) percent_char);
                cp = (wchar_t *) wschr(cp, (int) percent_char);
                if (cp != NULL) {
                        result->patterns_total++;
                        cp++;
                } else {
                        break;
                }
        }
        result->patterns_total++;

        /* allocate storage for patterns */
        result->patterns = (Name *) getmem(sizeof(Name) * result->patterns_total

        /* then create patterns */
        cp = wcb.get_string();
        pattern = 0;
        while (true) {
                cp1 = (wchar_t *) wcschr(cp, (int) percent_char);
                cp1 = (wchar_t *) wschr(cp, (int) percent_char);
                if (cp1 != NULL) {
                        result->patterns[pattern] = GETNAME(cp, cp1 - cp);
                        cp = cp1 + 1;
                        pattern++;
                } else {
                        result->patterns[pattern] = GETNAME(cp, (int) target->ha
                        break;
                }
        }

        Wstring wcb1;

        /* build dependencies list */
        for (nvp = depes; nvp != NULL; nvp = nvp->next) {
                for (i = 0; i < nvp->used; i++) {
                        depe = ALLOC(Percent);
                        depe->next = NULL;
                        depe->patterns = NULL;
                        depe->patterns_total = 0;
                        depe->name = nvp->names[i];
                        depe->dependencies = NULL;
                        depe->command_template = NULL;
                        depe->being_expanded = false;
                        depe->target_group = NULL;

                        *depe_tail = depe;
                        depe_tail = &depe->next;

                        if (depe->name->percent) {
                                /* get patterns count */
                                wcb1.init(depe->name);
                                cp = wcb1.get_string();
                                while (true) {
                                        cp = (wchar_t *) wcschr(cp, (int) percen
                                        cp = (wchar_t *) wschr(cp, (int) percent
                                        if (cp != NULL) {
                                                depe->patterns_total++;
                                                cp++;
                                        } else {
                                                break;
                                        }
```

```
                                }
                                depe->patterns_total++;

                                /* allocate storage for patterns */
                                depe->patterns = (Name *) getmem(sizeof(Name) *

                                /* then create patterns */
                                cp = wcb1.get_string();
                                pattern = 0;
                                while (true) {
                                        cp1 = (wchar_t *) wcschr(cp, (int) perce
                                        cp1 = (wchar_t *) wschr(cp, (int) percen
                                        if (cp1 != NULL) {
                                                depe->patterns[pattern] = GETNAM
                                                cp = cp1 + 1;
                                                pattern++;
                                        } else {
                                                depe->patterns[pattern] = GETNAM
                                                break;
                                        }
                                }
                        }
                }
        }

        /* Find the end of the percent list and append the new pattern */
        for (insert = &percent_list; (*insert) != NULL; insert = &(*insert)->nex
        *insert = result;

        if (trace_reader) {
                (void) printf("%s:", result->name->string_mb);

                for (depe = result->dependencies; depe != NULL; depe = depe->nex
                        (void) printf(" %s", depe->name->string_mb);
                }

                (void) printf("\n");

                print_rule(command);
        }

        return result;
}
```

______***unchanged_portion_omitted_***

```
/*
 *      make_relative(to, result)
 *
 *      Given a file name compose a relative path name from it to the
 *      current directory.
 *
 *      Parameters:
 *              to              The path we want to make relative
 *              result          Where to put the resulting relative path
 *
 *      Global variables used:
 */
static void
make_relative(wchar_t *to, wchar_t *result)
{
        wchar_t                 *from;
        wchar_t                 *allocated;
        wchar_t                 *cp;
        wchar_t                 *tocomp;
        int                     ncomps;
        int                     i;
        int                     len;

        /* Check if the path is already relative. */
        if (to[0] != (int) slash_char) {
                (void) wcscpy(result, to);
                (void) wscpy(result, to);
                return;
        }

        MBSTOWCS(wcs_buffer, get_current_path());
        from = allocated = (wchar_t *) wcsdup(wcs_buffer);
        from = allocated = (wchar_t *) wsdup(wcs_buffer);

        /*
         * Find the number of components in the from name.
         * ncomp = number of slashes + 1.
         */
        ncomps = 1;
        for (cp = from; *cp != (int) nul_char; cp++) {
                if (*cp == (int) slash_char) {
                        ncomps++;
                }
        }

        /*
         * See how many components match to determine how many "..",
         * if any, will be needed.
         */
        result[0] = (int) nul_char;
        tocomp = to;
        while ((*from != (int) nul_char) && (*from == *to)) {
                if (*from == (int) slash_char) {
                        ncomps--;
                        tocomp = &to[1];
                }
                from++;
                to++;
        }

        /*
         * Now for some special cases. Check for exact matches and
         * for either name terminating exactly.
         */
        if (*from == (int) nul_char) {
                if (*to == (int) nul_char) {
                        MBSTOWCS(wcs_buffer, ".");
                        (void) wcscpy(result, wcs_buffer);
                        (void) wscpy(result, wcs_buffer);
                        retmem(allocated);
                        return;
                }
                if (*to == (int) slash_char) {
                        ncomps--;
                        tocomp = &to[1];
                }
        } else if ((*from == (int) slash_char) && (*to == (int) nul_char)) {
                ncomps--;
                tocomp = to;
        }
        /* Add on the ".."s. */
        for (i = 0; i < ncomps; i++) {
                MBSTOWCS(wcs_buffer, "../");
                (void) wcscat(result, wcs_buffer);
                (void) wscat(result, wcs_buffer);
        }
```

```
        /* Add on the remainder of the to name, if any. */
        if (*tocomp == (int) nul_char) {
                len = wcslen(result);
                len = wslen(result);
                result[len - 1] = (int) nul_char;
        } else {
                (void) wcscat(result, tocomp);
                (void) wscat(result, tocomp);
        }
        retmem(allocated);
        return;
}
______unchanged_portion_omitted_

/*
 *      sh_transform(name, value)
 *
 *      Parameters:
 *              name    The name of the macro we might transform
 *              value   The value to transform
 *
 */
static void
sh_transform(Name *name, Name *value)
{
        /* Check if we need :sh transform */
        wchar_t         *colon;
        String_rec      command;
        String_rec      destination;
        wchar_t         buffer[1000];
        wchar_t         buffer1[1000];

        static wchar_t  colon_sh[4];
        static wchar_t  colon_shell[7];

        if (colon_sh[0] == (int) nul_char) {
                MBSTOWCS(colon_sh, ":sh");
                MBSTOWCS(colon_shell, ":shell");
        }
        Wstring nms((*name));
        wchar_t * wcb = nms.get_string();

        colon = (wchar_t *) wcsrchr(wcb, (int) colon_char);
        colon = (wchar_t *) wsrchr(wcb, (int) colon_char);
        if ((colon != NULL) && (IS_WEQUAL(colon, colon_sh) || IS_WEQUAL(colon, c
                INIT_STRING_FROM_STACK(destination, buffer);

                if(*value == NULL) {
                        buffer[0] = 0;
                } else {
                        Wstring wcb1((*value));
                        if (IS_WEQUAL(colon, colon_shell)) {
                                INIT_STRING_FROM_STACK(command, buffer1);
                                expand_value(*value, &command, false);
                        } else {
                                command.text.p = wcb1.get_string() + (*value)->h
                                command.text.end = command.text.p;
                                command.buffer.start = wcb1.get_string();
                                command.buffer.end = command.text.p;
                        }
                        sh_command2string(&command, &destination);
                }

                (*value) = GETNAME(destination.buffer.start, FIND_LENGTH);
                *colon = (int) nul_char;
                (*name) = GETNAME(wcb, FIND_LENGTH);
                *colon = (int) colon_char;
        }
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
    9647 Fri May 22 11:19:47 2015
new/usr/src/cmd/make/bin/rep.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
/*
 * 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 2003 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      rep.c
 *
 *      This file handles the .nse_depinfo file
 */

/*
 * Included files
 */
#include <mk/defs.h>
#include <mksh/misc.h>          /* retmem() */
#include <vroot/report.h>       /* NSE_DEPINFO */

/*
 * Static variables
 */
static  Recursive_make  recursive_list;
static  Recursive_make  *bpatch = &recursive_list;
static  Boolean         changed;

/*
 * File table of contents
 */


/*
 *      report_recursive_init()
 *
 *      Read the .nse_depinfo file and make a list of all the
 *      .RECURSIVE entries.
 *
 *      Parameters:
 *
 *      Static variables used:
 *              bpatch          Points to slot where next cell should be added
 *
```

```
 *      Global variables used:
 *              recursive_name  The Name ".RECURSIVE", compared against
 */

void
report_recursive_init(void)
{
        char            *search_dir;
        char            nse_depinfo[MAXPATHLEN];
        FILE            *fp;
        int             line_size, line_index;
        wchar_t         *line;
        wchar_t         *bigger_line;
        wchar_t         *colon;
        wchar_t         *dollar;
        Recursive_make  rp;

        /*
         * This routine can be called more than once,  don't do
         * anything after the first time.
         */
        if (depinfo_already_read) {
                return;
        } else {
                depinfo_already_read = true;
        }

        search_dir = getenv("NSE_DEP");
        if (search_dir == NULL) {
                return;
        }
        (void) sprintf(nse_depinfo, "%s/%s", search_dir, NSE_DEPINFO);
        fp = fopen(nse_depinfo, "r");
        if (fp == NULL) {
                return;
        }
        line_size = MAXPATHLEN;
        line_index = line_size - 1;
        line = ALLOC_WC(line_size);
        Wstring rns(recursive_name);
        wchar_t * wcb = rns.get_string();
        while (fgetws(line, line_size, fp) != NULL) {
                while (wcslen(line) == line_index) {
                        if (line[wcslen(line) - 1] == '\n') {
                while (wslen(line) == line_index) {
                        if (line[wslen(line) - 1] == '\n') {
                                continue;
                        }
                        bigger_line = ALLOC_WC(2 * line_size);
                        wcscpy(bigger_line, line);
                        wscpy(bigger_line, line);
                        retmem(line);
                        line = bigger_line;
                        if (fgetws(&line[line_index], line_size, fp) == NULL)
                                continue;
                        line_index = 2 * line_index;
                        line_size = 2 * line_size;
                }

                colon = (wchar_t *) wcschr(line, (int) colon_char);
                colon = (wchar_t *) wschr(line, (int) colon_char);
                if (colon == NULL) {
                        continue;
                }
                dollar = (wchar_t *) wcschr(line, (int) dollar_char);
                line[wcslen(line) - 1] = (int) nul_char;
                dollar = (wchar_t *) wschr(line, (int) dollar_char);
                line[wslen(line) - 1] = (int) nul_char;
                if (IS_WEQUALN(&colon[2], wcb,
                    (int) recursive_name->hash.length)) {
                        /*
                         * If this entry is an old entry, ignore it
                         */
                        MBSTOWCS(wcs_buffer, DEPINFO_FMT_VERSION);
                        if (dollar == NULL ||
                            !IS_WEQUALN(wcs_buffer, (dollar+1) - VER_LEN, VER_LE
                                continue;
                            }
                        rp = ALLOC(Recursive_make);
                        (void) memset((char *) rp, 0, sizeof (Recursive_make_rec
                        /*
                         * set conditional_macro_string if string is present
                         */
                        rp->oldline = (wchar_t *) wcsdup(line);
                        rp->oldline = (wchar_t *) wsdup(line);
                        if ( dollar != NULL ){
                                rp->cond_macrostring =
                                    (wchar_t *) wcsdup(dollar - VER_LEN + 1);
                                    (wchar_t *) wsdup(dollar - VER_LEN + 1);
                        }
                        /*
                         * get target name into recursive struct
                         */
                        *colon = (int) nul_char;
                        rp->target = (wchar_t *) wcsdup(line);
                        rp->target = (wchar_t *) wsdup(line);
                        *bpatch = rp;
                        bpatch = &rp->next;
                }
        }
        (void) fclose(fp);
}

/*
 *      report_recursive_dep(target, line)
 *
 *      Report a target as recursive.
 *
 *      Parameters:
 *              line            Dependency line reported
 *
 *      Static variables used:
 *              bpatch          Points to slot where next cell should be added
 *              changed         Written if report set changed
 */
void
report_recursive_dep(Name target, wchar_t *line)
{
        Recursive_make  rp;
        wchar_t         rec_buf[STRING_BUFFER_LENGTH];
        String_rec      string;

        INIT_STRING_FROM_STACK(string, rec_buf);
        cond_macros_into_string(target, &string);
        /*
         * find an applicable recursive entry, if there isn't one, create it
         */
        rp = find_recursive_target(target);
        if (rp == NULL) {
                rp = ALLOC(Recursive_make);
                (void) memset((char *) rp, 0, sizeof (Recursive_make_rec));
                wchar_t * wcb = get_wstring(target->string_mb); // XXX Tolik: ne
```

```
                rp->target = wcb;
                rp->newline = (wchar_t *) wcsdup(line);
                rp->cond_macrostring = (wchar_t *) wcsdup(rec_buf);
                rp->newline = (wchar_t *) wsdup(line);
                rp->cond_macrostring = (wchar_t *) wsdup(rec_buf);
                *bpatch = rp;
                bpatch = &rp->next;
                changed = true;
        } else {
                if ((rp->oldline != NULL) && !IS_WEQUAL(rp->oldline, line)) {
                        rp->newline = (wchar_t *) wcsdup(line);
                        rp->newline = (wchar_t *) wsdup(line);
                        changed = true;
                }
                rp->removed = false;
        }
}
______unchanged_portion_omitted_
```

```
**********************************************************
   22490 Fri May 22 11:19:47 2015
new/usr/src/cmd/make/include/mksh/defs.h
make: use the more modern wchar routines, not widec.h
**********************************************************
#ifndef _MKSH_DEFS_H
#define _MKSH_DEFS_H
/*
 * 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 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

#include <limits.h>             /* MB_LEN_MAX */
#include <stdio.h>
#include <stdlib.h>             /* wchar_t */
#include <string.h>             /* strcmp() */
#include <sys/param.h>          /* MAXPATHLEN */
#include <sys/types.h>          /* time_t, caddr_t */
#include <vroot/vroot.h>        /* pathpt */
#include <sys/time.h>           /* timestruc_t */
#include <errno.h>              /* errno */

#include <wctype.h>
#include <widec.h>


/*
 * A type and some utilities for boolean values
 */

#define false   BOOLEAN_false
#define true    BOOLEAN_true

typedef enum {
        false =         0,
        true =          1,
        failed =        0,
        succeeded =     1
} Boolean;
______unchanged_portion_omitted_

/*
 * CHAR_SEMANTICS_ENTRIES should be the number of entries above.
 * The last entry in char_semantics[] should be blank.
 */
#define CHAR_SEMANTICS_ENTRIES  27
```

```
/*
#define CHAR_SEMANTICS_STRING   "&*@'\\|[]:$=!>-\n#()%+?;^<'\""
 */

/*
 * Some utility macros
 */
#define ALLOC(x)                ((struct _##x *)getmem(sizeof (struct _##x)))
#define ALLOC_WC(x)             ((wchar_t *)getmem((x) * SIZEOFWCHAR_T))
#define FIND_LENGTH             -1
#define GETNAME(a,b)            getname_fn((a), (b), false)
#define IS_EQUAL(a,b)           (!strcmp((a), (b)))
#define IS_EQUALN(a,b,n)        (!strncmp((a), (b), (n)))
#define IS_WEQUAL(a,b)          (!wcscmp((a), (b)))
#define IS_WEQUALN(a,b,n)       (!wcsncmp((a), (b), (n)))
#define IS_WEQUAL(a,b)          (!wscmp((a), (b)))
#define IS_WEQUALN(a,b,n)       (!wsncmp((a), (b), (n)))
#define MBLEN(a)                mblen((a), MB_LEN_MAX)
#define MBSTOWCS(a,b)           (void) mbstowcs_with_check((a), (b), MAXPATHLEN)
#define MBTOWC(a,b)             mbtowc((a), (b), MB_LEN_MAX)
#define SIZEOFWCHAR_T           (sizeof (wchar_t))
#define VSIZEOF(v)              (sizeof (v) / sizeof ((v)[0]))
#define WCSTOMBS(a,b)           (void) wcstombs((a), (b), (MAXPATHLEN * MB_LEN_M
#define WCTOMB(a,b)             (void) wctomb((a), (b))
#define HASH(v, c)              (v = (v)*31 + (unsigned int)(c))

extern void mbstowcs_with_check(wchar_t *pwcs, const char *s, size_t n);

/*
 * Bits stored in funny vector to classify chars
 */
enum {
        dollar_sem =            0001,
        meta_sem =              0002,
        percent_sem =           0004,
        wildcard_sem =          0010,
        command_prefix_sem =    0020,
        special_macro_sem =     0040,
        colon_sem =             0100,
        parenleft_sem =         0200
};
______unchanged_portion_omitted_

#define STRING_BUFFER_LENGTH    1024
#define INIT_STRING_FROM_STACK(str, buf) { \
                        str.buffer.start = (buf); \
                        str.text.p = (buf); \
                        str.text.end = NULL; \
                        str.buffer.end = (buf) \
                          + (sizeof (buf)/SIZEOFWCHAR_T); \
                        str.free_after_use = false; \
                  }

#define APPEND_NAME(np, dest, len)      append_string((np)->string_mb, (dest), (

class Wstring {
        public:
                struct _String  string;
                wchar_t         string_buf[STRING_BUFFER_LENGTH];

        public:
                Wstring();
                Wstring(struct _Name * name);
                ~Wstring();

                void init(struct _Name * name);
                void init(wchar_t * name, unsigned length);
                unsigned length() {
                        return wcslen(string.buffer.start);
                        return wslen(string.buffer.start);
                };
                void append_to_str(struct _String * str, unsigned off, unsigned

                wchar_t * get_string() {
                        return string.buffer.start;
                };

                wchar_t * get_string(unsigned off) {
                        return string.buffer.start + off;
                };

                Boolean equaln(wchar_t * str, unsigned length);
                Boolean equal(wchar_t * str);
                Boolean equal(wchar_t * str, unsigned off);
                Boolean equal(wchar_t * str, unsigned off, unsigned length);

                Boolean equaln(Wstring * str, unsigned length);
                Boolean equal(Wstring * str);
                Boolean equal(Wstring * str, unsigned off);
                Boolean equal(Wstring * str, unsigned off, unsigned length);
};
```
______***unchanged_portion_omitted_***

```
**********************************************************
   15031 Fri May 22 11:19:48 2015
new/usr/src/cmd/make/lib/mksh/dosys.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
______unchanged_portion_omitted_

/*
 *      doshell(command, ignore_error)
 *
 *      Used to run command lines that include shell meta-characters.
 *      The make macro SHELL is supposed to contain a path to the shell.
 *
 *      Return value:
 *                              The pid of the process we started
 *
 *      Parameters:
 *              command         The command to run
 *              ignore_error    Should we abort on error?
 *
 *      Global variables used:
 *              filter_stderr   If -X is on we redirect stderr
 *              shell_name      The Name "SHELL", used to get the path to shell
 */
int
doshell(wchar_t *command, register Boolean ignore_error, char *stdout_file, char
{
        char                    *argv[6];
        int                     argv_index = 0;
        int                     cmd_argv_index;
        int                     length;
        char                    nice_prio_buf[MAXPATHLEN];
        register Name           shell = getvar(shell_name);
        register char           *shellname;
        char                    *tmp_mbs_buffer;


        if (IS_EQUAL(shell->string_mb, "")) {
                shell = shell_name;
        }
        if ((shellname = strrchr(shell->string_mb, (int) slash_char)) == NULL) {
                shellname = shell->string_mb;
        } else {
                shellname++;
        }

        /*
         * Only prepend the /usr/bin/nice command to the original command
         * if the nice priority, nice_prio, is NOT zero (0).
         * Nice priorities can be a positive or a negative number.
         */
        if (nice_prio != 0) {
                argv[argv_index++] = (char *)"nice";
                (void) sprintf(nice_prio_buf, "-%d", nice_prio);
                argv[argv_index++] = strdup(nice_prio_buf);
        }
        argv[argv_index++] = shellname;
        argv[argv_index++] = (char*)(ignore_error ? "-c" : "-ce");
        if ((length = wcslen(command)) >= MAXPATHLEN) {
        if ((length = wslen(command)) >= MAXPATHLEN) {
                tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
                (void) wcstombs(tmp_mbs_buffer, command, (length * MB_LEN_MAX) +
                cmd_argv_index = argv_index;
                argv[argv_index++] = strdup(tmp_mbs_buffer);
                retmem_mb(tmp_mbs_buffer);
        } else {
                WCSTOMBS(mbs_buffer, command);
                cmd_argv_index = argv_index;
                argv[argv_index++] = strdup(mbs_buffer);
        }
        argv[argv_index] = NULL;
        (void) fflush(stdout);
        if ((childPid = fork()) == 0) {
                enable_interrupt((void (*) (int)) SIG_DFL);
#if 0
                if (filter_stderr) {
                        redirect_stderr();
                }
#endif
                if (nice_prio != 0) {
                        (void) execve("/usr/bin/nice", argv, environ);
                        fatal_mksh(gettext("Could not load ‘/usr/bin/nice’: %s")
                              errmsg(errno));
                } else {
                        (void) execve(shell->string_mb, argv, environ);
                        fatal_mksh(gettext("Could not load Shell from ‘%s’: %s")
                              shell->string_mb,
                              errmsg(errno));
                }
        }
        if (childPid  == -1) {
                fatal_mksh(gettext("fork failed: %s"),
                      errmsg(errno));
        }
        retmem_mb(argv[cmd_argv_index]);
        return childPid;
}
______unchanged_portion_omitted_

/*
 *      doexec(command, ignore_error)
 *
 *      Will scan an argument string and split it into words
 *      thus building an argument list that can be passed to exec_ve()
 *
 *      Return value:
 *                              The pid of the process started here
 *
 *      Parameters:
 *              command         The command to run
 *              ignore_error    Should we abort on error?
 *
 *      Global variables used:
 *              filter_stderr   If -X is on we redirect stderr
 */
int
doexec(register wchar_t *command, register Boolean ignore_error, char *stdout_fi
{
        int                     arg_count = 5;
        char                    **argv;
        int                     length;
        char                    nice_prio_buf[MAXPATHLEN];
        register char           **p;
        wchar_t                 *q;
        register wchar_t        *t;
        char                    *tmp_mbs_buffer;

        /*
         * Only prepend the /usr/bin/nice command to the original command
         * if the nice priority, nice_prio, is NOT zero (0).
         * Nice priorities can be a positive or a negative number.
         */
        if (nice_prio != 0) {
                arg_count += 2;
        }
        for (t = command; *t != (int) nul_char; t++) {
                if (iswspace(*t)) {
                        arg_count++;
                }
        }
        argv = (char **)alloca(arg_count * (sizeof(char *)));
        /*
         * Reserve argv[0] for sh in case of exec_vp failure.
         * Don’t worry about prepending /usr/bin/nice command to argv[0].
         * In fact, doing it may cause the sh command to fail!
         */
        p = &argv[1];
        if ((length = wcslen(command)) >= MAXPATHLEN) {
        if ((length = wslen(command)) >= MAXPATHLEN) {
                tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
                (void) wcstombs(tmp_mbs_buffer, command, (length * MB_LEN_MAX) +
                argv[0] = strdup(tmp_mbs_buffer);
                retmem_mb(tmp_mbs_buffer);
        } else {
                WCSTOMBS(mbs_buffer, command);
                argv[0] = strdup(mbs_buffer);
        }

        if (nice_prio != 0) {
                *p++ = strdup("/usr/bin/nice");
                (void) sprintf(nice_prio_buf, "-%d", nice_prio);
                *p++ = strdup(nice_prio_buf);
        }
        /* Build list of argument words. */
        for (t = command; *t;) {
                if (p >= &argv[arg_count]) {
                        /* This should never happen, right? */
                        WCSTOMBS(mbs_buffer, command);
                        fatal_mksh(gettext("Command ‘%s’ has more than %d argume
                              mbs_buffer,
                              arg_count);
                }
                q = t;
                while (!iswspace(*t) && (*t != (int) nul_char)) {
                        t++;
                }
                if (*t) {
                        for (*t++ = (int) nul_char; iswspace(*t); t++);
                }
                if ((length = wcslen(q)) >= MAXPATHLEN) {
                if ((length = wslen(q)) >= MAXPATHLEN) {
                        tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
                        (void) wcstombs(tmp_mbs_buffer, q, (length * MB_LEN_MAX)
                        *p++ = strdup(tmp_mbs_buffer);
                        retmem_mb(tmp_mbs_buffer);
                } else {
                        WCSTOMBS(mbs_buffer, q);
                        *p++ = strdup(mbs_buffer);
                }
        }
        *p = NULL;

        /* Then exec the command with that argument list. */
        (void) fflush(stdout);
        if ((childPid = fork()) == 0) {
                enable_interrupt((void (*) (int)) SIG_DFL);
#if 0
                if (filter_stderr) {
```

```
                        redirect_stderr();
                }
#endif
                (void) exec_vp(argv[1], argv, environ, ignore_error, vroot_path)
                fatal_mksh(gettext("Cannot load command ‘%s’: %s"), argv[1], err
        }
        if (childPid  == -1) {
                fatal_mksh(gettext("fork failed: %s"),
                      errmsg(errno));
        }
        for (int i = 0; argv[i] != NULL; i++) {
                retmem_mb(argv[i]);
        }
        return childPid;
}
```

______unchanged_portion_omitted_

****
    2182 Fri May 22 11:19:48 2015
new/usr/src/cmd/make/lib/mksh/i18n.cc
make: use the more modern wchar routines, not widec.h
****
______***unchanged_portion_omitted_***

```
/*
 *      get_char_semantics_entry(ch)
 *
 *      Return value:
 *              The slot number in the array for special make chars,
 *              else the slot number of the last array entry.
 *
 *      Parameters:
 *              ch              The wide character
 *
 *      Global variables used:
 *              char_semantics_char[]   array of special wchar_t chars
 *                                      "&*@‘\\|[]:$=!>-\n#()%?;^<’\""
 */
int
get_char_semantics_entry(wchar_t ch)
{
        wchar_t         *char_sem_char;

        char_sem_char = (wchar_t *) wcschr(char_semantics_char, ch);
        char_sem_char = (wchar_t *) wschr(char_semantics_char, ch);
        if (char_sem_char == NULL) {
                /*
                 * Return the integer entry for the last slot,
                 * whose content is empty.
                 */
                return (CHAR_SEMANTICS_ENTRIES - 1);
        } else {
                return (char_sem_char - char_semantics_char);
        }
}
```

______***unchanged_portion_omitted_***

****
36552 Fri May 22 11:19:49 2015
new/usr/src/cmd/make/lib/mksh/macro.cc
make: use the more modern wchar routines, not widec.h
****

```
/*
 * 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 2006 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */


/*
 *      macro.cc
 *
 *      Handle expansion of make macros
 */

/*
 * Included files
 */
#include <mksh/dosys.h>         /* sh_command2string() */
#include <mksh/i18n.h>          /* get_char_semantics_value() */
#include <mksh/macro.h>
#include <mksh/misc.h>          /* retmem() */
#include <mksh/read.h>          /* get_next_block_fn() */

#include <widec.h>
#include <libintl.h>

/*
 * File table of contents
 */
static void     add_macro_to_global_list(Name macro_to_add);
static void     expand_value_with_daemon(Name, register Property macro, register

static void     init_arch_macros(void);
static void     init_mach_macros(void);
static Boolean  init_arch_done = false;
static Boolean  init_mach_done = false;


long env_alloc_num = 0;
long env_alloc_bytes = 0;

/*
 *      getvar(name)
```

```
 *
 *      Return expanded value of macro.
 *
 *      Return value:
 *                              The expanded value of the macro
 *
 *      Parameters:
 *              name            The name of the macro we want the value for
 *
 *      Global variables used:
 */
Name
getvar(register Name name)
{
        String_rec              destination;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        register Name           result;

        if ((name == host_arch) || (name == target_arch)) {
                if (!init_arch_done) {
                        init_arch_done = true;
                        init_arch_macros();
                }
        }
        if ((name == host_mach) || (name == target_mach)) {
                if (!init_mach_done) {
                        init_mach_done = true;
                        init_mach_macros();
                }
        }

        INIT_STRING_FROM_STACK(destination, buffer);
        expand_value(maybe_append_prop(name, macro_prop)->body.macro.value,
                     &destination,
                     false);
        result = GETNAME(destination.buffer.start, FIND_LENGTH);
        if (destination.free_after_use) {
                retmem(destination.buffer.start);
        }
        return result;
}

/*
 *      expand_value(value, destination, cmd)
 *
 *      Recursively expands all macros in the string value.
 *      destination is where the expanded value should be appended.
 *
 *      Parameters:
 *              value           The value we are expanding
 *              destination     Where to deposit the expansion
 *              cmd             If we are evaluating a command line we
 *                              turn \ quoting off
 *
 *      Global variables used:
 */
void
expand_value(Name value, register String destination, Boolean cmd)
{
        Source_rec              sourceb;
        register Source         source = &sourceb;
        register wchar_t        *source_p = NULL;
        register wchar_t        *source_end = NULL;
        wchar_t                 *block_start = NULL;
        int                     quote_seen = 0;
        if (value == NULL) {
                /*
                 * Make sure to get a string allocated even if it
                 * will be empty.
                 */
                MBSTOWCS(wcs_buffer, "");
                append_string(wcs_buffer, destination, FIND_LENGTH);
                destination->text.end = destination->text.p;
                return;
        }
        if (!value->dollar) {
                /*
                 * If the value we are expanding does not contain
                 * any $, we don't have to parse it.
                 */
                APPEND_NAME(value,
                        destination,
                        (int) value->hash.length
                );
                destination->text.end = destination->text.p;
                return;
        }

        if (value->being_expanded) {
                fatal_reader_mksh(gettext("Loop detected when expanding macro va
                             value->string_mb);
        }
        value->being_expanded = true;
        /* Setup the structure we read from */
        Wstring vals(value);
        sourceb.string.text.p = sourceb.string.buffer.start = wcsdup(vals.get_st
        sourceb.string.text.p = sourceb.string.buffer.start = wsdup(vals.get_str
        sourceb.string.free_after_use = true;
        sourceb.string.text.end =
          sourceb.string.buffer.end =
            sourceb.string.text.p + value->hash.length;
        sourceb.previous = NULL;
        sourceb.fd = -1;
        sourceb.inp_buf =
          sourceb.inp_buf_ptr =
            sourceb.inp_buf_end = NULL;
        sourceb.error_converting = false;
        /* Lift some pointers from the struct to local register variables */
        CACHE_SOURCE(0);
/* We parse the string in segments */
/* We read chars until we find a $, then we append what we have read so far */
/* (since last $ processing) to the destination. When we find a $ we call */
/* expand_macro() and let it expand that particular $ reference into dest */
        block_start = source_p;
        quote_seen = 0;
        for (; 1; source_p++) {
                switch (GET_CHAR()) {
                case backslash_char:
                        /* Quote $ in macro value */
                        if (!cmd) {
                                quote_seen = ~quote_seen;
                        }
                        continue;
                case dollar_char:
                        /* Save the plain string we found since */
                        /* start of string or previous $ */
                        if (quote_seen) {
                                append_string(block_start,
                                              destination,
                                              source_p - block_start - 1);
                                block_start = source_p;
```

```
                                break;
                        }
                        append_string(block_start,
                                      destination,
                                      source_p - block_start);
                        source->string.text.p = ++source_p;
                        UNCACHE_SOURCE();
                        /* Go expand the macro reference */
                        expand_macro(source, destination, sourceb.string.buffer.
                        CACHE_SOURCE(1);
                        block_start = source_p + 1;
                        break;
                case nul_char:
                        /* The string ran out. Get some more */
                        append_string(block_start,
                                      destination,
                                      source_p - block_start);
                        GET_NEXT_BLOCK_NOCHK(source);
                        if (source == NULL) {
                                destination->text.end = destination->text.p;
                                value->being_expanded = false;
                                return;
                        }
                        if (source->error_converting) {
                                fatal_reader_mksh("Internal error: Invalid byte
                        }
                        block_start = source_p;
                        source_p--;
                        continue;
                }
                quote_seen = 0;
        }
        retmem(sourceb.string.buffer.start);
}

/*
 *      expand_macro(source, destination, current_string, cmd)
 *
 *      Should be called with source->string.text.p pointing to
 *      the first char after the $ that starts a macro reference.
 *      source->string.text.p is returned pointing to the first char after
 *      the macro name.
 *      It will read the macro name, expanding any macros in it,
 *      and get the value. The value is then expanded.
 *      destination is a String that is filled in with the expanded macro.
 *      It may be passed in referencing a buffer to expand the macro into.
 *      Note that most expansions are done on demand, e.g. right
 *      before the command is executed and not while the file is
 *      being parsed.
 *
 *      Parameters:
 *              source          The source block that references the string
 *                              to expand
 *              destination     Where to put the result
 *              current_string  The string we are expanding, for error msg
 *              cmd             If we are evaluating a command line we
 *                              turn \ quoting off
 *
 *      Global variables used:
 *              funny           Vector of semantic tags for characters
 *              is_conditional  Set if a conditional macro is refd
 *              make_word_mentioned Set if the word "MAKE" is mentioned
 *              makefile_type   We deliver extra msg when reading makefiles
 *              query           The Name "?", compared against
 *              query_mentioned Set if the word "?" is mentioned
 */
```

```
void
expand_macro(register Source source, register String destination, wchar_t *curre
{
        static Name             make = (Name)NULL;
        static wchar_t          colon_sh[4];
        static wchar_t          colon_shell[7];
        String_rec              string;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        register wchar_t        *source_p = source->string.text.p;
        register wchar_t        *source_end = source->string.text.end;
        register int            closer = 0;
        wchar_t                 *block_start = (wchar_t *)NULL;
        int                     quote_seen = 0;
        register int            closer_level = 1;
        Name                    name = (Name)NULL;
        wchar_t                 *colon = (wchar_t *)NULL;
        wchar_t                 *percent = (wchar_t *)NULL;
        wchar_t                 *eq = (wchar_t *) NULL;
        Property                macro = NULL;
        wchar_t                 *p = (wchar_t*)NULL;
        String_rec              extracted;
        wchar_t                 extracted_string[MAXPATHLEN];
        wchar_t                 *left_head = NULL;
        wchar_t                 *left_tail = NULL;
        wchar_t                 *right_tail = NULL;
        int                     left_head_len = 0;
        int                     left_tail_len = 0;
        int                     tmp_len = 0;
        wchar_t                 *right_hand[128];
        int                     i = 0;
        enum {
                no_extract,
                dir_extract,
                file_extract
        }                       extraction = no_extract;
        enum {
                no_replace,
                suffix_replace,
                pattern_replace,
                sh_replace
        }                       replacement = no_replace;

        if (make == NULL) {
                MBSTOWCS(wcs_buffer, "MAKE");
                make = GETNAME(wcs_buffer, FIND_LENGTH);

                MBSTOWCS(colon_sh, ":sh");
                MBSTOWCS(colon_shell, ":shell");
        }

        right_hand[0] = NULL;

        /* First copy the (macro-expanded) macro name into string. */
        INIT_STRING_FROM_STACK(string, buffer);
recheck_first_char:
        /* Check the first char of the macro name to figure out what to do. */
        switch (GET_CHAR()) {
        case nul_char:
                GET_NEXT_BLOCK_NOCHK(source);
                if (source == NULL) {
                        WCSTOMBS(mbs_buffer, current_string);
                        fatal_reader_mksh(gettext("'$' at end of string '%s'"),
                                          mbs_buffer);
                }
                if (source->error_converting) {
                        fatal_reader_mksh("Internal error: Invalid byte sequence
```

```
                }
                goto recheck_first_char;
        case parenleft_char:
                /* Multi char name. */
                closer = (int) parenright_char;
                break;
        case braceleft_char:
                /* Multi char name. */
                closer = (int) braceright_char;
                break;
        case newline_char:
                fatal_reader_mksh(gettext("'$' at end of line"));
        default:
                /* Single char macro name. Just suck it up */
                append_char(*source_p, &string);
                source->string.text.p = source_p + 1;
                goto get_macro_value;
        }

        /* Handle multi-char macro names */
        block_start = ++source_p;
        quote_seen = 0;
        for (; 1; source_p++) {
                switch (GET_CHAR()) {
                case nul_char:
                        append_string(block_start,
                                      &string,
                                      source_p - block_start);
                        GET_NEXT_BLOCK_NOCHK(source);
                        if (source == NULL) {
                                if (current_string != NULL) {
                                        WCSTOMBS(mbs_buffer, current_string);
                                        fatal_reader_mksh(gettext("Unmatched '%c
                                                    closer ==
                                                    (int) braceright_char ?
                                                    (int) braceleft_char :
                                                    (int) parenleft_char,
                                                    mbs_buffer);
                                } else {
                                        fatal_reader_mksh(gettext("Premature EOF
                                }
                        }
                        if (source->error_converting) {
                                fatal_reader_mksh("Internal error: Invalid byte
                        }
                        block_start = source_p;
                        source_p--;
                        continue;
                case newline_char:
                        fatal_reader_mksh(gettext("Unmatched '%c' on line"),
                                     closer == (int) braceright_char ?
                                     (int) braceleft_char :
                                     (int) parenleft_char);
                case backslash_char:
                        /* Quote dollar in macro value. */
                        if (!cmd) {
                                quote_seen = ~quote_seen;
                        }
                        continue;
                case dollar_char:
                        /*
                         * Macro names may reference macros.
                         * This expands the value of such macros into the
                         * macro name string.
                         */
                        if (quote_seen) {
                                append_string(block_start,
                                              &string,
                                              source_p - block_start - 1);
                                block_start = source_p;
                                break;
                        }
                        append_string(block_start,
                                      &string,
                                      source_p - block_start);
                        source->string.text.p = ++source_p;
                        UNCACHE_SOURCE();
                        expand_macro(source, &string, current_string, cmd);
                        CACHE_SOURCE(0);
                        block_start = source_p;
                        source_p--;
                        break;
                case parenleft_char:
                        /* Allow nested pairs of () in the macro name. */
                        if (closer == (int) parenright_char) {
                                closer_level++;
                        }
                        break;
                case braceleft_char:
                        /* Allow nested pairs of {} in the macro name. */
                        if (closer == (int) braceright_char) {
                                closer_level++;
                        }
                        break;
                case parenright_char:
                case braceright_char:
                        /*
                         * End of the name. Save the string in the macro
                         * name string.
                         */
                        if ((*source_p == closer) && (--closer_level <= 0)) {
                                source->string.text.p = source_p + 1;
                                append_string(block_start,
                                              &string,
                                              source_p - block_start);
                                goto get_macro_value;
                        }
                        break;
                }
                quote_seen = 0;
        }
        /*
         * We got the macro name. We now inspect it to see if it
         * specifies any translations of the value.
         */
get_macro_value:
        name = NULL;
        /* First check if we have a $(@D) type translation. */
        if ((get_char_semantics_value(string.buffer.start[0]) &
             (int) special_macro_sem) &&
            (string.text.p - string.buffer.start >= 2) &&
            ((string.buffer.start[1] == 'D') ||
             (string.buffer.start[1] == 'F'))) {
                switch (string.buffer.start[1]) {
                case 'D':
                        extraction = dir_extract;
                        break;
                case 'F':
                        extraction = file_extract;
                        break;
                default:
                        WCSTOMBS(mbs_buffer, string.buffer.start);
```

```
                        fatal_reader_mksh(gettext("Illegal macro reference '%s'"
                                     mbs_buffer);
                }
                /* Internalize the macro name using the first char only. */
                name = GETNAME(string.buffer.start, 1);
                (void) wcscpy(string.buffer.start, string.buffer.start + 2);
                (void) wscpy(string.buffer.start, string.buffer.start + 2);
        }
        /* Check for other kinds of translations. */
        if ((colon = (wchar_t *) wcschr(string.buffer.start,
        if ((colon = (wchar_t *) wschr(string.buffer.start,
                                       (int) colon_char)) != NULL) {
                /*
                 * We have a $(FOO:.c=.o) type translation.
                 * Get the name of the macro proper.
                 */
                if (name == NULL) {
                        name = GETNAME(string.buffer.start,
                                       colon - string.buffer.start);
                }
                /* Pickup all the translations. */
                if (IS_WEQUAL(colon, colon_sh) || IS_WEQUAL(colon, colon_shell))
                        replacement = sh_replace;
                } else if ((svr4) ||
                           ((percent = (wchar_t *) wcschr(colon + 1,
                           ((percent = (wchar_t *) wschr(colon + 1,
                                                         (int) percent_char)) ==
                        while (colon != NULL) {
                                if ((eq = (wchar_t *) wcschr(colon + 1,
                                if ((eq = (wchar_t *) wschr(colon + 1,
                                                            (int) equal_char)) =
                                        fatal_reader_mksh(gettext("= missing fro
                                }
                                left_tail_len = eq - colon - 1;
                                if(left_tail) {
                                        retmem(left_tail);
                                }
                                left_tail = ALLOC_WC(left_tail_len + 1);
                                (void) wcsncpy(left_tail,
                                (void) wsncpy(left_tail,
                                              colon + 1,
                                              eq - colon - 1);
                                left_tail[eq - colon - 1] = (int) nul_char;
                                replacement = suffix_replace;
                                if ((colon = (wchar_t *) wcschr(eq + 1,
                                if ((colon = (wchar_t *) wschr(eq + 1,
                                                               (int) colon_char)
                                        tmp_len = colon - eq;
                                        if(right_tail) {
                                                retmem(right_tail);
                                        }
                                        right_tail = ALLOC_WC(tmp_len);
                                        (void) wcsncpy(right_tail,
                                        (void) wsncpy(right_tail,
                                                      eq + 1,
                                                      colon - eq - 1);
                                        right_tail[colon - eq - 1] =
                                          (int) nul_char;
                                } else {
                                        if(right_tail) {
                                                retmem(right_tail);
                                        }
                                        right_tail = ALLOC_WC(wcslen(eq) + 1);
                                        (void) wcscpy(right_tail, eq + 1);
                                        right_tail = ALLOC_WC(wslen(eq) + 1);
                                        (void) wscpy(right_tail, eq + 1);
                                }
                        }
                } else {
                        if ((eq = (wchar_t *) wcschr(colon + 1,
                        if ((eq = (wchar_t *) wschr(colon + 1,
                                                    (int) equal_char)) == NULL)
                                fatal_reader_mksh(gettext("= missing from replac
                        }
                        if ((percent = (wchar_t *) wcschr(colon + 1,
                        if ((percent = (wchar_t *) wschr(colon + 1,
                                                         (int) percent_char)) ==
                                fatal_reader_mksh(gettext("%% missing from repla
                        }
                        if (eq < percent) {
                                fatal_reader_mksh(gettext("%% missing from repla
                        }

                        if (percent > (colon + 1)) {
                                tmp_len = percent - colon;
                                if(left_head) {
                                        retmem(left_head);
                                }
                                left_head = ALLOC_WC(tmp_len);
                                (void) wcsncpy(left_head,
                                (void) wsncpy(left_head,
                                              colon + 1,
                                              percent - colon - 1);
                                left_head[percent-colon-1] = (int) nul_char;
                                left_head_len = percent-colon-1;
                        } else {
                                left_head = NULL;
                                left_head_len = 0;
                        }

                        if (eq > percent+1) {
                                tmp_len = eq - percent;
                                if(left_tail) {
                                        retmem(left_tail);
                                }
                                left_tail = ALLOC_WC(tmp_len);
                                (void) wcsncpy(left_tail,
                                (void) wsncpy(left_tail,
                                              percent + 1,
                                              eq - percent - 1);
                                left_tail[eq-percent-1] = (int) nul_char;
                                left_tail_len = eq-percent-1;
                        } else {
                                left_tail = NULL;
                                left_tail_len = 0;
                        }

                        if ((percent = (wchar_t *) wcschr(++eq,
                        if ((percent = (wchar_t *) wschr(++eq,
                                                         (int) percent_char)) ==

                                right_hand[0] = ALLOC_WC(wcslen(eq) + 1);
                                right_hand[0] = ALLOC_WC(wslen(eq) + 1);
                                right_hand[1] = NULL;
                                (void) wcscpy(right_hand[0], eq);
                                (void) wscpy(right_hand[0], eq);
                        } else {
                                i = 0;
                                do {
                                        right_hand[i] = ALLOC_WC(percent-eq+1);
                                        (void) wcsncpy(right_hand[i],
                                        (void) wsncpy(right_hand[i],
```

```
                                                      eq,
                                                      percent - eq);
                                        right_hand[i][percent-eq] =
                                          (int) nul_char;
                                        if (i++ >= VSIZEOF(right_hand)) {
                                                fatal_mksh(gettext("Too many %%
                                        }
                                        eq = percent + 1;
                                        if (eq[0] == (int) nul_char) {
                                                MBSTOWCS(wcs_buffer, "");
                                                right_hand[i] = (wchar_t *) wcsd
                                                right_hand[i] = (wchar_t *) wsdu
                                                i++;
                                                break;
                                        }
                                } while ((percent = (wchar_t *) wcschr(eq, (int)
                                } while ((percent = (wchar_t *) wschr(eq, (int)
                                if (eq[0] != (int) nul_char) {
                                        right_hand[i] = ALLOC_WC(wcslen(eq) + 1)
                                        (void) wcscpy(right_hand[i], eq);
                                        right_hand[i] = ALLOC_WC(wslen(eq) + 1);
                                        (void) wscpy(right_hand[i], eq);
                                        i++;
                                }
                                right_hand[i] = NULL;
                        }
                        replacement = pattern_replace;
                }
        }
        if (name == NULL) {
                /*
                 * No translations found.
                 * Use the whole string as the macro name.
                 */
                name = GETNAME(string.buffer.start,
                               string.text.p - string.buffer.start);
        }
        if (string.free_after_use) {
                retmem(string.buffer.start);
        }
        if (name == make) {
                make_word_mentioned = true;
        }
        if (name == query) {
                query_mentioned = true;
        }
        if ((name == host_arch) || (name == target_arch)) {
                if (!init_arch_done) {
                        init_arch_done = true;
                        init_arch_macros();
                }
        }
        if ((name == host_mach) || (name == target_mach)) {
                if (!init_mach_done) {
                        init_mach_done = true;
                        init_mach_macros();
                }
        }
        /* Get the macro value. */
        macro = get_prop(name->prop, macro_prop);
        if ((macro != NULL) && macro->body.macro.is_conditional) {
                conditional_macro_used = true;
                /*
                 * Add this conditional macro to the beginning of the
                 * global list.
                 */
                add_macro_to_global_list(name);
                if (makefile_type == reading_makefile) {
                        warning_mksh(gettext("Conditional macro ‘%s’ referenced
                                        name->string_mb, file_being_read, line_n
                }
        }
        /* Macro name read and parsed. Expand the value. */
        if ((macro == NULL) || (macro->body.macro.value == NULL)) {
                /* If the value is empty, we just get out of here. */
                goto exit;
        }
        if (replacement == sh_replace) {
                /* If we should do a :sh transform, we expand the command
                 * and process it.
                 */
                INIT_STRING_FROM_STACK(string, buffer);
                /* Expand the value into a local string buffer and run cmd. */
                expand_value_with_daemon(name, macro, &string, cmd);
                sh_command2string(&string, destination);
        } else if ((replacement != no_replace) || (extraction != no_extract)) {
                /*
                 * If there were any transforms specified in the macro
                 * name, we deal with them here.
                 */
                INIT_STRING_FROM_STACK(string, buffer);
                /* Expand the value into a local string buffer. */
                expand_value_with_daemon(name, macro, &string, cmd);
                /* Scan the expanded string. */
                p = string.buffer.start;
                while (*p != (int) nul_char) {
                        wchar_t         chr;

                        /*
                         * First skip over any white space and append
                         * that to the destination string.
                         */
                        block_start = p;
                        while ((*p != (int) nul_char) && iswspace(*p)) {
                                p++;
                        }
                        append_string(block_start,
                                      destination,
                                      p - block_start);
                        /* Then find the end of the next word. */
                        block_start = p;
                        while ((*p != (int) nul_char) && !iswspace(*p)) {
                                p++;
                        }
                        /* If we cant find another word we are done */
                        if (block_start == p) {
                                break;
                        }
                        /* Then apply the transforms to the word */
                        INIT_STRING_FROM_STACK(extracted, extracted_string);
                        switch (extraction) {
                        case dir_extract:
                                /*
                                 * $(@D) type transform. Extract the
                                 * path from the word. Deliver "." if
                                 * none is found.
                                 */
                                if (p != NULL) {
                                        chr = *p;
                                        *p = (int) nul_char;
                                }
                                eq = (wchar_t *) wcsrchr(block_start, (int) slas
```

```
                                eq = (wchar_t *) wsrchr(block_start, (int) slash
                                if (p != NULL) {
                                        *p = chr;
                                }
                                if ((eq == NULL) || (eq > p)) {
                                        MBSTOWCS(wcs_buffer, ".");
                                        append_string(wcs_buffer, &extracted, 1)
                                } else {
                                        append_string(block_start,
                                                      &extracted,
                                                      eq - block_start);
                                }
                                break;
                        case file_extract:
                                /*
                                 * $(@F) type transform. Remove the path
                                 * from the word if any.
                                 */
                                if (p != NULL) {
                                        chr = *p;
                                        *p = (int) nul_char;
                                }
                                eq = (wchar_t *) wcsrchr(block_start, (int) slas
                                eq = (wchar_t *) wsrchr(block_start, (int) slash
                                if (p != NULL) {
                                        *p = chr;
                                }
                                if ((eq == NULL) || (eq > p)) {
                                        append_string(block_start,
                                                      &extracted,
                                                      p - block_start);
                                } else {
                                        append_string(eq + 1,
                                                      &extracted,
                                                      p - eq - 1);
                                }
                                break;
                        case no_extract:
                                append_string(block_start,
                                              &extracted,
                                              p - block_start);
                                break;
                        }
                        switch (replacement) {
                        case suffix_replace:
                                /*
                                 * $(FOO:.o=.c) type transform.
                                 * Maybe replace the tail of the word.
                                 */
                                if (((extracted.text.p -
                                      extracted.buffer.start) >=
                                     left_tail_len) &&
                                    IS_WEQUALN(extracted.text.p - left_tail_len,
                                              left_tail,
                                              left_tail_len)) {
                                        append_string(extracted.buffer.start,
                                                      destination,
                                                      (extracted.text.p -
                                                       extracted.buffer.start)
                                                      - left_tail_len);
                                        append_string(right_tail,
                                                      destination,
                                                      FIND_LENGTH);
                                } else {
                                        append_string(extracted.buffer.start,
                                                      destination,
                                                      FIND_LENGTH);
                                }
                                break;
                        case pattern_replace:
                                /* $(X:a%b=c%d) type transform. */
                                if (((extracted.text.p -
                                      extracted.buffer.start) >=
                                     left_head_len+left_tail_len) &&
                                    IS_WEQUALN(left_head,
                                              extracted.buffer.start,
                                              left_head_len) &&
                                    IS_WEQUALN(left_tail,
                                              extracted.text.p - left_tail_len,
                                              left_tail_len)) {
                                        i = 0;
                                        while (right_hand[i] != NULL) {
                                                append_string(right_hand[i],
                                                              destination,
                                                              FIND_LENGTH);
                                                i++;
                                                if (right_hand[i] != NULL) {
                                                        append_string(extracted.
                                                                      start +
                                                                      left_head_
                                                                      destinatio
                                                                      (extracted
                                                }
                                        }
                                } else {
                                        append_string(extracted.buffer.start,
                                                      destination,
                                                      FIND_LENGTH);
                                }
                                break;
                        case no_replace:
                                append_string(extracted.buffer.start,
                                              destination,
                                              FIND_LENGTH);
                                break;
                        case sh_replace:
                                break;
                            }
                }
                if (string.free_after_use) {
                        retmem(string.buffer.start);
                }
        } else {
                /*
                 * This is for the case when the macro name did not
                 * specify transforms.
                 */
                if (!strncmp(name->string_mb, "GET", 3)) {
                        dollarget_seen = true;
                }
                dollarless_flag = false;
                if (!strncmp(name->string_mb, "<", 1) &&
                    dollarget_seen) {
                        dollarless_flag = true;
                        dollarget_seen = false;
                }
                expand_value_with_daemon(name, macro, destination, cmd);
        }
exit:
        if(left_tail) {
                retmem(left_tail);
        }
```

```
        if(right_tail) {
                retmem(right_tail);
        }
        if(left_head) {
                retmem(left_head);
        }
        i = 0;
        while (right_hand[i] != NULL) {
                retmem(right_hand[i]);
                i++;
        }
        *destination->text.p = (int) nul_char;
        destination->text.end = destination->text.p;
}
```

______***unchanged_portion_omitted_***

```
/*
 *      init_arch_macros(void)
 *
 *      Set the magic macros TARGET_ARCH, HOST_ARCH,
 *
 *      Parameters:
 *
 *      Global variables used:
 *                              host_arch   Property for magic macro HOST_ARCH
 *                              target_arch Property for magic macro TARGET_ARCH
 *
 *      Return value:
 *                              The function does not return a value, but can
 *                              call fatal() in case of error.
 */
static void
init_arch_macros(void)
{
        String_rec      result_string;
        wchar_t         wc_buf[STRING_BUFFER_LENGTH];
        char            mb_buf[STRING_BUFFER_LENGTH];
        FILE            *pipe;
        Name            value;
        int             set_host, set_target;
        const char      *mach_command = "/bin/mach";

        set_host = (get_prop(host_arch->prop, macro_prop) == NULL);
        set_target = (get_prop(target_arch->prop, macro_prop) == NULL);

        if (set_host || set_target) {
                INIT_STRING_FROM_STACK(result_string, wc_buf);
                append_char((int) hyphen_char, &result_string);

                if ((pipe = popen(mach_command, "r")) == NULL) {
                        fatal_mksh(gettext("Execute of %s failed"), mach_command
                }
                while (fgets(mb_buf, sizeof(mb_buf), pipe) != NULL) {
                        MBSTOWCS(wcs_buffer, mb_buf);
                        append_string(wcs_buffer, &result_string, wcslen(wcs_buf
                        append_string(wcs_buffer, &result_string, wslen(wcs_buff
                }
                if (pclose(pipe) != 0) {
                        fatal_mksh(gettext("Execute of %s failed"), mach_command
                }

                value = GETNAME(result_string.buffer.start, wcslen(result_string
                value = GETNAME(result_string.buffer.start, wslen(result_string.

                if (set_host) {
                        (void) setvar_daemon(host_arch, value, false, no_daemon,
                }
                if (set_target) {
                        (void) setvar_daemon(target_arch, value, false, no_daemo
                }
        }
}

/*
 *      init_mach_macros(void)
 *
 *      Set the magic macros TARGET_MACH, HOST_MACH,
 *
 *      Parameters:
 *
 *      Global variables used:
 *                              host_mach   Property for magic macro HOST_MACH
 *                              target_mach Property for magic macro TARGET_MACH
 *
 *      Return value:
 *                              The function does not return a value, but can
 *                              call fatal() in case of error.
 */
static void
init_mach_macros(void)
{
        String_rec      result_string;
        wchar_t         wc_buf[STRING_BUFFER_LENGTH];
        char            mb_buf[STRING_BUFFER_LENGTH];
        FILE            *pipe;
        Name            value;
        int             set_host, set_target;
        const char      *arch_command = "/bin/arch";

        set_host = (get_prop(host_mach->prop, macro_prop) == NULL);
        set_target = (get_prop(target_mach->prop, macro_prop) == NULL);

        if (set_host || set_target) {
                INIT_STRING_FROM_STACK(result_string, wc_buf);
                append_char((int) hyphen_char, &result_string);

                if ((pipe = popen(arch_command, "r")) == NULL) {
                        fatal_mksh(gettext("Execute of %s failed"), arch_command
                }
                while (fgets(mb_buf, sizeof(mb_buf), pipe) != NULL) {
                        MBSTOWCS(wcs_buffer, mb_buf);
                        append_string(wcs_buffer, &result_string, wcslen(wcs_buf
                        append_string(wcs_buffer, &result_string, wslen(wcs_buff
                }
                if (pclose(pipe) != 0) {
                        fatal_mksh(gettext("Execute of %s failed"), arch_command
                }

                value = GETNAME(result_string.buffer.start, wcslen(result_string
                value = GETNAME(result_string.buffer.start, wslen(result_string.

                if (set_host) {
                        (void) setvar_daemon(host_mach, value, false, no_daemon,
                }
                if (set_target) {
                        (void) setvar_daemon(target_mach, value, false, no_daemo
                }
        }
}
```

______unchanged_portion_omitted_

```
/*
 * We use a permanent buffer to reset SUNPRO_DEPENDENCIES value.
 */
char    *sunpro_dependencies_buf = NULL;
char    *sunpro_dependencies_oldbuf = NULL;
int     sunpro_dependencies_buf_size = 0;

/*
 *      setvar_daemon(name, value, append, daemon, strip_trailing_spaces)
 *
 *      Set a macro value, possibly supplying a daemon to be used
 *      when referencing the value.
 *
 *      Return value:
 *                              The property block with the new value
 *
 *      Parameters:
 *              name            Name of the macro to set
 *              value           The value to set
 *              append          Should we reset or append to the current value?
 *              daemon          Special treatment when reading the value
 *              strip_trailing_spaces from the end of value->string
 *              debug_level     Indicates how much tracing we should do
 *
 *      Global variables used:
 *              makefile_type   Used to check if we should enforce read only
 *              path_name       The Name "PATH", compared against
 *              virtual_root    The Name "VIRTUAL_ROOT", compared against
 *              vpath_defined   Set if the macro VPATH is set
 *              vpath_name      The Name "VPATH", compared against
 *              envvar          A list of environment vars with $ in value
 */
Property
setvar_daemon(register Name name, register Name value, Boolean append, Daemon da
{
        register Property       macro = maybe_append_prop(name, macro_prop);
        register Property       macro_apx = get_prop(name->prop, macro_append_pr
        int                     length = 0;
        String_rec              destination;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        register Chain          chain;
        Name                    val;
        wchar_t                 *val_string = (wchar_t*)NULL;
        Wstring                 wcb;


        if ((makefile_type != reading_nothing) &&
            macro->body.macro.read_only) {
                return macro;
        }
        /* Strip spaces from the end of the value */
        if (daemon == no_daemon) {
                if(value != NULL) {
                        wcb.init(value);
                        length = wcb.length();
                        val_string = wcb.get_string();
                }
                if ((length > 0) && iswspace(val_string[length-1])) {
                        INIT_STRING_FROM_STACK(destination, buffer);
                        buffer[0] = 0;
                        append_string(val_string, &destination, length);
                        if (strip_trailing_spaces) {
                                while ((length > 0) &&
                                       iswspace(destination.buffer.start[length-
                                        destination.buffer.start[--length] = 0;
                                }
                        }
                        value = GETNAME(destination.buffer.start, FIND_LENGTH);
                }
        }

        if(macro_apx != NULL) {
                val = macro_apx->body.macro_appendix.value;
        } else {
                val = macro->body.macro.value;
        }

        if (append) {
                /*
                 * If we are appending, we just tack the new value after
                 * the old one with a space in between.
                 */
                INIT_STRING_FROM_STACK(destination, buffer);
                buffer[0] = 0;
                if ((macro != NULL) && (val != NULL)) {
                        APPEND_NAME(val,
                                      &destination,
                                      (int) val->hash.length);
                        if (value != NULL) {
                                wcb.init(value);
                                if(wcb.length() > 0) {
                                        MBTOWC(wcs_buffer, " ");
                                        append_char(wcs_buffer[0], &destination)
                                }
                        }
                }
                if (value != NULL) {
                        APPEND_NAME(value,
                                      &destination,
                                      (int) value->hash.length);
                }
                value = GETNAME(destination.buffer.start, FIND_LENGTH);
                wcb.init(value);
                if (destination.free_after_use) {
                        retmem(destination.buffer.start);
                }
        }

        /* Debugging trace */
        if (debug_level > 1) {
                if (value != NULL) {
                        switch (daemon) {
                        case chain_daemon:
                                (void) printf("%s =", name->string_mb);
                                for (chain = (Chain) value;
                                     chain != NULL;
                                     chain = chain->next) {
                                        (void) printf(" %s", chain->name->string
                                }
                                (void) printf("\n");
                                break;
                        case no_daemon:
                                (void) printf("%s= %s\n",
                                              name->string_mb,
                                              value->string_mb);
                                break;
                        }
                } else {
                        (void) printf("%s =\n", name->string_mb);
                }
        }
        /* Set the new values in the macro property block */
/**/
```

```
        if(macro_apx != NULL) {
                macro_apx->body.macro_appendix.value = value;
                INIT_STRING_FROM_STACK(destination, buffer);
                buffer[0] = 0;
                if (value != NULL) {
                        APPEND_NAME(value,
                                      &destination,
                                      (int) value->hash.length);
                        if (macro_apx->body.macro_appendix.value_to_append != NU
                                MBTOWC(wcs_buffer, " ");
                                append_char(wcs_buffer[0], &destination);
                        }
                }
                if (macro_apx->body.macro_appendix.value_to_append != NULL) {
                        APPEND_NAME(macro_apx->body.macro_appendix.value_to_appe
                                      &destination,
                                      (int) macro_apx->body.macro_appendix.value
                }
                value = GETNAME(destination.buffer.start, FIND_LENGTH);
                if (destination.free_after_use) {
                        retmem(destination.buffer.start);
                }
        }
/**/
        macro->body.macro.value = value;
        macro->body.macro.daemon = daemon;
        /*
         * If the user changes the VIRTUAL_ROOT, we need to flush
         * the vroot package cache.
         */
        if (name == path_name) {
                flush_path_cache();
        }
        if (name == virtual_root) {
                flush_vroot_cache();
        }
        /* If this sets the VPATH we remember that */
        if ((name == vpath_name) &&
            (value != NULL) &&
            (value->hash.length > 0)) {
                vpath_defined = true;
        }
        /*
         * For environment variables we also set the
         * environment value each time.
         */
        if (macro->body.macro.exported) {
                static char     *env;

                if (!reading_environment && (value != NULL)) {
                        Envvar  p;

                        for (p = envvar; p != NULL; p = p->next) {
                                if (p->name == name) {
                                        p->value = value;
                                        p->already_put = false;
                                        goto found_it;
                                }
                        }
                        p = ALLOC(Envvar);
                        p->name = name;
                        p->value = value;
                        p->next = envvar;
                        p->env_string = NULL;
                        p->already_put = false;
                        envvar = p;
```

```
found_it:;
                } if (reading_environment || (value == NULL) || !value->dollar)
                        length = 2 + strlen(name->string_mb);
                        if (value != NULL) {
                                length += strlen(value->string_mb);
                        }
                        Property env_prop = maybe_append_prop(name, env_mem_prop
                        /*
                         * We use a permanent buffer to reset SUNPRO_DEPENDENCIE
                         */
                        if (!strncmp(name->string_mb, "SUNPRO_DEPENDENCIES", 19)
                                if (length >= sunpro_dependencies_buf_size) {
                                        sunpro_dependencies_buf_size=length*2;
                                        if (sunpro_dependencies_buf_size < 4096)
                                                sunpro_dependencies_buf_size = 4
                                        if (sunpro_dependencies_buf)
                                                sunpro_dependencies_oldbuf = sun
                                        sunpro_dependencies_buf=getmem(sunpro_de
                                }
                                env = sunpro_dependencies_buf;
                        } else {
                                env = getmem(length);
                        }
                        env_alloc_num++;
                        env_alloc_bytes += length;
                        (void) sprintf(env,
                                       "%s=%s",
                                       name->string_mb,
                                       value == NULL ?
                                         "" : value->string_mb);
                        (void) putenv(env);
                        env_prop->body.env_mem.value = env;
                        if (sunpro_dependencies_oldbuf) {
                                /* Return old buffer */
                                retmem_mb(sunpro_dependencies_oldbuf);
                                sunpro_dependencies_oldbuf = NULL;
                        }
                }
        }
        if (name == target_arch) {
                Name            ha = getvar(host_arch);
                Name            ta = getvar(target_arch);
                Name            vr = getvar(virtual_root);
                int             length;
                wchar_t         *new_value;
                wchar_t         *old_vr;
                Boolean         new_value_allocated = false;

                Wstring         ha_str(ha);
                Wstring         ta_str(ta);
                Wstring         vr_str(vr);

                wchar_t * wcb_ha = ha_str.get_string();
                wchar_t * wcb_ta = ta_str.get_string();
                wchar_t * wcb_vr = vr_str.get_string();

                length = 32 +
                  wcslen(wcb_ha) +
                    wcslen(wcb_ta) +
                      wcslen(wcb_vr);
                  wslen(wcb_ha) +
                    wslen(wcb_ta) +
                      wslen(wcb_vr);
                old_vr = wcb_vr;
                MBSTOWCS(wcs_buffer, "/usr/arch/");
                if (IS_WEQUALN(old_vr,
```

```
                               wcs_buffer,
                               wcslen(wcs_buffer))) {
                        old_vr = (wchar_t *) wcschr(old_vr, (int) colon_char) +
                               wslen(wcs_buffer))) {
                        old_vr = (wchar_t *) wschr(old_vr, (int) colon_char) + 1
                }
                if ( (ha == ta) || (wcslen(wcb_ta) == 0) ) {
                if ( (ha == ta) || (wslen(wcb_ta) == 0) ) {
                        new_value = old_vr;
                } else {
                        new_value = ALLOC_WC(length);
                        new_value_allocated = true;
                        WCSTOMBS(mbs_buffer, old_vr);
                        (void) swprintf(new_value, length * SIZEOFWCHAR_T,
                                        L"/usr/arch/%s/%s:%s",
                        (void) wsprintf(new_value,
                                        "/usr/arch/%s/%s:%s",
                                        ha->string_mb + 1,
                                        ta->string_mb + 1,
                                        mbs_buffer);
                }
                if (new_value[0] != 0) {
                        (void) setvar_daemon(virtual_root,
                                             GETNAME(new_value, FIND_LENGTH),
                                             false,
                                             no_daemon,
                                             true,
                                             debug_level);
                }
                if (new_value_allocated) {
                        retmem(new_value);
                }
        }
        return macro;
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   24135 Fri May 22 11:19:49 2015
new/usr/src/cmd/make/lib/mksh/misc.cc
make: use the more modern wchar routines, not widec.h
**********************************************************
______unchanged_portion_omitted_

/*
 *      getname_fn(name, len, dont_enter)
 *
 *      Hash a name string to the corresponding nameblock.
 *
 *      Return value:
 *                              The Name block for the string
 *
 *      Parameters:
 *              name            The string we want to internalize
 *              len             The length of that string
 *              dont_enter      Don't enter the name if it does not exist
 *
 *      Global variables used:
 *              funny           The vector of semantic tags for characters
 *              hashtab         The hashtable used for the nametable
 */
Name
getname_fn(wchar_t *name, register int len, register Boolean dont_enter, registe
{
        register int            length;
        register wchar_t        *cap = name;
        register Name           np;
        static Name_rec         empty_Name;
        char                    *tmp_mbs_buffer = NULL;
        char                    *mbs_name = mbs_buffer;

        /*
         * First figure out how long the string is.
         * If the len argument is -1 we count the chars here.
         */
        if (len == FIND_LENGTH) {
                length = wcslen(name);
                length = wslen(name);
        } else {
                length = len;
        }

        Wstring ws;
        ws.init(name, length);
        if (length >= MAXPATHLEN) {
                mbs_name = tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
        }
        (void) wcstombs(mbs_name, ws.get_string(), (length * MB_LEN_MAX) + 1);

        /* Look for the string */
        if (dont_enter || (foundp != 0)) {
                np = hashtab.lookup(mbs_name);
                if (foundp != 0) {
                        *foundp = (np != 0) ? true : false;
                }
                if ((np != 0) || dont_enter) {
                        if(tmp_mbs_buffer != NULL) {
                                retmem_mb(tmp_mbs_buffer);
                        }
                        return np;
                } else {
                        np = ALLOC(Name);
                }
```

```
        } else {
                Boolean found;
                np = hashtab.insert(mbs_name, found);
                if (found) {
                        if(tmp_mbs_buffer != NULL) {
                                retmem_mb(tmp_mbs_buffer);
                        }
                        return np;
                }
        }
        getname_struct_count += sizeof(struct _Name);
        *np = empty_Name;

        np->string_mb = strdup(mbs_name);
        if(tmp_mbs_buffer != NULL) {
                retmem_mb(tmp_mbs_buffer);
                mbs_name = tmp_mbs_buffer = NULL;
        }
        getname_bytes_count += strlen(np->string_mb) + 1;
        /* Fill in the new Name */
        np->stat.time = file_no_time;
        np->hash.length = length;
        /* Scan the namestring to classify it */
        for (cap = name, len = 0; --length >= 0;) {
                len |= get_char_semantics_value(*cap++);
        }
        np->dollar = BOOLEAN((len & (int) dollar_sem) != 0);
        np->meta = BOOLEAN((len & (int) meta_sem) != 0);
        np->percent = BOOLEAN((len & (int) percent_sem) != 0);
        np->wildcard = BOOLEAN((len & (int) wildcard_sem) != 0);
        np->colon = BOOLEAN((len & (int) colon_sem) != 0);
        np->parenleft = BOOLEAN((len & (int) parenleft_sem) != 0);
        getname_names_count++;
        return np;
}
______unchanged_portion_omitted_

/*
 *      append_string(from, to, length)
 *
 *      Append a C string to a make string expanding it if nessecary
 *
 *      Parameters:
 *              from            The source (C style) string
 *              to              The destination (make style) string
 *              length          The length of the from string
 *
 *      Global variables used:
 */
void
append_string(register wchar_t *from, register String to, register int length)
{
        if (length == FIND_LENGTH) {
                length = wcslen(from);
                length = wslen(from);
        }
        if (to->buffer.start == NULL) {
                expand_string(to, 32 + length);
        }
        if (to->buffer.end - to->text.p <= length) {
                expand_string(to,
                              (to->buffer.end - to->buffer.start) * 2 +
                              length);
        }
        if (length > 0) {
                (void) wcsncpy(to->text.p, from, length);
                (void) wsncpy(to->text.p, from, length);
                to->text.p += length;
        }
        *(to->text.p) = (int) nul_char;
}
______unchanged_portion_omitted_

/*
 *      expand_string(string, length)
 *
 *      Allocate more memory for strings that run out of space.
 *
 *      Parameters:
 *              string          The make style string we want to expand
 *              length          The new length we need
 *
 *      Global variables used:
 */
static void
expand_string(register String string, register int length)
{
        register wchar_t        *p;

        if (string->buffer.start == NULL) {
                /* For strings that have no memory allocated */
                string->buffer.start =
                  string->text.p =
                    string->text.end =
                      ALLOC_WC(length);
                string->buffer.end = string->buffer.start + length;
                string->text.p[0] = (int) nul_char;
                string->free_after_use = true;
                expandstring_count++;
                return;
        }
        if (string->buffer.end - string->buffer.start >= length) {
                /* If we really don’t need more memory. */
                return;
        }
        /*
         * Get more memory, copy the string and free the old buffer if
         * it is was malloc()’ed.
         */
        expandstring_count++;
        p = ALLOC_WC(length);
        (void) wcscpy(p, string->buffer.start);
        (void) wscpy(p, string->buffer.start);
        string->text.p = p + (string->text.p - string->buffer.start);
        string->text.end = p + (string->text.end - string->buffer.start);
        string->buffer.end = p + length;
        if (string->free_after_use) {
                retmem(string->buffer.start);
        }
        string->buffer.start = p;
        string->free_after_use = true;
}
______unchanged_portion_omitted_
```