```
**********************************************************
    1738 Wed May 20 11:32:37 2015
new/usr/src/cmd/make/bin/Makefile
make: unifdef for NSE (undefined)
**********************************************************
```

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

# Copyright 2015, Richard Lowe.

PROG=   make
OBJS=   ar.o            \
        depvar.o        \
        dist.o          \
        doname.o        \
        dosys.o         \
        files.o         \
        globals.o       \
        implicit.o      \
        macro.o         \
        main.o          \
        make.o          \
        misc.o          \
        nse_printdep.o  \
        nse.o           \
        parallel.o      \
        pmake.o         \
        read.o          \
        read2.o         \
        rep.o           \
        state.o

include ../../Makefile.cmd
include ../Makefile.com

LDLIBS += ../lib/mksh/libmksh.a ../lib/mksdmsi18n/libmksdmsi18n.a ../lib/vroot/l
LDLIBS += ../lib/bsd/libbsd.a -lc

CPPFLAGS += -D_FILE_OFFSET_BITS=64

ROOTLINKS = $(ROOTCCSBIN)/make $(ROOTXPG4BIN)/make $(ROOTBIN)/dmake $(ROOTCCSLIB
        $(ROOTLIB)/svr4.make

ROOTRULES = $(ROOTSHLIB)/make/make.rules $(ROOTSHLIB)/make/svr4.make.rules

all:    $(PROG)

install: all $(ROOTPROG) $(ROOTLINKS) $(ROOTRULES)

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

$(ROOTCCSBIN)/make:
        -$(RM) $@; $(SYMLINK) ../../bin/make $@

$(ROOTCCSLIB)/svr4.make:
        -$(RM) $@; $(SYMLINK) ../../bin/make $@
```



```
$(ROOTLIB)/svr4.make:
        -$(RM) $@; $(SYMLINK) ../bin/make $@

$(ROOTXPG4BIN)/make:
        -$(RM) $@; $(SYMLINK) ../../bin/make $@

$(ROOTBIN)/dmake:
        -$(RM) $@; $(SYMLINK) ./make $@

$(ROOTRULES) := FILEMODE = 0444

$(ROOTRULES): $(ROOTSHLIB)/make

$(ROOTSHLIB)/make: FRC
        $(INS.dir)

$(ROOTSHLIB)/make/%: %.file
        $(INS.rename)

lint:

clean:
        $(RM) $(OBJS)

FRC:

include ../../Makefile.targ
```

****
2685 Wed May 20 11:32:37 2015
new/usr/src/cmd/make/bin/depvar.cc
make: unifdef for NSE (undefined)
****
_____unchanged_portion_omitted_

```
 69 /*
 70  * The macro 'name' has been used in either the left-hand or
 71  * right-hand side of a dependency.  See if it is in the
 72  * list.  Two things are looked for.  Names given as args
 73  * to the -V list are checked so as to set the same/differ
 74  * output for the -P option.  Names given as macro=value
 75  * command-line args are checked and, if found, an NSE
 76  * warning is produced.
 77  */
 78 void
 79 depvar_dep_macro_used(Name name)
 80 {
 81         Depvar          dv;

 83         for (dv = depvar_list; dv != NULL; dv = dv->next) {
 84                 if (name == dv->name) {
 85 #ifdef NSE
 86                         if (dv->cmdline) {
 87                                 nse_dep_cmdmacro(dv->name->string);
 88                         }
 89 #endif
 85                         variant_deps = true;
 86                         break;
 87                 }
 88         }
 89 }

 96 #ifdef NSE
 97 /*
 98  * The macro 'name' has been used in either the argument
 99  * to a cd before a recursive make.  See if it was
100  * defined on the command-line and, if so, complain.
101  */
102 void
103 depvar_rule_macro_used(Name name)
104 {
105         Depvar          dv;

107         for (dv = depvar_list; dv != NULL; dv = dv->next) {
108                 if (name == dv->name) {
109                         if (dv->cmdline) {
110                                 nse_rule_cmdmacro(dv->name->string);
111                         }
112                         break;
113                 }
114         }
115 }
116 #endif

 92 /*
 93  * Print the results.  If any of the Dependency Variables
 94  * affected the dependencies then the dependencies potentially
 95  * differ because of these variables.
 96  */
 97 void
 98 depvar_print_results(void)
 99 {
100         if (variant_deps) {
101                 printf(catgets(catd, 1, 234, "differ\n"));
```



```
102         } else {
103                 printf(catgets(catd, 1, 235, "same\n"));
104         }
105 }
```

_____unchanged_portion_omitted_

```
**********************************************************
  104243 Wed May 20 11:32:38 2015
new/usr/src/cmd/make/bin/doname.cc
make: unifdef for NSE (undefined)
**********************************************************
______unchanged_portion_omitted_

 298 /*
 299  * DONE.
 300  *
 301  *      doname(target, do_get, implicit)
 302  *
 303  *      Chases all files the target depends on and builds any that
 304  *      are out of date. If the target is out of date it is then rebuilt.
 305  *
 306  *      Return value:
 307  *                              Indiates if build failed or nt
 308  *
 309  *      Parameters:
 310  *              target          Target to build
 311  *              do_get          Run sccs get is nessecary
 312  *              implicit        doname is trying to find an implicit rule
 313  *
 314  *      Global variables used:
 315  *              assign_done     True if command line assgnment has happened
 316  *              commands_done   Preserved for the case that we need local value
 317  *              debug_level     Should we trace make's actions?
 318  *              default_rule    The rule for ".DEFAULT", used as last resort
 319  *              empty_name      The Name "", used when looking for single sfx
 320  *              keep_state      Indicates that .KEEP_STATE is on
 321  *              parallel        True if building in parallel
 322  *              recursion_level Used for tracing
 323  *              report_dependencies make -P is on
 324  */
 325 Doname
 326 doname(register Name target, register Boolean do_get, register Boolean implicit,
 327 {
 328         Doname                  result = build_dont_know;
 329         Chain                   out_of_date_list = NULL;
 330 #ifdef TEAMWARE_MAKE_CMN
 331         Chain                   target_group;
 332 #endif
 333         Property                old_locals = NULL;
 334         register Property       line;
 335         Property                command = NULL;
 336         register Dependency     dependency;
 337         Name                    less = NULL;
 338         Name                    true_target = target;
 339         Name                    *automatics = NULL;
 340         register int            auto_count;
 341         Boolean                 rechecking_target = false;
 342         Boolean                 saved_commands_done;
 343         Boolean                 restart = false;
 344         Boolean                 save_parallel = parallel;
 345 #ifdef NSE
 346         Boolean                 save_readdep;
 347 #endif
 345         Boolean                 doing_subtree = false;

 347         Boolean                 recheck_conditionals = false;

 349         if (target->state == build_running) {
 350                 return build_running;
 351         }
 352         line = get_prop(target->prop, line_prop);
 353 #ifdef TEAMWARE_MAKE_CMN
```



```
 354         if (line != NULL) {
 355                 /*
 356                  * If this target is a member of target group and one of the
 357                  * other members of the group is running, mark this target
 358                  * as running.
 359                  */
 360                 for (target_group = line->body.line.target_group;
 361                      target_group != NULL;
 362                      target_group = target_group->next) {
 363                         if (is_running(target_group->name)) {
 364                                 target->state = build_running;
 365                                 add_pending(target,
 366                                             recursion_level,
 367                                             do_get,
 368                                             implicit,
 369                                             false);
 370                                 return build_running;
 371                         }
 372                 }
 373         }
 377 #ifdef NSE
 378         nse_check_file_backquotes(target->string);
 379 #endif
 374 #endif
 375         /*
 376          * If the target is a constructed one for a "::" target,
 377          * we need to consider that.
 378          */
 379         if (target->has_target_prop) {
 380                 true_target = get_prop(target->prop,
 381                                        target_prop)->body.target.target;
 382                 if (true_target->colon_splits > 0) {
 383                         /* Make sure we have a valid time for :: targets */
 384                         Property        time;

 386                         time = get_prop(true_target->prop, time_prop);
 387                         if (time != NULL) {
 388                                 true_target->stat.time = time->body.time.time;
 389                         }
 390                 }
 391         }
 392         (void) exists(true_target);
 393         /*
 394          * If the target has been processed, we don't need to do it again,
 395          * unless it depends on conditional macros or a delayed assignment,
 396          * or it has been done when KEEP_STATE is on.
 397          */
 398         if (target->state == build_ok) {
 399                 if((!keep_state || (!target->depends_on_conditional && !assign_d
 400                         return build_ok;
 401                 } else {
 402                         recheck_conditionals = true;
 403                 }
 404         }
 405         if (target->state == build_subtree) {
 406                 /* A dynamic macro subtree is being built */
 407                 target->state = build_dont_know;
 408                 doing_subtree = true;
 409                 if (!target->checking_subtree) {
 410                         /*
 411                          * This target has been started before and therefore
 412                          * not all dependencies have to be built.
 413                          */
 414                         restart = true;
 415                 }
 416         } else if (target->state == build_pending) {
```

```
417                 target->state = build_dont_know;
418                 restart = true;
419 /*
420 #ifdef TEAMWARE_MAKE_CMN
421         } else if (parallel &&
422                    keep_state &&
423                    (target->conditional_cnt > 0)) {
424             if (!parallel_ok(target, false)) {
425                 add_subtree(target, recursion_level, do_get, implicit);
426                 target->state = build_running;
427                 return build_running;
428             }
429 #endif
430  */
431         }
432         /*
433          * If KEEP_STATE is on, we have to rebuild the target if the
434          * building of it caused new automatic dependencies to be reported.
435          * This is where we restart the build.
436          */
437         if (line != NULL) {
438                 line->body.line.percent = NULL;
439         }
440 recheck_target:
441         /* Init all local variables */
442         result = build_dont_know;
443         out_of_date_list = NULL;
444         command = NULL;
445         less = NULL;
446         auto_count = 0;
447         if (!restart && line != NULL) {
448                 /*
449                  * If this target has never been built before, mark all
450                  * of the dependencies as never built.
451                  */
452                 for (dependency = line->body.line.dependencies;
453                      dependency != NULL;
454                      dependency = dependency->next) {
455                         dependency->built = false;
456                 }
457         }
458         /* Save the set of automatic depes defined for this target */
459         if (keep_state &&
460             (line != NULL) &&
461             (line->body.line.dependencies != NULL)) {
462                 Name *p;

464                 /*
465                  * First run thru the dependency list to see how many
466                  * autos there are.
467                  */
468                 for (dependency = line->body.line.dependencies;
469                      dependency != NULL;
470                      dependency = dependency->next) {
471                         if (dependency->automatic && !dependency->stale) {
472                                 auto_count++;
473                         }
474                 }
475                 /* Create vector to hold the current autos */
476                 automatics =
477                   (Name *) alloca((int) (auto_count * sizeof (Name)));
478                 /* Copy them */
479                 for (p = automatics, dependency = line->body.line.dependencies;
480                      dependency != NULL;
481                      dependency = dependency->next) {
482                         if (dependency->automatic && !dependency->stale) {
```



```
483                                 *p++ = dependency->name;
484                         }
485                 }
486         }
487         if (debug_level > 1) {
488                 (void) printf(NOCATGETS("%*sdoname(%s)\n"),
489                               recursion_level,
490                               "",
491                               target->string_mb);
492         }
493         recursion_level++;
494         /* Avoid infinite loops */
495         if (target->state == build_in_progress) {
496                 warning(catgets(catd, 1, 16, "Infinite loop: Target '%s' depends
497                         target->string_mb);
498                 return build_ok;
499         }
500         target->state = build_in_progress;

502         /* Activate conditional macros for the target */
503         if (!target->added_pattern_conditionals) {
504                 add_pattern_conditionals(target);
505                 target->added_pattern_conditionals = true;
506         }
507         if (target->conditional_cnt > 0) {
508                 old_locals = (Property) alloca(target->conditional_cnt *
509                                                sizeof (Property_rec));
510                 set_locals(target, old_locals);
511         }

513 /*
514  * after making the call to dynamic_dependecies unconditional we can handle
515  * target names that are same as file name. In this case $$@ in the
516  * dependencies did not mean anything. WIth this change it expands it
517  * as expected.
518  */
519         if (!target->has_depe_list_expanded)
520         {
527 #ifdef NSE
528                 save_readdep = reading_dependencies;
529                 reading_dependencies= true;
530 #endif
521                 dynamic_dependencies(target);
532 #ifdef NSE
533                 reading_dependencies= save_readdep;
534 #endif
522         }

524 /*
525  *      FIRST SECTION -- GO THROUGH DEPENDENCIES AND COLLECT EXPLICIT
526  *      COMMANDS TO RUN
527  */
528         if ((line = get_prop(target->prop, line_prop)) != NULL) {
529                 if (check_dependencies(&result,
530                                        line,
531                                        do_get,
532                                        target,
533                                        true_target,
534                                        doing_subtree,
535                                        &out_of_date_list,
536                                        old_locals,
537                                        implicit,
538                                        &command,
539                                        less,
540                                        rechecking_target,
541                                        recheck_conditionals)) {
```

```
542                         return build_running;
543                 }
544                 if (line->body.line.query != NULL) {
545                         delete_query_chain(line->body.line.query);
546                 }
547                 line->body.line.query = out_of_date_list;
548         }

550 #ifdef PARALLEL
551         if (doing_subtree) {
552                 parallel = false;
553         }
554 #endif

556 /*
557  * If the target is a :: type, do not try to find the rule for the target,
558  * all actions will be taken by separate branches.
559  * Else, we try to find an implicit rule using various methods,
560  * we quit as soon as one is found.
561  *
562  * [tolik, 12 Sep 2002] Do not try to find implicit rule for the target
563  * being rechecked - the target is being rechecked means that it already
564  * has explicit dependencies derived from an implicit rule found
565  * in previous step.
566  */
567         if (target->colon_splits == 0 && !rechecking_target) {
568                 /* Look for percent matched rule */
569                 if ((result == build_dont_know) &&
570                     (command == NULL)) {
571                         switch (find_percent_rule(
572                                         target,
573                                         &command,
574                                         recheck_conditionals)) {
575                         case build_failed:
576                                 result = build_failed;
577                                 break;
578 #ifdef TEAMWARE_MAKE_CMN
579                         case build_running:
580                                 target->state = build_running;
581                                 add_pending(target,
582                                             --recursion_level,
583                                             do_get,
584                                             implicit,
585                                             false);
586                                 if (target->conditional_cnt > 0) {
587                                         reset_locals(target,
588                                                      old_locals,
589                                                      get_prop(target->prop,
590                                                               conditional_prop),
591                                                      0);
592                                 }
593                                 return build_running;
594 #endif
595                         case build_ok:
596                                 result = build_ok;
597                                 break;
598                         }
599                 }
600                 /* Look for double suffix rule */
601                 if (result == build_dont_know) {
602                         Property member;

604                         if (target->is_member &&
605                             ((member = get_prop(target->prop, member_prop)) !=
606                              NULL)) {
607                                 switch (find_ar_suffix_rule(target,
```



```
608                                                 member->body.
609                                                 member.member,
610                                                 &command,
611                                                 recheck_conditionals)) {
612                                 case build_failed:
613                                         result = build_failed;
614                                         break;
615 #ifdef TEAMWARE_MAKE_CMN
616                                 case build_running:
617                                         target->state = build_running;
618                                         add_pending(target,
619                                                     --recursion_level,
620                                                     do_get,
621                                                     implicit,
622                                                     false);
623                                     if (target->conditional_cnt > 0) {
624                                         reset_locals(target,
625                                                      old_locals,
626                                                      get_prop(target->prop,
627                                                          conditional_prop),
628                                                      0);
629                                     }
630                                         return build_running;
631 #endif
632                                 default:
633                                         /* ALWAYS bind $% for old style */
634                                         /* ar rules */
635                                         if (line == NULL) {
636                                                 line =
637                                                   maybe_append_prop(target,
638                                                                     line_prop);
639                                         }
640                                         line->body.line.percent =
641                                           member->body.member.member;
642                                         break;
643                                 }
644                         } else {
645                                 switch (find_double_suffix_rule(target,
646                                                 &command,
647                                                 recheck_conditionals)) {
648                                 case build_failed:
649                                         result = build_failed;
650                                         break;
651 #ifdef TEAMWARE_MAKE_CMN
652                                 case build_running:
653                                         target->state = build_running;
654                                         add_pending(target,
655                                                     --recursion_level,
656                                                     do_get,
657                                                     implicit,
658                                                     false);
659                                         if (target->conditional_cnt > 0) {
660                                                 reset_locals(target,
661                                                              old_locals,
662                                                              get_prop(target->
663                                                                       prop,
664                                                                       conditiona
665                                                              0);
666                                         }
667                                         return build_running;
668 #endif
669                                 }
670                         }
671                 }
672                 /* Look for single suffix rule */
```

```
/* /tolik/
 * I commented !implicit to fix bug 1247448: Suffix Rules failed when combine wi
 * This caused problem with SVR4 tilde rules (infinite recursion). So I made som
 */
/* /tolik, 06.21.96/
 * Regression! See BugId 1255360
 * If more than one percent rules are defined for the same target then
 * the behaviour of 'make' with my previous fix may be different from one
 * of the 'old make'.
 * The global variable second_pass (maybe it should be an argument to doname())
 * is intended to avoid this regression. It is set in doname_check().
 * First, 'make' will work as it worked before. Only when it is
 * going to say "don't know how to make target" it sets second_pass to true and
 * run 'doname' again but now trying to use Single Suffix Rules.
 */
                if ((result == build_dont_know) && !automatic && (!implicit || s
                    ((line == NULL) ||
                     ((line->body.line.target != NULL) &&
                      !line->body.line.target->has_regular_dependency))) {
                        switch (find_suffix_rule(target,
                                                 target,
                                                 empty_name,
                                                 &command,
                                                 recheck_conditionals)) {
                        case build_failed:
                                result = build_failed;
                                break;
#ifdef TEAMWARE_MAKE_CMN
                        case build_running:
                                target->state = build_running;
                                add_pending(target,
                                            --recursion_level,
                                            do_get,
                                            implicit,
                                            false);
                                if (target->conditional_cnt > 0) {
                                        reset_locals(target,
                                                     old_locals,
                                                     get_prop(target->prop,
                                                              conditional_prop),
                                                     0);
                                }
                                return build_running;
#endif
                        }
                }
                /* Try to sccs get */
                if ((command == NULL) &&
                    (result == build_dont_know) &&
                    do_get) {
                        result = sccs_get(target, &command);
                }

                /* Use .DEFAULT rule if it is defined. */
                if ((command == NULL) &&
                    (result == build_dont_know) &&
                    (true_target->colons == no_colon) &&
                    default_rule &&
                    !implicit) {
                        /* Make sure we have a line prop */
                        line = maybe_append_prop(target, line_prop);
                        command = line;
                        Boolean out_of_date;
                        if (true_target->is_member) {
                                out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
                                                                        line->bo
```



```
                        } else {
                                out_of_date = (Boolean) OUT_OF_DATE(true_target-
                                                                    line->body.l
                        }
                        if (build_unconditional || out_of_date) {
                                line->body.line.is_out_of_date = true;
                                if (debug_level > 0) {
                                        (void) printf(catgets(catd, 1, 17, "%*sB
                                                      recursion_level,
                                                      "",
                                                      true_target->string_mb);
                                }
                        }
                        line->body.line.sccs_command = false;
                        line->body.line.command_template = default_rule;
                        line->body.line.target = true_target;
                        line->body.line.star = NULL;
                        line->body.line.less = true_target;
                        line->body.line.percent = NULL;
                }
        }

        /* We say "target up to date" if no cmd were executed for the target */
        if (!target->is_double_colon_parent) {
                commands_done = false;
        }

        silent = silent_all;
        ignore_errors = ignore_errors_all;
        if  (posix)
        {
          if  (!silent)
          {
            silent = (Boolean) target->silent_mode;
          }
          if  (!ignore_errors)
          {
            ignore_errors = (Boolean) target->ignore_error_mode;
          }
        }

        int doname_dyntarget = 0;
r_command:
        /* Run commands if any. */
        if ((command != NULL) &&
            (command->body.line.command_template != NULL)) {
                if (result != build_failed) {
                        result = run_command(command,
                                             (Boolean) ((parallel || save_parall
#ifdef NSE
                        nse_check_no_deps_no_rule(target,
                                get_prop(target->prop, line_prop), command);
#endif
                }
                switch (result) {
#ifdef TEAMWARE_MAKE_CMN
                case build_running:
                        add_running(target,
                                    true_target,
                                    command,
                                    --recursion_level,
                                    auto_count,
                                    automatics,
                                    do_get,
                                    implicit);
                        target->state = build_running;
```

```
802                         if ((line = get_prop(target->prop,
803                                         line_prop)) != NULL) {
804                                 if (line->body.line.query != NULL) {
805                                         delete_query_chain(line->body.line.query
806                                 }
807                                 line->body.line.query = NULL;
808                         }
809                         if (target->conditional_cnt > 0) {
810                                 reset_locals(target,
811                                         old_locals,
812                                         get_prop(target->prop,
813                                                 conditional_prop),
814                                         0);
815                         }
816                         return build_running;
817                 case build_serial:
818                         add_serial(target,
819                                    --recursion_level,
820                                    do_get,
821                                    implicit);
822                         target->state = build_running;
823                         line = get_prop(target->prop, line_prop);
824                         if (line != NULL) {
825                                 if (line->body.line.query != NULL) {
826                                         delete_query_chain(line->body.line.query
827                                 }
828                                 line->body.line.query = NULL;
829                         }
830                         if (target->conditional_cnt > 0) {
831                                 reset_locals(target,
832                                         old_locals,
833                                         get_prop(target->prop,
834                                                 conditional_prop),
835                                         0);
836                         }
837                         return build_running;
838 #endif
839                 case build_ok:
840                         /* If all went OK set a nice timestamp */
841                         if (true_target->stat.time == file_doesnt_exist) {
842                                 true_target->stat.time = file_max_time;
843                         }
844                         break;
845                 }
846         } else {
847                 /*
848                  * If no command was found for the target, and it doesn't
849                  * exist, and it is mentioned as a target in the makefile,
850                  * we say it is extremely new and that it is OK.
851                  */
852                 if (target->colons != no_colon) {
853                         if (true_target->stat.time == file_doesnt_exist){
854                                 true_target->stat.time = file_max_time;
855                         }
856                         result = build_ok;
857                 }
858                 /*
859                  * Trying dynamic targets.
860                  */
861                 if(!doname_dyntarget) {
862                         doname_dyntarget = 1;
863                         Name dtarg = find_dyntarget(target);
864                         if(dtarg!=NULL) {
865                                 if (!target->has_depe_list_expanded) {
866                                         dynamic_dependencies(target);
867                                 }
```



```
868                                 if ((line = get_prop(target->prop, line_prop)) !
869                                         if (check_dependencies(&result,
870                                                                 line,
871                                                                 do_get,
872                                                                 target,
873                                                                 true_target,
874                                                                 doing_subtree,
875                                                                 &out_of_date_list
876                                                                 old_locals,
877                                                                 implicit,
878                                                                 &command,
879                                                                 less,
880                                                                 rechecking_target
881                                                                 recheck_condition
882                                         {
883                                                 return build_running;
884                                         }
885                                         if (line->body.line.query != NULL) {
886                                                 delete_query_chain(line->body.li
887                                         }
888                                         line->body.line.query = out_of_date_list
889                                 }
890                                 goto r_command;
891                         }
892                 }
893                 /*
894                  * If the file exists, it is OK that we couldnt figure
895                  * out how to build it.
896                  */
897                 (void) exists(target);
898                 if ((target->stat.time != file_doesnt_exist) &&
899                     (result == build_dont_know)) {
900                         result = build_ok;
901                 }
902         }

904         /*
905          * Some of the following is duplicated in the function finish_doname.
906          * If anything is changed here, check to see if it needs to be
907          * changed there.
908          */
909         if ((line = get_prop(target->prop, line_prop)) != NULL) {
910                 if (line->body.line.query != NULL) {
911                         delete_query_chain(line->body.line.query);
912                 }
913                 line->body.line.query = NULL;
914         }
915         target->state = result;
916         parallel = save_parallel;
917         if (target->conditional_cnt > 0) {
918                 reset_locals(target,
919                              old_locals,
920                              get_prop(target->prop, conditional_prop),
921                              0);
922         }
923         recursion_level--;
924         if (target->is_member) {
925                 Property member;

927                 /* Propagate the timestamp from the member file to the member*/
928                 if ((target->stat.time != file_max_time) &&
929                     ((member = get_prop(target->prop, member_prop)) != NULL) &&
930                     (exists(member->body.member.member) > file_doesnt_exist)) {
931                         target->stat.time =
932                           member->body.member.member->stat.time;
933                 }
```

```
934         }
935         /*
936          * Check if we found any new auto dependencies when we
937          * built the target.
938          */
939         if ((result == build_ok) && check_auto_dependencies(target,
940                                                             auto_count,
941                                                             automatics)) {
942                 if (debug_level > 0) {
943                         (void) printf(catgets(catd, 1, 18, "%*sTarget ‘%s’ acqui
944                                       recursion_level,
945                                       "",
946                                       true_target->string_mb);
947                 }
948                 rechecking_target = true;
949                 saved_commands_done = commands_done;
950                 goto recheck_target;
951         }

953         if (rechecking_target && !commands_done) {
954                 commands_done = saved_commands_done;
955         }

957         return result;
958 }

960 /*
961  * DONE.
962  *
963  *      check_dependencies(result, line, do_get,
964  *                         target, true_target, doing_subtree, out_of_date_tail,
965  *                         old_locals, implicit, command, less, rechecking_target)
966  *
967  *      Return value:
968  *                              True returned if some dependencies left running
969  *
970  *      Parameters:
971  *              result          Pointer to cell we update if build failed
972  *              line            We get the dependencies from here
973  *              do_get          Allow use of sccs get in recursive doname()
974  *              target          The target to chase dependencies for
975  *              true_target     The real one for :: and lib(member)
976  *              doing_subtree   True if building a conditional macro subtree
977  *              out_of_date_tail Used to set the $? list
978  *              old_locals      Used for resetting the local macros
979  *              implicit        Called when scanning for implicit rules?
980  *              command         Place to stuff command
981  *              less            Set to $< value
982  *
983  *      Global variables used:
984  *              command_changed Set if we suspect .make.state needs rewrite
985  *              debug_level     Should we trace actions?
986  *              force           The Name " FORCE", compared against
987  *              recursion_level Used for tracing
988  *              rewrite_statefile Set if .make.state needs rewriting
989  *              wait_name       The Name ".WAIT", compared against
990  */
991 static Boolean
992 #ifdef TEAMWARE_MAKE_CMN
993 check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
994 #else
995 check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
996 #endif
997 {
998         Boolean                 dependencies_running;
999         register Dependency     dependency;
```



```
1000         Doname                  dep_result;
1001         Boolean                 dependency_changed = false;

1003         line->body.line.dependency_time = file_doesnt_exist;
1004         if (line->body.line.query != NULL) {
1005                 delete_query_chain(line->body.line.query);
1006         }
1007         line->body.line.query = NULL;
1008         line->body.line.is_out_of_date = false;
1009         dependencies_running = false;
1010         /*
1011          * Run thru all the dependencies and call doname() recursively
1012          * on each of them.
1013          */
1014         for (dependency = line->body.line.dependencies;
1015              dependency != NULL;
1016              dependency = dependency->next) {
1017                 Boolean this_dependency_changed = false;

1019                 if (!dependency->automatic &&
1020                     (rechecking_target || target->rechecking_target)) {
1021                         /*
1022                          * We only bother with the autos when rechecking
1023                          */
1024                         continue;
1025                 }

1027                 if (dependency->name == wait_name) {
1028                         /*
1029                          * The special target .WAIT means finish all of
1030                          * the prior dependencies before continuing.
1031                          */
1032                         if (dependencies_running) {
1033                                 break;
1034                         }
1035 #ifdef DISTRIBUTED
1036                 } else if ((!parallel_ok(dependency->name, false)) &&
1037                            (dependencies_running)) {
1038                         /*
1039                          * If we can’t execute the current dependency in
1040                          * parallel, hold off the dependency processing
1041                          * to preserve the order of the dependencies.
1042                          */
1043                         break;
1044 #endif
1045                 } else {
1046                         timestruc_t     depe_time = file_doesnt_exist;


1049                         if (true_target->is_member) {
1050                                 depe_time = exists(dependency->name);
1051                         }
1052                         if (dependency->built ||
1053                             (dependency->name->state == build_failed)) {
1054                                 dep_result = (Doname) dependency->name->state;
1055                         } else {
1073 #ifdef NSE
1074                                 nse_check_sccs(target->string,
1075                                                dependency->name->string);
1076                                 nse_check_derived_src(target,
1077                                                dependency->name->string,
1078                                                line->body.line.command_template)
1079 #endif
1056                                 dep_result = doname_check(dependency->name,
1057                                                           do_get,
1058                                                           false,
```

```
1059                                                    (Boolean) dependency->
1060                        }
1061                        if (true_target->is_member || dependency->name->is_membe
1062                                /* should compare only secs, cause lib members d
1063                                if (depe_time.tv_sec != dependency->name->stat.t
1064                                        this_dependency_changed =
1065                                          dependency_changed =
1066                                            true;
1067                                }
1068                        } else {
1069                                if (depe_time != dependency->name->stat.time) {
1070                                        this_dependency_changed =
1071                                          dependency_changed =
1072                                            true;
1073                                }
1074                        }
1075                        dependency->built = true;
1076                        switch (dep_result) {
1077                        case build_running:
1078                                dependencies_running = true;
1079                                continue;
1080                        case build_failed:
1081                                *result = build_failed;
1082                                break;
1083                        case build_dont_know:
1084 /*
1085  * If make can't figure out how to make a dependency, maybe the dependency
1086  * is out of date. In this case, we just declare the target out of date
1087  * and go on. If we really need the dependency, the make'ing of the target
1088  * will fail. This will only happen for automatic (hidden) dependencies.
1089  */
1090                                if(!recheck_conditionals) {
1091                                        line->body.line.is_out_of_date = true;
1092                                }
1093                                /*
1094                                 * Make sure the dependency is not saved
1095                                 * in the state file.
1096                                 */
1097                                dependency->stale = true;
1098                                rewrite_statefile =
1099                                  command_changed =
1100                                    true;
1101                                if (debug_level > 0) {
1102                                        (void) printf(catgets(catd, 1, 19, "Targ
1103                                                      true_target->string_mb,
1104                                                      dependency->name->string_mb
1105                                }
1106                                break;
1107                        }
1108                        if (dependency->name->depends_on_conditional) {
1109                                target->depends_on_conditional = true;
1110                        }
1111                        if (dependency->name == force) {
1112                                target->stat.time =
1113                                  dependency->name->stat.time;
1114                        }
1115                        /*
1116                         * Propagate new timestamp from "member" to
1117                         * "lib.a(member)".
1118                         */
1119                        (void) exists(dependency->name);

1121                        /* Collect the timestamp of the youngest dependency */
1122                        line->body.line.dependency_time =
1123                          MAX(dependency->name->stat.time,
1124                              line->body.line.dependency_time);
```



```
1126                        /* Correction: do not consider nanosecs for members */
1127                        if(true_target->is_member || dependency->name->is_member
1128                                line->body.line.dependency_time.tv_nsec = 0;
1129                        }

1131                        if (debug_level > 1) {
1132                                (void) printf(catgets(catd, 1, 20, "%*sDate(%s)=
1133                                              recursion_level,
1134                                              "",
1135                                              dependency->name->string_mb,
1136                                              time_to_string(dependency->name->
1137                                                             stat.time));
1138                                if (dependency->name->stat.time > line->body.lin
1139                                        (void) printf(catgets(catd, 1, 21, "%*sD
1140                                                      recursion_level,
1141                                                      "",
1142                                                      true_target->string_mb,
1143                                                      time_to_string(line->body.
1144                                                                     dependency_
1145                                }
1146                        }

1148                        /* Build the $? list */
1149                        if (true_target->is_member) {
1150                                if (this_dependency_changed == true) {
1151                                        true_target->stat.time = dependency->nam
1152                                        true_target->stat.time.tv_sec--;
1153                                } else {
1154                                        /* Dina:
1155                                         * The next statement is commented
1156                                         * out as a fix for bug #1051032.
1157                                         * if dependency hasn't changed
1158                                         * then there's no need to invalidate
1159                                         * true_target. This statemnt causes
1160                                         * make to take much longer to process
1161                                         * an already-built archive. Soren
1162                                         * said it was a quick fix for some
1163                                         * problem he doesn't remember.
1164                                        true_target->stat.time = file_no_time;
1165                                         */
1166                                        (void) exists(true_target);
1167                                }
1168                        } else {
1169                                (void) exists(true_target);
1170                        }
1171                        Boolean out_of_date;
1172                        if (true_target->is_member || dependency->name->is_membe
1173                                out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
1174                                                                        dependen
1175                        } else {
1176                                out_of_date = (Boolean) OUT_OF_DATE(true_target-
1177                                                                    dependency->
1178                        }
1179                        if ((build_unconditional || out_of_date) &&
1180                            (dependency->name != force) &&
1181                            (dependency->stale == false)) {
1182                                *out_of_date_tail = ALLOC(Chain);
1183                                if (dependency->name->is_member &&
1184                                    (get_prop(dependency->name->prop,
1185                                              member_prop) != NULL)) {
1186                                        (*out_of_date_tail)->name =
1187                                          get_prop(dependency->name->prop,
1188                                                   member_prop)->
1189                                            body.member.member;
1190                                } else {
```

```
1191                                        (*out_of_date_tail)->name =
1192                                          dependency->name;
1193                                }
1194                                (*out_of_date_tail)->next = NULL;
1195                                out_of_date_tail = &(*out_of_date_tail)->next;
1196                                if (debug_level > 0) {
1197                                        if (dependency->name->stat.time == file_
1198                                                (void) printf(catgets(catd, 1, 2
1199                                                        recursion_level,
1200                                                        "",
1201                                                        true_target->strin
1202                                                        dependency->name->
1203                                        } else {
1204                                                (void) printf(catgets(catd, 1, 2
1205                                                        recursion_level,
1206                                                        "",
1207                                                        true_target->strin
1208                                                        dependency->name->
1209                                        }
1210                                }
1211                        }
1212                        if (dependency->name == force) {
1213                                force->stat.time =
1214                                  file_max_time;
1215                                force->state = build_dont_know;
1216                        }
1217                }
1218        }
1219 #ifdef TEAMWARE_MAKE_CMN
1220        if (dependencies_running) {
1221                if (doing_subtree) {
1222                        if (target->conditional_cnt > 0) {
1223                                reset_locals(target,
1224                                        old_locals,
1225                                        get_prop(target->prop,
1226                                                conditional_prop),
1227                                        0);
1228                        }
1229                        return true;
1230                } else {
1231                        target->state = build_running;
1232                        add_pending(target,
1233                                    --recursion_level,
1234                                    do_get,
1235                                    implicit,
1236                                    false);
1237                        if (target->conditional_cnt > 0) {
1238                                reset_locals(target,
1239                                        old_locals,
1240                                        get_prop(target->prop,
1241                                                conditional_prop),
1242                                        0);
1243                        }
1244                        return true;
1245                }
1246        }
1247 #endif
1248        /*
1249         * Collect the timestamp of the youngest double colon target
1250         * dependency.
1251         */
1252        if (target->is_double_colon_parent) {
1253                for (dependency = line->body.line.dependencies;
1254                     dependency != NULL;
1255                     dependency = dependency->next) {
1256                        Property        tmp_line;
```



```
1258                        if ((tmp_line = get_prop(dependency->name->prop, line_pr
1259                                if(tmp_line->body.line.dependency_time != file_m
1260                                        target->stat.time =
1261                                          MAX(tmp_line->body.line.dependency_tim
1262                                              target->stat.time);
1263                                }
1264                        }
1265                }
1266        }
1267        if ((true_target->is_member) && (dependency_changed == true)) {
1268                true_target->stat.time = file_no_time;
1269        }
1270        /*
1271         * After scanning all the dependencies, we check the rule
1272         * if we found one.
1273         */
1274        if (line->body.line.command_template != NULL) {
1275                if (line->body.line.command_template_redefined) {
1276                        warning(catgets(catd, 1, 24, "Too many rules defined for
1277                                target->string_mb);
1278                }
1279                *command = line;
1280                /* Check if the target is out of date */
1281                Boolean out_of_date;
1282                if (true_target->is_member) {
1283                        out_of_date = (Boolean) OUT_OF_DATE_SEC(true_target->sta
1284                                                                line->body.line.
1285                } else {
1286                        out_of_date = (Boolean) OUT_OF_DATE(true_target->stat.ti
1287                                                            line->body.line.depe
1288                }
1289                if (build_unconditional || out_of_date){
1290                        if(!recheck_conditionals) {
1291                                line->body.line.is_out_of_date = true;
1292                        }
1293                }
1294                line->body.line.sccs_command = false;
1295                line->body.line.target = true_target;
1296                if(gnu_style) {

1298                        // set $< for explicit rule
1299                        if(line->body.line.dependencies != NULL) {
1300                                less = line->body.line.dependencies->name;
1301                        }

1303                        // set $* for explicit rule
1304                        Name                    target_body;
1305                        Name                    tt = true_target;
1306                        Property                member;
1307                        register wchar_t        *target_end;
1308                        register Dependency     suffix;
1309                        register int            suffix_length;
1310                        Wstring                 targ_string;
1311                        Wstring                 suf_string;

1313                        if (true_target->is_member &&
1314                            ((member = get_prop(target->prop, member_prop)) !=
1315                             NULL)) {
1316                                tt = member->body.member.member;
1317                        }
1318                        targ_string.init(tt);
1319                        target_end = targ_string.get_string() + tt->hash.length;
1320                        for (suffix = suffixes; suffix != NULL; suffix = suffix-
1321                                suffix_length = suffix->name->hash.length;
1322                                suf_string.init(suffix->name);
```

```
1323                                if (tt->hash.length < suffix_length) {
1324                                        continue;
1325                                } else if (!IS_WEQUALN(suf_string.get_string(),
1326                                                (target_end - suffix_length),
1327                                                suffix_length)) {
1328                                        continue;
1329                                }
1330                                target_body = GETNAME(
1331                                        targ_string.get_string(),
1332                                        (int)(tt->hash.length - suffix_length)
1333                                );
1334                                line->body.line.star = target_body;
1335                        }

1337                        // set result = build_ok so that implicit rules are not
1338                        if(*result == build_dont_know) {
1339                                *result = build_ok;
1340                        }
1341                }
1342                if (less != NULL) {
1343                        line->body.line.less = less;
1344                }
1345        }

1347        return false;
1348 }

1350 /*
1351  *      dynamic_dependencies(target)
1352  *
1353  *      Checks if any dependency contains a macro ref
1354  *      If so, it replaces the dependency with the expanded version.
1355  *      Here, "$@" gets translated to target->string. That is
1356  *      the current name on the left of the colon in the
1357  *      makefile.  Thus,
1358  *              xyz:    s.$@.c
1359  *      translates into
1360  *              xyz:    s.xyz.c
1361  *
1362  *      Also, "$(@F)" translates to the same thing without a preceeding
1363  *      directory path (if one exists).
1364  *      Note, to enter "$@" on a dependency line in a makefile
1365  *      "$$@" must be typed. This is because make expands
1366  *      macros in dependency lists upon reading them.
1367  *      dynamic_dependencies() also expands file wildcards.
1368  *      If there are any Shell meta characters in the name,
1369  *      search the directory, and replace the dependency
1370  *      with the set of files the pattern matches
1371  *
1372  *      Parameters:
1373  *              target          Target to sanitize dependencies for
1374  *
1375  *      Global variables used:
1376  *              c_at            The Name "@", used to set macro value
1377  *              debug_level     Should we trace actions?
1378  *              dot             The Name ".", used to read directory
1379  *              recursion_level Used for tracing
1380  */
1381 void
1382 dynamic_dependencies(Name target)
1383 {
1384        wchar_t                 pattern[MAXPATHLEN];
1385        register wchar_t        *p;
1386        Property                line;
1387        register Dependency     dependency;
1388        register Dependency     *remove;
```



```
1389        String_rec              string;
1390        wchar_t                 buffer[MAXPATHLEN];
1391        register Boolean        set_at = false;
1392        register wchar_t        *start;
1393        Dependency              new_depe;
1394        register Boolean        reuse_cell;
1395        Dependency              first_member;
1396        Name                    directory;
1397        Name                    lib;
1398        Name                    member;
1399        Property                prop;
1400        Name                    true_target = target;
1401        wchar_t                 *library;

1403        if ((line = get_prop(target->prop, line_prop)) == NULL) {
1404                return;
1405        }
1406        /* If the target is constructed from a "::" target we consider that */
1407        if (target->has_target_prop) {
1408                true_target = get_prop(target->prop,
1409                                       target_prop)->body.target.target;
1410        }
1411        /* Scan all dependencies and process the ones that contain "$" chars */
1412        for (dependency = line->body.line.dependencies;
1413             dependency != NULL;
1414             dependency = dependency->next) {
1415                if (!dependency->name->dollar) {
1416                        continue;
1417                }
1418                target->has_depe_list_expanded = true;

1420                /* The make macro $@ is bound to the target name once per */
1421                /* invocation of dynamic_dependencies() */
1422                if (!set_at) {
1423                        (void) SETVAR(c_at, true_target, false);
1424                        set_at = true;
1425                }
1426                /* Expand this dependency string */
1427                INIT_STRING_FROM_STACK(string, buffer);
1428                expand_value(dependency->name, &string, false);
1429                /* Scan the expanded string. It could contain whitespace */
1430                /* which mean it expands to several dependencies */
1431                start = string.buffer.start;
1432                while (iswspace(*start)) {
1433                        start++;
1434                }
1435                /* Remove the cell (later) if the macro was empty */
1436                if (start[0] == (int) nul_char) {
1437                        dependency->name = NULL;
1438                }

1440 /* azv 10/26/95 to fix bug BID_1170218 */
1441                if ((start[0] == (int) period_char) &&
1442                    (start[1] == (int) slash_char)) {
1443                        start += 2;
1444                }
1445 /* azv */

1447                first_member = NULL;
1448                /* We use the original dependency cell for the first */
1449                /* dependency from the expansion */
1450                reuse_cell = true;
1451                /* We also have to deal with dependencies that expand to */
1452                /* lib.a(members) notation */
1453                for (p = start; *p != (int) nul_char; p++) {
1454                        if ((*p == (int) parenleft_char)) {
```

```
1455                                    lib = GETNAME(start, p - start);
1456                                    lib->is_member = true;
1457                                    first_member = dependency;
1458                                    start = p + 1;
1459                                    while (iswspace(*start)) {
1460                                            start++;
1461                                    }
1462                                    break;
1463                            }
1464                    }
1465                    do {
1466                        /* First skip whitespace */
1467                            for (p = start; *p != (int) nul_char; p++) {
1468                                    if ((*p == (int) nul_char) ||
1469                                        iswspace(*p) ||
1470                                        (*p == (int) parenright_char)) {
1471                                            break;
1472                                    }
1473                            }
1474                            /* Enter dependency from expansion */
1475                            if (p != start) {
1476                                    /* Create new dependency cell if */
1477                                    /* this is not the first dependency */
1478                                    /* picked from the expansion */
1479                                    if (!reuse_cell) {
1480                                            new_depe = ALLOC(Dependency);
1481                                            new_depe->next = dependency->next;
1482                                            new_depe->automatic = false;
1483                                            new_depe->stale = false;
1484                                            new_depe->built = false;
1485                                            dependency->next = new_depe;
1486                                            dependency = new_depe;
1487                                    }
1488                                    reuse_cell = false;
1489                                    /* Internalize the dependency name */
1490                                    // tolik. Fix for bug 4110429: inconsistent expa
1491                                    // include "//" and "/./"
1492                                    //dependency->name = GETNAME(start, p - start);
1493                                    dependency->name = normalize_name(start, p - sta
1494                                    if ((debug_level > 0) &&
1495                                        (first_member == NULL)) {
1496                                            (void) printf(catgets(catd, 1, 25, "%*sD
1497                                                          recursion_level,
1498                                                          "",
1499                                                          dependency->name->string_m
1500                                                          true_target->string_mb);
1501                                    }
1502                                    for (start = p; iswspace(*start); start++);
1503                                    p = start;
1504                            }
1505                    } while ((*p != (int) nul_char) &&
1506                             (*p != (int) parenright_char));
1507                    /* If the expansion was of lib.a(members) format we now */
1508                    /* enter the proper member cells */
1509                    if (first_member != NULL) {
1510                            /* Scan the new dependencies and transform them from */
1511                            /* "foo" to "lib.a(foo)" */
1512                            for (; 1; first_member = first_member->next) {
1513                                    /* Build "lib.a(foo)" name */
1514                                    INIT_STRING_FROM_STACK(string, buffer);
1515                                    APPEND_NAME(lib,
1516                                                &string,
1517                                                (int) lib->hash.length);
1518                                    append_char((int) parenleft_char, &string);
1519                                    APPEND_NAME(first_member->name,
1520                                                &string,
```



```
1521                                                FIND_LENGTH);
1522                                    append_char((int) parenright_char, &string);
1523                                    member = first_member->name;
1524                                    /* Replace "foo" with "lib.a(foo)" */
1525                                    first_member->name =
1526                                      GETNAME(string.buffer.start, FIND_LENGTH);
1527                                    if (string.free_after_use) {
1528                                            retmem(string.buffer.start);
1529                                    }
1530                                    if (debug_level > 0) {
1531                                            (void) printf(catgets(catd, 1, 26, "%*sD
1532                                                          recursion_level,
1533                                                          "",
1534                                                          first_member->name->
1535                                                          string_mb,
1536                                                          true_target->string_mb);
1537                                    }
1538                                    first_member->name->is_member = lib->is_member;
1539                                    /* Add member property to member */
1540                                    prop = maybe_append_prop(first_member->name,
1541                                                             member_prop);
1542                                    prop->body.member.library = lib;
1543                                    prop->body.member.entry = NULL;
1544                                    prop->body.member.member = member;
1545                                    if (first_member == dependency) {
1546                                            break;
1547                                    }
1548                            }
1549                    }
1550            }
1551            Wstring wcb;
1552            /* Then scan all the dependencies again. This time we want to expand */
1553            /* shell file wildcards */
1554            for (remove = &line->body.line.dependencies, dependency = *remove;
1555                 dependency != NULL;
1556                 dependency = *remove) {
1557                    if (dependency->name == NULL) {
1558                            dependency = *remove = (*remove)->next;
1559                            continue;
1560                    }
1561                    /* If dependency name string contains shell wildcards */
1562                    /* replace the name with the expansion */
1563                    if (dependency->name->wildcard) {
1588 #ifdef NSE
1589                            nse_wildcard(target->string, dependency->name->string);
1590 #endif
1564                            wcb.init(dependency->name);
1565                            if ((start = (wchar_t *) wschr(wcb.get_string(),
1566                                                           (int) parenleft_char)) != NULL) {
1567                                    /* lib(*) type pattern */
1568                                    library = buffer;
1569                                    (void) wsncpy(buffer,
1570                                                  wcb.get_string(),
1571                                                  start - wcb.get_string());
1572                                    buffer[start-wcb.get_string()] =
1573                                      (int) nul_char;
1574                                    (void) wsncpy(pattern,
1575                                                  start + 1,
1576 (int) (dependency->name->hash.length-(start-wcb.get_string())-2));
1577                                    pattern[dependency->name->hash.length -
1578                                            (start-wcb.get_string()) - 2] =
1579                                              (int) nul_char;
1580                            } else {
1581                                    library = NULL;
1582                                    (void) wsncpy(pattern,
1583                                                  wcb.get_string(),
```

```
1584                                            (int) dependency->name->hash.lengt
1585                                    pattern[dependency->name->hash.length] =
1586                                      (int) nul_char;
1587                            }
1588                            start = (wchar_t *) wsrchr(pattern, (int) slash_char);
1589                            if (start == NULL) {
1590                                    directory = dot;
1591                                    p = pattern;
1592                            } else {
1593                                    directory = GETNAME(pattern, start-pattern);
1594                                    p = start+1;
1595                            }
1596                            /* The expansion is handled by the read_dir() routine*/
1597                            if (read_dir(directory, p, line, library)) {
1598                                    *remove = (*remove)->next;
1599                            } else {
1600                                    remove = &dependency->next;
1601                            }
1602                    } else {
1603                            remove = &dependency->next;
1604                    }
1605            }

1607            /* Then unbind $@ */
1608            (void) SETVAR(c_at, (Name) NULL, false);
1609    }
______unchanged_portion_omitted_

1834    /*
1835     *      execute_serial(line)
1836     *
1837     *      Runs thru the command line for the target and
1838     *      executes the rules one by one.
1839     *
1840     *      Return value:
1841     *                              The result of the command build
1842     *
1843     *      Parameters:
1844     *              line            The command to execute
1845     *
1846     *      Static variables used:
1847     *
1848     *      Global variables used:
1849     *              continue_after_error -k flag
1850     *              do_not_exec_rule -n flag
1851     *              report_dependencies -P flag
1852     *              silent          Don't echo commands before executing
1853     *              temp_file_name  Temp file for auto dependencies
1854     *              vpath_defined   If true, translate path for command
1855     */
1856    Doname
1857    execute_serial(Property line)
1858    {
1859            int                     child_pid = 0;
1860    #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
1861            Avo_MToolJobResultMsg   *job_result_msg;
1862            RWCollectable           *xdr_msg;
1863    #endif
1864            Boolean                 printed_serial;
1865            Doname                  result = build_ok;
1866            Cmd_line                rule, cmd_tail, command = NULL;
1867            char                    mbstring[MAXPATHLEN];
1868            int                     filed;
1869            Name                    target = line->body.line.target;

1871            SEND_MTOOL_MSG(
```



```
1872                    if (!sent_rsrc_info_msg) {
1873                            if (userName[0] == '\0') {
1874                                    avo_get_user(userName, NULL);
1875                            }
1876                            if (hostName[0] == '\0') {
1877                                    strcpy(hostName, avo_hostname());
1878                            }
1879                            send_rsrc_info_msg(1, hostName, userName);
1880                            sent_rsrc_info_msg = 1;
1881                    }
1882                    send_job_start_msg(line);
1883                    job_result_msg = new Avo_MToolJobResultMsg();
1884            );

1886            target->has_recursive_dependency = false;
1887            // We have to create a copy of the rules chain for processing because
1888            // the original one can be destroyed during .make.state file rereading.
1889            for (rule = line->body.line.command_used;
1890                 rule != NULL;
1891                 rule = rule->next) {
1892                    if (command == NULL) {
1893                            command = cmd_tail = ALLOC(Cmd_line);
1894                    } else {
1895                            cmd_tail->next = ALLOC(Cmd_line);
1896                            cmd_tail = cmd_tail->next;
1897                    }
1898                    *cmd_tail = *rule;
1899            }
1900            if (command) {
1901                    cmd_tail->next = NULL;
1902            }
1903            for (rule = command; rule != NULL; rule = rule->next) {
1904                    if (posix && (touch || quest) && !rule->always_exec) {
1905                            continue;
1906                    }
1907                    if (vpath_defined) {
1908                            rule->command_line =
1909                              vpath_translation(rule->command_line);
1910                    }
1911                    /* Echo command line, maybe. */
1912                    if ((rule->command_line->hash.length > 0) &&
1913                        !silent &&
1914                        (!rule->silent || do_not_exec_rule) &&
1915                        (report_dependencies_level == 0)) {
1916                            (void) printf("%s\n", rule->command_line->string_mb);
1917                            SEND_MTOOL_MSG(
1918                                    job_result_msg->appendOutput(AVO_STRDUP(rule->co
1919                            );
1920                    }
1921                    if (rule->command_line->hash.length > 0) {
1922                            SEND_MTOOL_MSG(
1923                                    (void) sprintf(mbstring,
1924                                                   NOCATGETS("%s/make.stdout.%d.%d.
1925                                                   tmpdir,
1926                                                   getpid(),
1927                                                   file_number++);

1929                                    int tmp_fd = mkstemp(mbstring);
1930                                    if(tmp_fd) {
1931                                            (void) close(tmp_fd);
1932                                    }

1934                                    stdout_file = strdup(mbstring);
1935                                    stderr_file = NULL;
1936                                    child_pid = pollResults(stdout_file,
1937                                                            (char *)NULL,
```

```
1938                                                    (char *)NULL);
1939                         );
1940                         /* Do assignment if command line prefixed with "=" */
1941                         if (rule->assign) {
1942                                 result = build_ok;
1943                                 do_assign(rule->command_line, target);
1944                         } else if (report_dependencies_level == 0) {
1945                                 /* Execute command line. */
1946 #ifdef DISTRIBUTED
1947                                 setvar_envvar((Avo_DoJobMsg *)NULL);
1948 #else
1949                                 setvar_envvar();
1950 #endif
1951                                 result = dosys(rule->command_line,
1952                                                 (Boolean) rule->ignore_error,
1953                                                 (Boolean) rule->make_refd,
1954                                                 /* ds 98.04.23 bug #4085164. make
1955                                                 false,
1956                                                 /* BOOLEAN(rule->silent &&
1957                                                         rule->ignore_error), */
1958                                                 (Boolean) rule->always_exec,
1959                                                 target,
1960                                                 send_mtool_msgs);
1988 #ifdef NSE
1989                                 nse_did_recursion= false;
1990 #endif
1961                                 check_state(temp_file_name);
1992 #ifdef NSE
1993                                 nse_check_cd(line);
1994 #endif
1962                         }
1963                         SEND_MTOOL_MSG(
1964                                 append_job_result_msg(job_result_msg);
1965                                 if (child_pid > 0) {
1966                                         kill(child_pid, SIGUSR1);
1967                                         while (!((waitpid(child_pid, 0, 0) == -1
1968                                                 && (errno == ECHILD)));
1969                                 }
1970                                 child_pid = 0;
1971                                 (void) unlink(stdout_file);
1972                                 retmem_mb(stdout_file);
1973                                 stdout_file = NULL;
1974                         );
1975                 } else {
1976                         result = build_ok;
1977                 }
1978                 if (result == build_failed) {
1979                         if (silent || rule->silent) {
1980                                 (void) printf(catgets(catd, 1, 242, "The followi
1981                                         rule->command_line->string_mb);
1982                                 SEND_MTOOL_MSG(
1983                                         job_result_msg->appendOutput(AVO_STRDUP(
1984                                         job_result_msg->appendOutput(AVO_STRDUP(
1985                                 );
1986                         }
1987                         if (!rule->ignore_error && !ignore_errors) {
1988                                 if (!continue_after_error) {
1989                                         SEND_MTOOL_MSG(
1990                                                 job_result_msg->setResult(job_ms
1991                                                 xdr_msg = (RWCollectable*)
1992                                                         job_result_msg;
1993                                                 xdr(&xdrs, xdr_msg);
1994                                                 (void) fflush(mtool_msgs_fp);
1995                                                 delete job_result_msg;
1996                                         );
1997                                         fatal(catgets(catd, 1, 244, "Command fai
```



```
1998                                                         target->string_mb);
1999                                 }
2000                                 /*
2001                                  * Make sure a failing command is not
2002                                  * saved in .make.state.
2003                                  */
2004                                 line->body.line.command_used = NULL;
2005                                 break;
2006                         } else {
2007                                 result = build_ok;
2008                         }
2009                 }
2010         }
2011         for (rule = command; rule != NULL; rule = cmd_tail) {
2012                 cmd_tail = rule->next;
2013                 free(rule);
2014         }
2015         command = NULL;
2016         SEND_MTOOL_MSG(
2017                 job_result_msg->setResult(job_msg_id, (result == build_ok) ? 0 :
2018                 xdr_msg = (RWCollectable*) job_result_msg;
2019                 xdr(&xdrs, xdr_msg);
2020                 (void) fflush(mtool_msgs_fp);

2022                 delete job_result_msg;
2023         );
2024         if (temp_file_name != NULL) {
2025                 free_name(temp_file_name);
2026         }
2027         temp_file_name = NULL;

2029         Property spro = get_prop(sunpro_dependencies->prop, macro_prop);
2030         if(spro != NULL) {
2031                 Name val = spro->body.macro.value;
2032                 if(val != NULL) {
2033                         free_name(val);
2034                         spro->body.macro.value = NULL;
2035                 }
2036         }
2037         spro = get_prop(sunpro_dependencies->prop, env_mem_prop);
2038         if(spro) {
2039                 char *val = spro->body.env_mem.value;
2040                 if(val != NULL) {
2041                         /*
2042                          * Do not return memory allocated for SUNPRO_DEPENDENCIE
2043                          * It will be returned in setvar_daemon() in macro.cc
2044                          */
2045                         //      retmem_mb(val);
2046                         spro->body.env_mem.value = NULL;
2047                 }
2048         }
2049
2050         return result;
2051 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    4960 Wed May 20 11:32:38 2015
new/usr/src/cmd/make/bin/globals.cc
make: unifdef for NSE (undefined)
**********************************************************
```

```
/*
 * 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.
 */

/*
 *      globals.cc
 *
 *      This declares all global variables
 */

/*
 * Included files
 */
#include <nl_types.h>
#include <mk/defs.h>
#include <sys/stat.h>

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Global variables used by make only
 */
        FILE            *dependency_report_file;

/*
 * Global variables used by make
 */
        Boolean         allrules_read=false;
        Name            posix_name;
        Name            svr4_name;
        Boolean         sdot_target;    /* used to identify s.m(/M)akefile */
        Boolean         all_parallel;                   /* TEAMWARE_MAKE_CMN */
        Boolean         assign_done;
        int foo;
```



```
        Boolean         build_failed_seen;
#ifdef DISTRIBUTED
        Boolean         building_serial;
#endif
        Name            built_last_make_run;
        Name            c_at;
#ifdef DISTRIBUTED
        Boolean         called_make = false;
#endif
        Boolean         cleanup;
        Boolean         close_report;
        Boolean         command_changed;
        Boolean         commands_done;
        Chain           conditional_targets;
        Name            conditionals;
        Boolean         continue_after_error;           /* '-k' */
        Property        current_line;
        Name            current_make_version;
        Name            current_target;
        short           debug_level;
        Cmd_line        default_rule;
        Name            default_rule_name;
        Name            default_target_to_build;
        Name            dmake_group;
        Name            dmake_max_jobs;
        Name            dmake_mode;
        DMake_mode      dmake_mode_type;
        Name            dmake_output_mode;
        DMake_output_mode       output_mode = txt1_mode;
        Name            dmake_odir;
        Name            dmake_rcfile;
        Name            done;
        Name            dot;
        Name            dot_keep_state;
        Name            dot_keep_state_file;
        Name            empty_name;
        Boolean         fatal_in_progress;
        int             file_number;
#if 0
        Boolean         filter_stderr;                  /* '-X' */
#endif
        Name            force;
        Name            ignore_name;
        Boolean         ignore_errors;                  /* '-i' */
        Boolean         ignore_errors_all;              /* '-i' */
        Name            init;
        int             job_msg_id;
        Boolean         keep_state;
        Name            make_state;
#ifdef TEAMWARE_MAKE_CMN
        timestruc_t     make_state_before;
#endif
        Dependency      makefiles_used;
        Name            makeflags;
//      Boolean         make_state_locked; // Moved to lib/mksh
        Name            make_version;
        char            mbs_buffer2[(MAXPATHLEN * MB_LEN_MAX)];
        char            *mbs_ptr;
        char            *mbs_ptr2;
        int             mtool_msgs_fd;
        Boolean         depinfo_already_read = false;
#ifdef NSE
        Name            derived_src;
        Boolean         nse;                            /* NSE on */
        Name            nse_backquote_seen;
        char            nse_depinfo_lockfile[MAXPATHLEN];
```

```
128         Boolean         nse_depinfo_locked;
129         Boolean         nse_did_recursion;
130         Name            nse_shell_var_used;
131         Boolean         nse_watch_vars = false;
132         wchar_t         current_makefile[MAXPATHLEN];
133 #endif
123         Boolean         no_action_was_taken = true;     /* true if we've not **
124                                                         ** run any command   */

126         Boolean         no_parallel = false;            /* TEAMWARE_MAKE_CMN */
127 #ifdef SGE_SUPPORT
128         Boolean         grid = false;                   /* TEAMWARE_MAKE_CMN */
129 #endif
130         Name            no_parallel_name;
131         Name            not_auto;
132         Boolean         only_parallel;                  /* TEAMWARE_MAKE_CMN */
133         Boolean         parallel;                       /* TEAMWARE_MAKE_CMN */
134         Name            parallel_name;
135         Name            localhost_name;
136         int             parallel_process_cnt;
137         Percent         percent_list;
138         Dyntarget       dyntarget_list;
139         Name            plus;
140         Name            pmake_machinesfile;
141         Name            precious;
142         Name            primary_makefile;
143         Boolean         quest;                          /* '-q' */
144         short           read_trace_level;
145         Boolean         reading_dependencies = false;
146         Name            recursive_name;
147         int             recursion_level;
148         short           report_dependencies_level = 0;  /* -P */
149         Boolean         report_pwd;
150         Boolean         rewrite_statefile;
151         Running         running_list;
152         char            *sccs_dir_path;
153         Name            sccs_get_name;
154         Name            sccs_get_posix_name;
155         Cmd_line        sccs_get_rule;
156         Cmd_line        sccs_get_org_rule;
157         Cmd_line        sccs_get_posix_rule;
158         Name            get_name;
159         Cmd_line        get_rule;
160         Name            get_posix_name;
161         Cmd_line        get_posix_rule;
162         Boolean         send_mtool_msgs;                /* '-K' */
163         Boolean         all_precious;
164         Boolean         silent_all;                     /* '-s' */
165         Boolean         report_cwd;                     /* '-w' */
166         Boolean         silent;                         /* '-s' */
167         Name            silent_name;
168         char            *stderr_file = NULL;
169         char            *stdout_file = NULL;
170 #ifdef SGE_SUPPORT
171         char            script_file[MAXPATHLEN] = "";
172 #endif
173         Boolean         stdout_stderr_same;
174         Dependency      suffixes;
175         Name            suffixes_name;
176         Name            sunpro_dependencies;
177         Boolean         target_variants;
178         const char      *tmpdir = NOCATGETS("/tmp");
179         const char      *temp_file_directory = NOCATGETS(".");
180         Name            temp_file_name;
181         short           temp_file_number;
182         time_t          timing_start;
```



```
183         wchar_t         *top_level_target;
184         Boolean         touch;                          /* '-t' */
185         Boolean         trace_reader;                   /* '-D' */
186         Boolean         build_unconditional;            /* '-u' */
187         pathpt          vroot_path = VROOT_DEFAULT;
188         Name            wait_name;
189         wchar_t         wcs_buffer2[MAXPATHLEN];
190         wchar_t         *wcs_ptr;
191         wchar_t         *wcs_ptr2;
192         nl_catd         catd;
193         long int        hostid;

195 /*
196  * File table of contents
197  */
```

```
**********************************************************
   43414 Wed May 20 11:32:39 2015
new/usr/src/cmd/make/bin/implicit.cc
make: unifdef for NSE (undefined)
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
  23  * Use is subject to license terms.
  24  */

  26 /*
  27  *      implicit.c
  28  *
  29  *      Handle suffix and percent rules
  30  */

  32 /*
  33  * Included files
  34  */
  35 #include <mk/defs.h>
  36 #include <mksh/macro.h>         /* expand_value() */
  37 #include <mksh/misc.h>          /* retmem() */

  39 /*
  40  * Defined macros
  41  */

  43 /*
  44  * typedefs & structs
  45  */

  47 /*
  48  * Static variables
  49  */
  50 static  wchar_t         WIDE_NULL[1] = {(wchar_t) nul_char};

  52 /*
  53  * File table of contents
  54  */
  55 extern  Doname          find_suffix_rule(Name target, Name target_body, Name tar
  56 extern  Doname          find_ar_suffix_rule(register Name target, Name true_targ
  57 extern  Doname          find_double_suffix_rule(register Name target, Property *
  58 extern  void            build_suffix_list(register Name target_suffix);
  59 extern  Doname          find_percent_rule(register Name target, Property *comman
  60 static  void            create_target_group_and_dependencies_list(Name target, P
  61 static  Boolean         match_found_with_pattern(Name target, Percent pat_rule,
```



```
  62 static  void            construct_string_from_pattern(Percent pat_rule, String p
  63 static  Boolean         dependency_exists(Name target, Property line);
  64 extern  Property        maybe_append_prop(Name, Property_id);
  65 extern  void            add_target_to_chain(Name target, Chain * query);

  67 /*
  68  *      find_suffix_rule(target, target_body, target_suffix, command, rechecking
  69  *
  70  *      Does the lookup for single and double suffix rules.
  71  *      It calls build_suffix_list() to build the list of possible suffixes
  72  *      for the given target.
  73  *      It then scans the list to find the first possible source file that
  74  *      exists. This is done by concatenating the body of the target name
  75  *      (target name less target suffix) and the source suffix and checking
  76  *      if the resulting file exists.
  77  *
  78  *      Return value:
  79  *                              Indicates if search failed or not
  80  *
  81  *      Parameters:
  82  *              target          The target we need a rule for
  83  *              target_body     The target name without the suffix
  84  *              target_suffix   The suffix of the target
  85  *              command         Pointer to slot to deposit cmd in if found
  86  *              rechecking      true if we are rechecking target which depends
  87  *                              on conditional macro and keep_state is set
  88  *
  89  *      Global variables used:
  90  *              debug_level     Indicates how much tracing to do
  91  *              recursion_level Used for tracing
  92  */

  94 extern int printf (const char *, ...);

  96 static Boolean actual_doname = false;

  98 /* /tolik/
  99  * fix bug 1247448: Suffix Rules failed when combine with Pattern Matching Rules
 100  * When make attemps to apply % rule it didn't look for a single suffix rule bec
 101  * if "doname" is called from "find_percent_rule" argument "implicit" is set to
 102  * and find_suffix_rule was not called. I've commented the checking of "implicit
 103  * in "doname" and make got infinite recursion for SVR4 tilde rules.
 104  * Usage of "we_are_in_tilde" is intended to avoid this recursion.
 105  */

 107 static Boolean we_are_in_tilde = false;

 109 Doname
 110 find_suffix_rule(Name target, Name target_body, Name target_suffix, Property *co
 111 {
 112         static wchar_t          static_string_buf_3M [ 3 * MAXPATHLEN ];
 113         Name                    true_target = target;
 114         wchar_t                 *sourcename = (wchar_t*)static_string_buf_3M;
 115         register wchar_t        *put_suffix;
 116         register Property       source_suffix;
 117         register Name           source;
 118         Doname                  result;
 119         register Property       line;
 120         extern Boolean          tilde_rule;
 121         Boolean                 name_found = true;
 122         Boolean                 posix_tilde_attempt = true;
 123         int                     src_len = MAXPATHLEN + strlen(target_body->strin

 125         /*
 126          * To avoid infinite recursion
 127          */
```

```
128         if(we_are_in_tilde) {
129                 we_are_in_tilde = false;
130                 return(build_dont_know);
131         }

133         /*
134          * If the target is a constructed one for a "::" target,
135          * we need to consider that.
136          */
137         if (target->has_target_prop) {
138                 true_target = get_prop(target->prop,
139                                        target_prop)->body.target.target;
140         }
141         if (debug_level > 1) {
142                 (void) printf(NOCATGETS("%*sfind_suffix_rule(%s,%s,%s)\n"),
143                               recursion_level,
144                               "",
145                               true_target->string_mb,
146                               target_body->string_mb,
147                               target_suffix->string_mb);
148         }
149         if (command != NULL) {
150                 if ((true_target->suffix_scan_done == true) && (*command == NULL
151                         return build_ok;
152                 }
153         }
154         true_target->suffix_scan_done = true;
155         /*
156          * Enter all names from the directory where the target lives as
157          * files that makes sense.
158          * This will make finding the synthesized source possible.
159          */
160         read_directory_of_file(target_body);
161         /* Cache the suffixes for this target suffix if not done. */
162         if (!target_suffix->has_read_suffixes) {
163                 build_suffix_list(target_suffix);
164         }
165         /* Preload the sourcename vector with the head of the target name. */
166         if (src_len >= sizeof(static_string_buf_3M)) {
167                 sourcename = ALLOC_WC(src_len);
168         }
169         (void) mbstowcs(sourcename,
170                       target_body->string_mb,
171                       (int) target_body->hash.length);
172         put_suffix = sourcename + target_body->hash.length;
173         /* Scan the suffix list for the target if one exists. */
174         if (target_suffix->has_suffixes) {
175 posix_attempts:
176                 for (source_suffix = get_prop(target_suffix->prop,
177                                               suffix_prop);
178                      source_suffix != NULL;
179                      source_suffix = get_prop(source_suffix->next,
180                                               suffix_prop)) {
181                         /* Build the synthesized source name. */
182                         (void) mbstowcs(put_suffix,
183                                         source_suffix->body.
184                                         suffix.suffix->string_mb,
185                                         (int) source_suffix->body.
186                                         suffix.suffix->hash.length);
187                         put_suffix[source_suffix->body.
188                                    suffix.suffix->hash.length] =
189                           (int) nul_char;
190                         if (debug_level > 1) {
191                                WCSTOMBS(mbs_buffer, sourcename);
192                                (void) printf(catgets(catd, 1, 218, "%*sTrying %
193                                              recursion_level,
```



```
194                                               "",
195                                               mbs_buffer);
196                         }
197                         source = getname_fn(sourcename, FIND_LENGTH, false, &nam
198                         /*
199                          * If the source file is not registered as
200                          * a file, this source suffix did not match.
201                          */
202                         if(vpath_defined && !posix && !svr4) {
203                                 (void) exists(source);
204                         }
205                         if (!source->stat.is_file) {
206                            if(!(posix|svr4))
207                            {
208                                 if(!name_found) {
209                                         free_name(source);
210                                 }
211                                 continue;
212                            }

214                            /* following code will ensure that the corresponding
215                            ** tilde rules are executed when corresponding s. fil
216                            ** exists in the current directory. Though the curren
217                            ** target ends with a ~ character, there wont be any
218                            ** any file in the current directory with that suffix
219                            ** as it's fictitious. Even if it exists, it'll
220                            ** execute all the rules for the ~ target.
221                            */

223                            if(source->string_mb[source->hash.length - 1] == '~'
224                               ( svr4 || posix_tilde_attempt ) )
225                            {
226                                 char *p, *np;
227                                 char *tmpbuf;

229                                 tmpbuf = getmem(source->hash.length + 8);
230                                 /* + 8 to add "s." or "SCCS/s." */
231                                 memset(tmpbuf,0,source->hash.length + 8);
232                                 source->string_mb[source->hash.length - 1] = '\0
233                                 if(p = (char *) memchr((char *)source->string_mb
234                                 {
235                                   while(1) {
236                                     if(np = (char *) memchr((char *)p+1,'/',sour
237                                       p = np;
238                                     } else {break;}
239                                   }
240                                   /* copy everything including '/' */
241                                   strncpy(tmpbuf, source->string_mb, p - source-
242                                   strcat(tmpbuf, NOCATGETS("s."));
243                                   strcat(tmpbuf, p+1);
244                                   retmem((wchar_t *) source->string_mb);
245                                   source->string_mb = tmpbuf;
246
247                                 } else {
248                                   strcpy(tmpbuf, NOCATGETS("s."));
249                                   strcat(tmpbuf, source->string_mb);
250                                   retmem((wchar_t *) source->string_mb);
251                                   source->string_mb = tmpbuf;
252
253                                 }
254                                 source->hash.length = strlen(source->string_mb);
255                                 if(exists(source) == file_doesnt_exist)
256                                   continue;
257                                 tilde_rule = true;
258                                 we_are_in_tilde = true;
259                            } else {
```

```
260                                 if(!name_found) {
261                                         free_name(source);
262                                 }
263                                 continue;
264                             }
265                         } else {
266                            if(posix && posix_tilde_attempt) {
267                                 if(exists(source) == file_doesnt_exist) {
268                                         if(!name_found) {
269                                                 free_name(source);
270                                         }
271                                         continue;
272                                 }
273                             }
274                         }
275
276                         if (command != NULL) {
277                                 if(!name_found) {
278                                         store_name(source);
279                                 }
280                                 /*
281                                  * The source file is a file.
282                                  * Make sure it is up to date.
283                                  */
284                                 if (dependency_exists(source,
285                                                       get_prop(target->prop,
286                                                                line_prop))) {
287                                         result = (Doname) source->state;
288                                 } else {
289 #ifdef NSE
290                                         nse_check_derived_src(target, source->st
291                                          source_suffix->body.suffix.command_temp
292 #endif
289 #if 0  /* with_squiggle sends false, which is buggy. : djay */
290                                         result = doname(source,
291                                                         (Boolean) source_suffix-
292                                                         suffix.suffix->with_squi
293                                                         true);
294 #else
295                                         result = doname(source,
296                                                         true,
297                                                         true);
298 #endif
299                                 }
300                         } else {
301                                 result = target_can_be_built(source);
302
303                                 if (result == build_ok) {
304                                         return result;
305                                 } else {
306                                         if(!name_found) {
307                                                 free_name(source);
308                                         }
309                                         continue;
310                                 }
311                         }
312
313                         switch (result) {
314                         case build_dont_know:
315                                 /*
316                                  * If we still can't build the source,
317                                  * this rule is not a match,
318                                  * try the next one.
319                                  */
320                                 if (source->stat.time == file_doesnt_exist) {
321                                         if(!name_found) {
```



```
322                                                 free_name(source);
323                                         }
324                                         continue;
325                                 }
326                         case build_running:
327                                 if(!name_found) {
328                                         store_name(source);
329                                 }
330                                 true_target->suffix_scan_done = false;
331                                 line = maybe_append_prop(target, line_prop);
332                                 enter_dependency(line, source, false);
333                                 line->body.line.target = true_target;
334                                 return build_running;
335                         case build_ok:
336                                 if(!name_found) {
337                                         store_name(source);
338                                 }
339                                 break;
340                         case build_failed:
341                                 if(!name_found) {
342                                         store_name(source);
343                                 }
344                                 if (sourcename != static_string_buf_3M) {
345                                         retmem(sourcename);
346                                 }
347                                 return build_failed;
348                         }
349
350                         if (debug_level > 1) {
351                                 WCSTOMBS(mbs_buffer, sourcename);
352                                 (void) printf(catgets(catd, 1, 219, "%*sFound %s
353                                               recursion_level,
354                                               "",
355                                               mbs_buffer);
356                         }
357
358                         if (source->depends_on_conditional) {
359                                 target->depends_on_conditional = true;
360                         }
361 /*
362  * Since it is possible that the same target is built several times during
363  * the make run, we have to patch the target with all information we found
364  * here. Thus, the target will have an explicit rule the next time around.
365  */
366                         line = maybe_append_prop(target, line_prop);
367                         if (*command == NULL) {
368                                 *command = line;
369                         }
370                         if ((source->stat.time > (*command)->body.line.dependenc
371                             (debug_level > 1)) {
372                                 (void) printf(catgets(catd, 1, 220, "%*sDate(%s)
373                                               recursion_level,
374                                               "",
375                                               source->string_mb,
376                                               time_to_string(source->
377                                                              stat.time),
378                                               true_target->string_mb,
379                                               time_to_string((*command)->
380                                                              body.line.
381                                                              dependency_time));
382                         }
383                         /*
384                          * Determine if this new dependency made the
385                          * target out of date.
386                          */
387                         (*command)->body.line.dependency_time =
```

```
388                           MAX((*command)->body.line.dependency_time,
389                               source->stat.time);
390                   Boolean out_of_date;
391                   if (target->is_member) {
392                           out_of_date = (Boolean) OUT_OF_DATE_SEC(target->
393                                                                   (*comman
394                   } else {
395                           out_of_date = (Boolean) OUT_OF_DATE(target->stat
396                                                               (*command)->
397                   }
398                   if (build_unconditional || out_of_date) {
399                           if(!rechecking) {
400                                   line->body.line.is_out_of_date = true;
401                           }
402                           if (debug_level > 0) {
403                                   (void) printf(catgets(catd, 1, 221, "%*s
404                                                 recursion_level,
405                                                 "",
406                                                 true_target->string_mb,
407                                                 source_suffix->body.suffix
408                                                 target_suffix->string_mb,
409                                                 source->string_mb);
410                           }
411                   }
412                   /*
413                    * Add the implicit rule as the target's explicit
414                    * rule if none actually given, and register
415                    * dependency.
416                    * The time checking above really should be
417                    * conditional on actual use of implicit rule
418                    * as well.
419                    */
420                   line->body.line.sccs_command = false;
421                   if (line->body.line.command_template == NULL) {
422                           line->body.line.command_template =
423                             source_suffix->body.suffix.command_template;
424                   }
425                   enter_dependency(line, source, false);
426                   line->body.line.target = true_target;
427                   /*
428                    * Also make sure the rule is built with
429                    * $* and $< bound properly.
430                    */
431                   line->body.line.star = target_body;
432                   if(svr4|posix) {
433                     char * p;
434                     char tstr[256];
435                     extern Boolean dollarless_flag;
436                     extern Name dollarless_value;

438                     if(tilde_rule) {
439                         MBSTOWCS(wcs_buffer, NOCATGETS(source->string_mb))
440                         dollarless_value = GETNAME(wcs_buffer,FIND_LENGTH)
441                     }
442                     else {
443                             dollarless_flag = false;
444                     }
445                   }
446                   line->body.line.less = source;
447                   line->body.line.percent = NULL;
448                   add_target_to_chain(source, &(line->body.line.query));
449                   if (sourcename != static_string_buf_3M) {
450                           retmem(sourcename);
451                   }
452                   return build_ok;
453           }
```



```
454                   if(posix && posix_tilde_attempt) {
455                           posix_tilde_attempt = false;
456                           goto posix_attempts;
457                   }
458                   if ((command != NULL) &&
459                       ((*command) != NULL) &&
460                       ((*command)->body.line.star == NULL)) {
461                           (*command)->body.line.star = target_body;
462                   }
463           }
464           if (sourcename != static_string_buf_3M) {
465                   retmem(sourcename);
466           }
467           /* Return here in case no rule matched the target */
468           return build_dont_know;
469   }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   99422 Wed May 20 11:32:39 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef for NSE (undefined)
**********************************************************
_____unchanged_portion_omitted_
```

```
167 #endif

169 extern  Name            normalize_name(register wchar_t *name_string, register i

171 extern  int             main(int, char * []);

173 static  void            append_makeflags_string(Name, String);
174 static  void            doalarm(int);
175 static  void            enter_argv_values(int , char **, ASCII_Dyn_Array *);
176 static  void            make_targets(int, char **, Boolean);
177 static  int             parse_command_option(char);
178 static  void            read_command_options(int, char **);
179 static  void            read_environment(Boolean);
180 static  void            read_files_and_state(int, char **);
181 static  Boolean         read_makefile(Name, Boolean, Boolean, Boolean);
182 static  void            report_recursion(Name);
183 static  void            set_sgs_support(void);
184 static  void            setup_for_projectdir(void);
185 static  void            setup_makeflags_argv(void);
186 static  void            report_dir_enter_leave(Boolean entering);

188 extern void expand_value(Name, register String , Boolean);

190 #ifdef DISTRIBUTED
191         extern  int             dmake_ofd;
192         extern  FILE*           dmake_ofp;
193         extern  int             rxmPid;
194         extern  XDR             xdrs_out;
195 #endif
196 #ifdef TEAMWARE_MAKE_CMN
197         extern  char            verstring[];
198 #endif

200 jmp_buf jmpbuffer;
201 extern nl_catd catd;

203 /*
204  *      main(argc, argv)
205  *
206  *      Parameters:
207  *              argc                    You know what this is
208  *              argv                    You know what this is
209  *
210  *      Static variables used:
211  *              list_all_targets        make -T seen
212  *              trace_status            make -p seen
213  *
214  *      Global variables used:
215  *              debug_level             Should we trace make actions?
216  *              keep_state              Set if .KEEP_STATE seen
217  *              makeflags               The Name "MAKEFLAGS", used to get macro
218  *              remote_command_name     Name of remote invocation cmd ("on")
219  *              running_list            List of parallel running processes
220  *              stdout_stderr_same      true if stdout and stderr are the same
221  *              auto_dependencies       The Name "SUNPRO_DEPENDENCIES"
222  *              temp_file_directory     Set to the dir where we create tmp file
223  *              trace_reader            Set to reflect tracing status
224  *              working_on_targets      Set when building user targets
225  */
226 int
```



```
227 main(int argc, char *argv[])
228 {
229         /*
230          * cp is a -> to the value of the MAKEFLAGS env var,
231          * which has to be regular chars.
232          */
233         register char           *cp;
234         char                    make_state_dir[MAXPATHLEN];
235         Boolean                 parallel_flag = false;
236         char                    *prognameptr;
237         char                    *slash_ptr;
238         mode_t                  um;
239         int                     i;
240 #ifdef TEAMWARE_MAKE_CMN
241         struct itimerval        value;
242         char                    def_dmakerc_path[MAXPATHLEN];
243         Name                    dmake_name, dmake_name2;
244         Name                    dmake_value, dmake_value2;
245         Property                prop, prop2;
246         struct stat             statbuf;
247         int                     statval;
248 #endif

250 #ifndef PARALLEL
251         struct stat             out_stat, err_stat;
252 #endif
253         hostid = gethostid();
254 #ifdef TEAMWARE_MAKE_CMN
255         avo_get_user(NULL, NULL); // Initialize user name
256 #endif
257         bsd_signals();

259         (void) setlocale(LC_ALL, "");


262 #ifdef DMAKE_STATISTICS
263         if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
264                 getname_stat = true;
265         }
266 #endif


269         /*
270          * avo_init() sets the umask to 0.  Save it here and restore
271          * it after the avo_init() call.
272          */
273 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
274         um = umask(0);
275         avo_init(argv[0]);
276         umask(um);

278 #ifdef USE_DMS_CCR
279         usageTracking = new Avo_usage_tracking(NOCATGETS("dmake"), argc, argv);
280 #else
281         cleanup = new Avo_cleanup(NOCATGETS("dmake"), argc, argv);
282 #endif
283 #endif

285 #if defined(TEAMWARE_MAKE_CMN)
286         catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
287         libcli_init();

289 #ifdef _CHECK_UPDATE_H
290         /* This is for dmake only (not for Solaris make).
291          * Check (in background) if there is an update (dmake patch)
292          * and inform user
```

```
293          */
294         {
295                 Avo_err         *err;
296                 char            *dir;
297                 err = avo_find_run_dir(&dir);
298                 if (AVO_OK == err) {
299                         AU_check_update_service(NOCATGETS("Dmake"), dir);
300                 }
301         }
302 #endif /* _CHECK_UPDATE_H */
303 #endif

305 // ---> fprintf(stderr, catgets(catd, 15, 666, "--- SUN make ---\n"));


308 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
309 /*
310  * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
311  * from avo_util library.
312  */
313         libmksdmsi18n_init();
314 #ifdef USE_DMS_CCR
315         libpubdmsi18n_init();
316 #endif
317 #endif


320 #ifndef TEAMWARE_MAKE_CMN
321         textdomain(NOCATGETS("SUNW_SPRO_MAKE"));
322 #endif /* TEAMWARE_MAKE_CMN */

324 #ifdef TEAMWARE_MAKE_CMN
325         g_argc = argc;
326         g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
327         for (i = 0; i < argc; i++) {
328                 g_argv[i] = argv[i];
329         }
330         g_argv[i] = NULL;
331 #endif /* TEAMWARE_MAKE_CMN */

333         /*
334          * Set argv_zero_string to some form of argv[0] for
335          * recursive MAKE builds.
336          */

338         if (*argv[0] == (int) slash_char) {
339                 /* argv[0] starts with a slash */
340                 argv_zero_string = strdup(argv[0]);
341         } else if (strchr(argv[0], (int) slash_char) == NULL) {
342                 /* argv[0] contains no slashes */
343                 argv_zero_string = strdup(argv[0]);
344         } else {
345                 /*
346                  * argv[0] contains at least one slash,
347                  * but doesn't start with a slash
348                  */
349                 char    *tmp_current_path;
350                 char    *tmp_string;

352                 tmp_current_path = get_current_path();
353                 tmp_string = getmem(strlen(tmp_current_path) + 1 +
354                                     strlen(argv[0]) + 1);
355                 (void) sprintf(tmp_string,
356                                "%s/%s",
357                                tmp_current_path,
358                                argv[0]);
```



```
359                 argv_zero_string = strdup(tmp_string);
360                 retmem_mb(tmp_string);
361         }

363         /*
364          * The following flags are reset if we don't have the
365          * (.nse_depinfo or .make.state) files locked and only set
366          * AFTER the file has been locked. This ensures that if the user
367          * interrupts the program while file_lock() is waiting to lock
368          * the file, the interrupt handler doesn't remove a lock
369          * that doesn't belong to us.
370          */
371         make_state_lockfile = NULL;
372         make_state_locked = false;

374 #ifdef NSE
375         nse_depinfo_lockfile[0] = '\0';
376         nse_depinfo_locked = false;
377 #endif

375         /*
376          * look for last slash char in the path to look at the binary
377          * name. This is to resolve the hard link and invoke make
378          * in svr4 mode.
379          */

381         /* Sun OS make standart */
382         svr4 = false;
383         posix = false;
384         if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
385                 svr4 = false;
386                 posix = true;
387         } else {
388                 prognameptr = strrchr(argv[0], '/');
389                 if(prognameptr) {
390                         prognameptr++;
391                 } else {
392                         prognameptr = argv[0];
393                 }
394                 if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
395                         svr4 = true;
396                         posix = false;
397                 }
398         }
399         if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
400            svr4 = true;
401            posix = false;
402         }

404         /*
405          * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
406          */
407         char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
408         if (dmake_compat_mode_var != NULL) {
409                 if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
410                         gnu_style = true;
411                 }
412                 //svr4 = false;
413                 //posix = false;
414         }

416         /*
417          * Temporary directory set up.
418          */
419         char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
420         if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
```

```
421                 strcpy(mbs_buffer, tmpdir_var);
422                 for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
423                         *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
424                         *tmpdir_var = '\0');
425                 if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
426                         sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
427                                 mbs_buffer, getpid());
428                         int fd = mkstemp(mbs_buffer2);
429                         if (fd >= 0) {
430                                 close(fd);
431                                 unlink(mbs_buffer2);
432                                 tmpdir = strdup(mbs_buffer);
433                         }
434                 }
435         }

437 #ifndef PARALLEL
438         /* find out if stdout and stderr point to the same place */
439         if (fstat(1, &out_stat) < 0) {
440                 fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
441         }
442         if (fstat(2, &err_stat) < 0) {
443                 fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
444         }
445         if ((out_stat.st_dev == err_stat.st_dev) &&
446             (out_stat.st_ino == err_stat.st_ino)) {
447                 stdout_stderr_same = true;
448         } else {
449                 stdout_stderr_same = false;
450         }
451 #else
452         stdout_stderr_same = false;
453 #endif
454         /* Make the vroot package scan the path using shell semantics */
455         set_path_style(0);

457         setup_char_semantics();

459         setup_for_projectdir();

461         /*
462          * If running with .KEEP_STATE, curdir will be set with
463          * the connected directory.
464          */
465         (void) atexit(cleanup_after_exit);

467         load_cached_names();

469 /*
470  *      Set command line flags
471  */
472         setup_makeflags_argv();
473         read_command_options(mf_argc, mf_argv);
474         read_command_options(argc, argv);
475         if (debug_level > 0) {
476                 cp = getenv(makeflags->string_mb);
477                 (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
478         }

480         setup_interrupt(handle_interrupt);

482         read_files_and_state(argc, argv);

484 #ifdef TEAMWARE_MAKE_CMN
485         /*
486          * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
```



```
487          */
488         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
489         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
490         prop2 = get_prop(dmake_name2->prop, macro_prop);
491         if (prop2 == NULL) {
492                 /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
493                 output_mode = txt1_mode;
494         } else {
495                 dmake_value2 = prop2->body.macro.value;
496                 if ((dmake_value2 == NULL) ||
497                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
498                         output_mode = txt1_mode;
499                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
500                         output_mode = txt2_mode;
501                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))
502                         output_mode = html1_mode;
503                 } else {
504                         warning(catgets(catd, 1, 352, "Unsupported value '%s' fo
505                               dmake_value2->string_mb);
506                 }
507         }
508         /*
509          * Find the dmake_mode: distributed, parallel, or serial.
510          */
511     if ((!pmake_cap_r_specified) &&
512         (!pmake_machinesfile_specified)) {
513         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
514         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
515         prop2 = get_prop(dmake_name2->prop, macro_prop);
516         if (prop2 == NULL) {
517                 /* DMAKE_MODE not defined, default to distributed mode */
518                 dmake_mode_type = distributed_mode;
519                 no_parallel = false;
520         } else {
521                 dmake_value2 = prop2->body.macro.value;
522                 if ((dmake_value2 == NULL) ||
523                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed"))
524                         dmake_mode_type = distributed_mode;
525                         no_parallel = false;
526                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel
527                         dmake_mode_type = parallel_mode;
528                         no_parallel = false;
529 #ifdef SGE_SUPPORT
530                         grid = false;
531                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("grid")))
532                         dmake_mode_type = parallel_mode;
533                         no_parallel = false;
534                         grid = true;
535 #endif
536                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
537                         dmake_mode_type = serial_mode;
538                         no_parallel = true;
539                 } else {
540                         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
541                 }
542         }

544         if ((!list_all_targets) &&
545             (report_dependencies_level == 0)) {
546                 /*
547                  * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
548                  * They could be defined in the env, in the makefile, or on the
549                  * command line.
550                  * If neither is defined, and $(HOME)/.dmakerc does not exists,
551                  * then print a message, and default to parallel mode.
552                  */
```

```
553 #ifdef DISTRIBUTED
554                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
555                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
556                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
557                 dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
558                 if ((((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
559                      ((dmake_value = prop->body.macro.value) == NULL)) &&
560                     (((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL)
561                      ((dmake_value2 = prop2->body.macro.value) == NULL))) {
562                         Boolean empty_dmakerc = true;
563                         char *homedir = getenv(NOCATGETS("HOME"));
564                         if ((homedir != NULL) && (strlen(homedir) < (sizeof(def_
565                                 sprintf(def_dmakerc_path, NOCATGETS("%s/.dmakerc
566                                 if ((((statval = stat(def_dmakerc_path, &statbuf
567                                         ((statval == 0) && (statbuf.st_size == 0
568                                 } else {
569                                         Avo_dmakerc     *rcfile = new Avo_dmaker
570                                         Avo_err         *err = rcfile->read(def_
571                                         if (err) {
572                                                 fatal(err->str);
573                                         }
574                                         empty_dmakerc = rcfile->was_empty();
575                                         delete rcfile;
576                                 }
577                         }
578                         if (empty_dmakerc) {
579                                 if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
580                                         putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE
581                                         (void) fprintf(stdout, catgets(catd, 1,
582                                         (void) fprintf(stdout, catgets(catd, 1,
583                                 }
584                                 dmake_mode_type = parallel_mode;
585                                 no_parallel = false;
586                         }
587                 }
588 #else
589                 if(dmake_mode_type == distributed_mode) {
590                         (void) fprintf(stdout, NOCATGETS("dmake: Distributed mod
591                         (void) fprintf(stdout, NOCATGETS("       Defaulting to p
592                         dmake_mode_type = parallel_mode;
593                         no_parallel = false;
594                 }
595 #endif  /* DISTRIBUTED */
596         }
597     }
598 #endif

600 #ifdef TEAMWARE_MAKE_CMN
601         parallel_flag = true;
602         /* XXX - This is a major hack for DMake/Licensing. */
603         if (getenv(NOCATGETS("DMAKE_CHILD")) == NULL) {
604                 if (!avo_cli_search_license(argv[0], dmake_exit_callback, TRUE,
605                         /*
606                          * If the user can not get a TeamWare license,
607                          * default to serial mode.
608                          */
609                         dmake_mode_type = serial_mode;
610                         no_parallel = true;
611                 } else {
612                         putenv(NOCATGETS("DMAKE_CHILD=TRUE"));
613                 }
614                 start_time = time(NULL);
615                 /*
616                  * XXX - Hack to disable SIGALRM's from licensing library's
617                  *       setitimer().
618                  */
```



```
619                 value.it_interval.tv_sec = 0;
620                 value.it_interval.tv_usec = 0;
621                 value.it_value.tv_sec = 0;
622                 value.it_value.tv_usec = 0;
623                 (void) setitimer(ITIMER_REAL, &value, NULL);
624         }

626 //
627 // If dmake is running with -t option, set dmake_mode_type to serial.
628 // This is done because doname() calls touch_command() that runs serially.
629 // If we do not do that, maketool will have problems.
630 //
631         if(touch) {
632                 dmake_mode_type = serial_mode;
633                 no_parallel = true;
634         }
635 #else
636         parallel_flag = false;
637 #endif

639 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
640         /*
641          * Check whether stdout and stderr are physically same.
642          * This is in order to decide whether we need to redirect
643          * stderr separately from stdout.
644          * This check is performed only if __DMAKE_SEPARATE_STDERR
645          * is not set. This variable may be used in order to preserve
646          * the 'old' behaviour.
647          */
648         out_err_same = true;
649         char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
650         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
651                 struct stat stdout_stat;
652                 struct stat stderr_stat;
653                 if( (fstat(1, &stdout_stat) == 0)
654                  && (fstat(2, &stderr_stat) == 0) )
655                 {
656                         if( (stdout_stat.st_dev != stderr_stat.st_dev)
657                          || (stdout_stat.st_ino != stderr_stat.st_ino) )
658                         {
659                                 out_err_same = false;
660                         }
661                 }
662         }
663 #endif

665 #ifdef DISTRIBUTED
666         /*
667          * At this point, DMake should startup an rxm with any and all
668          * DMake command line options. Rxm will, among other things,
669          * read the rc file.
670          */
671         if ((!list_all_targets) &&
672             (report_dependencies_level == 0) &&
673             (dmake_mode_type == distributed_mode)) {
674                 startup_rxm();
675         }
676 #endif
677
678 /*
679  *      Enable interrupt handler for alarms
680  */
681         (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

683 /*
684  *      Check if make should report
```

```
 */
        if (getenv(sunpro_dependencies->string_mb) != NULL) {
                FILE    *report_file;

                report_dependency("");
                report_file = get_report_file();
                if ((report_file != NULL) && (report_file != (FILE*)-1)) {
                        (void) fprintf(report_file, "\n");
                }
        }

/*
 *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
 *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly
 *      and NSE_DEP.
 */
        if (keep_state) {
                maybe_append_prop(sunpro_dependencies, macro_prop)->
                  body.macro.exported = true;
#ifdef NSE
                (void) setenv(NOCATGETS("NSE_DEP"), get_current_path());
#endif
        } else {
                maybe_append_prop(sunpro_dependencies, macro_prop)->
                  body.macro.exported = false;
        }

        working_on_targets = true;
        if (trace_status) {
                dump_make_state();
                fclose(stdout);
                fclose(stderr);
                exit_status = 0;
                exit(0);
        }
        if (list_all_targets) {
                dump_target_list();
                fclose(stdout);
                fclose(stderr);
                exit_status = 0;
                exit(0);
        }
        trace_reader = false;

        /*
         * Set temp_file_directory to the directory the .make.state
         * file is written to.
         */
        if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
                temp_file_directory = strdup(get_current_path());
        } else {
                *slash_ptr = (int) nul_char;
                (void) strcpy(make_state_dir, make_state->string_mb);
                *slash_ptr = (int) slash_char;
                   /* when there is only one slash and it's the first
                   ** character, make_state_dir should point to '/'.
                   */
                if(make_state_dir[0] == '\0') {
                   make_state_dir[0] = '/';
                   make_state_dir[1] = '\0';
                }
                if (make_state_dir[0] == (int) slash_char) {
                        temp_file_directory = strdup(make_state_dir);
                } else {
                        char    tmp_current_path2[MAXPATHLEN];
```



```
                        (void) sprintf(tmp_current_path2,
                                        "%s/%s",
                                        get_current_path(),
                                        make_state_dir);
                        temp_file_directory = strdup(tmp_current_path2);
                }
        }

#ifdef DISTRIBUTED
        building_serial = false;
#endif

        report_dir_enter_leave(true);

        make_targets(argc, argv, parallel_flag);

        report_dir_enter_leave(false);

#ifdef NSE
        exit(nse_exit_status());
#else
        if (build_failed_ever_seen) {
                if (posix) {
                        exit_status = 1;
                }
                exit(1);
        }
        exit_status = 0;
        exit(0);
#endif
        /* NOTREACHED */
}

/*
 *      cleanup_after_exit()
 *
 *      Called from exit(), performs cleanup actions.
 *
 *      Parameters:
 *              status          The argument exit() was called with
 *              arg             Address of an argument vector to
 *                              cleanup_after_exit()
 *
 *      Global variables used:
 *              command_changed Set if we think .make.state should be rewritten
 *              current_line    Is set we set commands_changed
 *              do_not_exec_rule
 *                              True if -n flag on
 *              done            The Name ".DONE", rule we run
 *              keep_state      Set if .KEEP_STATE seen
 *              parallel        True if building in parallel
 *              quest           If -q is on we do not run .DONE
 *              report_dependencies
 *                              True if -P flag on
 *              running_list    List of parallel running processes
 *              temp_file_name  The temp file is removed, if any
 *              catd    the message catalog file
 *              usage_tracking  Should have been constructed in main()
 *                              should destroyed just before exiting
 */
extern "C" void
cleanup_after_exit(void)
{
        Running         rp;
#ifdef NSE
        char            push_cmd[NSE_TFS_PUSH_LEN + 3 +
```

```
820                                 (MAXPATHLEN * MB_LEN_MAX) + 12];
821         char            *active;
822 #endif

807 extern long     getname_bytes_count;
808 extern long     getname_names_count;
809 extern long     getname_struct_count;
810 extern long     freename_bytes_count;
811 extern long     freename_names_count;
812 extern long     freename_struct_count;
813 extern long     other_alloc;

815 extern long     env_alloc_num;
816 extern long     env_alloc_bytes;


819 #ifdef DMAKE_STATISTICS
820 if(getname_stat) {
821         printf(NOCATGETS(">>> Getname statistics:\n"));
822         printf(NOCATGETS("  Allocated:\n"));
823         printf(NOCATGETS("        Names: %ld\n"), getname_names_count);
824         printf(NOCATGETS("      Strings: %ld Kb (%ld bytes)\n"), getname_bytes_c
825         printf(NOCATGETS("      Structs: %ld Kb (%ld bytes)\n"), getname_struct_
826         printf(NOCATGETS("  Total bytes: %ld Kb (%ld bytes)\n"), getname_struct_

828         printf(NOCATGETS("\n  Unallocated: %ld\n"), freename_names_count);
829         printf(NOCATGETS("        Names: %ld\n"), freename_names_count);
830         printf(NOCATGETS("      Strings: %ld Kb (%ld bytes)\n"), freename_bytes_
831         printf(NOCATGETS("      Structs: %ld Kb (%ld bytes)\n"), freename_struct
832         printf(NOCATGETS("  Total bytes: %ld Kb (%ld bytes)\n"), freename_struct

834         printf(NOCATGETS("\n  Total used: %ld Kb (%ld bytes)\n"), (getname_struc

836         printf(NOCATGETS("\n>>> Other:\n"));
837         printf(
838                 NOCATGETS("       Env (%ld): %ld Kb (%ld bytes)\n"),
839                 env_alloc_num,
840                 env_alloc_bytes/1000,
841                 env_alloc_bytes
842         );

844 }
845 #endif

847 /*
848 #ifdef DISTRIBUTED
849     if (get_parent() == TRUE) {
850 #endif
851  */

853         parallel = false;
854         /* If we used the SVR4_MAKE, don't build .DONE or .FAILED */
855         if (!getenv(USE_SVR4_MAKE)){
856             /* Build the target .DONE or .FAILED if we caught an error */
857             if (!quest && !list_all_targets) {
858                 Name            failed_name;

860                 MBSTOWCS(wcs_buffer, NOCATGETS(".FAILED"));
861                 failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
862                 if ((exit_status != 0) && (failed_name->prop != NULL)) {
863 #ifdef TEAMWARE_MAKE_CMN
864                         /*
865                          * [tolik] switch DMake to serial mode
866                          */
867                         dmake_mode_type = serial_mode;
868                         no_parallel = true;
```



```
869 #endif
870                         (void) doname(failed_name, false, true);
871                 } else {
872                     if (!trace_status) {
873 #ifdef TEAMWARE_MAKE_CMN
874                         /*
875                          * Switch DMake to serial mode
876                          */
877                         dmake_mode_type = serial_mode;
878                         no_parallel = true;
879 #endif
880                         (void) doname(done, false, true);
881                     }
882                 }
883             }
884         }
885         /*
886          * Remove the temp file utilities report dependencies thru if it
887          * is still around
888          */
889         if (temp_file_name != NULL) {
890                 (void) unlink(temp_file_name->string_mb);
891         }
892         /*
893          * Do not save the current command in .make.state if make
894          * was interrupted.
895          */
896         if (current_line != NULL) {
897                 command_changed = true;
898                 current_line->body.line.command_used = NULL;
899         }
900         /*
901          * For each parallel build process running, remove the temp files
902          * and zap the command line so it won't be put in .make.state
903          */
904         for (rp = running_list; rp != NULL; rp = rp->next) {
905                 if (rp->temp_file != NULL) {
906                         (void) unlink(rp->temp_file->string_mb);
907                 }
908                 if (rp->stdout_file != NULL) {
909                         (void) unlink(rp->stdout_file);
910                         retmem_mb(rp->stdout_file);
911                         rp->stdout_file = NULL;
912                 }
913                 if (rp->stderr_file != NULL) {
914                         (void) unlink(rp->stderr_file);
915                         retmem_mb(rp->stderr_file);
916                         rp->stderr_file = NULL;
917                 }
918                 command_changed = true;
919 /*
920                 line = get_prop(rp->target->prop, line_prop);
921                 if (line != NULL) {
922                         line->body.line.command_used = NULL;
923                 }
924  */
925         }
926         /* Remove the statefile lock file if the file has been locked */
927         if ((make_state_lockfile != NULL) && (make_state_locked)) {
928                 (void) unlink(make_state_lockfile);
929                 make_state_lockfile = NULL;
930                 make_state_locked = false;
931         }
932         /* Write .make.state */
933         write_state_file(1, (Boolean) 1);
```

```
#ifdef TEAMWARE_MAKE_CMN
        // Deleting the usage tracking object sends the usage mail
#ifdef USE_DMS_CCR
        //usageTracking->setExitStatus(exit_status, NULL);
        //delete usageTracking;
#else
        cleanup->set_exit_status(exit_status);
        delete cleanup;
#endif
#endif

#ifdef NSE
        /* If running inside an activated environment, push the */
        /* .nse_depinfo file (if written) */
        active = getenv(NSE_VARIANT_ENV);
        if (keep_state &&
            (active != NULL) &&
            !IS_EQUAL(active, NSE_RT_SOURCE_NAME) &&
            !do_not_exec_rule &&
            (report_dependencies_level == 0)) {
                (void) sprintf(push_cmd,
                                "%s %s/%s",
                                NSE_TFS_PUSH,
                                get_current_path(),
                                NSE_DEPINFO);
                (void) system(push_cmd);
        }
#endif

/*
#ifdef DISTRIBUTED
    }
#endif
 */

#if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
        job_adjust_fini();
#endif

#ifdef TEAMWARE_MAKE_CMN
        catclose(catd);
#endif
#ifdef DISTRIBUTED
        if (rxmPid > 0) {
                // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
                Avo_AcknowledgeMsg acknowledgeMsg;
                RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;

                int xdrResult = xdr(&xdrs_out, msg);

                if (xdrResult) {
                        fflush(dmake_ofp);
                } else {
/*
                        fatal(catgets(catd, 1, 266, "couldn't tell rxm to exit")
 */
                        kill(rxmPid, SIGTERM);
                }

                waitpid(rxmPid, NULL, 0);
                rxmPid = 0;
        }
#endif
}

/*
```



```
 *      handle_interrupt()
 *
 *      This is where C-C traps are caught.
 *
 *      Parameters:
 *
 *      Global variables used (except DMake 1.0):
 *              current_target          Sometimes the current target is removed
 *              do_not_exec_rule        But not if -n is on
 *              quest                   or -q
 *              running_list            List of parallel running processes
 *              touch                   Current target is not removed if -t on
 */
void
handle_interrupt(int)
{
        Property                member;
        Running                 rp;

        (void) fflush(stdout);
#ifdef DISTRIBUTED
        if (rxmPid > 0) {
                // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
                Avo_AcknowledgeMsg acknowledgeMsg;
                RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;

                int xdrResult = xdr(&xdrs_out, msg);

                if (xdrResult) {
                        fflush(dmake_ofp);
                } else {
                        kill(rxmPid, SIGTERM);
                        rxmPid = 0;
                }
        }
#endif
        if (childPid > 0) {
                kill(childPid, SIGTERM);
                childPid = -1;
        }
        for (rp = running_list; rp != NULL; rp = rp->next) {
                if (rp->state != build_running) {
                        continue;
                }
                if (rp->pid > 0) {
                        kill(rp->pid, SIGTERM);
                        rp->pid = -1;
                }
        }
        if (getpid() == getpgrp()) {
                bsd_signal(SIGTERM, SIG_IGN);
                kill (-getpid(), SIGTERM);
        }
#ifdef TEAMWARE_MAKE_CMN
        /* Clean up all parallel/distributed children already finished */
        finish_children(false);
#endif

        /* Make sure the processes running under us terminate first */

        while (wait((int *) NULL) != -1);
        /* Delete the current targets unless they are precious */
        if ((current_target != NULL) &&
            current_target->is_member &&
            ((member = get_prop(current_target->prop, member_prop)) != NULL)) {
                current_target = member->body.member.library;
```

```
1049         }
1050         if (!do_not_exec_rule &&
1051             !touch &&
1052             !quest &&
1053             (current_target != NULL) &&
1054             !(current_target->stat.is_precious || all_precious)) {

1056 /* BID_1030811 */
1057 /* azv 16 Oct 95 */
1058                 current_target->stat.time = file_no_time;

1060                 if (exists(current_target) != file_doesnt_exist) {
1061                         (void) fprintf(stderr,
1062                                        "\n*** %s ",
1063                                        current_target->string_mb);
1064                         if (current_target->stat.is_dir) {
1065                                 (void) fprintf(stderr,
1066                                                catgets(catd, 1, 168, "not remove
1067                                                current_target->string_mb);
1068                         } else if (unlink(current_target->string_mb) == 0) {
1069                                 (void) fprintf(stderr,
1070                                                catgets(catd, 1, 169, "removed.\n
1071                                                current_target->string_mb);
1072                         } else {
1073                                 (void) fprintf(stderr,
1074                                                catgets(catd, 1, 170, "could not
1075                                                current_target->string_mb,
1076                                                errmsg(errno));
1077                         }
1078                 }
1079         }
1080         for (rp = running_list; rp != NULL; rp = rp->next) {
1081                 if (rp->state != build_running) {
1082                         continue;
1083                 }
1084                 if (rp->target->is_member &&
1085                     ((member = get_prop(rp->target->prop, member_prop)) !=
1086                      NULL)) {
1087                         rp->target = member->body.member.library;
1088                 }
1089                 if (!do_not_exec_rule &&
1090                     !touch &&
1091                     !quest &&
1092                     !(rp->target->stat.is_precious || all_precious)) {

1094                         rp->target->stat.time = file_no_time;
1095                         if (exists(rp->target) != file_doesnt_exist) {
1096                                 (void) fprintf(stderr,
1097                                                "\n*** %s ",
1098                                                rp->target->string_mb);
1099                                 if (rp->target->stat.is_dir) {
1100                                         (void) fprintf(stderr,
1101                                                        catgets(catd, 1, 171, "no
1102                                                        rp->target->string_mb);
1103                                 } else if (unlink(rp->target->string_mb) == 0) {
1104                                         (void) fprintf(stderr,
1105                                                        catgets(catd, 1, 172, "re
1106                                                        rp->target->string_mb);
1107                                 } else {
1108                                         (void) fprintf(stderr,
1109                                                        catgets(catd, 1, 173, "co
1110                                                        rp->target->string_mb,
1111                                                        errmsg(errno));
1112                                 }
1113                         }
1114                 }
```



```
1115         }

1117 #ifdef SGE_SUPPORT
1118         /* Remove SGE script file */
1119         if (grid) {
1120                 unlink(script_file);
1121         }
1122 #endif

1124         /* Have we locked .make.state or .nse_depinfo? */
1125         if ((make_state_lockfile != NULL) && (make_state_locked)) {
1126                 unlink(make_state_lockfile);
1127                 make_state_lockfile = NULL;
1128                 make_state_locked = false;
1129         }
1165 #ifdef NSE
1166         if ((nse_depinfo_lockfile[0] != '\0') && (nse_depinfo_locked)) {
1167                 unlink(nse_depinfo_lockfile);
1168                 nse_depinfo_lockfile[0] = '\0';
1169                 nse_depinfo_locked = false;
1170         }
1171 #endif
1130         /*
1131          * Re-read .make.state file (it might be changed by recursive make)
1132          */
1133         check_state(NULL);

1135         report_dir_enter_leave(false);

1137         exit_status = 2;
1138         exit(2);
1139 }
_____unchanged_portion_omitted_

1584 /*
1585  *      parse_command_option(ch)
1586  *
1587  *      Parse make command line options.
1588  *
1589  *      Return value:
1590  *                              Indicates if any -f -c or -M were seen
1591  *
1592  *      Parameters:
1593  *              ch              The character to parse
1594  *
1595  *      Static variables used:
1596  *              dmake_group_specified   Set for make -g
1597  *              dmake_max_jobs_specified        Set for make -j
1598  *              dmake_mode_specified    Set for make -m
1599  *              dmake_add_mode_specified        Set for make -x
1600  *              dmake_compat_mode_specified     Set for make -x SUN_MAKE_COMPAT_
1601  *              dmake_output_mode_specified     Set for make -x DMAKE_OUTPUT_MOD
1602  *              dmake_odir_specified    Set for make -o
1603  *              dmake_rcfile_specified  Set for make -c
1604  *              env_wins                Set for make -e
1605  *              ignore_default_mk       Set for make -r
1606  *              trace_status            Set for make -p
1607  *
1608  *      Global variables used:
1609  *              .make.state path & name set for make -K
1610  *              continue_after_error    Set for make -k
1611  *              debug_level             Set for make -d
1612  *              do_not_exec_rule        Set for make -n
1613  *              filter_stderr           Set for make -X
1614  *              ignore_errors_all       Set for make -i
1615  *              no_parallel             Set for make -R
```

```
 *              quest                   Set for make -q
 *              read_trace_level        Set for make -D
 *              report_dependencies     Set for make -P
 *              send_mtool_msgs         Set for make -K
 *              silent_all              Set for make -s
 *              touch                   Set for make -t
 */
static int
parse_command_option(register char ch)
{
        static int              invert_next = 0;
        int                     invert_this = invert_next;

        invert_next = 0;
        switch (ch) {
        case '-':                       /* Ignore "--" */
                return 0;
        case '~':                       /* Invert next option */
                invert_next = 1;
                return 0;
        case 'B':                       /* Obsolete */
                return 0;
        case 'b':                       /* Obsolete */
                return 0;
        case 'c':                       /* Read alternative dmakerc file */
                if (invert_this) {
                        dmake_rcfile_specified = false;
                } else {
                        dmake_rcfile_specified = true;
                }
                return 2;
        case 'D':                       /* Show lines read */
                if (invert_this) {
                        read_trace_level--;
                } else {
                        read_trace_level++;
                }
                return 0;
        case 'd':                       /* Debug flag */
                if (invert_this) {
                        debug_level--;
                } else {
                        debug_level++;
                }
                return 0;
#ifdef NSE
        case 'E':
                if (invert_this) {
                        nse = false;
                } else {
                        nse = true;
                }
                nse_init_source_suffixes();
                return 0;
#endif
        case 'e':                       /* Environment override flag */
                if (invert_this) {
                        env_wins = false;
                } else {
                        env_wins = true;
                }
                return 0;
        case 'f':                       /* Read alternative makefile(s) */
                return 1;
        case 'g':                       /* Use alternative DMake group */
                if (invert_this) {
```



```
                        dmake_group_specified = false;
                } else {
                        dmake_group_specified = true;
                }
                return 4;
        case 'i':                       /* Ignore errors */
                if (invert_this) {
                        ignore_errors_all = false;
                } else {
                        ignore_errors_all = true;
                }
                return 0;
        case 'j':                       /* Use alternative DMake max jobs */
                if (invert_this) {
                        dmake_max_jobs_specified = false;
                } else {
                        dmake_max_jobs_specified = true;
                }
                return 8;
        case 'K':                       /* Read alternative .make.state */
                return 256;
        case 'k':                       /* Keep making even after errors */
                if (invert_this) {
                        continue_after_error = false;
                } else {
                        continue_after_error = true;
                        continue_after_error_ever_seen = true;
                }
                return 0;
        case 'M':                       /* Read alternative make.machines file
                if (invert_this) {
                        pmake_machinesfile_specified = false;
                } else {
                        pmake_machinesfile_specified = true;
                        dmake_mode_type = parallel_mode;
                        no_parallel = false;
                }
                return 16;
        case 'm':                       /* Use alternative DMake build mode */
                if (invert_this) {
                        dmake_mode_specified = false;
                } else {
                        dmake_mode_specified = true;
                }
                return 32;
        case 'x':                       /* Use alternative DMake mode */
                if (invert_this) {
                        dmake_add_mode_specified = false;
                } else {
                        dmake_add_mode_specified = true;
                }
                return 1024;
        case 'N':                       /* Reverse -n */
                if (invert_this) {
                        do_not_exec_rule = true;
                } else {
                        do_not_exec_rule = false;
                }
                return 0;
        case 'n':                       /* Print, not exec commands */
                if (invert_this) {
                        do_not_exec_rule = false;
                } else {
                        do_not_exec_rule = true;
                }
                return 0;
```

```
#ifndef PARALLEL
        case 'O':                       /* Send job start & result msgs */
                if (invert_this) {
                        send_mtool_msgs = false;
                } else {
#ifdef DISTRIBUTED
                        send_mtool_msgs = true;
#endif
                }
                return 128;
#endif
        case 'o':                       /* Use alternative dmake output dir */
                if (invert_this) {
                        dmake_odir_specified = false;
                } else {
                        dmake_odir_specified = true;
                }
                return 512;
        case 'P':                       /* Print for selected targets */
                if (invert_this) {
                        report_dependencies_level--;
                } else {
                        report_dependencies_level++;
                }
                return 0;
        case 'p':                       /* Print description */
                if (invert_this) {
                        trace_status = false;
                        do_not_exec_rule = false;
                } else {
                        trace_status = true;
                        do_not_exec_rule = true;
                }
                return 0;
        case 'q':                       /* Question flag */
                if (invert_this) {
                        quest = false;
                } else {
                        quest = true;
                }
                return 0;
        case 'R':                       /* Don't run in parallel */
#ifdef TEAMWARE_MAKE_CMN
                if (invert_this) {
                        pmake_cap_r_specified = false;
                        no_parallel = false;
                } else {
                        pmake_cap_r_specified = true;
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
                }
#else
                warning(catgets(catd, 1, 182, "Ignoring ParallelMake -R option")
#endif
                return 0;
        case 'r':                       /* Turn off internal rules */
                if (invert_this) {
                        ignore_default_mk = false;
                } else {
                        ignore_default_mk = true;
                }
                return 0;
        case 'S':                       /* Reverse -k */
                if (invert_this) {
                        continue_after_error = true;
                } else {
```



```
                        continue_after_error = false;
                        stop_after_error_ever_seen = true;
                }
                return 0;
        case 's':                       /* Silent flag */
                if (invert_this) {
                        silent_all = false;
                } else {
                        silent_all = true;
                }
                return 0;
        case 'T':                       /* Print target list */
                if (invert_this) {
                        list_all_targets = false;
                        do_not_exec_rule = false;
                } else {
                        list_all_targets = true;
                        do_not_exec_rule = true;
                }
                return 0;
        case 't':                       /* Touch flag */
                if (invert_this) {
                        touch = false;
                } else {
                        touch = true;
                }
                return 0;
        case 'u':                       /* Unconditional flag */
                if (invert_this) {
                        build_unconditional = false;
                } else {
                        build_unconditional = true;
                }
                return 0;
        case 'V':                       /* SVR4 mode */
                svr4 = true;
                return 0;
        case 'v':                       /* Version flag */
                if (invert_this) {
                } else {
#ifdef TEAMWARE_MAKE_CMN
                        fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
                        exit_status = 0;
                        exit(0);
#else
                        warning(catgets(catd, 1, 324, "Ignoring DistributedMake
#endif
                }
                return 0;
        case 'w':                       /* Unconditional flag */
                if (invert_this) {
                        report_cwd = false;
                } else {
                        report_cwd = true;
                }
                return 0;
#if 0
        case 'X':                       /* Filter stdout */
                if (invert_this) {
                        filter_stderr = false;
                } else {
                        filter_stderr = true;
                }
                return 0;
#endif
        default:
```

```
1870                    break;
1871            }
1872            return 0;
1873    }
______unchanged_portion_omitted_

1986    /*
1987     *      read_files_and_state(argc, argv)
1988     *
1989     *      Read the makefiles we care about and the environment
1990     *      Also read the = style command line options
1991     *
1992     *      Parameters:
1993     *              argc            You know what this is
1994     *              argv            You know what this is
1995     *
1996     *      Static variables used:
1997     *              env_wins        make -e, determines if env vars are RO
1998     *              ignore_default_mk make -r, determines if make.rules is read
1999     *              not_auto_depen  dwight
2000     *
2001     *      Global variables used:
2002     *              default_target_to_build Set to first proper target from file
2003     *              do_not_exec_rule Set to false when makfile is made
2004     *              dot             The Name ".", used to read current dir
2005     *              empty_name      The Name "", use as macro value
2006     *              keep_state      Set if KEEP_STATE is in environment
2007     *              make_state      The Name ".make.state", used to read file
2008     *              makefile_type   Set to type of file being read
2009     *              makeflags       The Name "MAKEFLAGS", used to set macro value
2010     *              not_auto        dwight
2063     *              nse             Set if NSE_ENV is in the environment
2011     *              read_trace_level Checked to se if the reader should trace
2012     *              report_dependencies If -P is on we do not read .make.state
2013     *              trace_reader    Set if reader should trace
2014     *              virtual_root    The Name "VIRTUAL_ROOT", used to check value
2015     *      */
2016    static void
2017    read_files_and_state(int argc, char **argv)
2018    {
2019            wchar_t                 buffer[1000];
2020            wchar_t                 buffer_posix[1000];
2021            register char           ch;
2022            register char           *cp;
2023            Property                def_make_macro = NULL;
2024            Name                    def_make_name;
2025            Name                    default_makefile;
2026            String_rec              dest;
2027            wchar_t                 destbuffer[STRING_BUFFER_LENGTH];
2028            register int            i;
2029            register int            j;
2030            Name                    keep_state_name;
2031            int                     length;
2032            Name                    Makefile;
2033            register Property       macro;
2034            struct stat             make_state_stat;
2035            Name                    makefile_name;
2036            register int            makefile_next = 0;
2037            register Boolean        makefile_read = false;
2038            String_rec              makeflags_string;
2039            String_rec              makeflags_string_posix;
2040            String_rec *            makeflags_string_current;
2041            Name                    makeflags_value_saved;
2042            register Name           name;
2043            Name                    new_make_value;
2044            Boolean                 save_do_not_exec_rule;
```



```
2045            Name                    sdotMakefile;
2046            Name                    sdotmakefile_name;
2047            static wchar_t          state_file_str;
2048            static char             state_file_str_mb[MAXPATHLEN];
2049            static struct _Name     state_filename;
2050            Boolean                 temp;
2051            char                    tmp_char;
2052            wchar_t                 *tmp_wcs_buffer;
2053            register Name           value;
2054            ASCII_Dyn_Array         makeflags_and_macro;
2055            Boolean                 is_xpg4;

2057    /*
2058     *      Remember current mode. It may be changed after reading makefile
2059     *      and we will have to correct MAKEFLAGS variable.
2060     */
2061            is_xpg4 = posix;

2063            MBSTOWCS(wcs_buffer, NOCATGETS("KEEP_STATE"));
2064            keep_state_name = GETNAME(wcs_buffer, FIND_LENGTH);
2065            MBSTOWCS(wcs_buffer, NOCATGETS("Makefile"));
2066            Makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2067            MBSTOWCS(wcs_buffer, NOCATGETS("makefile"));
2068            makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
2069            MBSTOWCS(wcs_buffer, NOCATGETS("s.makefile"));
2070            sdotmakefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
2071            MBSTOWCS(wcs_buffer, NOCATGETS("s.Makefile"));
2072            sdotMakefile = GETNAME(wcs_buffer, FIND_LENGTH);

2074    /*
2075     *      Set flag if NSE is active
2076     */
2130    #ifdef NSE
2131            if (getenv(NOCATGETS("NSE_ENV")) != NULL) {
2132                    nse = true;
2133            }
2134    #endif

2078    /*
2079     *      initialize global dependency entry for .NOT_AUTO
2080     */
2081            not_auto_depen->next = NULL;
2082            not_auto_depen->name = not_auto;
2083            not_auto_depen->automatic = not_auto_depen->stale = false;

2085    /*
2086     *      Read internal definitions and rules.
2087     */
2088            if (read_trace_level > 1) {
2089                    trace_reader = true;
2090            }
2091            if (!ignore_default_mk) {
2092                    if (svr4) {
2093                            MBSTOWCS(wcs_buffer, NOCATGETS("svr4.make.rules"));
2094                            default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2095                    } else {
2096                            MBSTOWCS(wcs_buffer, NOCATGETS("make.rules"));
2097                            default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2098                    }
2099                    default_makefile->stat.is_file = true;

2101                    (void) read_makefile(default_makefile,
2102                                         true,
2103                                         false,
2104                                         true);
2105            }
```

```
2107         /*
2108          * If the user did not redefine the MAKE macro in the
2109          * default makefile (make.rules), then we'd like to
2110          * change the macro value of MAKE to be some form
2111          * of argv[0] for recursive MAKE builds.
2112          */
2113         MBSTOWCS(wcs_buffer, NOCATGETS("MAKE"));
2114         def_make_name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2115         def_make_macro = get_prop(def_make_name->prop, macro_prop);
2116         if ((def_make_macro != NULL) &&
2117             (IS_EQUAL(def_make_macro->body.macro.value->string_mb,
2118                       NOCATGETS("make")))) {
2119                 MBSTOWCS(wcs_buffer, argv_zero_string);
2120                 new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2121                 (void) SETVAR(def_make_name,
2122                               new_make_value,
2123                               false);
2124         }

2126         default_target_to_build = NULL;
2127         trace_reader = false;

2129 /*
2130  *      Read environment args. Let file args which follow override unless
2131  *      -e option seen. If -e option is not mentioned.
2132  */
2133         read_environment(env_wins);
2134         if (getvar(virtual_root)->hash.length == 0) {
2135                 maybe_append_prop(virtual_root, macro_prop)
2136                   ->body.macro.exported = true;
2137                 MBSTOWCS(wcs_buffer, "/");
2138                 (void) SETVAR(virtual_root,
2139                               GETNAME(wcs_buffer, FIND_LENGTH),
2140                               false);
2141         }

2143 /*
2144  * We now scan mf_argv and argv to see if we need to set
2145  * any of the DMake-added options/variables in MAKEFLAGS.
2146  */

2148         makeflags_and_macro.start = 0;
2149         makeflags_and_macro.size = 0;
2150         enter_argv_values(mf_argc, mf_argv, &makeflags_and_macro);
2151         enter_argv_values(argc, argv, &makeflags_and_macro);

2153 /*
2154  *      Set MFLAGS and MAKEFLAGS
2155  *
2156  *      Before reading makefile we do not know exactly which mode
2157  *      (posix or not) is used. So prepare two MAKEFLAGS strings
2158  *      for both posix and solaris modes because they are different.
2159  */
2160         INIT_STRING_FROM_STACK(makeflags_string, buffer);
2161         INIT_STRING_FROM_STACK(makeflags_string_posix, buffer_posix);
2162         append_char((int) hyphen_char, &makeflags_string);
2163         append_char((int) hyphen_char, &makeflags_string_posix);

2165         switch (read_trace_level) {
2166         case 2:
2167                 append_char('D', &makeflags_string);
2168                 append_char('D', &makeflags_string_posix);
2169         case 1:
2170                 append_char('D', &makeflags_string);
2171                 append_char('D', &makeflags_string_posix);
```



```
2172         }
2173         switch (debug_level) {
2174         case 2:
2175                 append_char('d', &makeflags_string);
2176                 append_char('d', &makeflags_string_posix);
2177         case 1:
2178                 append_char('d', &makeflags_string);
2179                 append_char('d', &makeflags_string_posix);
2180         }
2239 #ifdef NSE
2240         if (nse) {
2241                 append_char('E', &makeflags_string);
2242         }
2243 #endif
2181         if (env_wins) {
2182                 append_char('e', &makeflags_string);
2183                 append_char('e', &makeflags_string_posix);
2184         }
2185         if (ignore_errors_all) {
2186                 append_char('i', &makeflags_string);
2187                 append_char('i', &makeflags_string_posix);
2188         }
2189         if (continue_after_error) {
2190                 if (stop_after_error_ever_seen) {
2191                         append_char('S', &makeflags_string_posix);
2192                         append_char((int) space_char, &makeflags_string_posix);
2193                         append_char((int) hyphen_char, &makeflags_string_posix);
2194                 }
2195                 append_char('k', &makeflags_string);
2196                 append_char('k', &makeflags_string_posix);
2197         } else {
2198                 if (stop_after_error_ever_seen
2199                     && continue_after_error_ever_seen) {
2200                         append_char('k', &makeflags_string_posix);
2201                         append_char((int) space_char, &makeflags_string_posix);
2202                         append_char((int) hyphen_char, &makeflags_string_posix);
2203                         append_char('S', &makeflags_string_posix);
2204                 }
2205         }
2206         if (do_not_exec_rule) {
2207                 append_char('n', &makeflags_string);
2208                 append_char('n', &makeflags_string_posix);
2209         }
2210         switch (report_dependencies_level) {
2211         case 4:
2212                 append_char('P', &makeflags_string);
2213                 append_char('P', &makeflags_string_posix);
2214         case 3:
2215                 append_char('P', &makeflags_string);
2216                 append_char('P', &makeflags_string_posix);
2217         case 2:
2218                 append_char('P', &makeflags_string);
2219                 append_char('P', &makeflags_string_posix);
2220         case 1:
2221                 append_char('P', &makeflags_string);
2222                 append_char('P', &makeflags_string_posix);
2223         }
2224         if (trace_status) {
2225                 append_char('p', &makeflags_string);
2226                 append_char('p', &makeflags_string_posix);
2227         }
2228         if (quest) {
2229                 append_char('q', &makeflags_string);
2230                 append_char('q', &makeflags_string_posix);
2231         }
2232         if (silent_all) {
```

```
2233                    append_char('s', &makeflags_string);
2234                    append_char('s', &makeflags_string_posix);
2235            }
2236            if (touch) {
2237                    append_char('t', &makeflags_string);
2238                    append_char('t', &makeflags_string_posix);
2239            }
2240            if (build_unconditional) {
2241                    append_char('u', &makeflags_string);
2242                    append_char('u', &makeflags_string_posix);
2243            }
2244            if (report_cwd) {
2245                    append_char('w', &makeflags_string);
2246                    append_char('w', &makeflags_string_posix);
2247            }
2248 #ifndef PARALLEL
2249            /* -c dmake_rcfile */
2250            if (dmake_rcfile_specified) {
2251                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2252                    dmake_rcfile = GETNAME(wcs_buffer, FIND_LENGTH);
2253                    append_makeflags_string(dmake_rcfile, &makeflags_string);
2254                    append_makeflags_string(dmake_rcfile, &makeflags_string_posix);
2255            }
2256            /* -g dmake_group */
2257            if (dmake_group_specified) {
2258                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
2259                    dmake_group = GETNAME(wcs_buffer, FIND_LENGTH);
2260                    append_makeflags_string(dmake_group, &makeflags_string);
2261                    append_makeflags_string(dmake_group, &makeflags_string_posix);
2262            }
2263            /* -j dmake_max_jobs */
2264            if (dmake_max_jobs_specified) {
2265                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
2266                    dmake_max_jobs = GETNAME(wcs_buffer, FIND_LENGTH);
2267                    append_makeflags_string(dmake_max_jobs, &makeflags_string);
2268                    append_makeflags_string(dmake_max_jobs, &makeflags_string_posix)
2269            }
2270            /* -m dmake_mode */
2271            if (dmake_mode_specified) {
2272                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2273                    dmake_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2274                    append_makeflags_string(dmake_mode, &makeflags_string);
2275                    append_makeflags_string(dmake_mode, &makeflags_string_posix);
2276            }
2277            /* -x dmake_compat_mode */
2278 //         if (dmake_compat_mode_specified) {
2279 //                 MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE_COMPAT_MODE"));
2280 //                 dmake_compat_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2281 //                 append_makeflags_string(dmake_compat_mode, &makeflags_string);
2282 //                 append_makeflags_string(dmake_compat_mode, &makeflags_string_pos
2283 //         }
2284            /* -x dmake_output_mode */
2285            if (dmake_output_mode_specified) {
2286                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
2287                    dmake_output_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2288                    append_makeflags_string(dmake_output_mode, &makeflags_string);
2289                    append_makeflags_string(dmake_output_mode, &makeflags_string_pos
2290            }
2291            /* -o dmake_odir */
2292            if (dmake_odir_specified) {
2293                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2294                    dmake_odir = GETNAME(wcs_buffer, FIND_LENGTH);
2295                    append_makeflags_string(dmake_odir, &makeflags_string);
2296                    append_makeflags_string(dmake_odir, &makeflags_string_posix);
2297            }
2298            /* -M pmake_machinesfile */
```



```
2299            if (pmake_machinesfile_specified) {
2300                    MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
2301                    pmake_machinesfile = GETNAME(wcs_buffer, FIND_LENGTH);
2302                    append_makeflags_string(pmake_machinesfile, &makeflags_string);
2303                    append_makeflags_string(pmake_machinesfile, &makeflags_string_po
2304            }
2305            /* -R */
2306            if (pmake_cap_r_specified) {
2307                    append_char((int) space_char, &makeflags_string);
2308                    append_char((int) hyphen_char, &makeflags_string);
2309                    append_char('R', &makeflags_string);
2310                    append_char((int) space_char, &makeflags_string_posix);
2311                    append_char((int) hyphen_char, &makeflags_string_posix);
2312                    append_char('R', &makeflags_string_posix);
2313            }
2314 #endif

2316 /*
2317  *      Make sure MAKEFLAGS is exported
2318  */
2319            maybe_append_prop(makeflags, macro_prop)->
2320              body.macro.exported = true;

2322            if (makeflags_string.buffer.start[1] != (int) nul_char) {
2323                    if (makeflags_string.buffer.start[1] != (int) space_char) {
2324                            MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2325                            (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2326                                    GETNAME(makeflags_string.buffer.start,
2327                                            FIND_LENGTH),
2328                                    false);
2329                    } else {
2330                            MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2331                            (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2332                                    GETNAME(makeflags_string.buffer.start + 2,
2333                                            FIND_LENGTH),
2334                                    false);
2335                    }
2336            }

2338 /*
2339  *      Add command line macro to POSIX makeflags_string
2340  */
2341            if (makeflags_and_macro.start) {
2342                    tmp_char = (char) space_char;
2343                    cp = makeflags_and_macro.start;
2344                    do {
2345                            append_char(tmp_char, &makeflags_string_posix);
2346                    } while ( tmp_char = *cp++ );
2347                    retmem_mb(makeflags_and_macro.start);
2348            }

2350 /*
2351  *      Now set the value of MAKEFLAGS macro in accordance
2352  *      with current mode.
2353  */
2354            macro = maybe_append_prop(makeflags, macro_prop);
2355            temp = (Boolean) macro->body.macro.read_only;
2356            macro->body.macro.read_only = false;
2357            if(posix || gnu_style) {
2358                    makeflags_string_current = &makeflags_string_posix;
2359            } else {
2360                    makeflags_string_current = &makeflags_string;
2361            }
2362            if (makeflags_string_current->buffer.start[1] == (int) nul_char) {
2363                    makeflags_value_saved =
2364                            GETNAME( makeflags_string_current->buffer.start + 1
```

```
2365                                , FIND_LENGTH
2366                                );
2367        } else {
2368                if (makeflags_string_current->buffer.start[1] != (int) space_cha
2369                        makeflags_value_saved =
2370                                GETNAME( makeflags_string_current->buffer.start
2371                                       , FIND_LENGTH
2372                                       );
2373                } else {
2374                        makeflags_value_saved =
2375                                GETNAME( makeflags_string_current->buffer.start
2376                                       , FIND_LENGTH
2377                                       );
2378                }
2379        }
2380        (void) SETVAR( makeflags
2381                     , makeflags_value_saved
2382                     , false
2383                     );
2384        macro->body.macro.read_only = temp;

2386 /*
2387  *      Read command line "-f" arguments and ignore -c, g, j, K, M, m, O and o a
2388  */
2389        save_do_not_exec_rule = do_not_exec_rule;
2390        do_not_exec_rule = false;
2391        if (read_trace_level > 0) {
2392                trace_reader = true;
2393        }

2395        for (i = 1; i < argc; i++) {
2396                if (argv[i] &&
2397                    (argv[i][0] == (int) hyphen_char) &&
2398                    (argv[i][1] == 'f') &&
2399                    (argv[i][2] == (int) nul_char)) {
2400                        argv[i] = NULL;         /* Remove -f */
2401                        if (i >= argc - 1) {
2402                                fatal(catgets(catd, 1, 190, "No filename argumen
2403                        }
2404                        MBSTOWCS(wcs_buffer, argv[++i]);
2405                        primary_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2406                        (void) read_makefile(primary_makefile, true, true, true)
2407                        argv[i] = NULL;         /* Remove filename */
2408                        makefile_read = true;
2409                } else if (argv[i] &&
2410                           (argv[i][0] == (int) hyphen_char) &&
2411                           (argv[i][1] == 'c' ||
2412                            argv[i][1] == 'g' ||
2413                            argv[i][1] == 'j' ||
2414                            argv[i][1] == 'K' ||
2415                            argv[i][1] == 'M' ||
2416                            argv[i][1] == 'm' ||
2417                            argv[i][1] == 'O' ||
2418                            argv[i][1] == 'o') &&
2419                           (argv[i][2] == (int) nul_char)) {
2420                        argv[i] = NULL;
2421                        argv[++i] = NULL;
2422                }
2423        }

2425 /*
2426  *      If no command line "-f" args then look for "makefile", and then for
2427  *      "Makefile" if "makefile" isn't found.
2428  */
2429        if (!makefile_read) {
2430                (void) read_dir(dot,
```



```
2431                                (wchar_t *) NULL,
2432                                (Property) NULL,
2433                                (wchar_t *) NULL);
2434            if (!posix) {
2435                if (makefile_name->stat.is_file) {
2436                        if (Makefile->stat.is_file) {
2437                                warning(catgets(catd, 1, 310, "Both 'makefile' a
2438                        }
2439                        primary_makefile = makefile_name;
2440                        makefile_read = read_makefile(makefile_name,
2441                                                      false,
2442                                                      false,
2443                                                      true);
2444                }
2445                if (!makefile_read &&
2446                    Makefile->stat.is_file) {
2447                        primary_makefile = Makefile;
2448                        makefile_read = read_makefile(Makefile,
2449                                                      false,
2450                                                      false,
2451                                                      true);
2452                }
2453            } else {

2455                enum sccs_stat save_m_has_sccs = NO_SCCS;
2456                enum sccs_stat save_M_has_sccs = NO_SCCS;

2458                if (makefile_name->stat.is_file) {
2459                        if (Makefile->stat.is_file) {
2460                                warning(catgets(catd, 1, 191, "Both 'makefile' a
2461                        }
2462                }
2463                if (makefile_name->stat.is_file) {
2464                        if (makefile_name->stat.has_sccs == NO_SCCS) {
2465                           primary_makefile = makefile_name;
2466                           makefile_read = read_makefile(makefile_name,
2467                                                         false,
2468                                                         false,
2469                                                         true);
2470                        } else {
2471                          save_m_has_sccs = makefile_name->stat.has_sccs;
2472                          makefile_name->stat.has_sccs = NO_SCCS;
2473                          primary_makefile = makefile_name;
2474                          makefile_read = read_makefile(makefile_name,
2475                                                        false,
2476                                                        false,
2477                                                        true);
2478                        }
2479                }
2480                if (!makefile_read &&
2481                    Makefile->stat.is_file) {
2482                        if (Makefile->stat.has_sccs == NO_SCCS) {
2483                           primary_makefile = Makefile;
2484                           makefile_read = read_makefile(Makefile,
2485                                                         false,
2486                                                         false,
2487                                                         true);
2488                        } else {
2489                          save_M_has_sccs = Makefile->stat.has_sccs;
2490                          Makefile->stat.has_sccs = NO_SCCS;
2491                          primary_makefile = Makefile;
2492                          makefile_read = read_makefile(Makefile,
2493                                                        false,
2494                                                        false,
2495                                                        true);
2496                        }
```

```
2497                }
2498                if (!makefile_read &&
2499                        makefile_name->stat.is_file) {
2500                           makefile_name->stat.has_sccs = save_m_has_sccs;
2501                           primary_makefile = makefile_name;
2502                           makefile_read = read_makefile(makefile_name,
2503                                                         false,
2504                                                         false,
2505                                                         true);
2506                }
2507                if (!makefile_read &&
2508                    Makefile->stat.is_file) {
2509                           Makefile->stat.has_sccs = save_M_has_sccs;
2510                           primary_makefile = Makefile;
2511                           makefile_read = read_makefile(Makefile,
2512                                                         false,
2513                                                         false,
2514                                                         true);
2515                }
2516            }
2517        }
2518        do_not_exec_rule = save_do_not_exec_rule;
2519        allrules_read = makefile_read;
2520        trace_reader = false;

2522 /*
2523  *      Now get current value of MAKEFLAGS and compare it with
2524  *      the saved value we set before reading makefile.
2525  *      If they are different then MAKEFLAGS is subsequently set by
2526  *      makefile, just leave it there. Otherwise, if make mode
2527  *      is changed by using .POSIX target in makefile we need
2528  *      to correct MAKEFLAGS value.
2529  */
2530        Name mf_val = getvar(makeflags);
2531        if( (posix != is_xpg4)
2532         && (!strcmp(mf_val->string_mb, makeflags_value_saved->string_mb)))
2533        {
2534                if (makeflags_string_posix.buffer.start[1] == (int) nul_char) {
2535                        (void) SETVAR(makeflags,
2536                                      GETNAME(makeflags_string_posix.buffer.star
2537                                              FIND_LENGTH),
2538                                      false);
2539                } else {
2540                        if (makeflags_string_posix.buffer.start[1] != (int) spac
2541                                (void) SETVAR(makeflags,
2542                                              GETNAME(makeflags_string_posix.buf
2543                                                      FIND_LENGTH),
2544                                              false);
2545                        } else {
2546                                (void) SETVAR(makeflags,
2547                                              GETNAME(makeflags_string_posix.buf
2548                                                      FIND_LENGTH),
2549                                              false);
2550                        }
2551                }
2552        }

2554        if (makeflags_string.free_after_use) {
2555                retmem(makeflags_string.buffer.start);
2556        }
2557        if (makeflags_string_posix.free_after_use) {
2558                retmem(makeflags_string_posix.buffer.start);
2559        }
2560        makeflags_string.buffer.start = NULL;
2561        makeflags_string_posix.buffer.start = NULL;
```



```
2563        if (posix) {
2564                /*
2565                 * If the user did not redefine the ARFLAGS macro in the
2566                 * default makefile (make.rules), then we'd like to
2567                 * change the macro value of ARFLAGS to be in accordance
2568                 * with "POSIX" requirements.
2569                 */
2570                MBSTOWCS(wcs_buffer, NOCATGETS("ARFLAGS"));
2571                name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2572                macro = get_prop(name->prop, macro_prop);
2573                if ((macro != NULL) && /* Maybe (macro == NULL) || ? */
2574                    (IS_EQUAL(macro->body.macro.value->string_mb,
2575                              NOCATGETS("rv")))) {
2576                        MBSTOWCS(wcs_buffer, NOCATGETS("-rv"));
2577                        value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2578                        (void) SETVAR(name,
2579                                      value,
2580                                      false);
2581                }
2582        }

2584        if (!posix && !svr4) {
2585                set_sgs_support();
2586        }


2589 /*
2590  *      Make sure KEEP_STATE is in the environment if KEEP_STATE is on.
2591  */
2592        macro = get_prop(keep_state_name->prop, macro_prop);
2593        if ((macro != NULL) &&
2594            macro->body.macro.exported) {
2595                keep_state = true;
2596        }
2597        if (keep_state) {
2598                if (macro == NULL) {
2599                        macro = maybe_append_prop(keep_state_name,
2600                                                  macro_prop);
2601                }
2602                macro->body.macro.exported = true;
2603                (void) SETVAR(keep_state_name,
2604                              empty_name,
2605                              false);

2607                /*
2608                 *      Read state file
2609                 */

2611                /* Before we read state, let's make sure we have
2612                ** right state file.
2613                */
2614                /* just in case macro references are used in make_state file
2615                ** name, we better expand them at this stage using expand_value.
2616                */
2617                INIT_STRING_FROM_STACK(dest, destbuffer);
2618                expand_value(make_state, &dest, false);

2620                make_state = GETNAME(dest.buffer.start, FIND_LENGTH);

2622                if(!stat(make_state->string_mb, &make_state_stat)) {
2623                   if(!(make_state_stat.st_mode & S_IFREG) ) {
2624                        /* copy the make_state structure to the other
2625                        ** and then let make_state point to the new
2626                        ** one.
2627                        */
2628                      memcpy(&state_filename, make_state,sizeof(state_filename))
```

```
2629                        state_filename.string_mb = state_file_str_mb;
2630                /* Just a kludge to avoid two slashes back to back */
2631                    if((make_state->hash.length == 1)&&
2632                              (make_state->string_mb[0] == '/')) {
2633                       make_state->hash.length = 0;
2634                       make_state->string_mb[0] = '\0';
2635                    }
2636                    sprintf(state_file_str_mb,NOCATGETS("%s%s"),
2637                     make_state->string_mb,NOCATGETS("/.make.state"));
2638                    make_state = &state_filename;
2639                      /* adjust the length to reflect the appended string */
2640                    make_state->hash.length += 12;
2641                  }
2642                } else { /* the file doesn't exist or no permission */
2643                   char tmp_path[MAXPATHLEN];
2644                   char *slashp;

2646                   if (slashp = strrchr(make_state->string_mb, '/')) {
2647                      strncpy(tmp_path, make_state->string_mb,
2648                              (slashp - make_state->string_mb));
2649                        tmp_path[slashp - make_state->string_mb]=0;
2650                      if(strlen(tmp_path)) {
2651                        if(stat(tmp_path, &make_state_stat)) {
2652                          warning(catgets(catd, 1, 192, "directory %s for .KEEP_
2653                        }
2654                        if (access(tmp_path, F_OK) != 0) {
2655                          warning(catgets(catd, 1, 193, "can't access dir %s"),t
2656                        }
2657                      }
2658                   }
2659                }
2660                if (report_dependencies_level != 1) {
2661                       Makefile_type   makefile_type_temp = makefile_type;
2662                       makefile_type = reading_statefile;
2663                       if (read_trace_level > 1) {
2664                              trace_reader = true;
2665                       }
2666                       (void) read_simple_file(make_state,
2667                                               false,
2668                                               false,
2669                                               false,
2670                                               false,
2671                                               false,
2672                                               true);
2673                       trace_reader = false;
2674                       makefile_type = makefile_type_temp;
2675                }
2676        }
2677 }
_____unchanged_portion_omitted_

2917 /*
2918  *      read_environment(read_only)
2919  *
2920  *      This routine reads the process environment when make starts and enters
2921  *      it as make macros. The environment variable SHELL is ignored.
2922  *
2923  *      Parameters:
2924  *              read_only       Should we make env vars read only?
2925  *
2926  *      Global variables used:
2927  *              report_pwd      Set if this make was started by other make
2928  */
2929 static void
2930 read_environment(Boolean read_only)
2931 {
```



```
2932        register char           **environment;
2933        int                     length;
2934        wchar_t                 *tmp_wcs_buffer;
2935        Boolean                 alloced_tmp_wcs_buffer = false;
2936        register wchar_t        *name;
2937        register wchar_t        *value;
2938        register Name           macro;
2939        Property                val;
2940        Boolean                 read_only_saved;

2942        reading_environment = true;
2943        environment = environ;
2944        for (; *environment; environment++) {
2945                read_only_saved = read_only;
2946                if ((length = strlen(*environment)) >= MAXPATHLEN) {
2947                        tmp_wcs_buffer = ALLOC_WC(length + 1);
2948                        alloced_tmp_wcs_buffer = true;
2949                        (void) mbstowcs(tmp_wcs_buffer, *environment, length + 1
2950                        name = tmp_wcs_buffer;
2951                } else {
2952                        MBSTOWCS(wcs_buffer, *environment);
2953                        name = wcs_buffer;
2954                }
2955                value = (wchar_t *) wschr(name, (int) equal_char);

2957                /*
2958                 * Looks like there's a bug in the system, but sometimes
2959                 * you can get blank lines in *environment.
2960                 */
2961                if (!value) {
2962                        continue;
2963                }
2964                MBSTOWCS(wcs_buffer2, NOCATGETS("SHELL="));
2965                if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2966                        continue;
2967                }
2968                MBSTOWCS(wcs_buffer2, NOCATGETS("MAKEFLAGS="));
2969                if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2970                        report_pwd = true;
2971                        /*
2972                         * In POSIX mode we do not want MAKEFLAGS to be readonly
2973                         * If the MAKEFLAGS macro is subsequently set by the mak
2974                         * it replaces the MAKEFLAGS variable currently found in
2975                         * environment.
2976                         * See Assertion 50 in section 6.2.5.3 of standard P1003
2977                         */
2978                        if(posix) {
2979                                read_only_saved = false;
2980                        }
2981                }

2983                /*
2984                 * We ignore SUNPRO_DEPENDENCIES. This environment variable is
2985                 * set by make and read by cpp which then writes info to
2986                 * .make.dependency.xxx.  When make is invoked by another make
2987                 * (recursive make), we don't want to read this because then
2988                 * the child make will end up writing to the parent
2989                 * directory's .make.state and clobbering them.
3047                 * We ignore SUNPRO_DEPENDENCIES and NSE_DEP. Those
3048                 * environment variables are set by make and read by
3049                 * cpp which then writes info to .make.dependency.xxx and
3050                 * .nse_depinfo. When make is invoked by another make
3051                 * (recursive make), we don't want to read this because
3052                 * then the child make will end up writing to the parent
3053                 * directory's .make.state and .nse_depinfo and clobbering
3054                 * them.
```

```
2990                  */
2991                 MBSTOWCS(wcs_buffer2, NOCATGETS("SUNPRO_DEPENDENCIES"));
2992                 if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2993                         continue;
2994                 }
2995 #ifdef NSE
2996                 MBSTOWCS(wcs_buffer2, NOCATGETS("NSE_DEP"));
2997                 if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2998                         continue;
2999                 }
3000 #endif

3001                 macro = GETNAME(name, value - name);
3002                 maybe_append_prop(macro, macro_prop)->body.macro.exported =
3003                   true;
3004                 if ((value == NULL) || ((value + 1)[0] == (int) nul_char)) {
3005                         val = setvar_daemon(macro,
3006                                             (Name) NULL,
3007                                             false, no_daemon, false, debug_level
3008                 } else {
3009                         val = setvar_daemon(macro,
3010                                             GETNAME(value + 1, FIND_LENGTH),
3011                                             false, no_daemon, false, debug_level
3012                 }
3013 #ifdef NSE
3014                 /*
3015                  * Must be after the call to setvar() as it sets
3016                  * imported to false.
3017                  */
3018                 maybe_append_prop(macro, macro_prop)->body.macro.imported = true
3019 #endif
3020                 val->body.macro.read_only = read_only_saved;
3021                 if (alloced_tmp_wcs_buffer) {
3022                         retmem(tmp_wcs_buffer);
3023                         alloced_tmp_wcs_buffer = false;
3024                 }
3025         }
3026         reading_environment = false;
3027 }

3028 /*
3029  *      read_makefile(makefile, complain, must_exist, report_file)
3030  *
3031  *      Read one makefile and check the result
3032  *
3033  *      Return value:
3034  *                              false is the read failed
3035  *
3036  *      Parameters:
3037  *              makefile        The file to read
3038  *              complain        Passed thru to read_simple_file()
3039  *              must_exist      Passed thru to read_simple_file()
3040  *              report_file     Passed thru to read_simple_file()
3041  *
3042  *      Global variables used:
3043  *              makefile_type   Set to indicate we are reading main file
3044  *              recursion_level Initialized
3045  */
3046 static Boolean
3047 read_makefile(register Name makefile, Boolean complain, Boolean must_exist, Bool
3048 {
3049         Boolean                 b;
3050
3051         makefile_type = reading_makefile;
3052         recursion_level = 0;
3053 #ifdef NSE
```



```
3054         wscpy(current_makefile, makefile->string);
3055 #endif
3056         reading_dependencies = true;
3057         b = read_simple_file(makefile, true, true, complain,
3058                              must_exist, report_file, false);
3059         reading_dependencies = false;
3060         return b;
3061 }
______unchanged_portion_omitted_

3062 /*
3063  *      report_recursion(target)
3064  *
3065  *      If this is a recursive make and the parent make has KEEP_STATE on
3066  *      this routine reports the dependency to the parent make
3067  *
3068  *      Parameters:
3069  *              target          Target to report
3070  *
3071  *      Global variables used:
3072  *              makefiles_used          List of makefiles read
3073  *              recursive_name          The Name ".RECURSIVE", printed
3074  *              report_dependency       dwight
3075  */
3076 static void
3077 report_recursion(register Name target)
3078 {
3079         register FILE           *report_file = get_report_file();

3080         if ((report_file == NULL) || (report_file == (FILE*)-1)) {
3081                 return;
3082         }
3083         if (primary_makefile == NULL) {
3084                 /*
3085                  * This can happen when there is no makefile and
3086                  * only implicit rules are being used.
3087                  */
3088 #ifdef NSE
3089                 nse_no_makefile(target);
3090 #endif
3091                 return;
3092         }
3093         (void) fprintf(report_file,
3094                        "%s: %s ",
3095                        get_target_being_reported_for(),
3096                        recursive_name->string_mb);
3097         report_dependency(get_current_path());
3098         report_dependency(target->string_mb);
3099         report_dependency(primary_makefile->string_mb);
3100         (void) fprintf(report_file, "\n");
3101 }
______unchanged_portion_omitted_
```

```
**********************************************************
   26169 Wed May 20 11:32:40 2015
new/usr/src/cmd/make/bin/misc.cc
make: unifdef for NSE (undefined)
**********************************************************
______unchanged_portion_omitted_

 568 /*************************************
 569  *
 570  *      main() support
 571  */

 573 /*
 574  *      load_cached_names()
 575  *
 576  *      Load the vector of cached names
 577  *
 578  *      Parameters:
 579  *
 580  *      Global variables used:
 581  *              Many many pointers to Name blocks.
 582  */
 583 void
 584 load_cached_names(void)
 585 {
 586         char            *cp;
 587         Name            dollar;

 589         /* Load the cached_names struct */
 590         MBSTOWCS(wcs_buffer, NOCATGETS(".BUILT_LAST_MAKE_RUN"));
 591         built_last_make_run = GETNAME(wcs_buffer, FIND_LENGTH);
 592         MBSTOWCS(wcs_buffer, NOCATGETS("@"));
 593         c_at = GETNAME(wcs_buffer, FIND_LENGTH);
 594         MBSTOWCS(wcs_buffer, NOCATGETS(" *conditionals* "));
 595         conditionals = GETNAME(wcs_buffer, FIND_LENGTH);
 596         /*
 597          * A version of make was released with NSE 1.0 that used
 598          * VERSION-1.1 but this version is identical to VERSION-1.0.
 599          * The version mismatch code makes a special case for this
 600          * situation.  If the version number is changed from 1.0
 601          * it should go to 1.2.
 602          */
 603         MBSTOWCS(wcs_buffer, NOCATGETS("VERSION-1.0"));
 604         current_make_version = GETNAME(wcs_buffer, FIND_LENGTH);
 605         MBSTOWCS(wcs_buffer, NOCATGETS(".SVR4"));
 606         svr4_name = GETNAME(wcs_buffer, FIND_LENGTH);
 607         MBSTOWCS(wcs_buffer, NOCATGETS(".POSIX"));
 608         posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
 609         MBSTOWCS(wcs_buffer, NOCATGETS(".DEFAULT"));
 610         default_rule_name = GETNAME(wcs_buffer, FIND_LENGTH);
 611 #ifdef NSE
 612         MBSTOWCS(wcs_buffer, NOCATGETS(".DERIVED_SRC"));
 613         derived_src= GETNAME(wcs_buffer, FIND_LENGTH);
 614 #endif
 611         MBSTOWCS(wcs_buffer, NOCATGETS("$"));
 612         dollar = GETNAME(wcs_buffer, FIND_LENGTH);
 613         MBSTOWCS(wcs_buffer, NOCATGETS(".DONE"));
 614         done = GETNAME(wcs_buffer, FIND_LENGTH);
 615         MBSTOWCS(wcs_buffer, NOCATGETS("."));
 616         dot = GETNAME(wcs_buffer, FIND_LENGTH);
 617         MBSTOWCS(wcs_buffer, NOCATGETS(".KEEP_STATE"));
 618         dot_keep_state = GETNAME(wcs_buffer, FIND_LENGTH);
 619         MBSTOWCS(wcs_buffer, NOCATGETS(".KEEP_STATE_FILE"));
 620         dot_keep_state_file = GETNAME(wcs_buffer, FIND_LENGTH);
 621         MBSTOWCS(wcs_buffer, NOCATGETS(""));
 622         empty_name = GETNAME(wcs_buffer, FIND_LENGTH);
```



```
 623         MBSTOWCS(wcs_buffer, NOCATGETS(" FORCE"));
 624         force = GETNAME(wcs_buffer, FIND_LENGTH);
 625         MBSTOWCS(wcs_buffer, NOCATGETS("HOST_ARCH"));
 626         host_arch = GETNAME(wcs_buffer, FIND_LENGTH);
 627         MBSTOWCS(wcs_buffer, NOCATGETS("HOST_MACH"));
 628         host_mach = GETNAME(wcs_buffer, FIND_LENGTH);
 629         MBSTOWCS(wcs_buffer, NOCATGETS(".IGNORE"));
 630         ignore_name = GETNAME(wcs_buffer, FIND_LENGTH);
 631         MBSTOWCS(wcs_buffer, NOCATGETS(".INIT"));
 632         init = GETNAME(wcs_buffer, FIND_LENGTH);
 633         MBSTOWCS(wcs_buffer, NOCATGETS(".LOCAL"));
 634         localhost_name = GETNAME(wcs_buffer, FIND_LENGTH);
 635         MBSTOWCS(wcs_buffer, NOCATGETS(".make.state"));
 636         make_state = GETNAME(wcs_buffer, FIND_LENGTH);
 637         MBSTOWCS(wcs_buffer, NOCATGETS("MAKEFLAGS"));
 638         makeflags = GETNAME(wcs_buffer, FIND_LENGTH);
 639         MBSTOWCS(wcs_buffer, NOCATGETS(".MAKE_VERSION"));
 640         make_version = GETNAME(wcs_buffer, FIND_LENGTH);
 641         MBSTOWCS(wcs_buffer, NOCATGETS(".NO_PARALLEL"));
 642         no_parallel_name = GETNAME(wcs_buffer, FIND_LENGTH);
 643         MBSTOWCS(wcs_buffer, NOCATGETS(".NOT_AUTO"));
 644         not_auto = GETNAME(wcs_buffer, FIND_LENGTH);
 645         MBSTOWCS(wcs_buffer, NOCATGETS(".PARALLEL"));
 646         parallel_name = GETNAME(wcs_buffer, FIND_LENGTH);
 647         MBSTOWCS(wcs_buffer, NOCATGETS("PATH"));
 648         path_name = GETNAME(wcs_buffer, FIND_LENGTH);
 649         MBSTOWCS(wcs_buffer, NOCATGETS("+"));
 650         plus = GETNAME(wcs_buffer, FIND_LENGTH);
 651         MBSTOWCS(wcs_buffer, NOCATGETS(".PRECIOUS"));
 652         precious = GETNAME(wcs_buffer, FIND_LENGTH);
 653         MBSTOWCS(wcs_buffer, NOCATGETS("?"));
 654         query = GETNAME(wcs_buffer, FIND_LENGTH);
 655         MBSTOWCS(wcs_buffer, NOCATGETS("^"));
 656         hat = GETNAME(wcs_buffer, FIND_LENGTH);
 657         MBSTOWCS(wcs_buffer, NOCATGETS(".RECURSIVE"));
 658         recursive_name = GETNAME(wcs_buffer, FIND_LENGTH);
 659         MBSTOWCS(wcs_buffer, NOCATGETS(".SCCS_GET"));
 660         sccs_get_name = GETNAME(wcs_buffer, FIND_LENGTH);
 661         MBSTOWCS(wcs_buffer, NOCATGETS(".SCCS_GET_POSIX"));
 662         sccs_get_posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
 663         MBSTOWCS(wcs_buffer, NOCATGETS(".GET"));
 664         get_name = GETNAME(wcs_buffer, FIND_LENGTH);
 665         MBSTOWCS(wcs_buffer, NOCATGETS(".GET_POSIX"));
 666         get_posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
 667         MBSTOWCS(wcs_buffer, NOCATGETS("SHELL"));
 668         shell_name = GETNAME(wcs_buffer, FIND_LENGTH);
 669         MBSTOWCS(wcs_buffer, NOCATGETS(".SILENT"));
 670         silent_name = GETNAME(wcs_buffer, FIND_LENGTH);
 671         MBSTOWCS(wcs_buffer, NOCATGETS(".SUFFIXES"));
 672         suffixes_name = GETNAME(wcs_buffer, FIND_LENGTH);
 673         MBSTOWCS(wcs_buffer, SUNPRO_DEPENDENCIES);
 674         sunpro_dependencies = GETNAME(wcs_buffer, FIND_LENGTH);
 675         MBSTOWCS(wcs_buffer, NOCATGETS("TARGET_ARCH"));
 676         target_arch = GETNAME(wcs_buffer, FIND_LENGTH);
 677         MBSTOWCS(wcs_buffer, NOCATGETS("TARGET_MACH"));
 678         target_mach = GETNAME(wcs_buffer, FIND_LENGTH);
 679         MBSTOWCS(wcs_buffer, NOCATGETS("VIRTUAL_ROOT"));
 680         virtual_root = GETNAME(wcs_buffer, FIND_LENGTH);
 681         MBSTOWCS(wcs_buffer, NOCATGETS("VPATH"));
 682         vpath_name = GETNAME(wcs_buffer, FIND_LENGTH);
 683         MBSTOWCS(wcs_buffer, NOCATGETS(".WAIT"));
 684         wait_name = GETNAME(wcs_buffer, FIND_LENGTH);

 686         wait_name->state = build_ok;

 688         /* Mark special targets so that the reader treats them properly */
```

```
689         svr4_name->special_reader = svr4_special;
690         posix_name->special_reader = posix_special;
691         built_last_make_run->special_reader = built_last_make_run_special;
692         default_rule_name->special_reader = default_special;
697 #ifdef NSE
698         derived_src->special_reader= derived_src_special;
699 #endif
693         dot_keep_state->special_reader = keep_state_special;
694         dot_keep_state_file->special_reader = keep_state_file_special;
695         ignore_name->special_reader = ignore_special;
696         make_version->special_reader = make_version_special;
697         no_parallel_name->special_reader = no_parallel_special;
698         parallel_name->special_reader = parallel_special;
699         localhost_name->special_reader = localhost_special;
700         precious->special_reader = precious_special;
701         sccs_get_name->special_reader = sccs_get_special;
702         sccs_get_posix_name->special_reader = sccs_get_posix_special;
703         get_name->special_reader = get_special;
704         get_posix_name->special_reader = get_posix_special;
705         silent_name->special_reader = silent_special;
706         suffixes_name->special_reader = suffixes_special;

708         /* The value of $$ is $ */
709         (void) SETVAR(dollar, dollar, false);
710         dollar->dollar = false;

712         /* Set the value of $(SHELL) */
713         if (posix) {
714           MBSTOWCS(wcs_buffer, NOCATGETS("/usr/xpg4/bin/sh"));
715         } else {
716           MBSTOWCS(wcs_buffer, NOCATGETS("/bin/sh"));
717         }
718         (void) SETVAR(shell_name, GETNAME(wcs_buffer, FIND_LENGTH), false);

720         /*
721          * Use " FORCE" to simulate a FRC dependency for :: type
722          * targets with no dependencies.
723          */
724         (void) append_prop(force, line_prop);
725         force->stat.time = file_max_time;

727         /* Make sure VPATH is defined before current dir is read */
728         if ((cp = getenv(vpath_name->string_mb)) != NULL) {
729                 MBSTOWCS(wcs_buffer, cp);
730                 (void) SETVAR(vpath_name,
731                               GETNAME(wcs_buffer, FIND_LENGTH),
732                               false);
733         }

735         /* Check if there is NO PATH variable. If not we construct one. */
736         if (getenv(path_name->string_mb) == NULL) {
737                 vroot_path = NULL;
738                 add_dir_to_path(NOCATGETS("."), &vroot_path, -1);
739                 add_dir_to_path(NOCATGETS("/bin"), &vroot_path, -1);
740                 add_dir_to_path(NOCATGETS("/usr/bin"), &vroot_path, -1);
741         }
742 }
```

_____***unchanged_portion_omitted_***

****
8764 Wed May 20 11:32:41 2015
new/usr/src/cmd/make/bin/nse_printdep.cc
make: unifdef for NSE (undefined)
****
_____***unchanged_portion_omitted_***

```
 141 /*
 142  *      print_deps(target, line, go_recursive)
 143  *
 144  *      Print a regular dependency list.  Append to this information which
 145  *      indicates whether or not the target is recursive.
 146  *
 147  *      Parameters:
 148  *              target          target to print dependencies for
 149  *              line            We get the dependency list from here
 150  *              go_recursive    Should we show all dependencies recursively?
 151  *
 152  *      Global variables used:
 153  *              recursive_name  The Name ".RECURSIVE", printed
 154  */
 155 static void
 156 print_deps(register Name target, register Property line)
 157 {
 158         register Dependency     dep;

 160         if ((target->dependency_printed) ||
 161             (target == force)) {
 162                 return;
 163         }
 164         target->dependency_printed = true;

 166         /* only print entries that are actually derived and are not leaf
 167          * files and are not the result of sccs get.
 168          */
 169         if (should_print_dep(line)) {
 170 #ifdef NSE
 171                 nse_check_no_deps_no_rule(target, line, line);
 172 #endif
 170                 if ((report_dependencies_level == 2) ||
 171                     (report_dependencies_level == 4)) {
 172                         if (is_out_of_date(line)) {
 173                                 (void) printf("1 ");
 174                         } else {
 175                                 (void) printf("0 ");
 176                         }
 177                 }
 178                 print_filename(target);
 179                 (void) printf(":\t");
 180                 print_deplist(line->body.line.dependencies);
 181                 print_rec_info(target);
 182                 (void) printf("\n");
 183                 for (dep = line->body.line.dependencies;
 184                      dep != NULL;
 185                      dep = dep->next) {
 186                         print_deps(dep->name,
 187                                    get_prop(dep->name->prop, line_prop));
 188                 }
 189         }
 190 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   56895 Wed May 20 11:32:42 2015
new/usr/src/cmd/make/bin/read.cc
make: unifdef for NSE (undefined)
**********************************************************
```

```
/*
 * 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 <avo/avo_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() */


/*
 * 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);

#ifdef NSE
        if (report_file){
                wscpy(previous_current_makefile, current_makefile);
                wscpy(current_makefile, wcb);
        }
#endif
        if (max_include_depth++ >= 40) {
                fatal(catgets(catd, 1, 66, "Too many nested include statements")
        }
```

```
122         if (makefile->body.makefile.contents != NULL) {
123                 retmem(makefile->body.makefile.contents);
124         }
125         source->inp_buf =
126           source->inp_buf_ptr =
127             source->inp_buf_end = NULL;
128         source->error_converting = false;
129         makefile->body.makefile.contents = NULL;
130         makefile->body.makefile.size = 0;
131         if ((makefile_name->hash.length != 1) ||
132             (wcb[0] != (int) hyphen_char)) {
133                 if ((makefile->body.makefile.contents == NULL) &&
134                     (doname_it)) {
135                         if (makefile_path == NULL) {
136                                 add_dir_to_path(".",
137                                                 &makefile_path,
138                                                 -1);
139                                 add_dir_to_path(NOCATGETS("/usr/share/lib/make")
140                                                 &makefile_path,
141                                                 -1);
142                                 add_dir_to_path(NOCATGETS("/etc/default"),
143                                                 &makefile_path,
144                                                 -1);
145                         }
146                         save_makefile_type = makefile_type;
147                         makefile_type = reading_nothing;
148                         if (doname(makefile_name, true, false) == build_dont_kno
149                                 /* Try normalized filename */
150                                 string_start=get_wstring(makefile_name->string_m
151                                 for (string_end=string_start+1; *string_end != L
152                                 normalized_makefile_name=normalize_name(string_s
153                                 if ((strcmp(makefile_name->string_mb, normalized
154                                         (doname(normalized_makefile_name, true,
155                                         n = access_vroot(makefile_name->string_m
156                                                  4,
157                                                  chase_path ?
158                                                  makefile_path : NULL,
159                                                  VROOT_DEFAULT);
160                                         if (n == 0) {
161                                                 get_vroot_path((char **) NULL,
162                                                        &path,
163                                                        (char **) NULL);
164                                                 if ((path[0] == (int) period_cha
165                                                     (path[1] == (int) slash_char
166                                                          path += 2;
167                                                 }
168                                                 MBSTOWCS(wcs_buffer, path);
169                                                 makefile_name = GETNAME(wcs_buff
170                                                                 FIND_LENGTH);
171                                         }
172                                 }
173                                 retmem(string_start);
174                                 /*
175                                  * Commented out: retmem_mb(normalized_makefile_
176                                  * We have to return this memory, but it seems t
177                                  * in dmake or in Sun C++ 5.7 compiler (it works
178                                  * is compiled using Sun C++ 5.6).
179                                  */
180                                 // retmem_mb(normalized_makefile_name->string_mb
181                         }
182                         makefile_type = save_makefile_type;
183                 }
184                 source->string.free_after_use = false;
185                 source->previous = NULL;
186                 source->already_expanded = false;
187                 /* Lock the file for read, but not when -n. */
```



```
188                 if (lock_makefile &&
189                     !do_not_exec_rule) {
190 
191                         make_state_lockfile = getmem(strlen(make_state->string_
192                         (void) sprintf(make_state_lockfile,
193                                                 NOCATGETS("%s.lock"),
194                                                 make_state->string_mb);
195                         (void) file_lock(make_state->string_mb,
196                                          make_state_lockfile,
197                                          (int *) &make_state_locked,
198                                          0);
199                         if(!make_state_locked) {
200                                 printf(NOCATGETS("-- NO LOCKING for read\n"));
201                                 retmem_mb(make_state_lockfile);
202                                 make_state_lockfile = 0;
203                                 return failed;
204                         }
205                 }
206                 if (makefile->body.makefile.contents == NULL) {
207                         save_makefile_type = makefile_type;
208                         makefile_type = reading_nothing;
209                         if ((doname_it) &&
210                             (doname(makefile_name, true, false) == build_failed)
211                                 if (complain) {
212                                         (void) fprintf(stderr,
213 #ifdef DISTRIBUTED
214                                                        catgets(catd, 1, 67, "dma
215 #else
216                                                        catgets(catd, 1, 237, "ma
217 #endif
218                                                        makefile_name->string_mb)
219                                 }
220                                 max_include_depth--;
221                                 makefile_type = save_makefile_type;
222                                 return failed;
223                         }
224                         makefile_type = save_makefile_type;
225                         //
226                         // Before calling exists() make sure that we have the ri
227                         //
228                         makefile_name->stat.time = file_no_time;
229 
230                         if (exists(makefile_name) == file_doesnt_exist) {
231                                 if (complain ||
232                                     (makefile_name->stat.stat_errno != ENOENT))
233                                         if (must_exist) {
234                                                 fatal(catgets(catd, 1, 68, "Can'
235                                                       makefile_name->string_mb,
236                                                       errmsg(makefile_name->
237                                                              stat.stat_errno));
238                                         } else {
239                                                 warning(catgets(catd, 1, 69, "Ca
240                                                         makefile_name->string_mb
241                                                         errmsg(makefile_name->
242                                                                stat.stat_errno))
243                                         }
244                                 }
245                                 max_include_depth--;
246                                 if(make_state_locked && (make_state_lockfile !=
247                                         (void) unlink(make_state_lockfile);
248                                         retmem_mb(make_state_lockfile);
249                                         make_state_lockfile = NULL;
250                                         make_state_locked = false;
251                                 }
252                                 retmem(wcb);
253                                 retmem_mb((char *)source);
```

```
254                             return failed;
255                     }
256                     /*
257                      * These values are the size and bytes of
258                      * the MULTI-BYTE makefile.
259                      */
260                     orig_makefile->body.makefile.size =
261                       makefile->body.makefile.size =
262                         source->bytes_left_in_file =
263                           makefile_name->stat.size;
264                     if (report_file) {
265                             for (dpp = &makefiles_used;
266                                  *dpp != NULL;
267                                  dpp = &(*dpp)->next);
268                             dp = ALLOC(Dependency);
269                             dp->next = NULL;
270                             dp->name = makefile_name;
271                             dp->automatic = false;
272                             dp->stale = false;
273                             dp->built = false;
274                             *dpp = dp;
275                     }
276                     source->fd = open_vroot(makefile_name->string_mb,
277                                             O_RDONLY,
278                                             0,
279                                             NULL,
280                                             VROOT_DEFAULT);
281                     if (source->fd < 0) {
282                             if (complain || (errno != ENOENT)) {
283                                     if (must_exist) {
284                                             fatal(catgets(catd, 1, 70, "Can'
285                                                   makefile_name->string_mb,
286                                                   errmsg(errno));
287                                     } else {
288                                             warning(catgets(catd, 1, 71, "Ca
289                                                     makefile_name->string_mb
290                                                     errmsg(errno));
291                                     }
292                             }
293                             max_include_depth--;
294                             return failed;
295                     }
296                     (void) fcntl(source->fd, F_SETFD, 1);
297                     orig_makefile->body.makefile.contents =
298                       makefile->body.makefile.contents =
299                         source->string.text.p =
300                           source->string.buffer.start =
301                             ALLOC_WC((int) (makefile_name->stat.size + 2));
302                     if (makefile_type == reading_cpp_file) {
303                             forget_after_parse = true;
304                     }
305                     source->string.text.end = source->string.text.p;
306                     source->string.buffer.end =
307                       source->string.text.p + makefile_name->stat.size;
308             } else {
309                     /* Do we ever reach here? */
310                     source->fd = -1;
311                     source->string.text.p =
312                       source->string.buffer.start =
313                         makefile->body.makefile.contents;
314                     source->string.text.end =
315                       source->string.buffer.end =
316                         source->string.text.p + makefile->body.makefile.size
317                     source->bytes_left_in_file =
318                       makefile->body.makefile.size;
319             }
```



```
320                     file_being_read = wcb;
321             } else {
322                     char            *stdin_text_p;
323                     char            *stdin_text_end;
324                     char            *stdin_buffer_start;
325                     char            *stdin_buffer_end;
326                     char            *p_mb;
327                     int             num_mb_chars;
328                     size_t          num_wc_chars;
329
330                     MBSTOWCS(wcs_buffer, NOCATGETS("Standard in"));
331                     makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
332                     /*
333                      * Memory to read standard in, then convert it
334                      * to wide char strings.
335                      */
336                     stdin_buffer_start =
337                       stdin_text_p = getmem(length = 1024);
338                     stdin_buffer_end = stdin_text_p + length;
339                     MBSTOWCS(wcs_buffer, NOCATGETS("standard input"));
340                     file_being_read = (wchar_t *) wsdup(wcs_buffer);
341                     line_number = 0;
342                     while ((n = read(fileno(stdin),
343                                      stdin_text_p,
344                                      length)) > 0) {
345                             length -= n;
346                             stdin_text_p += n;
347                             if (length == 0) {
348                                     p_mb = getmem(length = 1024 +
349                                                   (stdin_buffer_end -
350                                                    stdin_buffer_start));
351                                     (void) strncpy(p_mb,
352                                                    stdin_buffer_start,
353                                                    (stdin_buffer_end -
354                                                     stdin_buffer_start));
355                                     retmem_mb(stdin_buffer_start);
356                                     stdin_text_p = p_mb +
357                                       (stdin_buffer_end - stdin_buffer_start);
358                                     stdin_buffer_start = p_mb;
359                                     stdin_buffer_end =
360                                       stdin_buffer_start + length;
361                                     length = 1024;
362                             }
363                     }
364                     if (n < 0) {
365                             fatal(catgets(catd, 1, 72, "Error reading standard input
366                                   errmsg(errno));
367                     }
368                     stdin_text_p = stdin_buffer_start;
369                     stdin_text_end = stdin_buffer_end - length;
370                     num_mb_chars = stdin_text_end - stdin_text_p;
371
372                     /*
373                      * Now, convert the sequence of multibyte chars into
374                      * a sequence of corresponding wide character codes.
375                      */
376                     source->string.free_after_use = false;
377                     source->previous = NULL;
378                     source->bytes_left_in_file = 0;
379                     source->fd = -1;
380                     source->already_expanded = false;
381                     source->string.buffer.start =
382                       source->string.text.p = ALLOC_WC(num_mb_chars + 1);
383                     source->string.buffer.end =
384                         source->string.text.p + num_mb_chars;
385                     num_wc_chars = mbstowcs(source->string.text.p,
```

```
386                                         stdin_text_p,
387                                         num_mb_chars);
388                 if ((int) num_wc_chars >= 0) {
389                         source->string.text.end =
390                           source->string.text.p + num_wc_chars;
391                 }
392                 (void) retmem_mb(stdin_text_p);
393         }
394         line_number = 1;
395         if (trace_reader) {
396                 (void) printf(catgets(catd, 1, 73, ">>>>>>>>>>>>>>>> Reading mak
397                               makefile_name->string_mb);
398         }
399         parse_makefile(makefile_name, source);
400         if (trace_reader) {
401                 (void) printf(catgets(catd, 1, 74, ">>>>>>>>>>>>>>>> End of make
402                               makefile_name->string_mb);
403         }
410 #ifdef NSE
411         if (report_file && (previous_current_makefile[0] != NULL)) {
412                 wscpy(current_makefile, previous_current_makefile);
413         }
414 #endif
404         if(file_being_read) {
405                 retmem(file_being_read);
406         }
407         file_being_read = previous_file_being_read;
408         line_number = previous_line_number;
409         makefile_type = reading_nothing;
410         max_include_depth--;
411         if (make_state_locked) {
412                 /* Unlock .make.state. */
413                 unlink(make_state_lockfile);
414                 make_state_locked = false;
415                 retmem_mb(make_state_lockfile);
416         }
417         if (forget_after_parse) {
418                 retmem(makefile->body.makefile.contents);
419                 makefile->body.makefile.contents = NULL;
420         }
421         retmem_mb((char *)source);
422         return succeeded;
423 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   51721 Wed May 20 11:32:42 2015
new/usr/src/cmd/make/bin/read2.cc
make: unifdef for NSE (undefined)
**********************************************************
______unchanged_portion_omitted_

 537 /*
 538  *      enter_dependencies(target, target_group, depes, command, separator)
 539  *
 540  *      Take one target and a list of dependencies and process the whole thing.
 541  *      The target might be special in some sense in which case that is handled
 542  *
 543  *      Parameters:
 544  *              target          The target we want to enter
 545  *              target_group    Non-NULL if target is part of a group this time
 546  *              depes           A list of dependencies for the target
 547  *              command         The command the target should be entered with
 548  *              separator       Indicates if this is a ":" or a "::" rule
 549  *
 550  *      Static variables used:
 551  *              built_last_make_run_seen If the previous target was
 552  *                                      .BUILT_LAST_MAKE_RUN we say to rewrite
 553  *                                      the state file later on
 554  *
 555  *      Global variables used:
 556  *              command_changed Set to indicate if .make.state needs rewriting
 557  *              default_target_to_build Set to the target if reading makefile
 558  *                                      and this is the first regular target
 559  *              force           The Name " FORCE", used with "::" targets
 560  *              makefile_type   We do different things for makefile vs. report
 561  *              not_auto        The Name ".NOT_AUTO", compared against
 562  *              recursive_name  The Name ".RECURSIVE", compared against
 563  *              temp_file_number Used to figure out when to clear stale
 564  *                                      automatic dependencies
 565  *              trace_reader    Indicates that we should echo stuff we read
 566  */
 567 void
 568 enter_dependencies(register Name target, Chain target_group, register Name_vecto
 569 {
 570         register int            i;
 571         register Property       line;
 572         Name                    name;
 573         Name                    directory;
 574         wchar_t                 *namep;
 575         char                    *mb_namep;
 576         Dependency              dp;
 577         Dependency              *dpp;
 578         Property                line2;
 579         wchar_t                 relative[MAXPATHLEN];
 580         register int            recursive_state;
 581         Boolean                 register_as_auto;
 582         Boolean                 not_auto_found;
 583         char                    *slash;
 584         Wstring                 depstr;

 586         /* Check if this is a .RECURSIVE line */
 587         if ((depes->used >= 3) &&
 588             (depes->names[0] == recursive_name)) {
 589 #ifdef NSE
 590                 nse_did_recursion= true;
 591 #endif
 589                 target->has_recursive_dependency = true;
 590                 depes->names[0] = NULL;
 591                 recursive_state = 0;
 592                 dp = NULL;
```



```
 593                 dpp = &dp;
 594                 /* Read the dependencies. They are "<directory> <target-made>*/
 595                 /* <makefile>*" */
 596                 for (; depes != NULL; depes = depes->next) {
 597                         for (i = 0; i < depes->used; i++) {
 598                                 if (depes->names[i] != NULL) {
 599                                         switch (recursive_state++) {
 600                                         case 0: /* Directory */
 601                                         {
 602                                                 depstr.init(depes->names[i]);
 603                                                 make_relative(depstr.get_string(
 604                                                                 relative);
 605                                                 directory =
 606                                                   GETNAME(relative,
 607                                                           FIND_LENGTH);
 608                                         }
 609                                                 break;
 610                                         case 1: /* Target */
 611                                                 name = depes->names[i];
 612                                                 break;
 613                                         default:        /* Makefiles */
 614                                                 *dpp = ALLOC(Dependency);
 615                                                 (*dpp)->next = NULL;
 616                                                 (*dpp)->name = depes->names[i];
 617                                                 (*dpp)->automatic = false;
 618                                                 (*dpp)->stale = false;
 619                                                 (*dpp)->built = false;
 620                                                 dpp = &((*dpp)->next);
 621                                                 break;
 622                                         }
 623                                 }
 624                         }
 625                 }
 626                 /* Check if this recursion already has been reported else */
 627                 /* enter the recursive prop for the target */
 628                 /* The has_built flag is used to tell if this .RECURSIVE */
 629                 /* was discovered from this run (read from a tmp file) */
 630                 /* or was from discovered from the original .make.state */
 631                 /* file */
 632                 for (line = get_prop(target->prop, recursive_prop);
 633                      line != NULL;
 634                      line = get_prop(line->next, recursive_prop)) {
 635                         if ((line->body.recursive.directory == directory) &&
 636                             (line->body.recursive.target == name)) {
 637                                 line->body.recursive.makefiles = dp;
 638                                 line->body.recursive.has_built =
 639                                   (Boolean)
 640                                     (makefile_type == reading_cpp_file);
 641                                 return;
 642                         }
 643                 }
 644                 line2 = append_prop(target, recursive_prop);
 645                 line2->body.recursive.directory = directory;
 646                 line2->body.recursive.target = name;
 647                 line2->body.recursive.makefiles = dp;
 648                 line2->body.recursive.has_built =
 649                     (Boolean) (makefile_type == reading_cpp_file);
 650                 line2->body.recursive.in_depinfo = false;
 651                 return;
 652         }
 653         /* If this is the first target that doesnt start with a "." in the */
 654         /* makefile we remember that */
 655         Wstring tstr(target);
 656         wchar_t * wcb = tstr.get_string();
 657         if ((makefile_type == reading_makefile) &&
 658             (default_target_to_build == NULL) &&
```

```
659             ((wcb[0] != (int) period_char) ||
660              wschr(wcb, (int) slash_char))) {

662 /* BID 1181577: $(EMPTY_MACRO) + $(EMPTY_MACRO):
663 ** The target with empty name cannot be default_target_to_build
664 */
665                 if (target->hash.length != 0)
666                         default_target_to_build = target;
667         }
668         /* Check if the line is ":" or "::" */
669         if (makefile_type == reading_makefile) {
670                 if (target->colons == no_colon) {
671                         target->colons = separator;
672                 } else {
673                         if (target->colons != separator) {
674                                 fatal_reader(catgets(catd, 1, 92, ":/:: conflict
675                                                 target->string_mb);
676                         }
677                 }
678                 if (target->colons == two_colon) {
679                         if (depes->used == 0) {
680                                 /* If this is a "::" type line with no */
681                                 /* dependencies we add one "FRC" type */
682                                 /* dependency for free */
683                                 depes->used = 1; /* Force :: targets with no
684                                                   * depes to always run */
685                                 depes->names[0] = force;
686                         }
687                         /* Do not delete "::" type targets when interrupted */
688                         target->stat.is_precious = true;
689                         /*
690                          * Build a synthetic target "<number>%target"
691                          * for "target".
692                          */
693                         mb_namep = getmem((int) (strlen(target->string_mb) + 10)
694                         namep = ALLOC_WC((int) (target->hash.length + 10));
695                         slash = strrchr(target->string_mb, (int) slash_char);
696                         if (slash == NULL) {
697                                 (void) sprintf(mb_namep,
698                                                 "%d@%s",
699                                                 target->colon_splits++,
700                                                 target->string_mb);
701                         } else {
702                                 *slash = 0;
703                                 (void) sprintf(mb_namep,
704                                                 "%s/%d@%s",
705                                                 target->string_mb,
706                                                 target->colon_splits++,
707                                                 slash + 1);
708                                 *slash = (int) slash_char;
709                         }
710                         MBSTOWCS(namep, mb_namep);
711                         retmem_mb(mb_namep);
712                         name = GETNAME(namep, FIND_LENGTH);
713                         retmem(namep);
714                         if (trace_reader) {
715                                 (void) printf("%s:\t", target->string_mb);
716                         }
717                         /* Make "target" depend on "<number>%target */
718                         line2 = maybe_append_prop(target, line_prop);
719                         enter_dependency(line2, name, true);
720                         line2->body.line.target = target;
721                         /* Put a prop on "<number>%target that makes */
722                         /* appear as "target" */
723                         /* when it is processed */
724                         maybe_append_prop(name, target_prop)->
```



```
725                           body.target.target = target;
726                         target->is_double_colon_parent = true;
727                         name->is_double_colon = true;
728                         name->has_target_prop = true;
729                         if (trace_reader) {
730                                 (void) printf("\n");
731                         }
732                         (target = name)->stat.is_file = true;
733                 }
734         }
735         /* This really is a regular dependency line. Just enter it */
736         line = maybe_append_prop(target, line_prop);
737         line->body.line.target = target;
738         /* Depending on what kind of makefile we are reading we have to */
739         /* treat things differently */
740         switch (makefile_type) {
741         case reading_makefile:
742                 /* Reading regular makefile. Just notice whether this */
743                 /* redefines the rule for the  target */
744                 if (command != NULL) {
745                         if (line->body.line.command_template != NULL) {
746                                 line->body.line.command_template_redefined =
747                                   true;
748                                 if ((wcb[0] == (int) period_char) &&
749                                     !wschr(wcb, (int) slash_char)) {
750                                         line->body.line.command_template =
751                                           command;
752                                 }
753                         } else {
754                                 line->body.line.command_template = command;
755                         }
756                 } else {
757                         if ((wcb[0] == (int) period_char) &&
758                             !wschr(wcb, (int) slash_char)) {
759                                 line->body.line.command_template = command;
760                         }
761                 }
762                 break;
763         case rereading_statefile:
764                 /* Rereading the statefile. We only enter thing that changed */
765                 /* since the previous time we read it */
766                 if (!built_last_make_run_seen) {
767                         for (Cmd_line next, cmd = command; cmd != NULL; cmd = ne
768                                 next = cmd->next;
769                                 free(cmd);
770                         }
771                         return;
772                 }
773                 built_last_make_run_seen = false;
774                 command_changed = true;
775                 target->ran_command = true;
776         case reading_statefile:
777                 /* Reading the statefile for the first time. Enter the rules */
778                 /* as "Commands used" not "templates to use" */
779                 if (command != NULL) {
780                         for (Cmd_line next, cmd = line->body.line.command_used;
781                              cmd != NULL; cmd = next) {
782                                 next = cmd->next;
783                                 free(cmd);
784                         }
785                         line->body.line.command_used = command;
786                 }
787         case reading_cpp_file:
788                 /* Reading report file from programs that reports */
789                 /* dependencies. If this is the first time the target is */
790                 /* read from this reportfile we clear all old */
```

```
791                 /* automatic depes */
792                 if (target->temp_file_number == temp_file_number) {
793                         break;
794                 }
795                 target->temp_file_number = temp_file_number;
796                 command_changed = true;
797                 if (line != NULL) {
798                         for (dp = line->body.line.dependencies;
799                              dp != NULL;
800                              dp = dp->next) {
801                                 if (dp->automatic) {
802                                         dp->stale = true;
803                                 }
804                         }
805                 }
806                 break;
807         default:
808                 fatal_reader(catgets(catd, 1, 93, "Internal error. Unknown makef
809                         makefile_type);
810         }
811         /* A target may only be involved in one target group */
812         if (line->body.line.target_group != NULL) {
813                 if (target_group != NULL) {
814                         fatal_reader(catgets(catd, 1, 94, "Too many target group
815                                 target->string_mb);
816                 }
817         } else {
818                 line->body.line.target_group = target_group;
819         }

821         if (trace_reader) {
822                 (void) printf("%s:\t", target->string_mb);
823         }
824         /* Enter the dependencies */
825         register_as_auto = BOOLEAN(makefile_type != reading_makefile);
826         not_auto_found = false;
827         for (;
828              (depes != NULL) && !not_auto_found;
829              depes = depes->next) {
830                 for (i = 0; i < depes->used; i++) {
831                         /* the dependency .NOT_AUTO signals beginning of
832                          * explicit dependancies which were put at end of
833                          * list in .make.state file - we stop entering
834                          * dependencies at this point
835                          */
836                         if (depes->names[i] == not_auto) {
837                                 not_auto_found = true;
838                                 break;
839                         }
840                         enter_dependency(line,
841                                          depes->names[i],
842                                          register_as_auto);
843                 }
844         }
845         if (trace_reader) {
846                 (void) printf("\n");
847                 print_rule(command);
848         }
849 }

851 /*
852  *      enter_dependency(line, depe, automatic)
853  *
854  *      Enter one dependency. Do not enter duplicates.
855  *
856  *      Parameters:
```



```
857  *              line            The line block that the dependeny is
858  *                              entered for
859  *              depe            The dependency to enter
860  *              automatic       Used to set the field "automatic"
861  *
862  *      Global variables used:
863  *              makefile_type   We do different things for makefile vs. report
864  *              trace_reader    Indicates that we should echo stuff we read
865  *              wait_name       The Name ".WAIT", compared against
866  */
867 void
868 enter_dependency(Property line, register Name depe, Boolean automatic)
869 {
870         register Dependency     dp;
871         register Dependency     *insert;

873         if (trace_reader) {
874                 (void) printf("%s ", depe->string_mb);
875         }
876         /* Find the end of the list and check for duplicates */
877         for (insert = &line->body.line.dependencies, dp = *insert;
878              dp != NULL;
879              insert = &dp->next, dp = *insert) {
880                 if ((dp->name == depe) && (depe != wait_name)) {
881                         if (dp->automatic) {
882                                 dp->automatic = automatic;
883                                 if (automatic) {
884                                         dp->built = false;
885                                         depe->stat.is_file = true;
889 #ifdef NSE
890                                         depe->has_parent= true;
891                                         depe->is_target= true;
892 #endif
886                                 }
887                         }
888                         dp->stale = false;
889                         return;
890                 }
891         }
892         /* Insert the new dependency since we couldnt find it */
893         dp = *insert = ALLOC(Dependency);
894         dp->name = depe;
895         dp->next = NULL;
896         dp->automatic = automatic;
897         dp->stale = false;
898         dp->built = false;
899         depe->stat.is_file = true;
907 #ifdef NSE
908         depe->has_parent= true;
909         depe->is_target= true;
910 #endif

901         if ((makefile_type == reading_makefile) &&
902             (line != NULL) &&
903             (line->body.line.target != NULL)) {
904                 line->body.line.target->has_regular_dependency = true;
916 #ifdef NSE
917                 line->body.line.target->is_target= true;
918 #endif
905         }
906 }
______unchanged_portion_omitted_


1087 /*
1088  *      special_reader(target, depes, command)
```

```
 *
 *      Read the pseudo targets make knows about
 *      This handles the special targets that should not be entered as regular
 *      target/dependency sets.
 *
 *      Parameters:
 *              target          The special target
 *              depes           The list of dependencies it was entered with
 *              command         The command it was entered with
 *
 *      Static variables used:
 *              built_last_make_run_seen Set to indicate .BUILT_LAST... seen
 *
 *      Global variables used:
 *              all_parallel    Set to indicate that everything runs parallel
 *              svr4            Set when ".SVR4" target is read
 *              svr4_name       The Name ".SVR4"
 *              posix           Set when ".POSIX" target is read
 *              posix_name      The Name ".POSIX"
 *              current_make_version The Name "<current version number>"
 *              default_rule    Set when ".DEFAULT" target is read
 *              default_rule_name The Name ".DEFAULT", used for tracing
 *              dot_keep_state  The Name ".KEEP_STATE", used for tracing
 *              ignore_errors   Set if ".IGNORE" target is read
 *              ignore_name     The Name ".IGNORE", used for tracing
 *              keep_state      Set if ".KEEP_STATE" target is read
 *              no_parallel_name The Name ".NO_PARALLEL", used for tracing
 *              only_parallel   Set to indicate only some targets runs parallel
 *              parallel_name   The Name ".PARALLEL", used for tracing
 *              precious        The Name ".PRECIOUS", used for tracing
 *              sccs_get_name   The Name ".SCCS_GET", used for tracing
 *              sccs_get_posix_name The Name ".SCCS_GET_POSIX", used for tracing
 *              get_name        The Name ".GET", used for tracing
 *              sccs_get_rule   Set when ".SCCS_GET" target is read
 *              silent          Set when ".SILENT" target is read
 *              silent_name     The Name ".SILENT", used for tracing
 *              trace_reader    Indicates that we should echo stuff we read
 */
void
special_reader(Name target, register Name_vector depes, Cmd_line command)
{
        register int            n;

        switch (target->special_reader) {

        case svr4_special:
                if (depes->used != 0) {
                        fatal_reader(catgets(catd, 1, 98, "Illegal dependencies
                                        target->string_mb);
                }
                svr4  = true;
                posix  = false;
                keep_state = false;
                all_parallel = false;
                only_parallel = false;
                if (trace_reader) {
                        (void) printf("%s:\n", svr4_name->string_mb);
                }
                break;

        case posix_special:
                if(svr4)
                  break;
                if (depes->used != 0) {
                        fatal_reader(catgets(catd, 1, 99, "Illegal dependencies
                                        target->string_mb);
```



```
                }
                posix  = true;
                        /* with posix on, use the posix get rule */
                sccs_get_rule = sccs_get_posix_rule;
                        /* turn keep state off being SunPro make specific */
                keep_state = false;
                /* Use /usr/xpg4/bin/sh on Solaris */
                MBSTOWCS(wcs_buffer, NOCATGETS("/usr/xpg4/bin/sh"));
                (void) SETVAR(shell_name, GETNAME(wcs_buffer, FIND_LENGTH), fals
                if (trace_reader) {
                        (void) printf("%s:\n", posix_name->string_mb);
                }
                break;

        case built_last_make_run_special:
                built_last_make_run_seen = true;
                break;

        case default_special:
                if (depes->used != 0) {
                        warning(catgets(catd, 1, 100, "Illegal dependency list f
                                target->string_mb);
                }
                default_rule = command;
                if (trace_reader) {
                        (void) printf("%s:\n",
                                        default_rule_name->string_mb);
                        print_rule(command);
                }
                break;

#ifdef NSE
        case derived_src_special:
                for (; depes != NULL; depes= depes->next)
                        for (n= 0; n < depes->used; n++) {
                                if (trace_reader)
                                        (void)printf("%s:\t%s\n",
                                        precious->string_mb,
                                        depes->names[n]->string_mb);
                                depes->names[n]->stat.is_derived_src= true;
                        };
                break;
#endif

        case ignore_special:
                if ((depes->used != 0) &&(!posix)){
                        fatal_reader(catgets(catd, 1, 101, "Illegal dependencies
                                        target->string_mb);
                }
                if (depes->used == 0)
                {
                   ignore_errors_all = true;
                }
                if(svr4) {
                  ignore_errors_all = true;
                  break;
                }
                for (; depes != NULL; depes = depes->next) {
                        for (n = 0; n < depes->used; n++) {
                                depes->names[n]->ignore_error_mode = true;
                        }
                }
                if (trace_reader) {
                        (void) printf("%s:\n", ignore_name->string_mb);
                }
                break;
```

```
1210        case keep_state_special:
1211                if(svr4)
1212                  break;
1213                        /* ignore keep state, being SunPro make specific */
1214                if(posix)
1215                  break;
1216                if (depes->used != 0) {
1217                        fatal_reader(catgets(catd, 1, 102, "Illegal dependencies
1218                                     target->string_mb);
1219                }
1220                keep_state = true;
1221                if (trace_reader) {
1222                        (void) printf("%s:\n",
1223                                      dot_keep_state->string_mb);
1224                }
1225                break;

1227        case keep_state_file_special:
1228                if(svr4)
1229                  break;
1230                if(posix)
1231                  break;
1232                        /* it's not necessary to specify KEEP_STATE, if this
1233                        ** is given, so set the keep_state.
1234                        */
1235                keep_state = true;
1236                if (depes->used != 0) {
1237                   if((!make_state) ||(!strcmp(make_state->string_mb,NOCATGETS("
1238                     make_state = depes->names[0];
1239                   }
1240                }
1241                break;
1242        case make_version_special:
1243                if(svr4)
1244                  break;
1245                if (depes->used != 1) {
1246                        fatal_reader(catgets(catd, 1, 103, "Illegal dependency l
1247                                     target->string_mb);
1248                }
1249                if (depes->names[0] != current_make_version) {
1250                        /*
1251                         * Special case the fact that version 1.0 and 1.1
1252                         * are identical.
1253                         */
1254                        if (!IS_EQUAL(depes->names[0]->string_mb,
1255                                      NOCATGETS("VERSION-1.1")) ||
1256                            !IS_EQUAL(current_make_version->string_mb,
1257                                      NOCATGETS("VERSION-1.0"))) {
1258                                /*
1259                                 * Version mismatches should cause the
1260                                 * .make.state file to be skipped.
1261                                 * This is currently not true - it is read
1262                                 * anyway.
1263                                 */
1264                                warning(catgets(catd, 1, 104, "Version mismatch
1265                                        current_make_version->string_mb,
1266                                        depes->names[0]->string_mb);
1267                        }
1268                }
1269                break;

1271        case no_parallel_special:
1272                if(svr4)
1273                  break;
1274                /* Set the no_parallel bit for all the targets on */
```



```
1275                /* the dependency list */
1276                if (depes->used == 0) {
1277                        /* only those explicitly made parallel */
1278                        only_parallel = true;
1279                        all_parallel = false;
1280                }
1281                for (; depes != NULL; depes = depes->next) {
1282                        for (n = 0; n < depes->used; n++) {
1283                                if (trace_reader) {
1284                                        (void) printf("%s:\t%s\n",
1285                                                      no_parallel_name->string_m
1286                                                      depes->names[n]->string_mb
1287                                }
1288                                depes->names[n]->no_parallel = true;
1289                                depes->names[n]->parallel = false;
1290                        }
1291                }
1292                break;

1294        case parallel_special:
1295                if(svr4)
1296                  break;
1297                if (depes->used == 0) {
1298                        /* everything runs in parallel */
1299                        all_parallel = true;
1300                        only_parallel = false;
1301                }
1302                /* Set the parallel bit for all the targets on */
1303                /* the dependency list */
1304                for (; depes != NULL; depes = depes->next) {
1305                        for (n = 0; n < depes->used; n++) {
1306                                if (trace_reader) {
1307                                        (void) printf("%s:\t%s\n",
1308                                                      parallel_name->string_mb,
1309                                                      depes->names[n]->string_mb
1310                                }
1311                                depes->names[n]->parallel = true;
1312                                depes->names[n]->no_parallel = false;
1313                        }
1314                }
1315                break;

1317        case localhost_special:
1318                if(svr4)
1319                  break;
1320                /* Set the no_parallel bit for all the targets on */
1321                /* the dependency list */
1322                if (depes->used == 0) {
1323                        /* only those explicitly made parallel */
1324                        only_parallel = true;
1325                        all_parallel = false;
1326                }
1327                for (; depes != NULL; depes = depes->next) {
1328                        for (n = 0; n < depes->used; n++) {
1329                                if (trace_reader) {
1330                                        (void) printf("%s:\t%s\n",
1331                                                      localhost_name->string_mb,
1332                                                      depes->names[n]->string_mb
1333                                }
1334                                depes->names[n]->no_parallel = true;
1335                                depes->names[n]->parallel = false;
1336                                depes->names[n]->localhost = true;
1337                        }
1338                }
1339                break;
```

```
1341         case precious_special:
1342                 if (depes->used == 0) {
1343                         /* everything is precious      */
1344                         all_precious = true;
1345                 } else {
1346                         all_precious = false;
1347                 }
1348                 if(svr4) {
1349                   all_precious = true;
1350                   break;
1351                 }
1352                 /* Set the precious bit for all the targets on */
1353                 /* the dependency list */
1354                 for (; depes != NULL; depes = depes->next) {
1355                         for (n = 0; n < depes->used; n++) {
1356                                 if (trace_reader) {
1357                                         (void) printf("%s:\t%s\n",
1358                                                         precious->string_mb,
1359                                                         depes->names[n]->string_mb
1360                                 }
1361                                 depes->names[n]->stat.is_precious = true;
1362                         }
1363                 }
1364                 break;

1366         case sccs_get_special:
1367                 if (depes->used != 0) {
1368                         fatal_reader(catgets(catd, 1, 105, "Illegal dependencies
1369                                         target->string_mb);
1370                 }
1371                 sccs_get_rule = command;
1372                 sccs_get_org_rule = command;
1373                 if (trace_reader) {
1374                         (void) printf("%s:\n", sccs_get_name->string_mb);
1375                         print_rule(command);
1376                 }
1377                 break;

1379         case sccs_get_posix_special:
1380                 if (depes->used != 0) {
1381                         fatal_reader(catgets(catd, 1, 106, "Illegal dependencies
1382                                         target->string_mb);
1383                 }
1384                 sccs_get_posix_rule = command;
1385                 if (trace_reader) {
1386                         (void) printf("%s:\n", sccs_get_posix_name->string_mb);
1387                         print_rule(command);
1388                 }
1389                 break;

1391         case get_posix_special:
1392                 if (depes->used != 0) {
1393                         fatal_reader(catgets(catd, 1, 107, "Illegal dependencies
1394                                         target->string_mb);
1395                 }
1396                 get_posix_rule = command;
1397                 if (trace_reader) {
1398                         (void) printf("%s:\n", get_posix_name->string_mb);
1399                         print_rule(command);
1400                 }
1401                 break;

1403         case get_special:
1404                 if(!svr4) {
1405                   break;
1406                 }
```



```
1407                 if (depes->used != 0) {
1408                         fatal_reader(catgets(catd, 1, 108, "Illegal dependencies
1409                                         target->string_mb);
1410                 }
1411                 get_rule = command;
1412                 sccs_get_rule = command;
1413                 if (trace_reader) {
1414                         (void) printf("%s:\n", get_name->string_mb);
1415                         print_rule(command);
1416                 }
1417                 break;

1419         case silent_special:
1420                 if ((depes->used != 0) && (!posix)){
1421                         fatal_reader(catgets(catd, 1, 109, "Illegal dependencies
1422                                         target->string_mb);
1423                 }
1424                 if (depes->used == 0)
1425                 {
1426                    silent_all = true;
1427                 }
1428                 if(svr4) {
1429                   silent_all = true;
1430                   break;
1431                 }
1432                 for (; depes != NULL; depes = depes->next) {
1433                         for (n = 0; n < depes->used; n++) {
1434                                 depes->names[n]->silent_mode = true;
1435                         }
1436                 }
1437                 if (trace_reader) {
1438                         (void) printf("%s:\n", silent_name->string_mb);
1439                 }
1440                 break;

1442         case suffixes_special:
1443                 read_suffixes_list(depes);
1444                 break;

1446         default:

1448                 fatal_reader(catgets(catd, 1, 110, "Internal error: Unknown spec
1449         }
1450 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    9646 Wed May 20 11:32:43 2015
new/usr/src/cmd/make/bin/rep.cc
make: unifdef for NSE (undefined)
**********************************************************
______unchanged_portion_omitted_
```

```
315 #ifdef NSE
316 /*
317  *      report_recursive_done()
318  *
319  *      Write the .nse_depinfo file.
320  *
321  *      Parameters:
322  *
323  *      Static variables used:
324  *              recursive_list  The list of targets
325  *              changed         Written if report set changed
326  *
327  *      Global variables used:
328  *              recursive_name  The Name ".RECURSIVE", compared against
329  */
330 void
331 report_recursive_done(void)
332 {
333         char            *search_dir;
334         char            nse_depinfo[MAXPATHLEN];
335         char            tmpfile[MAXPATHLEN];
336         FILE            *ofp;
337         FILE            *ifp;
338         wchar_t         *space;
339         wchar_t         *data;
340         wchar_t         *line;
341         wchar_t         *bigger_line;
342         int             line_size, line_index;
343         int             lock_err;
344         Recursive_make  rp;

346         if (changed == false) {
347                 return;
348         }

350         search_dir = getenv(NOCATGETS("NSE_DEP"));
351         if (search_dir == NULL) {
352                 return;
353         }
354         (void) sprintf(nse_depinfo, "%s/%s", search_dir, NSE_DEPINFO);
355         (void) sprintf(tmpfile, "%s.%d", nse_depinfo, getpid());
356         ofp = fopen(tmpfile, "w");
357         if (ofp == NULL) {
358                 (void) fprintf(stderr,
359                                 catgets(catd, 1, 116, "Cannot open ‘%s’ for writi
360                                 tmpfile);
361                 return;
362         }
363         (void) sprintf(nse_depinfo_lockfile,
364                        "%s/%s", search_dir, NSE_DEPINFO_LOCK);
365         if (lock_err = file_lock(nse_depinfo,
366                                  nse_depinfo_lockfile,
367                                  (int *) &nse_depinfo_locked, 0)) {
368                 (void) fprintf(stderr,
369                                 catgets(catd, 1, 117, "writing .RECURSIVE lines t
370                                 tmpfile);
371                 (void) fprintf(stderr,
372                                 catgets(catd, 1, 118, "To recover, merge .nse_dep
373                                 getpid(),
```



```
374                                 catgets(catd, 1, 119, "with .nse_depinfo"));
375         }

377         if (nse_depinfo_locked) {
378                 ifp = fopen(nse_depinfo, "r");
379                 if (ifp != NULL) {
380                         /*
381                          * Copy all the non-.RECURSIVE lines from
382                          * the old file to the new one.
383                          */
384                         line_size = MAXPATHLEN;
385                         line_index = line_size - 1;
386                         line = ALLOC_WC(line_size);
387                         while (fgetws(line, line_size, ifp) != NULL) {
388                                 while (wslen(line) == line_index) {
389                                         if (line[wslen(line) - 1] == '\n') {
390                                                 continue;
391                                         }
392                                         bigger_line = ALLOC_WC(2 * line_size);
393                                         wscpy(bigger_line, line);
394                                         retmem(line);
395                                         line = bigger_line;
396                                         if (fgetws(&line[line_index],
397                                                    line_size, ifp) == NULL)
398                                           continue;
399                                         line_index = 2 * line_index;
400                                         line_size = 2 * line_size;
401                                 }

403                                 space = wschr(line, (int) space_char);
404                                 if (space != NULL &&
405                                     IS_WEQUALN(&space[1],
406                                                recursive_name->string,
407                                                (int) recursive_name->hash.length)
408                                         continue;
409                                 }
410                                 WCSTOMBS(mbs_buffer, line);
411                                 (void) fprintf(ofp, "%s", mbs_buffer);
412                         }
413                         (void) fclose(ifp);
414                 }
415         }

417         /*
418          * Write out the .RECURSIVE lines.
419          */
420         for (rp = recursive_list; rp != NULL; rp = rp->next) {
421                 if (rp->removed) {
422                         continue;
423                 }
424                 if (rp->newline != NULL) {
425                         data = rp->newline;
426                 } else {
427                         data = rp->oldline;
428                 }
429                 if (data != NULL) {
430                         WCSTOMBS(mbs_buffer, data);
431                         (void) fprintf(ofp, "%s\n", mbs_buffer);
432                 }
433         }
434         (void) fclose(ofp);

436         if (nse_depinfo_locked) {
437                 (void) rename(tmpfile, nse_depinfo);
438                 (void) unlink(nse_depinfo_lockfile);
439                 nse_depinfo_locked = false;
```

```
440                 nse_depinfo_lockfile[0] = '\0';
441                 (void) chmod(nse_depinfo, 0666);
442         }
443 }
444 #endif // NSE

316 /* gather_recursive_deps()
317  *
318  *      Create or update list of recursive targets.
319  */
320 void
321 gather_recursive_deps(void)
322 {
323         Name_set::iterator      np, e;
324         String_rec              rec;
325         wchar_t                 rec_buf[STRING_BUFFER_LENGTH];
326         register Property       lines;
327         Boolean                 has_recursive;
328         Dependency              dp;

330         report_recursive_init();

332         /* Go thru all targets and dump recursive dependencies */
333         for (np = hashtab.begin(), e = hashtab.end(); np != e; np++) {
334                 if (np->has_recursive_dependency){
335                         has_recursive = false;
336                         /*
337                          * start .RECURSIVE line with target:
338                          */
339                         INIT_STRING_FROM_STACK(rec, rec_buf);
340                         APPEND_NAME(np, &rec, FIND_LENGTH);
341                         append_char((int) colon_char, &rec);
342                         append_char((int) space_char, &rec);
343
344                         for (lines = get_prop(np->prop,recursive_prop);
345                             lines != NULL;
346                             lines = get_prop(lines->next, recursive_prop)) {
347                                 /*
348                                  * if entry is already in depinfo
349                                  * file or entry was not built, ignore it
350                                  */
351                                 if (lines->body.recursive.in_depinfo)
352                                         continue;
353                                 if (!lines->body.recursive.has_built)
354                                         continue;
355                                 has_recursive = true;
356                                 lines->body.recursive.in_depinfo=true;
357
358                                 /*
359                                 * Write the remainder of the
360                                 * .RECURSIVE line
361                                 */
362                                 APPEND_NAME(recursive_name, &rec,
363                                     FIND_LENGTH);
364                                 append_char((int) space_char, &rec);
365                                 APPEND_NAME(lines->body.recursive.directory,
366                                         &rec, FIND_LENGTH);
367                                 append_char((int) space_char, &rec);
368                                 APPEND_NAME(lines->body.recursive.target,
369                                         &rec, FIND_LENGTH);
370                                 append_char((int) space_char, &rec);
371
372                                 /* Complete list of makefiles used */
373                                 for (dp = lines->body.recursive.makefiles;
374                                     dp != NULL;
375                                     dp = dp->next) {
```



```
376                                         APPEND_NAME(dp->name, &rec,  FIND_LENGTH
377                                         append_char((int) space_char, &rec);
378                                 }
379                         }
380                         /*
381                          * dump list of conditional targets,
382                          * and report recursive entry, if needed
383                          */
384                         cond_macros_into_string(np, &rec);
385                         if (has_recursive){
386                                 report_recursive_dep(np, rec.buffer.start);
387                         }

389                 } else if ( np->has_built ) {
390                         remove_recursive_dep(np);
391                 }
392         }
393 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   12480 Wed May 20 11:32:43 2015
new/usr/src/cmd/make/bin/state.cc
make: unifdef for NSE (undefined)
**********************************************************
______unchanged_portion_omitted_

  96 static  void            print_auto_depes(register Dependency dependency, registe

  98 /*
  99  *      write_state_file(report_recursive, exiting)
 100  *
 101  *      Write a new version of .make.state
 102  *
 103  *      Parameters:
 104  *              report_recursive        Should only be done at end of run
 105  *              exiting                 true if called from the exit handler
 106  *
 107  *      Global variables used:
 108  *              built_last_make_run The Name ".BUILT_LAST_MAKE_RUN", written
 109  *              command_changed If no command changed we do not need to write
 110  *              current_make_version The Name "<current version>", written
 111  *              do_not_exec_rule If -n is on we do not write statefile
 112  *              hashtab         The hashtable that contains all names
 113  *              keep_state      If .KEEP_STATE is no on we do not write file
 114  *              make_state      The Name ".make.state", used for opening file
 115  *              make_version    The Name ".MAKE_VERSION", written
 116  *              recursive_name  The Name ".RECURSIVE", written
 117  *              rewrite_statefile Indicates that something changed
 118  */

 120 void
 121 #ifdef NSE
 122 write_state_file(int report_recursive, Boolean exiting)
 123 #else
 121 write_state_file(int, Boolean exiting)
 125 #endif
 122 {
 123         register FILE           *fd;
 124         int                     lock_err;
 125         char                    buffer[MAXPATHLEN];
 126         char                    make_state_tempfile[MAXPATHLEN];
 127         jmp_buf                 long_jump;
 128         register int            attempts = 0;
 129         Name_set::iterator      np, e;
 130         register Property       lines;
 131         register int            m;
 132         Dependency              dependency;
 133         register Boolean        name_printed;
 134         Boolean                 built_this_run = false;
 135         char                    *target_name;
 136         int                     line_length;
 137         register Cmd_line       cp;


 140         if (!rewrite_statefile ||
 141             !command_changed ||
 142             !keep_state ||
 143             do_not_exec_rule ||
 144             (report_dependencies_level > 0)) {
 145                 return;
 146         }
 147         /* Lock the file for writing. */
 148         make_state_lockfile = getmem(strlen(make_state->string_mb) + strlen(NOCA
 149         (void) sprintf(make_state_lockfile,
 150                        NOCATGETS("%s.lock"),
```



```
 151                        make_state->string_mb);
 152         if (lock_err = file_lock(make_state->string_mb,
 153                                  make_state_lockfile,
 154                                  (int *) &make_state_locked, 0)) {
 155                 retmem_mb(make_state_lockfile);
 156                 make_state_lockfile = NULL;
 157
 158                 /*
 159                  * We need to make sure that we are not being
 160                  * called by the exit handler so we don't call
 161                  * it again.
 162                  */
 163
 164                 if (exiting) {
 165                         (void) sprintf(buffer, NOCATGETS("%s/.make.state.%d.XXXX
 166                         report_pwd = true;
 167                         warning(catgets(catd, 1, 60, "Writing to %s"), buffer);
 168                         int fdes = mkstemp(buffer);
 169                         if ((fdes < 0) || (fd = fdopen(fdes, "w")) == NULL) {
 170                                 fprintf(stderr,
 171                                         catgets(catd, 1, 61, "Could not open sta
 172                                         buffer,
 173                                         errmsg(errno));
 174                                 return;
 175                         }
 176                 } else {
 177                         report_pwd = true;
 178                         fatal(catgets(catd, 1, 62, "Can't lock .make.state"));
 179                 }
 180         }

 182         (void) sprintf(make_state_tempfile,
 183                        NOCATGETS("%s.tmp"),
 184                        make_state->string_mb);
 185         /* Delete old temporary statefile (in case it exists) */
 186         (void) unlink(make_state_tempfile);
 187         if ((fd = fopen(make_state_tempfile, "w")) == NULL) {
 188                 lock_err = errno; /* Save it! unlink() can change errno */
 189                 (void) unlink(make_state_lockfile);
 190                 retmem_mb(make_state_lockfile);
 191                 make_state_lockfile = NULL;
 192                 make_state_locked = false;
 193                 fatal(catgets(catd, 1, 59, "Could not open temporary statefile '
 194                       make_state_tempfile,
 195                       errmsg(lock_err));
 196         }
 201 #ifdef NSE
 202         if (nse) {
 203                 (void) fchmod(fileno(fd), 0666);
 204         }
 205 #endif
 197         /*
 198          * Set a trap for failed writes. If a write fails, the routine
 199          * will try saving the .make.state file under another name in /tmp.
 200          */
 201         if (setjmp(long_jump)) {
 202                 (void) fclose(fd);
 203                 if (attempts++ > 5) {
 204                         if ((make_state_lockfile != NULL) &&
 205                             make_state_locked) {
 206                                 (void) unlink(make_state_lockfile);
 207                                 retmem_mb(make_state_lockfile);
 208                                 make_state_lockfile = NULL;
 209                                 make_state_locked = false;
 210                         }
 211                         fatal(catgets(catd, 1, 63, "Giving up on writing statefi
```

```
212                 }
213                 sleep(10);
214                 (void) sprintf(buffer, NOCATGETS("%s/.make.state.%d.XXXXXX"), tm
215                 int fdes = mkstemp(buffer);
216                 if ((fdes < 0) || (fd = fdopen(fdes, "w")) == NULL) {
217                         fatal(catgets(catd, 1, 64, "Could not open statefile `%s
218                               buffer,
219                               errmsg(errno));
220                 }
221                 warning(catgets(catd, 1, 65, "Initial write of statefile failed.
222                         buffer);
223         }

225         /* Write the version stamp. */
226         XFWRITE(make_version->string_mb,
227                 strlen(make_version->string_mb),
228                 fd);
229         XPUTC(colon_char, fd);
230         XPUTC(tab_char, fd);
231         XFWRITE(current_make_version->string_mb,
232                 strlen(current_make_version->string_mb),
233                 fd);
234         XPUTC(newline_char, fd);

236         /*
237          * Go through all the targets, dump their dependencies and
238          * command used.
239          */
240         for (np = hashtab.begin(), e = hashtab.end(); np != e; np++) {
241                 /*
242                  * If the target has no command used nor dependencies,
243                  * we can go to the next one.
244                  */
245                 if ((lines = get_prop(np->prop, line_prop)) == NULL) {
246                         continue;
247                 }
248                 /* If this target is a special target, don't print. */
249                 if (np->special_reader != no_special) {
250                         continue;
251                 }
252                 /*
253                  * Find out if any of the targets dependencies should
254                  * be written to .make.state.
255                  */
256                 for (m = 0, dependency = lines->body.line.dependencies;
257                      dependency != NULL;
258                      dependency = dependency->next) {
259                         if (m = !dependency->stale
260                             && (dependency->name != force)
261 #ifndef PRINT_EXPLICIT_DEPEN
262                             && dependency->automatic
263 #endif
264                             ) {
265                                 break;
266                         }
267                 }
268                 /* Only print if dependencies listed. */
269                 if (m || (lines->body.line.command_used != NULL)) {
270                         name_printed = false;
271                         /*
272                          * If this target was built during this make run,
273                          * we mark it.
274                          */
275                         built_this_run = false;
276                         if (np->has_built) {
277                                 built_this_run = true;
```



```
278                                 XFWRITE(built_last_make_run->string_mb,
279                                         strlen(built_last_make_run->string_mb),
280                                         fd);
281                                 XPUTC(colon_char, fd);
282                                 XPUTC(newline_char, fd);
283                         }
284                         /* If the target has dependencies, we dump them. */
285                         target_name = escape_target_name(np);
286                         if (np->has_long_member_name) {
287                                 target_name =
288                                   get_prop(np->prop, long_member_name_prop)
289                                     ->body.long_member_name.member_name->
290                                       string_mb;
291                         }
292                         if (m) {
293                                 XFPUTS(target_name, fd);
294                                 XPUTC(colon_char, fd);
295                                 XFPUTS("\t", fd);
296                                 name_printed = true;
297                                 line_length = 0;
298                                 for (dependency =
299                                      lines->body.line.dependencies;
300                                      dependency != NULL;
301                                      dependency = dependency->next) {
302                                         print_auto_depes(dependency,
303                                                          fd,
304                                                          built_this_run,
305                                                          &line_length,
306                                                          target_name,
307                                                          long_jump);
308                                 }
309                                 XFPUTS("\n", fd);
310                         }
311                         /* If there is a command used, we dump it. */
312                         if (lines->body.line.command_used != NULL) {
313                                 /*
314                                  * Only write the target name if it
315                                  * wasn't done for the dependencies.
316                                  */
317                                 if (!name_printed) {
318                                         XFPUTS(target_name, fd);
319                                         XPUTC(colon_char, fd);
320                                         XPUTC(newline_char, fd);
321                                 }
322                                 /*
323                                  * Write the command lines.
324                                  * Prefix each textual line with a tab.
325                                  */
326                                 for (cp = lines->body.line.command_used;
327                                      cp != NULL;
328                                      cp = cp->next) {
329                                         char            *csp;
330                                         int             n;

332                                         XPUTC(tab_char, fd);
333                                         if (cp->command_line != NULL) {
334                                                 for (csp = cp->
335                                                            command_line->
336                                                            string_mb,
337                                                      n = strlen(cp->
338                                                                 command_line->
339                                                                 string_mb);
340                                                      n > 0;
341                                                      n--, csp++) {
342                                                         XPUTC(*csp, fd);
343                                                         if (*csp ==
```

```
                                                        (int) newline_char)
                                                            XPUTC(tab_char,
                                                                  fd);
                                                }
                                        }
                                }
                                XPUTC(newline_char, fd);
                        }
                }
                (void)free(target_name);
        }
    }
    if (fclose(fd) == EOF) {
        longjmp(long_jump, LONGJUMP_VALUE);
    }
    if (attempts == 0) {
        if (unlink(make_state->string_mb) != 0 && errno != ENOENT) {
            lock_err = errno; /* Save it! unlink() can change errno
            /* Delete temporary statefile */
            (void) unlink(make_state_tempfile);
            (void) unlink(make_state_lockfile);
            retmem_mb(make_state_lockfile);
            make_state_lockfile = NULL;
            make_state_locked = false;
            fatal(catgets(catd, 1, 356, "Could not delete old statef
                  make_state->string_mb,
                  errmsg(lock_err));
        }
        if (rename(make_state_tempfile, make_state->string_mb) != 0) {
            lock_err = errno; /* Save it! unlink() can change errno
            /* Delete temporary statefile */
            (void) unlink(make_state_tempfile);
            (void) unlink(make_state_lockfile);
            retmem_mb(make_state_lockfile);
            make_state_lockfile = NULL;
            make_state_locked = false;
            fatal(catgets(catd, 1, 357, "Could not rename ‘%s’ to ‘%
                  make_state_tempfile,
                  make_state->string_mb,
                  errmsg(lock_err));
        }
    }
    if ((make_state_lockfile != NULL) && make_state_locked) {
        (void) unlink(make_state_lockfile);
        retmem_mb(make_state_lockfile);
        make_state_lockfile = NULL;
        make_state_locked = false;
    }
#ifdef NSE
    if (report_recursive) {
        report_recursive_done();
    }
#endif
}
```

_____unchanged_portion_omitted_

```
**********************************************************
   15477 Wed May 20 11:32:44 2015
new/usr/src/cmd/make/include/mk/defs.h
make: unifdef for NSE (undefined)
**********************************************************
______unchanged_portion_omitted_
 414 #endif
 415 extern  void            handle_interrupt(int);
 416 extern  Boolean         is_running(Name target);
 417 extern  void            load_cached_names(void);
 418 extern  Boolean         parallel_ok(Name target, Boolean line_prop_must_exists);
 419 extern  void            print_dependencies(register Name target, register Proper
 420 extern  void            send_job_start_msg(Property line);
 421 extern  void            send_rsrc_info_msg(int max_jobs, char *hostname, char *u
 422 extern  void            print_value(register Name value, Daemon daemon);
 423 extern  timestruc_t&    read_archive(register Name target);
 424 extern  int             read_dir(Name dir, wchar_t *pattern, Property line, wcha
 425 extern  void            read_directory_of_file(register Name file);
 426 extern  int             read_make_machines(Name make_machines_name);
 427 extern  Boolean         read_simple_file(register Name makefile_name, register B
 428 extern  void            remove_recursive_dep(Name target);
 429 extern  void            report_recursive_dep(Name target, char *line);
 430 extern  void            report_recursive_done(void);
 431 extern  void            report_recursive_init(void);
 432 extern  Recursive_make  find_recursive_target(Name target);
 433 extern  void            reset_locals(register Name target, register Property old
 434 extern  void            set_locals(register Name target, register Property old_l
 435 extern  void            setvar_append(register Name name, register Name value);
 436 #ifdef DISTRIBUTED
 437 extern  void            setvar_envvar(Avo_DoJobMsg *dmake_job_msg);
 438 #else
 439 extern  void            setvar_envvar(void);
 440 #endif
 441 extern  void            special_reader(Name target, register Name_vector depes,
 442 extern  void            startup_rxm();
 443 extern  Doname          target_can_be_built(register Name target);
 444 extern  char            *time_to_string(const timestruc_t &time);
 445 extern  void            update_target(Property line, Doname result);
 446 extern  void            warning(char *, ...);
 447 extern  void            write_state_file(int report_recursive, Boolean exiting);
 448 extern  Name            vpath_translation(register Name cmd);

 450 #define DEPINFO_FMT_VERSION "VERS2$"
 451 #define VER_LEN strlen(DEPINFO_FMT_VERSION)

 453 #ifdef NSE

 455 /*
 456  *  NSE version for depinfo format
 457  */
 458 extern Boolean          nse;
 459 extern Name             nse_backquote_seen;
 460 extern Boolean          nse_did_recursion;
 461 extern Name             nse_shell_var_used;
 462 extern Boolean          nse_watch_vars;
 463 extern wchar_t          current_makefile[MAXPATHLEN];
 464 extern Boolean          nse_depinfo_locked;
 465 extern char             nse_depinfo_lockfile[MAXPATHLEN];
 466 extern Name             derived_src;

 468 extern void             depvar_dep_macro_used(Name);
 469 extern void             depvar_rule_macro_used(Name);
 470 extern Boolean          nse_backquotes(wchar_t *);
 471 extern void             nse_check_cd(Property);
 472 extern void             nse_check_derived_src(Name, wchar_t *, Cmd_line);
 473 extern void             nse_check_file_backquotes(wchar_t *);
```



```
 474 extern void             nse_check_no_deps_no_rule(Name, Property, Property);
 475 extern void             nse_check_sccs(wchar_t *, wchar_t *);
 476 extern void             nse_dep_cmdmacro(wchar_t *);
 477 extern int              nse_exit_status(void);
 478 extern void             nse_init_source_suffixes(void);
 479 extern void             nse_no_makefile(Name);
 480 extern void             nse_rule_cmdmacro(wchar_t *);
 481 extern void             nse_wildcard(wchar_t *, wchar_t *);
 482 #endif

 454 #endif
```

```
**********************************************************
   23141 Wed May 20 11:32:44 2015
new/usr/src/cmd/make/include/mksh/defs.h
make: unifdef for NSE (undefined)
**********************************************************
______unchanged_portion_omitted_

 330 /*
 331  * The specials are markers for targets that the reader should special case
 332  */
 333 typedef enum {
 334         no_special,
 335         built_last_make_run_special,
 336         default_special,
 337 #ifdef NSE
 338         derived_src_special,
 339 #endif
 337         get_posix_special,
 338         get_special,
 339         ignore_special,
 340         keep_state_file_special,
 341         keep_state_special,
 342         make_version_special,
 343         no_parallel_special,
 344         parallel_special,
 345         posix_special,
 346         precious_special,
 347         sccs_get_posix_special,
 348         sccs_get_special,
 349         silent_special,
 350         suffixes_special,
 351         svr4_special,
 352         localhost_special
 353 } Special;
______unchanged_portion_omitted_

 412 struct _Macro {
 413         /*
 414         * For "ABC = xyz" constructs
 415         * Name "ABC" get one macro prop
 416         */
 417         struct _Name            *value;
 421 #ifdef NSE
 422         Boolean                 imported:1;
 423 #endif
 418         Boolean                 exported:1;
 419         Boolean                 read_only:1;
 420         /*
 421         * This macro is defined conditionally
 422         */
 423         Boolean                 is_conditional:1;
 424         /*
 425         * The list for $? is stored as a structured list that
 426         * is translated into a string iff it is referenced.
 427         * This is why  some macro values need a daemon.
 428         */
 429         Daemon                  daemon:2;
 430 };
______unchanged_portion_omitted_

 444 struct _Name {
 445         struct _Property        *prop;          /* List of properties */
 446         char                    *string_mb;     /* Multi-byte name string */
 447         struct {
 448                 unsigned int            length;
 449         }                       hash;
```



```
 450         struct {
 451                 timestruc_t             time;           /* Modification */
 452                 int                     stat_errno;     /* error from "stat" */
 453                 off_t                   size;           /* Of file */
 454                 mode_t                  mode;           /* Of file */
 455                 Boolean                 is_file:1;
 456                 Boolean                 is_dir:1;
 457                 Boolean                 is_sym_link:1;
 458                 Boolean                 is_precious:1;
 465 #ifdef NSE
 466                 Boolean                 is_derived_src:1;
 467 #endif
 459                 enum sccs_stat          has_sccs:2;
 460         }                       stat;
 461         /*
 462          * Count instances of :: definitions for this target
 463          */
 464         short                   colon_splits;
 465         /*
 466          * We only clear the automatic depes once per target per report
 467          */
 468         short                   temp_file_number;
 469         /*
 470          * Count how many conditional macros this target has defined
 471          */
 472         short                   conditional_cnt;
 473         /*
 474          * A conditional macro was used when building this target
 475          */
 476         Boolean                 depends_on_conditional:1;
 477         /*
 478          * Pointer to list of conditional macros which were used to build
 479          * this target
 480          */
 481         struct _Macro_list      *conditional_macro_list;
 482         Boolean                 has_member_depe:1;
 483         Boolean                 is_member:1;
 484         /*
 485          * This target is a directory that has been read
 486          */
 487         Boolean                 has_read_dir:1;
 488         /*
 489          * This name is a macro that is now being expanded
 490          */
 491         Boolean                 being_expanded:1;
 492         /*
 493          * This name is a magic name that the reader must know about
 494          */
 495         Special                 special_reader:5;
 496         Doname                  state:3;
 497         Separator               colons:3;
 498         Boolean                 has_depe_list_expanded:1;
 499         Boolean                 suffix_scan_done:1;
 500         Boolean                 has_complained:1;       /* For sccs */
 501         /*
 502          * This target has been built during this make run
 503          */
 504         Boolean                 ran_command:1;
 505         Boolean                 with_squiggle:1;        /* for .SUFFIXES */
 506         Boolean                 without_squiggle:1;     /* for .SUFFIXES */
 507         Boolean                 has_read_suffixes:1;    /* Suffix list cached*/
 508         Boolean                 has_suffixes:1;
 509         Boolean                 has_target_prop:1;
 510         Boolean                 has_vpath_alias_prop:1;
 511         Boolean                 dependency_printed:1;   /* For dump_make_state()
 512         Boolean                 dollar:1;               /* In namestring */
```

```
513         Boolean                 meta:1;                 /* In namestring */
514         Boolean                 percent:1;              /* In namestring */
515         Boolean                 wildcard:1;             /* In namestring */
516         Boolean                 has_parent:1;
517         Boolean                 is_target:1;
518         Boolean                 has_built:1;
519         Boolean                 colon:1;                /* In namestring */
520         Boolean                 parenleft:1;            /* In namestring */
521         Boolean                 has_recursive_dependency:1;
522         Boolean                 has_regular_dependency:1;
523         Boolean                 is_double_colon:1;
524         Boolean                 is_double_colon_parent:1;
525         Boolean                 has_long_member_name:1;
526         /*
527          * allowed to run in parallel
528          */
529         Boolean                 parallel:1;
530         /*
531          * not allowed to run in parallel
532          */
533         Boolean                 no_parallel:1;
534         /*
535          * used in dependency_conflict
536          */
537         Boolean                 checking_subtree:1;
538         Boolean                 added_pattern_conditionals:1;
539         /*
540          * rechecking target for possible rebuild
541          */
542         Boolean                 rechecking_target:1;
543         /*
544          * build this target in silent mode
545          */
546         Boolean                 silent_mode:1;
547         /*
548          * build this target in ignore error mode
549          */
550         Boolean                 ignore_error_mode:1;
551         Boolean                 dont_activate_cond_values:1;
552         /*
553          * allowed to run serially on local host
554          */
555         Boolean                 localhost:1;
556 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1561 Wed May 20 11:32:45 2015
new/usr/src/cmd/make/include/vroot/report.h
make: unifdef for NSE (undefined)
**********************************************************
/*
 * 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 1994 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _REPORT_H_
#define _REPORT_H_

#include <stdio.h>

extern FILE     *get_report_file(void);
extern char     *get_target_being_reported_for(void);
extern void     report_dependency(const char *name);
extern int      file_lock(char *name, char *lockname, int *file_locked, int time
#ifdef NSE
extern char     *setenv(char *name, char *value);
#endif

#define SUNPRO_DEPENDENCIES "SUNPRO_DEPENDENCIES"
#define LD      "LD"
#define COMP    "COMP"

/*
 * These relate to Sun's ancient source control system that predated TeamWare,
 * named NSE.  They appear to be used regardless of its presence, however, and
 * so linger.
/* the following definitions define the interface between make and
 * NSE - the two systems must track each other.
 */
#define NSE_DEPINFO             ".nse_depinfo"
#define NSE_DEPINFO_LOCK        ".nse_depinfo.lock"
#define NSE_DEP_ENV             "NSE_DEP"
#define NSE_TFS_PUSH            "/usr/nse/bin/tfs_push"
#define NSE_TFS_PUSH_LEN        8
#define NSE_VARIANT_ENV         "NSE_VARIANT"
#define NSE_RT_SOURCE_NAME      "Shared_Source"

#endif
```

```
**********************************************************
   37303 Wed May 20 11:32:45 2015
new/usr/src/cmd/make/lib/mksh/macro.cc
make: unifdef for NSE (undefined)
**********************************************************
```

```
/*
 * 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 <mksdmsi18n/mksdmsi18n.h>      /* libmksdmsi18n_init() */

#include <widec.h>

/*
 * File table of contents
 */
static void     add_macro_to_global_list(Name macro_to_add);
#ifdef NSE
static void     expand_value_with_daemon(Name name, register Property macro, reg
#else
static void     expand_value_with_daemon(Name, register Property macro, register
#endif

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;
}
```

unchanged_portion_omitted_

```
/*
 *      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
```

```
 247  *              current_string  The string we are expanding, for error msg
 248  *              cmd             If we are evaluating a command line we
 249  *                              turn \ quoting off
 250  *
 251  *      Global variables used:
 252  *              funny           Vector of semantic tags for characters
 253  *              is_conditional  Set if a conditional macro is refd
 254  *              make_word_mentioned Set if the word "MAKE" is mentioned
 255  *              makefile_type   We deliver extra msg when reading makefiles
 256  *              query           The Name "?", compared against
 257  *              query_mentioned Set if the word "?" is mentioned
 258  */
 259 void
 260 expand_macro(register Source source, register String destination, wchar_t *curre
 261 {
 262         static Name             make = (Name)NULL;
 263         static wchar_t          colon_sh[4];
 264         static wchar_t          colon_shell[7];
 265         String_rec              string;
 266         wchar_t                 buffer[STRING_BUFFER_LENGTH];
 267         register wchar_t        *source_p = source->string.text.p;
 268         register wchar_t        *source_end = source->string.text.end;
 269         register int            closer = 0;
 270         wchar_t                 *block_start = (wchar_t *)NULL;
 271         int                     quote_seen = 0;
 272         register int            closer_level = 1;
 273         Name                    name = (Name)NULL;
 274         wchar_t                 *colon = (wchar_t *)NULL;
 275         wchar_t                 *percent = (wchar_t *)NULL;
 276         wchar_t                 *eq = (wchar_t *) NULL;
 277         Property                macro = NULL;
 278         wchar_t                 *p = (wchar_t*)NULL;
 279         String_rec              extracted;
 280         wchar_t                 extracted_string[MAXPATHLEN];
 281         wchar_t                 *left_head = NULL;
 282         wchar_t                 *left_tail = NULL;
 283         wchar_t                 *right_tail = NULL;
 284         int                     left_head_len = 0;
 285         int                     left_tail_len = 0;
 286         int                     tmp_len = 0;
 287         wchar_t                 *right_hand[128];
 288         int                     i = 0;
 289         enum {
 290                 no_extract,
 291                 dir_extract,
 292                 file_extract
 293         }                       extraction = no_extract;
 294         enum {
 295                 no_replace,
 296                 suffix_replace,
 297                 pattern_replace,
 298                 sh_replace
 299         }                       replacement = no_replace;

 301         if (make == NULL) {
 302                 MBSTOWCS(wcs_buffer, NOCATGETS("MAKE"));
 303                 make = GETNAME(wcs_buffer, FIND_LENGTH);

 305                 MBSTOWCS(colon_sh, NOCATGETS(":sh"));
 306                 MBSTOWCS(colon_shell, NOCATGETS(":shell"));
 307         }

 309         right_hand[0] = NULL;

 311         /* First copy the (macro-expanded) macro name into string. */
 312         INIT_STRING_FROM_STACK(string, buffer);
```



```
 313 recheck_first_char:
 314         /* Check the first char of the macro name to figure out what to do. */
 315         switch (GET_CHAR()) {
 316         case nul_char:
 317                 GET_NEXT_BLOCK_NOCHK(source);
 318                 if (source == NULL) {
 319                         WCSTOMBS(mbs_buffer, current_string);
 320                         fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 114, "'
 321                                           mbs_buffer);
 322                 }
 323                 if (source->error_converting) {
 324                         fatal_reader_mksh(NOCATGETS("Internal error: Invalid byt
 325                 }
 326                 goto recheck_first_char;
 327         case parenleft_char:
 328                 /* Multi char name. */
 329                 closer = (int) parenright_char;
 330                 break;
 331         case braceleft_char:
 332                 /* Multi char name. */
 333                 closer = (int) braceright_char;
 334                 break;
 335         case newline_char:
 336                 fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 115, "'$' at en
 337         default:
 338                 /* Single char macro name. Just suck it up */
 339                 append_char(*source_p, &string);
 340                 source->string.text.p = source_p + 1;
 341                 goto get_macro_value;
 342         }

 344         /* Handle multi-char macro names */
 345         block_start = ++source_p;
 346         quote_seen = 0;
 347         for (; 1; source_p++) {
 348                 switch (GET_CHAR()) {
 349                 case nul_char:
 350                         append_string(block_start,
 351                                       &string,
 352                                       source_p - block_start);
 353                         GET_NEXT_BLOCK_NOCHK(source);
 354                         if (source == NULL) {
 355                                 if (current_string != NULL) {
 356                                         WCSTOMBS(mbs_buffer, current_string);
 357                                         fatal_reader_mksh(catgets(libmksdmsi18n_
 358                                                           closer ==
 359                                                           (int) braceright_char ?
 360                                                           (int) braceleft_char :
 361                                                           (int) parenleft_char,
 362                                                           mbs_buffer);
 363                                 } else {
 364                                         fatal_reader_mksh(catgets(libmksdmsi18n_
 365                                 }
 366                         }
 367                         if (source->error_converting) {
 368                                 fatal_reader_mksh(NOCATGETS("Internal error: Inv
 369                         }
 370                         block_start = source_p;
 371                         source_p--;
 372                         continue;
 373                 case newline_char:
 374                         fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 118, "U
 375                                           closer == (int) braceright_char ?
 376                                           (int) braceleft_char :
 377                                           (int) parenleft_char);
 378                 case backslash_char:
```

```
379                         /* Quote dollar in macro value. */
380                         if (!cmd) {
381                                 quote_seen = ~quote_seen;
382                         }
383                         continue;
384                 case dollar_char:
385                         /*
386                          * Macro names may reference macros.
387                          * This expands the value of such macros into the
388                          * macro name string.
389                          */
390                         if (quote_seen) {
391                                 append_string(block_start,
392                                               &string,
393                                               source_p - block_start - 1);
394                                 block_start = source_p;
395                                 break;
396                         }
397                         append_string(block_start,
398                                       &string,
399                                       source_p - block_start);
400                         source->string.text.p = ++source_p;
401                         UNCACHE_SOURCE();
402                         expand_macro(source, &string, current_string, cmd);
403                         CACHE_SOURCE(0);
404                         block_start = source_p;
405                         source_p--;
406                         break;
407                 case parenleft_char:
408                         /* Allow nested pairs of () in the macro name. */
409                         if (closer == (int) parenright_char) {
410                                 closer_level++;
411                         }
412                         break;
413                 case braceleft_char:
414                         /* Allow nested pairs of {} in the macro name. */
415                         if (closer == (int) braceright_char) {
416                                 closer_level++;
417                         }
418                         break;
419                 case parenright_char:
420                 case braceright_char:
421                         /*
422                          * End of the name. Save the string in the macro
423                          * name string.
424                          */
425                         if ((*source_p == closer) && (--closer_level <= 0)) {
426                                 source->string.text.p = source_p + 1;
427                                 append_string(block_start,
428                                               &string,
429                                               source_p - block_start);
430                                 goto get_macro_value;
431                         }
432                         break;
433                 }
434                 quote_seen = 0;
435         }
436         /*
437          * We got the macro name. We now inspect it to see if it
438          * specifies any translations of the value.
439          */
440 get_macro_value:
441         name = NULL;
442         /* First check if we have a $(@D) type translation. */
443         if ((get_char_semantics_value(string.buffer.start[0]) &
444              (int) special_macro_sem) &&
```



```
445             (string.text.p - string.buffer.start >= 2) &&
446             ((string.buffer.start[1] == 'D') ||
447              (string.buffer.start[1] == 'F'))) {
448                 switch (string.buffer.start[1]) {
449                 case 'D':
450                         extraction = dir_extract;
451                         break;
452                 case 'F':
453                         extraction = file_extract;
454                         break;
455                 default:
456                         WCSTOMBS(mbs_buffer, string.buffer.start);
457                         fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 119, "I
458                                 mbs_buffer);
459                 }
460                 /* Internalize the macro name using the first char only. */
461                 name = GETNAME(string.buffer.start, 1);
462                 (void) wscpy(string.buffer.start, string.buffer.start + 2);
463         }
464         /* Check for other kinds of translations. */
465         if ((colon = (wchar_t *) wschr(string.buffer.start,
466                                        (int) colon_char)) != NULL) {
467                 /*
468                  * We have a $(FOO:.c=.o) type translation.
469                  * Get the name of the macro proper.
470                  */
471                 if (name == NULL) {
472                         name = GETNAME(string.buffer.start,
473                                        colon - string.buffer.start);
474                 }
475                 /* Pickup all the translations. */
476                 if (IS_WEQUAL(colon, colon_sh) || IS_WEQUAL(colon, colon_shell))
477                         replacement = sh_replace;
478                 } else if ((svr4) ||
479                            ((percent = (wchar_t *) wschr(colon + 1,
480                                                          (int) percent_char)) ==
481                         while (colon != NULL) {
482                                 if ((eq = (wchar_t *) wschr(colon + 1,
483                                                             (int) equal_char)) =
484                                         fatal_reader_mksh(catgets(libmksdmsi18n_
485                                 }
486                                 left_tail_len = eq - colon - 1;
487                                 if(left_tail) {
488                                         retmem(left_tail);
489                                 }
490                                 left_tail = ALLOC_WC(left_tail_len + 1);
491                                 (void) wsncpy(left_tail,
492                                               colon + 1,
493                                               eq - colon - 1);
494                                 left_tail[eq - colon - 1] = (int) nul_char;
495                                 replacement = suffix_replace;
496                                 if ((colon = (wchar_t *) wschr(eq + 1,
497                                                                (int) colon_char)
498                                         tmp_len = colon - eq;
499                                         if(right_tail) {
500                                                 retmem(right_tail);
501                                         }
502                                         right_tail = ALLOC_WC(tmp_len);
503                                         (void) wsncpy(right_tail,
504                                                       eq + 1,
505                                                       colon - eq - 1);
506                                         right_tail[colon - eq - 1] =
507                                           (int) nul_char;
508                                 } else {
509                                         if(right_tail) {
510                                                 retmem(right_tail);
```

```
511                                     }
512                                     right_tail = ALLOC_WC(wslen(eq) + 1);
513                                     (void) wscpy(right_tail, eq + 1);
514                             }
515                     }
516             } else {
517                     if ((eq = (wchar_t *) wschr(colon + 1,
518                                             (int) equal_char)) == NULL)
519                             fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1,
520                     }
521                     if ((percent = (wchar_t *) wschr(colon + 1,
522                                                     (int) percent_char)) ==
523                             fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1,
524                     }
525                     if (eq < percent) {
526                             fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1,
527                     }

529                     if (percent > (colon + 1)) {
530                             tmp_len = percent - colon;
531                             if(left_head) {
532                                     retmem(left_head);
533                             }
534                             left_head = ALLOC_WC(tmp_len);
535                             (void) wsncpy(left_head,
536                                             colon + 1,
537                                             percent - colon - 1);
538                             left_head[percent-colon-1] = (int) nul_char;
539                             left_head_len = percent-colon-1;
540                     } else {
541                             left_head = NULL;
542                             left_head_len = 0;
543                     }

545                     if (eq > percent+1) {
546                             tmp_len = eq - percent;
547                             if(left_tail) {
548                                     retmem(left_tail);
549                             }
550                             left_tail = ALLOC_WC(tmp_len);
551                             (void) wsncpy(left_tail,
552                                             percent + 1,
553                                             eq - percent - 1);
554                             left_tail[eq-percent-1] = (int) nul_char;
555                             left_tail_len = eq-percent-1;
556                     } else {
557                             left_tail = NULL;
558                             left_tail_len = 0;
559                     }

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

564                             right_hand[0] = ALLOC_WC(wslen(eq) + 1);
565                             right_hand[1] = NULL;
566                             (void) wscpy(right_hand[0], eq);
567                     } else {
568                             i = 0;
569                             do {
570                                     right_hand[i] = ALLOC_WC(percent-eq+1);
571                                     (void) wsncpy(right_hand[i],
572                                                     eq,
573                                                     percent - eq);
574                                     right_hand[i][percent-eq] =
575                                       (int) nul_char;
576                                     if (i++ >= VSIZEOF(right_hand)) {
```



```
577                                             fatal_mksh(catgets(libmksdmsi18n
578                                     }
579                                     eq = percent + 1;
580                                     if (eq[0] == (int) nul_char) {
581                                             MBSTOWCS(wcs_buffer, "");
582                                             right_hand[i] = (wchar_t *) wsdu
583                                             i++;
584                                             break;
585                                     }
586                                 } while ((percent = (wchar_t *) wschr(eq, (int)
587                                 if (eq[0] != (int) nul_char) {
588                                     right_hand[i] = ALLOC_WC(wslen(eq) + 1);
589                                     (void) wscpy(right_hand[i], eq);
590                                     i++;
591                                 }
592                                 right_hand[i] = NULL;
593                             }
594                             replacement = pattern_replace;
595                     }
596             }
597             if (name == NULL) {
598                     /*
599                      * No translations found.
600                      * Use the whole string as the macro name.
601                      */
602                     name = GETNAME(string.buffer.start,
603                                    string.text.p - string.buffer.start);
604             }
605             if (string.free_after_use) {
606                     retmem(string.buffer.start);
607             }
608             if (name == make) {
609                     make_word_mentioned = true;
610             }
611             if (name == query) {
612                     query_mentioned = true;
613             }
614             if ((name == host_arch) || (name == target_arch)) {
615                     if (!init_arch_done) {
616                             init_arch_done = true;
617                             init_arch_macros();
618                     }
619             }
620             if ((name == host_mach) || (name == target_mach)) {
621                     if (!init_mach_done) {
622                             init_mach_done = true;
623                             init_mach_macros();
624                     }
625             }
626             /* Get the macro value. */
627             macro = get_prop(name->prop, macro_prop);
632 #ifdef NSE
633             if (nse_watch_vars && nse && macro != NULL) {
634                     if (macro->body.macro.imported) {
635                             nse_shell_var_used= name;
636                     }
637                     if (macro->body.macro.value != NULL){
638                           if (nse_backquotes(macro->body.macro.value->string)) {
639                                   nse_backquote_seen= name;
640                           }
641                        }
642             }
643 #endif
628             if ((macro != NULL) && macro->body.macro.is_conditional) {
629                     conditional_macro_used = true;
630                     /*
```

```
631                  * Add this conditional macro to the beginning of the
632                  * global list.
633                  */
634                 add_macro_to_global_list(name);
635                 if (makefile_type == reading_makefile) {
636                         warning_mksh(catgets(libmksdmsi18n_catd, 1, 164, "Condit
637                                         name->string_mb, file_being_read, line_n
638                 }
639         }
640         /* Macro name read and parsed. Expand the value. */
641         if ((macro == NULL) || (macro->body.macro.value == NULL)) {
642                 /* If the value is empty, we just get out of here. */
643                 goto exit;
644         }
645         if (replacement == sh_replace) {
646                 /* If we should do a :sh transform, we expand the command
647                  * and process it.
648                  */
649                 INIT_STRING_FROM_STACK(string, buffer);
650                 /* Expand the value into a local string buffer and run cmd. */
651                 expand_value_with_daemon(name, macro, &string, cmd);
652                 sh_command2string(&string, destination);
653         } else if ((replacement != no_replace) || (extraction != no_extract)) {
654                 /*
655                  * If there were any transforms specified in the macro
656                  * name, we deal with them here.
657                  */
658                 INIT_STRING_FROM_STACK(string, buffer);
659                 /* Expand the value into a local string buffer. */
660                 expand_value_with_daemon(name, macro, &string, cmd);
661                 /* Scan the expanded string. */
662                 p = string.buffer.start;
663                 while (*p != (int) nul_char) {
664                         wchar_t         chr;

666                         /*
667                          * First skip over any white space and append
668                          * that to the destination string.
669                          */
670                         block_start = p;
671                         while ((*p != (int) nul_char) && iswspace(*p)) {
672                                 p++;
673                         }
674                         append_string(block_start,
675                                       destination,
676                                       p - block_start);
677                         /* Then find the end of the next word. */
678                         block_start = p;
679                         while ((*p != (int) nul_char) && !iswspace(*p)) {
680                                 p++;
681                         }
682                         /* If we cant find another word we are done */
683                         if (block_start == p) {
684                                 break;
685                         }
686                         /* Then apply the transforms to the word */
687                         INIT_STRING_FROM_STACK(extracted, extracted_string);
688                         switch (extraction) {
689                         case dir_extract:
690                                 /*
691                                  * $(@D) type transform. Extract the
692                                  * path from the word. Deliver "." if
693                                  * none is found.
694                                  */
695                                 if (p != NULL) {
696                                         chr = *p;
```



```
697                                         *p = (int) nul_char;
698                                 }
699                                 eq = (wchar_t *) wsrchr(block_start, (int) slash
700                                 if (p != NULL) {
701                                         *p = chr;
702                                 }
703                                 if ((eq == NULL) || (eq > p)) {
704                                         MBSTOWCS(wcs_buffer, ".");
705                                         append_string(wcs_buffer, &extracted, 1)
706                                 } else {
707                                         append_string(block_start,
708                                                       &extracted,
709                                                       eq - block_start);
710                                 }
711                                 break;
712                         case file_extract:
713                                 /*
714                                  * $(@F) type transform. Remove the path
715                                  * from the word if any.
716                                  */
717                                 if (p != NULL) {
718                                         chr = *p;
719                                         *p = (int) nul_char;
720                                 }
721                                 eq = (wchar_t *) wsrchr(block_start, (int) slash
722                                 if (p != NULL) {
723                                         *p = chr;
724                                 }
725                                 if ((eq == NULL) || (eq > p)) {
726                                         append_string(block_start,
727                                                       &extracted,
728                                                       p - block_start);
729                                 } else {
730                                         append_string(eq + 1,
731                                                       &extracted,
732                                                       p - eq - 1);
733                                 }
734                                 break;
735                         case no_extract:
736                                 append_string(block_start,
737                                               &extracted,
738                                               p - block_start);
739                                 break;
740                         }
741                         switch (replacement) {
742                         case suffix_replace:
743                                 /*
744                                  * $(FOO:.o=.c) type transform.
745                                  * Maybe replace the tail of the word.
746                                  */
747                                 if (((extracted.text.p -
748                                       extracted.buffer.start) >=
749                                      left_tail_len) &&
750                                     IS_WEQUALN(extracted.text.p - left_tail_len,
751                                                left_tail,
752                                                left_tail_len)) {
753                                         append_string(extracted.buffer.start,
754                                                       destination,
755                                                       (extracted.text.p -
756                                                        extracted.buffer.start)
757                                                       - left_tail_len);
758                                         append_string(right_tail,
759                                                       destination,
760                                                       FIND_LENGTH);
761                                 } else {
762                                         append_string(extracted.buffer.start,
```

```
763                                                 destination,
764                                                 FIND_LENGTH);
765                             }
766                             break;
767                     case pattern_replace:
768                             /* $(X:a%b=c%d) type transform. */
769                             if (((extracted.text.p -
770                                   extracted.buffer.start) >=
771                                  left_head_len+left_tail_len) &&
772                                 IS_WEQUALN(left_head,
773                                         extracted.buffer.start,
774                                         left_head_len) &&
775                                 IS_WEQUALN(left_tail,
776                                         extracted.text.p - left_tail_len,
777                                         left_tail_len)) {
778                                     i = 0;
779                                     while (right_hand[i] != NULL) {
780                                       append_string(right_hand[i],
781                                                     destination,
782                                                     FIND_LENGTH);
783                                       i++;
784                                       if (right_hand[i] != NULL) {
785                                               append_string(extracted.
786                                                             start +
787                                                             left_head_
788                                                             destinatio
789                                                             (extracted
790                                       }
791                                     }
792                             } else {
793                                     append_string(extracted.buffer.start,
794                                                   destination,
795                                                   FIND_LENGTH);
796                             }
797                             break;
798                     case no_replace:
799                             append_string(extracted.buffer.start,
800                                           destination,
801                                           FIND_LENGTH);
802                             break;
803                     case sh_replace:
804                             break;
805                         }
806             }
807             if (string.free_after_use) {
808                     retmem(string.buffer.start);
809             }
810     } else {
811             /*
812              * This is for the case when the macro name did not
813              * specify transforms.
814              */
815             if (!strncmp(name->string_mb, NOCATGETS("GET"), 3)) {
816                     dollarget_seen = true;
817             }
818             dollarless_flag = false;
819             if (!strncmp(name->string_mb, "<", 1) &&
820                 dollarget_seen) {
821                     dollarless_flag = true;
822                     dollarget_seen = false;
823             }
824             expand_value_with_daemon(name, macro, destination, cmd);
825     }
826 exit:
827     if(left_tail) {
828             retmem(left_tail);
```



```
829     }
830     if(right_tail) {
831             retmem(right_tail);
832     }
833     if(left_head) {
834             retmem(left_head);
835     }
836     i = 0;
837     while (right_hand[i] != NULL) {
838             retmem(right_hand[i]);
839             i++;
840     }
841     *destination->text.p = (int) nul_char;
842     destination->text.end = destination->text.p;
843 }
```

*_____unchanged_portion_omitted_*

```
884 /*
885  *      init_arch_macros(void)
886  *
887  *      Set the magic macros TARGET_ARCH, HOST_ARCH,
888  *
889  *      Parameters:
890  *
891  *      Global variables used:
892  *                              host_arch   Property for magic macro HOST_ARCH
893  *                              target_arch Property for magic macro TARGET_ARCH
894  *
895  *      Return value:
896  *                              The function does not return a value, but can
897  *                              call fatal() in case of error.
898  */
899 static void
900 init_arch_macros(void)
901 {
902     String_rec      result_string;
903     wchar_t         wc_buf[STRING_BUFFER_LENGTH];
904     char            mb_buf[STRING_BUFFER_LENGTH];
905     FILE            *pipe;
906     Name            value;
907     int             set_host, set_target;
924 #ifdef NSE
925     Property        macro;
926 #endif
908     const char      *mach_command = NOCATGETS("/bin/mach");

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

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

917             if ((pipe = popen(mach_command, "r")) == NULL) {
918                     fatal_mksh(catgets(libmksdmsi18n_catd, 1, 185, "Execute
919             }
920             while (fgets(mb_buf, sizeof(mb_buf), pipe) != NULL) {
921                     MBSTOWCS(wcs_buffer, mb_buf);
922                     append_string(wcs_buffer, &result_string, wslen(wcs_buff
923             }
924             if (pclose(pipe) != 0) {
925                     fatal_mksh(catgets(libmksdmsi18n_catd, 1, 186, "Execute
926             }

928             value = GETNAME(result_string.buffer.start, wslen(result_string.
```

```
 949 #ifdef NSE
 950                 macro = setvar_daemon(host_arch, value, false, no_daemon, true,
 951                 macro->body.macro.imported= true;
 952                 macro = setvar_daemon(target_arch, value, false, no_daemon, true
 953                 macro->body.macro.imported= true;
 954 #else
 930                 if (set_host) {
 931                         (void) setvar_daemon(host_arch, value, false, no_daemon,
 932                 }
 933                 if (set_target) {
 934                         (void) setvar_daemon(target_arch, value, false, no_daemo
 935                 }
 961 #endif
 936         }
 937 }
_____unchanged_portion_omitted_

 994 /*
 995  *      expand_value_with_daemon(name, macro, destination, cmd)
 996  *
 997  *      Checks for daemons and then maybe calls expand_value().
 998  *
 999  *      Parameters:
1000  *              name            Name of the macro  (Added by the NSE)
1001  *              macro           The property block with the value to expand
1002  *              destination     Where the result should be deposited
1003  *              cmd             If we are evaluating a command line we
1004  *                              turn \ quoting off
1005  *
1006  *      Global variables used:
1007  */
1008 static void
1035 #ifdef NSE
1036 expand_value_with_daemon(Name name, register Property macro, register String des
1037 #else
1009 expand_value_with_daemon(Name, register Property macro, register String destinat
1039 #endif
1010 {
1011         register Chain          chain;

1043 #ifdef NSE
1044         if (reading_dependencies) {
1045                 /*
1046                  * Processing the dependencies themselves
1047                  */
1048                 depvar_dep_macro_used(name);
1049         } else {
1050                 /*
1051                  * Processing the rules for the targets
1052                  * the nse_watch_vars flags chokes off most
1053                  * checks.  it is true only when processing
1054                  * the output from a recursive make run
1055                  * which is all we are interested in here.
1056                  */
1057                 if (nse_watch_vars) {
1058                         depvar_rule_macro_used(name);
1059                 }
1060          }
1061 #endif

1014         switch (macro->body.macro.daemon) {
1015         case no_daemon:
1016                 if (!svr4 && !posix) {
1017                         expand_value(macro->body.macro.value, destination, cmd);
1018                 } else {
1019                         if (dollarless_flag && tilde_rule) {
```



```
1020                                 expand_value(dollarless_value, destination, cmd)
1021                                 dollarless_flag = false;
1022                                 tilde_rule = false;
1023                         } else {
1024                                 expand_value(macro->body.macro.value, destinatio
1025                         }
1026                 }
1027                 return;
1028         case chain_daemon:
1029                 /* If this is a $? value we call the daemon to translate the */
1030                 /* list of names to a string */
1031                 for (chain = (Chain) macro->body.macro.value;
1032                      chain != NULL;
1033                      chain = chain->next) {
1034                         APPEND_NAME(chain->name,
1035                                     destination,
1036                                     (int) chain->name->hash.length);
1037                         if (chain->next != NULL) {
1038                                 append_char((int) space_char, destination);
1039                         }
1040                 }
1041                 return;
1042         }
1043 }

1045 /*
1046  * We use a permanent buffer to reset SUNPRO_DEPENDENCIES value.
1047  */
1048 char    *sunpro_dependencies_buf = NULL;
1049 char    *sunpro_dependencies_oldbuf = NULL;
1050 int     sunpro_dependencies_buf_size = 0;

1052 /*
1053  *      setvar_daemon(name, value, append, daemon, strip_trailing_spaces)
1054  *
1055  *      Set a macro value, possibly supplying a daemon to be used
1056  *      when referencing the value.
1057  *
1058  *      Return value:
1059  *                              The property block with the new value
1060  *
1061  *      Parameters:
1062  *              name            Name of the macro to set
1063  *              value           The value to set
1064  *              append          Should we reset or append to the current value?
1065  *              daemon          Special treatment when reading the value
1066  *              strip_trailing_spaces from the end of value->string
1067  *              debug_level     Indicates how much tracing we should do
1068  *
1069  *      Global variables used:
1070  *              makefile_type   Used to check if we should enforce read only
1071  *              path_name       The Name "PATH", compared against
1072  *              virtual_root    The Name "VIRTUAL_ROOT", compared against
1073  *              vpath_defined   Set if the macro VPATH is set
1074  *              vpath_name      The Name "VPATH", compared against
1075  *              envvar          A list of environment vars with $ in value
1076  */
1077 Property
1078 setvar_daemon(register Name name, register Name value, Boolean append, Daemon da
1079 {
1080         register Property       macro = maybe_append_prop(name, macro_prop);
1081         register Property       macro_apx = get_prop(name->prop, macro_append_pr
1082         int                     length = 0;
1083         String_rec              destination;
1084         wchar_t                 buffer[STRING_BUFFER_LENGTH];
1085         register Chain          chain;
```

```
1086         Name                    val;
1087         wchar_t                 *val_string = (wchar_t*)NULL;
1088         Wstring                 wcb;

1139 #ifdef NSE
1140         macro->body.macro.imported = false;
1141 #endif

1091         if ((makefile_type != reading_nothing) &&
1092             macro->body.macro.read_only) {
1093                 return macro;
1094         }
1095         /* Strip spaces from the end of the value */
1096         if (daemon == no_daemon) {
1097                 if(value != NULL) {
1098                         wcb.init(value);
1099                         length = wcb.length();
1100                         val_string = wcb.get_string();
1101                 }
1102                 if ((length > 0) && iswspace(val_string[length-1])) {
1103                         INIT_STRING_FROM_STACK(destination, buffer);
1104                         buffer[0] = 0;
1105                         append_string(val_string, &destination, length);
1106                         if (strip_trailing_spaces) {
1107                                 while ((length > 0) &&
1108                                        iswspace(destination.buffer.start[length-
1109                                         destination.buffer.start[--length] = 0;
1110                                 }
1111                         }
1112                         value = GETNAME(destination.buffer.start, FIND_LENGTH);
1113                 }
1114         }
1115
1116         if(macro_apx != NULL) {
1117                 val = macro_apx->body.macro_appendix.value;
1118         } else {
1119                 val = macro->body.macro.value;
1120         }

1122         if (append) {
1123                 /*
1124                  * If we are appending, we just tack the new value after
1125                  * the old one with a space in between.
1126                  */
1127                 INIT_STRING_FROM_STACK(destination, buffer);
1128                 buffer[0] = 0;
1129                 if ((macro != NULL) && (val != NULL)) {
1130                         APPEND_NAME(val,
1131                                     &destination,
1132                                     (int) val->hash.length);
1133                         if (value != NULL) {
1134                                 wcb.init(value);
1135                                 if(wcb.length() > 0) {
1136                                         MBTOWC(wcs_buffer, " ");
1137                                         append_char(wcs_buffer[0], &destination)
1138                                 }
1139                         }
1140                 }
1141                 if (value != NULL) {
1142                         APPEND_NAME(value,
1143                                     &destination,
1144                                     (int) value->hash.length);
1145                 }
1146                 value = GETNAME(destination.buffer.start, FIND_LENGTH);
1147                 wcb.init(value);
1148                 if (destination.free_after_use) {
```



```
1149                         retmem(destination.buffer.start);
1150                 }
1151         }

1153         /* Debugging trace */
1154         if (debug_level > 1) {
1155                 if (value != NULL) {
1156                         switch (daemon) {
1157                         case chain_daemon:
1158                                 (void) printf("%s =", name->string_mb);
1159                                 for (chain = (Chain) value;
1160                                      chain != NULL;
1161                                      chain = chain->next) {
1162                                    (void) printf(" %s", chain->name->string
1163                                 }
1164                                 (void) printf("\n");
1165                                 break;
1166                         case no_daemon:
1167                                 (void) printf("%s= %s\n",
1168                                               name->string_mb,
1169                                               value->string_mb);
1170                                 break;
1171                         }
1172                 } else {
1173                         (void) printf("%s =\n", name->string_mb);
1174                 }
1175         }
1176         /* Set the new values in the macro property block */
1177 /**/
1178         if(macro_apx != NULL) {
1179                 macro_apx->body.macro_appendix.value = value;
1180                 INIT_STRING_FROM_STACK(destination, buffer);
1181                 buffer[0] = 0;
1182                 if (value != NULL) {
1183                         APPEND_NAME(value,
1184                                     &destination,
1185                                     (int) value->hash.length);
1186                         if (macro_apx->body.macro_appendix.value_to_append != NU
1187                                 MBTOWC(wcs_buffer, " ");
1188                                 append_char(wcs_buffer[0], &destination);
1189                         }
1190                 }
1191                 if (macro_apx->body.macro_appendix.value_to_append != NULL) {
1192                         APPEND_NAME(macro_apx->body.macro_appendix.value_to_appe
1193                                     &destination,
1194                                     (int) macro_apx->body.macro_appendix.value
1195                 }
1196                 value = GETNAME(destination.buffer.start, FIND_LENGTH);
1197                 if (destination.free_after_use) {
1198                         retmem(destination.buffer.start);
1199                 }
1200         }
1201 /**/
1202         macro->body.macro.value = value;
1203         macro->body.macro.daemon = daemon;
1204         /*
1205          * If the user changes the VIRTUAL_ROOT, we need to flush
1206          * the vroot package cache.
1207          */
1208         if (name == path_name) {
1209                 flush_path_cache();
1210         }
1211         if (name == virtual_root) {
1212                 flush_vroot_cache();
1213         }
1214         /* If this sets the VPATH we remember that */
```

```
1215         if ((name == vpath_name) &&
1216             (value != NULL) &&
1217             (value->hash.length > 0)) {
1218                 vpath_defined = true;
1219         }
1220         /*
1221          * For environment variables we also set the
1222          * environment value each time.
1223          */
1224         if (macro->body.macro.exported) {
1225                 static char     *env;

1227 #ifdef DISTRIBUTED
1228                 if (!reading_environment && (value != NULL)) {
1229 #else
1230                 if (!reading_environment && (value != NULL) && value->dollar) {
1231 #endif
1232                         Envvar  p;

1234                         for (p = envvar; p != NULL; p = p->next) {
1235                                 if (p->name == name) {
1236                                         p->value = value;
1237                                         p->already_put = false;
1238                                         goto found_it;
1239                                 }
1240                         }
1241                         p = ALLOC(Envvar);
1242                         p->name = name;
1243                         p->value = value;
1244                         p->next = envvar;
1245                         p->env_string = NULL;
1246                         p->already_put = false;
1247                         envvar = p;
1248 found_it:;
1249 #ifdef DISTRIBUTED
1250                 }
1251                 if (reading_environment || (value == NULL) || !value->dollar) {
1252 #else
1253                 } else {
1254 #endif
1255                         length = 2 + strlen(name->string_mb);
1256                         if (value != NULL) {
1257                                 length += strlen(value->string_mb);
1258                         }
1259                         Property env_prop = maybe_append_prop(name, env_mem_prop
1260                         /*
1261                          * We use a permanent buffer to reset SUNPRO_DEPENDENCIE
1262                          */
1263                         if (!strncmp(name->string_mb, NOCATGETS("SUNPRO_DEPENDEN
1264                                 if (length >= sunpro_dependencies_buf_size) {
1265                                         sunpro_dependencies_buf_size=length*2;
1266                                         if (sunpro_dependencies_buf_size < 4096)
1267                                                 sunpro_dependencies_buf_size = 4
1268                                         if (sunpro_dependencies_buf)
1269                                                 sunpro_dependencies_oldbuf = sun
1270                                         sunpro_dependencies_buf=getmem(sunpro_de
1271                                 }
1272                                 env = sunpro_dependencies_buf;
1273                         } else {
1274                                 env = getmem(length);
1275                         }
1276                         env_alloc_num++;
1277                         env_alloc_bytes += length;
1278                         (void) sprintf(env,
1279                                        "%s=%s",
1280                                        name->string_mb,
```



```
1281                                        value == NULL ?
1282                                          "" : value->string_mb);
1283                         (void) putenv(env);
1284                         env_prop->body.env_mem.value = env;
1285                         if (sunpro_dependencies_oldbuf) {
1286                                 /* Return old buffer */
1287                                 retmem_mb(sunpro_dependencies_oldbuf);
1288                                 sunpro_dependencies_oldbuf = NULL;
1289                         }
1290                 }
1291         }
1292         if (name == target_arch) {
1293                 Name            ha = getvar(host_arch);
1294                 Name            ta = getvar(target_arch);
1295                 Name            vr = getvar(virtual_root);
1296                 int             length;
1297                 wchar_t         *new_value;
1298                 wchar_t         *old_vr;
1299                 Boolean         new_value_allocated = false;

1301                 Wstring         ha_str(ha);
1302                 Wstring         ta_str(ta);
1303                 Wstring         vr_str(vr);

1305                 wchar_t * wcb_ha = ha_str.get_string();
1306                 wchar_t * wcb_ta = ta_str.get_string();
1307                 wchar_t * wcb_vr = vr_str.get_string();

1309                 length = 32 +
1310                   wslen(wcb_ha) +
1311                     wslen(wcb_ta) +
1312                       wslen(wcb_vr);
1313                 old_vr = wcb_vr;
1314                 MBSTOWCS(wcs_buffer, NOCATGETS("/usr/arch/"));
1315                 if (IS_WEQUALN(old_vr,
1316                                wcs_buffer,
1317                                wslen(wcs_buffer))) {
1318                         old_vr = (wchar_t *) wschr(old_vr, (int) colon_char) + 1
1319                 }
1320                 if ( (ha == ta) || (wslen(wcb_ta) == 0) ) {
1321                         new_value = old_vr;
1322                 } else {
1323                         new_value = ALLOC_WC(length);
1324                         new_value_allocated = true;
1325                         WCSTOMBS(mbs_buffer, old_vr);
1326                         (void) wsprintf(new_value,
1327                                         NOCATGETS("/usr/arch/%s/%s:%s"),
1328                                         ha->string_mb + 1,
1329                                         ta->string_mb + 1,
1330                                         mbs_buffer);
1331                 }
1332                 if (new_value[0] != 0) {
1333                         (void) setvar_daemon(virtual_root,
1334                                              GETNAME(new_value, FIND_LENGTH),
1335                                              false,
1336                                              no_daemon,
1337                                              true,
1338                                              debug_level);
1339                 }
1340                 if (new_value_allocated) {
1341                         retmem(new_value);
1342                 }
1343         }
1344         return macro;
1345 }
```

_____***unchanged_portion_omitted_***