```
**********************************************************
     579 Wed May 20 11:58:26 2015
new/usr/src/cmd/make/Makefile.com
make: unifdef for TEAMWARE_MAKE_CMN (defined)
**********************************************************
#
# 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.

MAKE_INCLUDE= $(SRC)/cmd/make/include
MAKE_DEFS= -DSYSV -DINTER -DTEAMWARE_MAKE_CMN
$(RELEASE_BUILD)MAKE_DEFS += -DNDEBUG
CFLAGS += $(CCVERBOSE)
CPPFLAGS += -I$(MAKE_INCLUDE) $(MAKE_DEFS)
```

```
**********************************************************
   94650 Wed May 20 11:58:27 2015
new/usr/src/cmd/make/bin/doname.cc
make: unifdef for TEAMWARE_MAKE_CMN (defined)
**********************************************************
______unchanged_portion_omitted_

/*
 * DONE.
 *
 *      doname(target, do_get, implicit)
 *
 *      Chases all files the target depends on and builds any that
 *      are out of date. If the target is out of date it is then rebuilt.
 *
 *      Return value:
 *                              Indiates if build failed or nt
 *
 *      Parameters:
 *              target          Target to build
 *              do_get          Run sccs get is nessecary
 *              implicit        doname is trying to find an implicit rule
 *
 *      Global variables used:
 *              assign_done     True if command line assgnment has happened
 *              commands_done   Preserved for the case that we need local value
 *              debug_level     Should we trace make's actions?
 *              default_rule    The rule for ".DEFAULT", used as last resort
 *              empty_name      The Name "", used when looking for single sfx
 *              keep_state      Indicates that .KEEP_STATE is on
 *              parallel        True if building in parallel
 *              recursion_level Used for tracing
 *              report_dependencies make -P is on
 */
Doname
doname(register Name target, register Boolean do_get, register Boolean implicit,
{
        Doname                  result = build_dont_know;
        Chain                   out_of_date_list = NULL;
#ifdef TEAMWARE_MAKE_CMN
        Chain                   target_group;
#endif
        Property                old_locals = NULL;
        register Property       line;
        Property                command = NULL;
        register Dependency     dependency;
        Name                    less = NULL;
        Name                    true_target = target;
        Name                    *automatics = NULL;
        register int            auto_count;
        Boolean                 rechecking_target = false;
        Boolean                 saved_commands_done;
        Boolean                 restart = false;
        Boolean                 save_parallel = parallel;
        Boolean                 doing_subtree = false;

        Boolean                 recheck_conditionals = false;

        if (target->state == build_running) {
                return build_running;
        }
        line = get_prop(target->prop, line_prop);
#ifdef TEAMWARE_MAKE_CMN
        if (line != NULL) {
                /*
                 * If this target is a member of target group and one of the
```

```
                 * other members of the group is running, mark this target
                 * as running.
                 */
                for (target_group = line->body.line.target_group;
                     target_group != NULL;
                     target_group = target_group->next) {
                        if (is_running(target_group->name)) {
                                target->state = build_running;
                                add_pending(target,
                                            recursion_level,
                                            do_get,
                                            implicit,
                                            false);
                                return build_running;
                        }
                }
        }
#endif
        /*
         * If the target is a constructed one for a "::" target,
         * we need to consider that.
         */
        if (target->has_target_prop) {
                true_target = get_prop(target->prop,
                                       target_prop)->body.target.target;
                if (true_target->colon_splits > 0) {
                        /* Make sure we have a valid time for :: targets */
                        Property        time;

                        time = get_prop(true_target->prop, time_prop);
                        if (time != NULL) {
                                true_target->stat.time = time->body.time.time;
                        }
                }
        }
        (void) exists(true_target);
        /*
         * If the target has been processed, we don't need to do it again,
         * unless it depends on conditional macros or a delayed assignment,
         * or it has been done when KEEP_STATE is on.
         */
        if (target->state == build_ok) {
                if((!keep_state || (!target->depends_on_conditional && !assign_d
                        return build_ok;
                } else {
                        recheck_conditionals = true;
                }
        }
        if (target->state == build_subtree) {
                /* A dynamic macro subtree is being built */
                target->state = build_dont_know;
                doing_subtree = true;
                if (!target->checking_subtree) {
                        /*
                         * This target has been started before and therefore
                         * not all dependencies have to be built.
                         */
                        restart = true;
                }
        } else if (target->state == build_pending) {
                target->state = build_dont_know;
                restart = true;
/*
#ifdef TEAMWARE_MAKE_CMN
        } else if (parallel &&
                   keep_state &&
                   (target->conditional_cnt > 0)) {
            if (!parallel_ok(target, false)) {
                add_subtree(target, recursion_level, do_get, implicit);
                target->state = build_running;
                return build_running;
            }
#endif
 */
        }
        /*
         * If KEEP_STATE is on, we have to rebuild the target if the
         * building of it caused new automatic dependencies to be reported.
         * This is where we restart the build.
         */
        if (line != NULL) {
                line->body.line.percent = NULL;
        }
recheck_target:
        /* Init all local variables */
        result = build_dont_know;
        out_of_date_list = NULL;
        command = NULL;
        less = NULL;
        auto_count = 0;
        if (!restart && line != NULL) {
                /*
                 * If this target has never been built before, mark all
                 * of the dependencies as never built.
                 */
                for (dependency = line->body.line.dependencies;
                     dependency != NULL;
                     dependency = dependency->next) {
                        dependency->built = false;
                }
        }
        /* Save the set of automatic depes defined for this target */
        if (keep_state &&
            (line != NULL) &&
            (line->body.line.dependencies != NULL)) {
                Name *p;

                /*
                 * First run thru the dependency list to see how many
                 * autos there are.
                 */
                for (dependency = line->body.line.dependencies;
                     dependency != NULL;
                     dependency = dependency->next) {
                        if (dependency->automatic && !dependency->stale) {
                                auto_count++;
                        }
                }
                /* Create vector to hold the current autos */
                automatics =
                  (Name *) alloca((int) (auto_count * sizeof (Name)));
                /* Copy them */
                for (p = automatics, dependency = line->body.line.dependencies;
                     dependency != NULL;
                     dependency = dependency->next) {
                        if (dependency->automatic && !dependency->stale) {
                                *p++ = dependency->name;
                        }
                }
        }
        if (debug_level > 1) {
                (void) printf(NOCATGETS("%*sdoname(%s)\n"),
```

```
                              recursion_level,
                              "",
                              target->string_mb);
        }
        recursion_level++;
        /* Avoid infinite loops */
        if (target->state == build_in_progress) {
                warning(catgets(catd, 1, 16, "Infinite loop: Target ‘%s’ depends
                        target->string_mb);
                return build_ok;
        }
        target->state = build_in_progress;

        /* Activate conditional macros for the target */
        if (!target->added_pattern_conditionals) {
                add_pattern_conditionals(target);
                target->added_pattern_conditionals = true;
        }
        if (target->conditional_cnt > 0) {
                old_locals = (Property) alloca(target->conditional_cnt *
                                               sizeof (Property_rec));
                set_locals(target, old_locals);
        }

/*
 * after making the call to dynamic_dependecies unconditional we can handle
 * target names that are same as file name. In this case $$@ in the
 * dependencies did not mean anything. WIth this change it expands it
 * as expected.
 */
        if (!target->has_depe_list_expanded)
        {
                dynamic_dependencies(target);
        }

/*
 *      FIRST SECTION -- GO THROUGH DEPENDENCIES AND COLLECT EXPLICIT
 *      COMMANDS TO RUN
 */
        if ((line = get_prop(target->prop, line_prop)) != NULL) {
                if (check_dependencies(&result,
                                       line,
                                       do_get,
                                       target,
                                       true_target,
                                       doing_subtree,
                                       &out_of_date_list,
                                       old_locals,
                                       implicit,
                                       &command,
                                       less,
                                       rechecking_target,
                                       recheck_conditionals)) {
                        return build_running;
                }
                if (line->body.line.query != NULL) {
                        delete_query_chain(line->body.line.query);
                }
                line->body.line.query = out_of_date_list;
        }


/*
 * If the target is a :: type, do not try to find the rule for the target,
 * all actions will be taken by separate branches.
 * Else, we try to find an implicit rule using various methods,
 * we quit as soon as one is found.
 *
 * [tolik, 12 Sep 2002] Do not try to find implicit rule for the target
 * being rechecked - the target is being rechecked means that it already
 * has explicit dependencies derived from an implicit rule found
 * in previous step.
 */
        if (target->colon_splits == 0 && !rechecking_target) {
                /* Look for percent matched rule */
                if ((result == build_dont_know) &&
                    (command == NULL)) {
                        switch (find_percent_rule(
                                        target,
                                        &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
                        case build_ok:
                                result = build_ok;
                                break;
                        }
                }
                /* Look for double suffix rule */
                if (result == build_dont_know) {
                        Property member;

                        if (target->is_member &&
                            ((member = get_prop(target->prop, member_prop)) !=
                             NULL)) {
                                switch (find_ar_suffix_rule(target,
                                                member->body.
                                                member.member,
                                                &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
                                default:
                                        /* ALWAYS bind $% for old style */
                                        /* ar rules */
                                        if (line == NULL) {
                                                line =
                                                  maybe_append_prop(target,
                                                                    line_prop);
                                        }
                                        line->body.line.percent =
                                          member->body.member.member;
                                        break;
                                }
                        } else {
                                switch (find_double_suffix_rule(target,
                                                &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,
                                                                      conditiona
                                                             0);
                                        }
                                        return build_running;
#endif
                                }
                        }
                }
                /* 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
                }
                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;
                        if ((line = get_prop(target->prop,
                                             line_prop)) != NULL) {
                                if (line->body.line.query != NULL) {
                                        delete_query_chain(line->body.line.query
                                }
                                line->body.line.query = NULL;
                        }
                        if (target->conditional_cnt > 0) {
                                reset_locals(target,
                                             old_locals,
                                             get_prop(target->prop,
                                                     conditional_prop),
                                             0);
                        }
                        return build_running;
                case build_serial:
                        add_serial(target,
                                   --recursion_level,
                                   do_get,
                                   implicit);
                        target->state = build_running;
                        line = get_prop(target->prop, line_prop);
                        if (line != NULL) {
                                if (line->body.line.query != NULL) {
                                        delete_query_chain(line->body.line.query
                                }
                                line->body.line.query = NULL;
                        }
                        if (target->conditional_cnt > 0) {
                                reset_locals(target,
                                             old_locals,
                                             get_prop(target->prop,
                                                     conditional_prop),
                                             0);
                        }
                        return build_running;
#endif
                case build_ok:
                        /* If all went OK set a nice timestamp */
                        if (true_target->stat.time == file_doesnt_exist) {
                                true_target->stat.time = file_max_time;
                        }
                        break;
                }
        } else {
                /*
                 * If no command was found for the target, and it doesn't
                 * exist, and it is mentioned as a target in the makefile,
                 * we say it is extremely new and that it is OK.
                 */
                if (target->colons != no_colon) {
                        if (true_target->stat.time == file_doesnt_exist){
                                true_target->stat.time = file_max_time;
                        }
                        result = build_ok;
                }
                /*
                 * Trying dynamic targets.
                 */
                if(!doname_dyntarget) {
                        doname_dyntarget = 1;
                        Name dtarg = find_dyntarget(target);
                        if(dtarg!=NULL) {
                                if (!target->has_depe_list_expanded) {
                                        dynamic_dependencies(target);
                                }
                                if ((line = get_prop(target->prop, line_prop)) !
                                        if (check_dependencies(&result,
                                                               line,
                                                               do_get,
                                                               target,
                                                               true_target,
                                                               doing_subtree,
                                                               &out_of_date_list
                                                               old_locals,
                                                               implicit,
                                                               &command,
                                                               less,
                                                               rechecking_target
                                                               recheck_condition
                                        {
                                                return build_running;
                                        }
                                        if (line->body.line.query != NULL) {
                                                delete_query_chain(line->body.li
                                        }
                                        line->body.line.query = out_of_date_list
                                }
```

```
                                goto r_command;
                        }
                }
                /*
                 * If the file exists, it is OK that we couldnt figure
                 * out how to build it.
                 */
                (void) exists(target);
                if ((target->stat.time != file_doesnt_exist) &&
                    (result == build_dont_know)) {
                        result = build_ok;
                }
        }

        /*
         * Some of the following is duplicated in the function finish_doname.
         * If anything is changed here, check to see if it needs to be
         * changed there.
         */
        if ((line = get_prop(target->prop, line_prop)) != NULL) {
                if (line->body.line.query != NULL) {
                        delete_query_chain(line->body.line.query);
                }
                line->body.line.query = NULL;
        }
        target->state = result;
        parallel = save_parallel;
        if (target->conditional_cnt > 0) {
                reset_locals(target,
                             old_locals,
                             get_prop(target->prop, conditional_prop),
                             0);
        }
        recursion_level--;
        if (target->is_member) {
                Property member;

                /* Propagate the timestamp from the member file to the member*/
                if ((target->stat.time != file_max_time) &&
                    ((member = get_prop(target->prop, member_prop)) != NULL) &&
                    (exists(member->body.member.member) > file_doesnt_exist)) {
                        target->stat.time =
                          member->body.member.member->stat.time;
                }
        }
        /*
         * Check if we found any new auto dependencies when we
         * built the target.
         */
        if ((result == build_ok) && check_auto_dependencies(target,
                                                            auto_count,
                                                            automatics)) {
                if (debug_level > 0) {
                        (void) printf(catgets(catd, 1, 18, "%*sTarget '%s' acqui
                                      recursion_level,
                                      "",
                                      true_target->string_mb);
                }
                rechecking_target = true;
                saved_commands_done = commands_done;
                goto recheck_target;
        }

        if (rechecking_target && !commands_done) {
                commands_done = saved_commands_done;
        }
        return result;
}

/*
 * DONE.
 *
 *      check_dependencies(result, line, do_get,
 *                      target, true_target, doing_subtree, out_of_date_tail,
 *                      old_locals, implicit, command, less, rechecking_target)
 *
 *      Return value:
 *                              True returned if some dependencies left running
 *
 *      Parameters:
 *              result          Pointer to cell we update if build failed
 *              line            We get the dependencies from here
 *              do_get          Allow use of sccs get in recursive doname()
 *              target          The target to chase dependencies for
 *              true_target     The real one for :: and lib(member)
 *              doing_subtree   True if building a conditional macro subtree
 *              out_of_date_tail Used to set the $? list
 *              old_locals      Used for resetting the local macros
 *              implicit        Called when scanning for implicit rules?
 *              command         Place to stuff command
 *              less            Set to $< value
 *
 *      Global variables used:
 *              command_changed Set if we suspect .make.state needs rewrite
 *              debug_level     Should we trace actions?
 *              force           The Name " FORCE", compared against
 *              recursion_level Used for tracing
 *              rewrite_statefile Set if .make.state needs rewriting
 *              wait_name       The Name ".WAIT", compared against
 */
static Boolean
#ifdef TEAMWARE_MAKE_CMN
check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
#else
check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
#endif
{
        Boolean                 dependencies_running;
        register Dependency     dependency;
        Doname                  dep_result;
        Boolean                 dependency_changed = false;

        line->body.line.dependency_time = file_doesnt_exist;
        if (line->body.line.query != NULL) {
                delete_query_chain(line->body.line.query);
        }
        line->body.line.query = NULL;
        line->body.line.is_out_of_date = false;
        dependencies_running = false;
        /*
         * Run thru all the dependencies and call doname() recursively
         * on each of them.
         */
        for (dependency = line->body.line.dependencies;
             dependency != NULL;
             dependency = dependency->next) {
                Boolean this_dependency_changed = false;

                if (!dependency->automatic &&
                    (rechecking_target || target->rechecking_target)) {
                        /*
```

```
                         * We only bother with the autos when rechecking
                         */
                        continue;
                }

                if (dependency->name == wait_name) {
                        /*
                         * The special target .WAIT means finish all of
                         * the prior dependencies before continuing.
                         */
                        if (dependencies_running) {
                                break;
                        }
                } else {
                        timestruc_t     depe_time = file_doesnt_exist;


                        if (true_target->is_member) {
                                depe_time = exists(dependency->name);
                        }
                        if (dependency->built ||
                            (dependency->name->state == build_failed)) {
                                dep_result = (Doname) dependency->name->state;
                        } else {
                                dep_result = doname_check(dependency->name,
                                                          do_get,
                                                          false,
                                                          (Boolean) dependency->
                        }
                        if (true_target->is_member || dependency->name->is_membe
                                /* should compare only secs, cause lib members d
                                if (depe_time.tv_sec != dependency->name->stat.t
                                        this_dependency_changed =
                                          dependency_changed =
                                            true;
                                }
                        } else {
                                if (depe_time != dependency->name->stat.time) {
                                        this_dependency_changed =
                                          dependency_changed =
                                            true;
                                }
                        }
                        dependency->built = true;
                        switch (dep_result) {
                        case build_running:
                                dependencies_running = true;
                                continue;
                        case build_failed:
                                *result = build_failed;
                                break;
                        case build_dont_know:
/*
 * If make can't figure out how to make a dependency, maybe the dependency
 * is out of date. In this case, we just declare the target out of date
 * and go on. If we really need the dependency, the make'ing of the target
 * will fail. This will only happen for automatic (hidden) dependencies.
 */
                                if(!recheck_conditionals) {
                                        line->body.line.is_out_of_date = true;
                                }
                                /*
                                 * Make sure the dependency is not saved
                                 * in the state file.
                                 */
                                dependency->stale = true;
                                rewrite_statefile =
                                  command_changed =
                                    true;
                                if (debug_level > 0) {
                                        (void) printf(catgets(catd, 1, 19, "Targ
                                                     true_target->string_mb,
                                                     dependency->name->string_mb
                                }
                                break;
                        }
                        if (dependency->name->depends_on_conditional) {
                                target->depends_on_conditional = true;
                        }
                        if (dependency->name == force) {
                                target->stat.time =
                                  dependency->name->stat.time;
                        }
                        /*
                         * Propagate new timestamp from "member" to
                         * "lib.a(member)".
                         */
                        (void) exists(dependency->name);

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

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

                        if (debug_level > 1) {
                                (void) printf(catgets(catd, 1, 20, "%*sDate(%s)=
                                              recursion_level,
                                              "",
                                              dependency->name->string_mb,
                                              time_to_string(dependency->name->
                                                             stat.time));
                                if (dependency->name->stat.time > line->body.lin
                                        (void) printf(catgets(catd, 1, 21, "%*sD
                                                      recursion_level,
                                                      "",
                                                      true_target->string_mb,
                                                      time_to_string(line->body.
                                                                     dependency_
                                }
                        }

                        /* Build the $? list */
                        if (true_target->is_member) {
                                if (this_dependency_changed == true) {
                                        true_target->stat.time = dependency->nam
                                        true_target->stat.time.tv_sec--;
                                } else {
                                        /* Dina:
                                         * The next statement is commented
                                         * out as a fix for bug #1051032.
                                         * if dependency hasn't changed
                                         * then there's no need to invalidate
                                         * true_target. This statemnt causes
                                         * make to take much longer to process
                                         * an already-built archive. Soren
                                         * said it was a quick fix for some
                                         * problem he doesn't remember.
```

```
                                        true_target->stat.time = file_no_time;
                                         */
                                        (void) exists(true_target);
                                }
                        } else {
                                (void) exists(true_target);
                        }
                        Boolean out_of_date;
                        if (true_target->is_member || dependency->name->is_membe
                                out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
                                                                        dependen
                        } else {
                                out_of_date = (Boolean) OUT_OF_DATE(true_target-
                                                                    dependency->
                        }
                        if ((build_unconditional || out_of_date) &&
                            (dependency->name != force) &&
                            (dependency->stale == false)) {
                                *out_of_date_tail = ALLOC(Chain);
                                if (dependency->name->is_member &&
                                    (get_prop(dependency->name->prop,
                                              member_prop) != NULL)) {
                                        (*out_of_date_tail)->name =
                                          get_prop(dependency->name->prop,
                                                   member_prop)->
                                                     body.member.member;
                                } else {
                                        (*out_of_date_tail)->name =
                                          dependency->name;
                                }
                                (*out_of_date_tail)->next = NULL;
                                out_of_date_tail = &(*out_of_date_tail)->next;
                                if (debug_level > 0) {
                                        if (dependency->name->stat.time == file_
                                                (void) printf(catgets(catd, 1, 2
                                                              recursion_level,
                                                              "",
                                                              true_target->strin
                                                              dependency->name->
                                        } else {
                                                (void) printf(catgets(catd, 1, 2
                                                              recursion_level,
                                                              "",
                                                              true_target->strin
                                                              dependency->name->
                                        }
                                }
                        }
                        if (dependency->name == force) {
                                force->stat.time =
                                  file_max_time;
                                force->state = build_dont_know;
                        }
                }
        }
#ifdef TEAMWARE_MAKE_CMN
        if (dependencies_running) {
                if (doing_subtree) {
                        if (target->conditional_cnt > 0) {
                                reset_locals(target,
                                             old_locals,
                                             get_prop(target->prop,
                                                      conditional_prop),
                                             0);
                        }
                        return true;
                } else {
                        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 true;
                }
        }
#endif
        /*
         * Collect the timestamp of the youngest double colon target
         * dependency.
         */
        if (target->is_double_colon_parent) {
                for (dependency = line->body.line.dependencies;
                     dependency != NULL;
                     dependency = dependency->next) {
                        Property        tmp_line;

                        if ((tmp_line = get_prop(dependency->name->prop, line_pr
                                if(tmp_line->body.line.dependency_time != file_m
                                        target->stat.time =
                                          MAX(tmp_line->body.line.dependency_tim
                                              target->stat.time);
                                }
                        }
                }
        }
        if ((true_target->is_member) && (dependency_changed == true)) {
                true_target->stat.time = file_no_time;
        }
        /*
         * After scanning all the dependencies, we check the rule
         * if we found one.
         */
        if (line->body.line.command_template != NULL) {
                if (line->body.line.command_template_redefined) {
                        warning(catgets(catd, 1, 24, "Too many rules defined for
                                target->string_mb);
                }
                *command = line;
                /* Check if the target is out of date */
                Boolean out_of_date;
                if (true_target->is_member) {
                        out_of_date = (Boolean) OUT_OF_DATE_SEC(true_target->sta
                                                                line->body.line.
                } else {
                        out_of_date = (Boolean) OUT_OF_DATE(true_target->stat.ti
                                                            line->body.line.depe
                }
                if (build_unconditional || out_of_date){
                        if(!recheck_conditionals) {
                                line->body.line.is_out_of_date = true;
                        }
                }
                line->body.line.sccs_command = false;
                line->body.line.target = true_target;
```

```
                if(gnu_style) {

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

                        // set $* for explicit rule
                        Name                    target_body;
                        Name                    tt = true_target;
                        Property                member;
                        register wchar_t        *target_end;
                        register Dependency     suffix;
                        register int            suffix_length;
                        Wstring                 targ_string;
                        Wstring                 suf_string;

                        if (true_target->is_member &&
                            ((member = get_prop(target->prop, member_prop)) !=
                             NULL)) {
                                tt = member->body.member.member;
                        }
                        targ_string.init(tt);
                        target_end = targ_string.get_string() + tt->hash.length;
                        for (suffix = suffixes; suffix != NULL; suffix = suffix-
                                suffix_length = suffix->name->hash.length;
                                suf_string.init(suffix->name);
                                if (tt->hash.length < suffix_length) {
                                        continue;
                                } else if (!IS_WEQUALN(suf_string.get_string(),
                                                (target_end - suffix_length),
                                                suffix_length)) {
                                        continue;
                                }
                                target_body = GETNAME(
                                        targ_string.get_string(),
                                        (int)(tt->hash.length - suffix_length)
                                );
                                line->body.line.star = target_body;
                        }

                        // set result = build_ok so that implicit rules are not
                        if(*result == build_dont_know) {
                                *result = build_ok;
                        }
                }
                if (less != NULL) {
                        line->body.line.less = less;
                }
        }

        return false;
}
```
______***unchanged_portion_omitted_***

```
/*
 * DONE.
 *
 *      run_command(line)
 *
 *      Takes one Cmd_line and runs the commands from it.
 *
 *      Return value:
 *                              Indicates if the command failed or not
 *
 *      Parameters:
```

```
 *              line            The command line to run
 *
 *      Global variables used:
 *              commands_done   Set if we do run command
 *              current_line    Set to the line we run a command from
 *              current_target  Set to the target we run a command for
 *              file_number     Used to form temp file name
 *              keep_state      Indicates that .KEEP_STATE is on
 *              make_state      The Name ".make.state", used to check timestamp
 *              parallel        True if currently building in parallel
 *              parallel_process_cnt Count of parallel processes running
 *              quest           Indicates that make -q is on
 *              rewrite_statefile Set if we do run a command
 *              sunpro_dependencies The Name "SUNPRO_DEPENDENCIES", set value
 *              temp_file_directory Used to form temp fie name
 *              temp_file_name  Set to the name of the temp file
 *              touch           Indicates that make -t is on
 */
static Doname
run_command(register Property line, Boolean)
{
        register Doname         result = build_ok;
        register Boolean        remember_only = false;
        register Name           target = line->body.line.target;
        wchar_t                 *string;
        char                    tmp_file_path[MAXPATHLEN];

        if (!line->body.line.is_out_of_date && target->rechecking_target) {
                target->rechecking_target = false;
                return build_ok;
        }

        /*
         * Build the command if we know the target is out of date,
         * or if we want to check cmd consistency.
         */
        if (line->body.line.is_out_of_date || keep_state) {
                /* Hack for handling conditional macros in DMake. */
                if (!line->body.line.dont_rebuild_command_used) {
                        build_command_strings(target, line);
                }
        }
        /* Never mind */
        if (!line->body.line.is_out_of_date) {
                return build_ok;
        }
        /* If quest, then exit(1) because the target is out of date */
        if (quest) {
                if (posix) {
#ifdef TEAMWARE_MAKE_CMN
                        result = execute_parallel(line, true);
#else
                        result = execute_serial(line);
#endif
                }
                exit_status = 1;
                exit(1);
        }
        /* We actually had to do something this time */
        rewrite_statefile = commands_done = true;
        /*
         * If this is an sccs command, we have to do some extra checking
         * and possibly complain. If the file can't be gotten because it's
         * checked out, we complain and behave as if the command was
         * executed eventhough we ignored the command.
         */
```

```
        if (!touch &&
            line->body.line.sccs_command &&
            (target->stat.time != file_doesnt_exist) &&
            ((target->stat.mode & 0222) != 0)) {
                fatal(catgets(catd, 1, 27, "%s is writable so it cannot be sccs
                      target->string_mb);
                target->has_complained = remember_only = true;
        }
        /*
         * If KEEP_STATE is on, we make sure we have the timestamp for
         * .make.state. If .make.state changes during the command run,
         * we reread .make.state after the command. We also setup the
         * environment variable that asks utilities to report dependencies.
         */
        if (!touch &&
            keep_state &&
            !remember_only) {
                (void) exists(make_state);
                if((strlen(temp_file_directory) == 1) &&
                        (temp_file_directory[0] == '/')) {
                   tmp_file_path[0] = '\0';
                } else {
                   strcpy(tmp_file_path, temp_file_directory);
                }
                sprintf(mbs_buffer,
                                NOCATGETS("%s/.make.dependency.%08x.%d.%d"),
                                tmp_file_path,
                                hostid,
                                getpid(),
                                file_number++);
                MBSTOWCS(wcs_buffer, mbs_buffer);
                Boolean fnd;
                temp_file_name = getname_fn(wcs_buffer, FIND_LENGTH, false, &fnd
                temp_file_name->stat.is_file = true;
                int len = 2*MAXPATHLEN + strlen(target->string_mb) + 2;
                wchar_t *to = string = ALLOC_WC(len);
                for (wchar_t *from = wcs_buffer; *from != (int) nul_char; ) {
                        if (*from == (int) space_char) {
                                *to++ = (int) backslash_char;
                        }
                        *to++ = *from++;
                }
                *to++ = (int) space_char;
                MBSTOWCS(to, target->string_mb);
                Name sprodep_name = getname_fn(string, FIND_LENGTH, false, &fnd)
                (void) SETVAR(sunpro_dependencies,
                              sprodep_name,
                              false);
                retmem(string);
        } else {
                temp_file_name = NULL;
        }

        /*
         * In case we are interrupted, we need to know what was going on.
         */
        current_target = target;
        /*
         * We also need to be able to save an empty command instead of the
         * interrupted one in .make.state.
         */
        current_line = line;
        if (remember_only) {
                /* Empty block!!! */
        } else if (touch) {
                result = touch_command(line, target, result);
                if (posix) {
#ifdef TEAMWARE_MAKE_CMN
                        result = execute_parallel(line, true);
#else
                        result = execute_serial(line);
#endif
                }
        } else {
                /*
                 * If this is not a touch run, we need to execute the
                 * proper command(s) for the target.
                 */
#ifdef TEAMWARE_MAKE_CMN
                if (parallel) {
                        if (!parallel_ok(target, true)) {
                                /*
                                 * We are building in parallel, but
                                 * this target must be built in serial.
                                 */
                                /*
                                 * If nothing else is building,
                                 * do this one, else wait.
                                 */
                                if (parallel_process_cnt == 0) {
#ifdef TEAMWARE_MAKE_CMN
                                        result = execute_parallel(line, true, ta
#else
                                        result = execute_serial(line);
#endif
                                } else {
                                        current_target = NULL;
                                        current_line = NULL;
/*
                                        line->body.line.command_used = NULL;
 */
                                        line->body.line.dont_rebuild_command_use
                                        return build_serial;
                                }
                        } else {
                                result = execute_parallel(line, false);
                                switch (result) {
                                case build_running:
                                        return build_running;
                                case build_serial:
                                        if (parallel_process_cnt == 0) {
#ifdef TEAMWARE_MAKE_CMN
                                                result = execute_parallel(line,
#else
                                                result = execute_serial(line);
#endif
                                        } else {
                                                current_target = NULL;
                                                current_line = NULL;
                                                target->parallel = false;
                                                line->body.line.command_used =
                                                                        NULL;
                                                return build_serial;
                                        }
                                }
                        }
                } else {
#endif
#ifdef TEAMWARE_MAKE_CMN
                        result = execute_parallel(line, true, target->localhost)
#else
                        result = execute_serial(line);
```

```
#endif
#ifdef TEAMWARE_MAKE_CMN
                }
#endif
        }
        temp_file_name = NULL;
        if (report_dependencies_level == 0){
                update_target(line, result);
        }
        current_target = NULL;
        current_line = NULL;
        return result;
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
    4576 Wed May 20 11:58:27 2015
new/usr/src/cmd/make/bin/globals.cc
make: unifdef for TEAMWARE_MAKE_CMN (defined)
**********************************************************
/*
 * 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;
        Boolean         all_parallel;                   /* TEAMWARE_MAKE_CMN */
        Boolean         assign_done;
```

```
        int foo;
        Boolean         build_failed_seen;
        Name            built_last_make_run;
        Name            c_at;
        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;
        Boolean         no_action_was_taken = true;     /* true if we've not **
                                                        ** run any command   */

        Boolean         no_parallel = false;
        Boolean         no_parallel = false;            /* TEAMWARE_MAKE_CMN */
        Name            no_parallel_name;
        Name            not_auto;
        Boolean         only_parallel;
        Boolean         parallel;
        Boolean         only_parallel;                  /* TEAMWARE_MAKE_CMN */
        Boolean         parallel;                       /* TEAMWARE_MAKE_CMN */
        Name            parallel_name;
        Name            localhost_name;
        int             parallel_process_cnt;
        Percent         percent_list;
        Dyntarget       dyntarget_list;
        Name            plus;
        Name            pmake_machinesfile;
        Name            precious;
        Name            primary_makefile;
        Boolean         quest;                          /* '-q' */
        short           read_trace_level;
        Boolean         reading_dependencies = false;
        Name            recursive_name;
        int             recursion_level;
        short           report_dependencies_level = 0;  /* -P */
        Boolean         report_pwd;
        Boolean         rewrite_statefile;
        Running         running_list;
        char            *sccs_dir_path;
        Name            sccs_get_name;
        Name            sccs_get_posix_name;
        Cmd_line        sccs_get_rule;
        Cmd_line        sccs_get_org_rule;
        Cmd_line        sccs_get_posix_rule;
        Name            get_name;
        Cmd_line        get_rule;
        Name            get_posix_name;
        Cmd_line        get_posix_rule;
        Boolean         send_mtool_msgs;                /* '-K' */
        Boolean         all_precious;
        Boolean         silent_all;                     /* '-s' */
        Boolean         report_cwd;                     /* '-w' */
        Boolean         silent;                         /* '-s' */
        Name            silent_name;
        char            *stderr_file = NULL;
        char            *stdout_file = NULL;
        Boolean         stdout_stderr_same;
        Dependency      suffixes;
        Name            suffixes_name;
        Name            sunpro_dependencies;
        Boolean         target_variants;
        const char      *tmpdir = NOCATGETS("/tmp");
        const char      *temp_file_directory = NOCATGETS(".");
        Name            temp_file_name;
        short           temp_file_number;
        time_t          timing_start;
        wchar_t         *top_level_target;
        Boolean         touch;                          /* '-t' */
        Boolean         trace_reader;                   /* '-D' */
        Boolean         build_unconditional;            /* '-u' */
        pathpt          vroot_path = VROOT_DEFAULT;
        Name            wait_name;
        wchar_t         wcs_buffer2[MAXPATHLEN];
        wchar_t         *wcs_ptr;
        wchar_t         *wcs_ptr2;
        nl_catd         catd;
        long int        hostid;

/*
 * File table of contents
 */
```

```
**********************************************************
   43284 Wed May 20 11:58:28 2015
new/usr/src/cmd/make/bin/implicit.cc
make: unifdef for TEAMWARE_MAKE_CMN (defined)
**********************************************************
______unchanged_portion_omitted_

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

        /* If the target is constructed for a "::" target we consider that */
        if (target->has_target_prop) {
                true_target = get_prop(target->prop,
                                       target_prop)->body.target.target;
        }
        if (target->has_long_member_name) {
                true_target = get_prop(target->prop,
                                       long_member_name_prop)->body.long_member_
        }
        if (debug_level > 1) {
                (void) printf(catgets(catd, 1, 222, "%*sLooking for %% rule for
                              recursion_level,
                              "",
                              true_target->string_mb);
```

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

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

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

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

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

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

                                if (depe_to_check == empty_name) {
                                                result = build_ok;
                                } else {
                                        if (debug_level > 1) {
                                                (void) printf(catgets(catd, 1, 2
                                                              recursion_level,
                                                              "",
                                                              depe_to_check->str
                                        }

                                        pat_rule->being_expanded = true;

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

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

                                        /* restore debug_level */
                                        debug_level = save_debug_level;
                                }

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

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

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

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

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

        if (debug_level > 1) {
                (void) printf(catgets(catd, 1, 224, "%*sMatched %s:"),
                                      recursion_level,
                                      "",
                                      target->string_mb);

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

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

                (void) printf(catgets(catd, 1, 225, " from: %s:"),
                              pat_rule->name->string_mb);

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

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

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

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

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

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

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

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

                        if ((depe->name->stat.time > line->body.line.dependency_
                            (debug_level > 1)) {
                                (void) printf(catgets(catd, 1, 226, "%*sDate(%s)
                                              recursion_level,
                                              "",
                                              depe->name->string_mb,
                                              time_to_string(depe->name->stat.ti
                                              true_target->string_mb,
                                              time_to_string(line->body.line.dep
                        }

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

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

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

                                if (debug_level > 0) {
                                        (void) printf(catgets(catd, 1, 227, "%*s
                                                      recursion_level,
                                                      "",
                                                      true_target->string_mb,
                                                      pat_rule->name->string_mb)

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

                                        (void) printf(catgets(catd, 1, 228, " be
                                                      depe->name->string_mb);
                                }
                        }
                }
        } else {
                if ((true_target->stat.time <= file_doesnt_exist) ||
                    (true_target->stat.time < line->body.line.dependency_time))
                        if(!rechecking) {
                                line->body.line.is_out_of_date = true;
                        }
                        if (debug_level > 0) {
                                (void) printf(catgets(catd, 1, 229, "%*sBuilding
                                              recursion_level,
                                              "",
                                              true_target->string_mb,
                                              pat_rule->name->string_mb,
                                              (target->stat.time > file_doesnt_e
                                              catgets(catd, 1, 230, "because it
                                              catgets(catd, 1, 236, "because it
                        }
                }
        }

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

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

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

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

#ifdef TEAMWARE_MAKE_CMN
        /*
         * This #ifdef fixes a dmake bug, but introduces bugid 1136156.
         */
        return result;
#else
        return build_ok;
#endif
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   90998 Wed May 20 11:58:28 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef for TEAMWARE_MAKE_CMN (defined)
**********************************************************
/*
 * 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.
 */

/*
 *      main.cc
 *
 *      make program main routine plus some helper routines
 */

/*
 * Included files
 */
#if defined(TEAMWARE_MAKE_CMN)
#       include <avo/intl.h>
#endif

#include <bsd/bsd.h>            /* bsd_signal() */


#include <locale.h>             /* setlocale() */
#include <mk/defs.h>
#include <mksdmsi18n/mksdmsi18n.h>      /* libmksdmsi18n_init() */
#include <mksh/macro.h>         /* getvar() */
#include <mksh/misc.h>          /* getmem(), setup_char_semantics() */

#if defined(TEAMWARE_MAKE_CMN)
#endif

#include <pwd.h>                /* getpwnam() */
#include <setjmp.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/errno.h>          /* ENOENT */
#include <sys/stat.h>           /* fstat() */
#include <fcntl.h>              /* open() */

#       include <sys/systeminfo.h>      /* sysinfo() */

#include <sys/types.h>          /* stat() */
```

```
#include <sys/wait.h>           /* wait() */
#include <unistd.h>             /* execv(), unlink(), access() */
#include <vroot/report.h>       /* report_dependency(), get_report_file() */

// From read2.cc
extern  Name            normalize_name(register wchar_t *name_string, register i

// From parallel.cc
#if defined(TEAMWARE_MAKE_CMN)
#define MAXJOBS_ADJUST_RFE4694000

#ifdef MAXJOBS_ADJUST_RFE4694000
extern void job_adjust_fini();
#endif /* MAXJOBS_ADJUST_RFE4694000 */
#endif /* TEAMWARE_MAKE_CMN */


/*
 * Defined macros
 */
#define MAKE_PREFIX             NOCATGETS("/usr")
#define LD_SUPPORT_ENV_VAR      NOCATGETS("SGS_SUPPORT_32")
#define LD_SUPPORT_ENV_VAR_32   NOCATGETS("SGS_SUPPORT_32")
#define LD_SUPPORT_ENV_VAR_64   NOCATGETS("SGS_SUPPORT_64")
#define LD_SUPPORT_MAKE_LIB     NOCATGETS("libmakestate.so.1")
#define LD_SUPPORT_MAKE_LIB_DIR NOCATGETS("/lib")
#define LD_SUPPORT_MAKE_LIB_DIR_64      NOCATGETS("/64")

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static  char            *argv_zero_string;
static  Boolean         build_failed_ever_seen;
static  Boolean         continue_after_error_ever_seen; /* '-k' */
static  Boolean         dmake_group_specified;          /* '-g' */
static  Boolean         dmake_max_jobs_specified;       /* '-j' */
static  Boolean         dmake_mode_specified;           /* '-m' */
static  Boolean         dmake_add_mode_specified;       /* '-x' */
static  Boolean         dmake_output_mode_specified;    /* '-x DMAKE_OUTPUT_MODE
static  Boolean         dmake_compat_mode_specified;    /* '-x SUN_MAKE_COMPAT_M
static  Boolean         dmake_odir_specified;           /* '-o' */
static  Boolean         dmake_rcfile_specified;         /* '-c' */
static  Boolean         env_wins;                       /* '-e' */
static  Boolean         ignore_default_mk;              /* '-r' */
static  Boolean         list_all_targets;               /* '-T' */
static  int             mf_argc;
static  char            **mf_argv;
static  Dependency_rec  not_auto_depen_struct;
static  Dependency      not_auto_depen = &not_auto_depen_struct;
static  Boolean         pmake_cap_r_specified;          /* '-R' */
static  Boolean         pmake_machinesfile_specified;   /* '-M' */
static  Boolean         stop_after_error_ever_seen;     /* '-S' */
static  Boolean         trace_status;                   /* '-p' */

#ifdef DMAKE_STATISTICS
static  Boolean         getname_stat = false;
#endif

#if defined(TEAMWARE_MAKE_CMN)
        static  time_t          start_time;
        static  int             g_argc;
        static  char            **g_argv;
```

```
#endif

/*
 * File table of contents
 */
        extern "C" void         cleanup_after_exit(void);

#ifdef TEAMWARE_MAKE_CMN
extern "C" {
        extern  void            dmake_exit_callback(void);
        extern  void            dmake_message_callback(char *);
}
#endif

extern  Name            normalize_name(register wchar_t *name_string, register i

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

static  void            append_makeflags_string(Name, String);
static  void            doalarm(int);
static  void            enter_argv_values(int , char **, ASCII_Dyn_Array *);
static  void            make_targets(int, char **, Boolean);
static  int             parse_command_option(char);
static  void            read_command_options(int, char **);
static  void            read_environment(Boolean);
static  void            read_files_and_state(int, char **);
static  Boolean         read_makefile(Name, Boolean, Boolean, Boolean);
static  void            report_recursion(Name);
static  void            set_sgs_support(void);
static  void            setup_for_projectdir(void);
static  void            setup_makeflags_argv(void);
static  void            report_dir_enter_leave(Boolean entering);

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

static const char       verstring[] = "illumos make";
#ifdef TEAMWARE_MAKE_CMN
        static const char       verstring[] = "illumos make";
#endif

jmp_buf jmpbuffer;
extern nl_catd catd;

/*
 *      main(argc, argv)
 *
 *      Parameters:
 *              argc                    You know what this is
 *              argv                    You know what this is
 *
 *      Static variables used:
 *              list_all_targets        make -T seen
 *              trace_status            make -p seen
 *
 *      Global variables used:
 *              debug_level             Should we trace make actions?
 *              keep_state              Set if .KEEP_STATE seen
 *              makeflags               The Name "MAKEFLAGS", used to get macro
 *              remote_command_name     Name of remote invocation cmd ("on")
 *              running_list            List of parallel running processes
 *              stdout_stderr_same      true if stdout and stderr are the same
 *              auto_dependencies       The Name "SUNPRO_DEPENDENCIES"
 *              temp_file_directory     Set to the dir where we create tmp file
 *              trace_reader            Set to reflect tracing status
 *              working_on_targets      Set when building user targets
 */
```

```
int
main(int argc, char *argv[])
{
        /*
         * cp is a -> to the value of the MAKEFLAGS env var,
         * which has to be regular chars.
         */
        register char           *cp;
        char                    make_state_dir[MAXPATHLEN];
        Boolean                 parallel_flag = false;
        char                    *prognameptr;
        char                    *slash_ptr;
        mode_t                  um;
        int                     i;
#ifdef TEAMWARE_MAKE_CMN
        struct itimerval        value;
        char                    def_dmakerc_path[MAXPATHLEN];
        Name                    dmake_name, dmake_name2;
        Name                    dmake_value, dmake_value2;
        Property                prop, prop2;
        struct stat             statbuf;
        int                     statval;
#endif

        struct stat             out_stat, err_stat;
        hostid = gethostid();
        bsd_signals();

        (void) setlocale(LC_ALL, "");


#ifdef DMAKE_STATISTICS
        if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
                getname_stat = true;
        }
#endif

#if defined(TEAMWARE_MAKE_CMN)
        catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
#endif

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


#if defined(TEAMWARE_MAKE_CMN)
/*
 * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
 * from avo_util library.
 */
        libmksdmsi18n_init();
#endif


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

#ifdef TEAMWARE_MAKE_CMN
        g_argc = argc;
        g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
        for (i = 0; i < argc; i++) {
                g_argv[i] = argv[i];
        }
        g_argv[i] = NULL;
#endif /* TEAMWARE_MAKE_CMN */
```

```
        /*
         * Set argv_zero_string to some form of argv[0] for
         * recursive MAKE builds.
         */

        if (*argv[0] == (int) slash_char) {
                /* argv[0] starts with a slash */
                argv_zero_string = strdup(argv[0]);
        } else if (strchr(argv[0], (int) slash_char) == NULL) {
                /* argv[0] contains no slashes */
                argv_zero_string = strdup(argv[0]);
        } else {
                /*
                 * argv[0] contains at least one slash,
                 * but doesn't start with a slash
                 */
                char    *tmp_current_path;
                char    *tmp_string;

                tmp_current_path = get_current_path();
                tmp_string = getmem(strlen(tmp_current_path) + 1 +
                                    strlen(argv[0]) + 1);
                (void) sprintf(tmp_string,
                               "%s/%s",
                               tmp_current_path,
                               argv[0]);
                argv_zero_string = strdup(tmp_string);
                retmem_mb(tmp_string);
        }

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


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

        /* Sun OS make standart */
        svr4 = false;
        posix = false;
        if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
                svr4 = false;
                posix = true;
        } else {
                prognameptr = strrchr(argv[0], '/');
                if(prognameptr) {
                        prognameptr++;
                } else {
                        prognameptr = argv[0];
                }
                if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
                        svr4 = true;
                        posix = false;
                }
        }
```

```
        if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
           svr4 = true;
           posix = false;
        }

        /*
         * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
         */
        char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
        if (dmake_compat_mode_var != NULL) {
                if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
                        gnu_style = true;
                }
                //svr4 = false;
                //posix = false;
        }

        /*
         * Temporary directory set up.
         */
        char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
        if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
                strcpy(mbs_buffer, tmpdir_var);
                for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
                        *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
                        *tmpdir_var = '\0');
                if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
                        sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
                                mbs_buffer, getpid());
                        int fd = mkstemp(mbs_buffer2);
                        if (fd >= 0) {
                                close(fd);
                                unlink(mbs_buffer2);
                                tmpdir = strdup(mbs_buffer);
                        }
                }
        }

        /* find out if stdout and stderr point to the same place */
        if (fstat(1, &out_stat) < 0) {
                fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
        }
        if (fstat(2, &err_stat) < 0) {
                fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
        }
        if ((out_stat.st_dev == err_stat.st_dev) &&
            (out_stat.st_ino == err_stat.st_ino)) {
                stdout_stderr_same = true;
        } else {
                stdout_stderr_same = false;
        }
        /* Make the vroot package scan the path using shell semantics */
        set_path_style(0);

        setup_char_semantics();

        setup_for_projectdir();

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

        load_cached_names();
```

```
/*
 *      Set command line flags
 */
        setup_makeflags_argv();
        read_command_options(mf_argc, mf_argv);
        read_command_options(argc, argv);
        if (debug_level > 0) {
                cp = getenv(makeflags->string_mb);
                (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
        }

        setup_interrupt(handle_interrupt);

        read_files_and_state(argc, argv);

#ifdef TEAMWARE_MAKE_CMN
        /*
         * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
         */
        MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
        dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
        prop2 = get_prop(dmake_name2->prop, macro_prop);
        if (prop2 == NULL) {
                /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
                output_mode = txt1_mode;
        } else {
                dmake_value2 = prop2->body.macro.value;
                if ((dmake_value2 == NULL) ||
                    (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
                        output_mode = txt1_mode;
                } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
                        output_mode = txt2_mode;
                } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))
                        output_mode = html1_mode;
                } else {
                        warning(catgets(catd, 1, 352, "Unsupported value ‘%s’ fo
                              dmake_value2->string_mb);
                }
        }
        /*
         * Find the dmake_mode: distributed, parallel, or serial.
         */
    if ((!pmake_cap_r_specified) &&
        (!pmake_machinesfile_specified)) {
        MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
        dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
        prop2 = get_prop(dmake_name2->prop, macro_prop);
        if (prop2 == NULL) {
                /* DMAKE_MODE not defined, default to distributed mode */
                dmake_mode_type = distributed_mode;
                no_parallel = false;
        } else {
                dmake_value2 = prop2->body.macro.value;
                if ((dmake_value2 == NULL) ||
                    (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed"))
                        dmake_mode_type = distributed_mode;
                        no_parallel = false;
                } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel
                        dmake_mode_type = parallel_mode;
                        no_parallel = false;
                } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
                } else {
                        fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
                }
```

```
        }

        if ((!list_all_targets) &&
            (report_dependencies_level == 0)) {
                /*
                 * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
                 * They could be defined in the env, in the makefile, or on the
                 * command line.
                 * If neither is defined, and $(HOME)/.dmakerc does not exists,
                 * then print a message, and default to parallel mode.
                 */
                if(dmake_mode_type == distributed_mode) {
                        dmake_mode_type = parallel_mode;
                        no_parallel = false;
                }
        }
    }
#endif

#ifdef TEAMWARE_MAKE_CMN
        parallel_flag = true;
        putenv(strdup(NOCATGETS("DMAKE_CHILD=TRUE")));

//
// If dmake is running with -t option, set dmake_mode_type to serial.
// This is done because doname() calls touch_command() that runs serially.
// If we do not do that, maketool will have problems.
//
        if(touch) {
                dmake_mode_type = serial_mode;
                no_parallel = true;
        }
#else
        parallel_flag = false;
#endif

#if defined (TEAMWARE_MAKE_CMN)
        /*
         * Check whether stdout and stderr are physically same.
         * This is in order to decide whether we need to redirect
         * stderr separately from stdout.
         * This check is performed only if __DMAKE_SEPARATE_STDERR
         * is not set. This variable may be used in order to preserve
         * the ’old’ behaviour.
         */
        out_err_same = true;
        char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
        if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
                struct stat stdout_stat;
                struct stat stderr_stat;
                if( (fstat(1, &stdout_stat) == 0)
                 && (fstat(2, &stderr_stat) == 0) )
                {
                        if( (stdout_stat.st_dev != stderr_stat.st_dev)
                         || (stdout_stat.st_ino != stderr_stat.st_ino) )
                        {
                                out_err_same = false;
                        }
                }
        }
#endif


/*
 *      Enable interrupt handler for alarms
 */
```

```
        (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

/*
 *      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.
 */
        if (keep_state) {
                maybe_append_prop(sunpro_dependencies, macro_prop)->
                  body.macro.exported = true;
        } 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);
                }
        }


        report_dir_enter_leave(true);

        make_targets(argc, argv, parallel_flag);

        report_dir_enter_leave(false);

        if (build_failed_ever_seen) {
                if (posix) {
                        exit_status = 1;
                }
                exit(1);
        }
        exit_status = 0;
        exit(0);
        /* NOTREACHED */
}
```
______***unchanged_portion_omitted_***
```
#endif

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

                MBSTOWCS(wcs_buffer, NOCATGETS(".FAILED"));
                failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
                if ((exit_status != 0) && (failed_name->prop != NULL)) {
#ifdef TEAMWARE_MAKE_CMN
                        /*
                         * [tolik] switch DMake to serial mode
                         */
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
#endif
                        (void) doname(failed_name, false, true);
                } else {
                    if (!trace_status) {
#ifdef TEAMWARE_MAKE_CMN
                        /*
                         * Switch DMake to serial mode
                         */
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
#endif
                        (void) doname(done, false, true);
                    }
                }
            }
        }
        /*
         * Remove the temp file utilities report dependencies thru if it
         * is still around
         */
        if (temp_file_name != NULL) {
                (void) unlink(temp_file_name->string_mb);
        }
        /*
         * Do not save the current command in .make.state if make
         * was interrupted.
         */
        if (current_line != NULL) {
                command_changed = true;
                current_line->body.line.command_used = NULL;
        }
        /*
         * For each parallel build process running, remove the temp files
         * and zap the command line so it won’t be put in .make.state
         */
        for (rp = running_list; rp != NULL; rp = rp->next) {
                if (rp->temp_file != NULL) {
                        (void) unlink(rp->temp_file->string_mb);
                }
                if (rp->stdout_file != NULL) {
                        (void) unlink(rp->stdout_file);
                        retmem_mb(rp->stdout_file);
                        rp->stdout_file = NULL;
                }
                if (rp->stderr_file != NULL) {
                        (void) unlink(rp->stderr_file);
                        retmem_mb(rp->stderr_file);
                        rp->stderr_file = NULL;
                }
                command_changed = true;
/*
                line = get_prop(rp->target->prop, line_prop);
                if (line != NULL) {
                        line->body.line.command_used = NULL;
                }
 */
        }
        /* Remove the statefile lock file if the file has been locked */
        if ((make_state_lockfile != NULL) && (make_state_locked)) {
                (void) unlink(make_state_lockfile);
                make_state_lockfile = NULL;
                make_state_locked = false;
        }
        /* Write .make.state */
        write_state_file(1, (Boolean) 1);

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

#ifdef TEAMWARE_MAKE_CMN
        catclose(catd);
#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);
        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;
        }
        if (!do_not_exec_rule &&
            !touch &&
            !quest &&
            (current_target != NULL) &&
            !(current_target->stat.is_precious || all_precious)) {

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

                if (exists(current_target) != file_doesnt_exist) {
                        (void) fprintf(stderr,
                                       "\n*** %s ",
                                       current_target->string_mb);
                        if (current_target->stat.is_dir) {
                                (void) fprintf(stderr,
                                               catgets(catd, 1, 168, "not remove
                                               current_target->string_mb);
                        } else if (unlink(current_target->string_mb) == 0) {
                                (void) fprintf(stderr,
                                               catgets(catd, 1, 169, "removed.\n
                                               current_target->string_mb);
                        } else {
                                (void) fprintf(stderr,
                                               catgets(catd, 1, 170, "could not
                                               current_target->string_mb,
                                               errmsg(errno));
                        }
                }
        }
        for (rp = running_list; rp != NULL; rp = rp->next) {
                if (rp->state != build_running) {
                        continue;
                }
                if (rp->target->is_member &&
                    ((member = get_prop(rp->target->prop, member_prop)) !=
                     NULL)) {
                        rp->target = member->body.member.library;
                }
                if (!do_not_exec_rule &&
                    !touch &&
                    !quest &&
                    !(rp->target->stat.is_precious || all_precious)) {

                        rp->target->stat.time = file_no_time;
                        if (exists(rp->target) != file_doesnt_exist) {
                                (void) fprintf(stderr,
                                               "\n*** %s ",
                                               rp->target->string_mb);
                                if (rp->target->stat.is_dir) {
                                        (void) fprintf(stderr,
                                                       catgets(catd, 1, 171, "no
                                                       rp->target->string_mb);
                                } else if (unlink(rp->target->string_mb) == 0) {
                                        (void) fprintf(stderr,
                                                       catgets(catd, 1, 172, "re
                                                       rp->target->string_mb);
                                } else {
                                        (void) fprintf(stderr,
                                                       catgets(catd, 1, 173, "co
                                                       rp->target->string_mb,
                                                       errmsg(errno));
                                }
                        }
                }
        }


        /* Have we locked .make.state or .nse_depinfo? */
        if ((make_state_lockfile != NULL) && (make_state_locked)) {
                unlink(make_state_lockfile);
                make_state_lockfile = NULL;
                make_state_locked = false;
        }
        /*
         * Re-read .make.state file (it might be changed by recursive make)
         */
        check_state(NULL);

        report_dir_enter_leave(false);

        exit_status = 2;
        exit(2);
}
```

______unchanged_portion_omitted_

```
/*
 *      read_command_options(argc, argv)
 *
 *      Scan the cmd line options and process the ones that start with "-"
 *
 *      Return value:
 *                              -M argument, if any
```

```
 *
 *      Parameters:
 *              argc            You know what this is
 *              argv            You know what this is
 *
 *      Global variables used:
 */
static void
read_command_options(register int argc, register char **argv)
{
        register int            ch;
        int                     current_optind = 1;
        int                     last_optind_with_double_hyphen = 0;
        int                     last_optind;
        int                     last_current_optind;
        register int            i;
        register int            j;
        register int            k;
        register int            makefile_next = 0; /*
                                                    * flag to note options:
                                                    * -c, f, g, j, m, o
                                                    */
        const char              *tptr;
        const char              *CMD_OPTS;

        extern char             *optarg;
        extern int              optind, opterr, optopt;

#define SUNPRO_CMD_OPTS "-~Bbc:Ddef:g:ij:K:kM:m:NnO:o:PpqRrSsTtuVvwx:"

#ifdef TEAMWARE_MAKE_CMN
#       define SVR4_CMD_OPTS   "-c:ef:g:ij:km:nO:o:pqrsTtVv"
#else
#       define SVR4_CMD_OPTS   "-ef:iknpqrstV"
#endif

        /*
         * Added V in SVR4_CMD_OPTS also, which is going to be a hidden
         * option, just to make sure that the getopt doesn't fail when some
         * users leave their USE_SVR4_MAKE set and try to use the makefiles
         * that are designed to issue commands like $(MAKE) -V. Anyway it
         * sets the same flag but ensures that getopt doesn't fail.
         */

        opterr = 0;
        optind = 1;
        while (1) {
                last_optind=optind;                     /* Save optind and curre
                last_current_optind=current_optind;     /* in case we have to re
                if (svr4) {
                        CMD_OPTS=SVR4_CMD_OPTS;
                        ch = getopt(argc, argv, SVR4_CMD_OPTS);
                } else {
                        CMD_OPTS=SUNPRO_CMD_OPTS;
                        ch = getopt(argc, argv, SUNPRO_CMD_OPTS);
                }
                if (ch == EOF) {
                        if(optind < argc) {
                                /*
                                 * Fixing bug 4102537:
                                 *    Strange behaviour of command make using --
                                 * Not all argv have been processed
                                 * Skip non-flag argv and continue processing.
                                 */
                                optind++;
                                current_optind++;
                                continue;
                        } else {
                                break;
                        }

                }
                if (ch == ’?’) {
                        if (optopt == ’-’) {
                                /* Bug 5060758: getopt() changed behavior (s10_6
                                 * and now we have to deal with cases when optio
                                 * with double hyphen appear here, from -$(MAKEF
                                 */
                                i = current_optind;
                                if (argv[i][0] == ’-’) {
                                  if (argv[i][1] == ’-’) {
                                    if (argv[i][2] != ’\0’) {
                                      /* Check if this option is allowed */
                                      tptr = strchr(CMD_OPTS, argv[i][2]);
                                      if (tptr) {
                                        if (last_optind_with_double_hyphen != cu
                                          /* This is first time we are trying to
                                           * problem with this option. If we com
                                           * time, we will go to fatal error.
                                           */
                                          last_optind_with_double_hyphen = curre

                                          /* Eliminate first hyphen character */
                                          for (j=0; argv[i][j] != ’\0’; j++) {
                                            argv[i][j] = argv[i][j+1];
                                          }

                                          /* Repeat the processing of this argum
                                          optind=last_optind;
                                          current_optind=last_current_optind;
                                          continue;
                                        }
                                      }
                                    }
                                  }
                                }
                        }
                }

                if (ch == ’?’) {
                        if (svr4) {
#ifdef TEAMWARE_MAKE_CMN
                                fprintf(stderr,
                                        catgets(catd, 1, 267, "Usage : dmake [ -
                                fprintf(stderr,
                                        catgets(catd, 1, 268, "              [ -
                                fprintf(stderr,
                                        catgets(catd, 1, 269, "              [ -
#else
                                fprintf(stderr,
                                        catgets(catd, 1, 270, "Usage : make [ -f
                                fprintf(stderr,
                                        catgets(catd, 1, 271, "             [ -s
#endif
                                tptr = strchr(SVR4_CMD_OPTS, optopt);
                        } else {
#ifdef TEAMWARE_MAKE_CMN
                                fprintf(stderr,
                                        catgets(catd, 1, 272, "Usage : dmake [ -
                                fprintf(stderr,
                                        catgets(catd, 1, 273, "              [ -
                                fprintf(stderr,
```

```
                                        catgets(catd, 1, 274, "              [ -
                                fprintf(stderr,
                                        catgets(catd, 1, 275, "              [ -
#else
                                fprintf(stderr,
                                        catgets(catd, 1, 276, "Usage : make [ -f
                                fprintf(stderr,
                                        catgets(catd, 1, 277, "             [ -e
                                fprintf(stderr,
                                        catgets(catd, 1, 278, "             [ -u
#endif
                                tptr = strchr(SUNPRO_CMD_OPTS, optopt);
                        }
                        if (!tptr) {
                                fatal(catgets(catd, 1, 279, "Unknown option ‘-%c
                        } else {
                                fatal(catgets(catd, 1, 280, "Missing argument af
                        }
                }



                makefile_next |= parse_command_option(ch);
                /*
                 * If we’re done processing all of the options of
                 * ONE argument string...
                 */
                if (current_optind < optind) {
                        i = current_optind;
                        k = 0;
                        /* If there’s an argument for an option... */
                        if ((optind - current_optind) > 1) {
                                k = i + 1;
                        }
                        switch (makefile_next) {
                        case 0:
                                argv[i] = NULL;
                                /* This shouldn’t happen */
                                if (k) {
                                        argv[k] = NULL;
                                }
                                break;
                        case 1: /* -f seen */
                                argv[i] = (char *)NOCATGETS("-f");
                                break;
                        case 2: /* -c seen */
                                argv[i] = (char *)NOCATGETS("-c");
#ifndef TEAMWARE_MAKE_CMN
                                warning(catgets(catd, 1, 281, "Ignoring Distribu
#endif
                                break;
                        case 4: /* -g seen */
                                argv[i] = (char *)NOCATGETS("-g");
#ifndef TEAMWARE_MAKE_CMN
                                warning(catgets(catd, 1, 282, "Ignoring Distribu
#endif
                                break;
                        case 8: /* -j seen */
                                argv[i] = (char *)NOCATGETS("-j");
#ifndef TEAMWARE_MAKE_CMN
                                warning(catgets(catd, 1, 283, "Ignoring Distribu
#endif
                                break;
                        case 16: /* -M seen */
                                argv[i] = (char *)NOCATGETS("-M");
#ifndef TEAMWARE_MAKE_CMN
```

```
                                warning(catgets(catd, 1, 284, "Ignoring Parallel
#endif
                                break;
                        case 32: /* -m seen */
                                argv[i] = (char *)NOCATGETS("-m");
#ifndef TEAMWARE_MAKE_CMN
                                warning(catgets(catd, 1, 285, "Ignoring Distribu
#endif
                                break;
                        case 128: /* -O seen */
                                argv[i] = (char *)NOCATGETS("-O");
                                break;
                        case 256: /* -K seen */
                                argv[i] = (char *)NOCATGETS("-K");
                                break;
                        case 512:       /* -o seen */
                                argv[i] = (char *)NOCATGETS("-o");
#ifndef TEAMWARE_MAKE_CMN
                                warning(catgets(catd, 1, 311, "Ignoring Distribu
#endif
                                break;
                        case 1024: /* -x seen */
                                argv[i] = (char *)NOCATGETS("-x");
#ifndef TEAMWARE_MAKE_CMN
                                warning(catgets(catd, 1, 353, "Ignoring Distribu
#endif
                                break;
                        default: /* > 1 of -c, f, g, j, K, M, m, O, o, x seen */
                                fatal(catgets(catd, 1, 286, "Illegal command lin
                        }

                        makefile_next = 0;
                        current_optind = optind;
                }
        }
}
______unchanged_portion_omitted_

/*
 *      parse_command_option(ch)
 *
 *      Parse make command line options.
 *
 *      Return value:
 *                              Indicates if any -f -c or -M were seen
 *
 *      Parameters:
 *              ch              The character to parse
 *
 *      Static variables used:
 *              dmake_group_specified   Set for make -g
 *              dmake_max_jobs_specified        Set for make -j
 *              dmake_mode_specified    Set for make -m
 *              dmake_add_mode_specified        Set for make -x
 *              dmake_compat_mode_specified     Set for make -x SUN_MAKE_COMPAT_
 *              dmake_output_mode_specified     Set for make -x DMAKE_OUTPUT_MOD
 *              dmake_odir_specified    Set for make -o
 *              dmake_rcfile_specified  Set for make -c
 *              env_wins                Set for make -e
 *              ignore_default_mk       Set for make -r
 *              trace_status            Set for make -p
 *
 *      Global variables used:
 *              .make.state path & name set for make -K
 *              continue_after_error    Set for make -k
 *              debug_level             Set for make -d
```

```
 *              do_not_exec_rule        Set for make -n
 *              filter_stderr           Set for make -X
 *              ignore_errors_all       Set for make -i
 *              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;
        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;
        case ’O’:                       /* Send job start & result msgs */
                if (invert_this) {
                        send_mtool_msgs = false;
                } else {
                }
                return 128;
```

```
        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:
                break;
        }
        return 0;
}
______unchanged_portion_omitted_

/*
 *      make_targets(argc, argv, parallel_flag)
 *
 *      Call doname on the specified targets
 *
```

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

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

        }
#endif
        for (i = 1; i < argc; i++) {
                if ((cp = argv[i]) != NULL) {
                        target_to_make_found = true;
                        if ((cp[0] == (int) period_char) &&
                            (cp[1] == (int) slash_char)) {
                                cp += 2;
                        }
                                 if((cp[0] == (int) ' ') &&
                                    (cp[1] == (int) '-') &&
                                    (cp[2] == (int) ' ') &&
                                    (cp[3] == (int) '-')) {
                                    argv[i] = NULL;
                                        continue;
                                }
                        MBSTOWCS(wcs_buffer, cp);
                        default_target_to_build = normalize_name(wcs_buffer, wsl
                        top_level_target = get_wstring(default_target_to_build->
                        report_recursion(default_target_to_build);
                        commands_done = false;
                        if (parallel) {
                                result = (Doname) default_target_to_build->state
```

```
                        } else {
                                result = doname_check(default_target_to_build,
                                                      true,
                                                      false,
                                                      false);
                        }
                        gather_recursive_deps();
                        if (build_failed_seen) {
                                build_failed_ever_seen = true;
                                warning(catgets(catd, 1, 200, "Target '%s' not r
                                        default_target_to_build->string_mb);
                        }
                        build_failed_seen = false;
                        if (report_dependencies_level > 0) {
                                print_dependencies(default_target_to_build,
                                                   get_prop(default_target_to_bu
                                                            line_prop));
                        }
                        default_target_to_build->stat.time =
                          file_no_time;
                        if (default_target_to_build->colon_splits > 0) {
                                default_target_to_build->state =
                                  build_dont_know;
                        }
                        if (!parallel &&
                            /* !commands_done && */
                            (result == build_ok) &&
                            !quest &&
                            (report_dependencies_level == 0) /*  &&
                            (exists(default_target_to_build) > file_doesnt_exist
                                if (posix) {
                                        if (!commands_done) {
                                                (void) printf(catgets(catd, 1, 2
                                                              default_target_to_
                                        } else {
                                                if (no_action_was_taken) {
                                                        (void) printf(catgets(ca
                                                                      default_ta
                                                }
                                        }
                                } else {
                                        if (!commands_done &&
                                            (exists(default_target_to_build) > f
                                                (void) printf(catgets(catd, 1, 2
                                                              default_target_to_
                                        }
                                }
                        }
                }
        }

/*
 *      If no file arguments have been encountered,
 *      make the first name encountered that doesnt start with a dot
 */
        if (!target_to_make_found) {
                if (default_target_to_build == NULL) {
                        fatal(catgets(catd, 1, 202, "No arguments to build"));
                }
                commands_done = false;
                top_level_target = get_wstring(default_target_to_build->string_m
                report_recursion(default_target_to_build);


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


#ifdef TEAMWARE_MAKE_CMN
                result = doname_parallel(default_target_to_build, true, false);
#else
                result = doname_check(default_target_to_build, true,
                                      false, false);
#endif
                gather_recursive_deps();
                if (build_failed_seen) {
                        build_failed_ever_seen = true;
                        warning(catgets(catd, 1, 203, "Target ‘%s’ not remade be
                                default_target_to_build->string_mb);
                }
                build_failed_seen = false;
                if (report_dependencies_level > 0) {
                        print_dependencies(default_target_to_build,
                                           get_prop(default_target_to_build->
                                                    prop,
                                                    line_prop));
                }
                default_target_to_build->stat.time = file_no_time;
                if (default_target_to_build->colon_splits > 0) {
                        default_target_to_build->state = build_dont_know;
                }
                if (/* !commands_done && */
                    (result == build_ok) &&
                    !quest &&
                    (report_dependencies_level == 0) /*  &&
                    (exists(default_target_to_build) > file_doesnt_exist)  */) {
                        if (posix) {
                                if (!commands_done) {
                                        (void) printf(catgets(catd, 1, 299, "‘%s
                                                      default_target_to_build->s
                                } else {
                                        if (no_action_was_taken) {
                                                (void) printf(catgets(catd, 1, 3
                                                              default_target_to_
                                        }
                                }
                        } else {
                                if (!commands_done &&
                                    (exists(default_target_to_build) > file_does
                                        (void) printf(catgets(catd, 1, 301, "‘%s
                                                      default_target_to_build->s
                                }
                        }
                }
        }
}
```
______***unchanged_portion_omitted_***
```
#endif


static void
report_dir_enter_leave(Boolean entering)
{
        char    rcwd[MAXPATHLEN];
static  char *  mlev = NULL;
        char *  make_level_str = NULL;
        int     make_level_val = 0;

        make_level_str = getenv(NOCATGETS("MAKELEVEL"));
        if(make_level_str) {
```

```
                make_level_val = atoi(make_level_str);
        }
        if(mlev == NULL) {
                mlev = (char*) malloc(MAXPATHLEN);
        }
        if(entering) {
                sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val + 1);
        } else {
                make_level_val--;
                sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val);
        }
        putenv(mlev);

        if(report_cwd) {
                if(make_level_val <= 0) {
                        if(entering) {
#ifdef TEAMWARE_MAKE_CMN
                                sprintf( rcwd
                                       , catgets(catd, 1, 329, "dmake: Entering
                                       , get_current_path());
#else
                                sprintf( rcwd
                                       , catgets(catd, 1, 330, "make: Entering d
                                       , get_current_path());
#endif
                        } else {
#ifdef TEAMWARE_MAKE_CMN
                                sprintf( rcwd
                                       , catgets(catd, 1, 331, "dmake: Leaving d
                                       , get_current_path());
#else
                                sprintf( rcwd
                                       , catgets(catd, 1, 332, "make: Leaving di
                                       , get_current_path());
#endif
                        }
                } else {
                        if(entering) {
#ifdef TEAMWARE_MAKE_CMN
                                sprintf( rcwd
                                       , catgets(catd, 1, 333, "dmake[%d]: Enter
                                       , make_level_val, get_current_path());
#else
                                sprintf( rcwd
                                       , catgets(catd, 1, 334, "make[%d]: Enteri
                                       , make_level_val, get_current_path());
#endif
                        } else {
#ifdef TEAMWARE_MAKE_CMN
                                sprintf( rcwd
                                       , catgets(catd, 1, 335, "dmake[%d]: Leavi
                                       , make_level_val, get_current_path());
#else
                                sprintf( rcwd
                                       , catgets(catd, 1, 336, "make[%d]: Leavin
                                       , make_level_val, get_current_path());
#endif
                        }
                }
                printf(NOCATGETS("%s"), rcwd);
        }
}
```
______***unchanged_portion_omitted_***

****
25535 Wed May 20 11:58:29 2015
new/usr/src/cmd/make/bin/misc.cc
make: unifdef for TEAMWARE_MAKE_CMN (defined)
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      misc.cc
 *
 *      This file contains various unclassified routines. Some main groups:
 *              getname
 *              Memory allocation
 *              String handling
 *              Property handling
 *              Error message handling
 *              Make internal state dumping
 *              main routine support
 */

/*
 * Included files
 */
#include <errno.h>
#include <mk/defs.h>
#include <mksh/macro.h>         /* SETVAR() */
#include <mksh/misc.h>          /* enable_interrupt() */
#include <stdarg.h>             /* va_list, va_start(), va_end() */
#include <vroot/report.h>       /* SUNPRO_DEPENDENCIES */


#ifdef TEAMWARE_MAKE_CMN
#define MAXJOBS_ADJUST_RFE4694000

#ifdef MAXJOBS_ADJUST_RFE4694000
extern void job_adjust_fini();
#endif /* MAXJOBS_ADJUST_RFE4694000 */
#endif /* TEAMWARE_MAKE_CMN */


/*
 * Defined macros
 */
```

```
/*
 * typedefs & structs
 */

/*
 * Static variables
 */

/*
 * File table of contents
 */
static  void            print_rule(register Name target);
static  void            print_target_n_deps(register Name target);

/*****************************************
 *
 *      getname
 */

/*****************************************
 *
 *      Memory allocation
 */

/*
 *      free_chain()
 *
 *      frees a chain of Name_vector's
 *
 *      Parameters:
 *              ptr             Pointer to the first element in the chain
 *                              to be freed.
 *
 *      Global variables used:
 */
void
free_chain(Name_vector ptr)
{
        if (ptr != NULL) {
                if (ptr->next != NULL) {
                        free_chain(ptr->next);
                }
                free((char *) ptr);
        }
}

/*****************************************
 *
 *      String manipulation
 */

/*****************************************
 *
 *      Nameblock property handling
 */

/*****************************************
 *
 *      Error message handling
 */

/*
 *      fatal(format, args...)
 *
 *      Print a message and die
 *
 *      Parameters:
 *              format          printf type format string
 *              args            Arguments to match the format
 *
 *      Global variables used:
 *              fatal_in_progress Indicates if this is a recursive call
 *              parallel_process_cnt Do we need to wait for anything?
 *              report_pwd      Should we report the current path?
 */
/*VARARGS*/
void
fatal(const char *message, ...)
{
        va_list args;

        va_start(args, message);
        (void) fflush(stdout);
        (void) fprintf(stderr, catgets(catd, 1, 263, "make: Fatal error: "));
        (void) vfprintf(stderr, message, args);
        (void) fprintf(stderr, "\n");
        va_end(args);
        if (report_pwd) {
                (void) fprintf(stderr,
                               catgets(catd, 1, 156, "Current working directory
                               get_current_path());
        }
        (void) fflush(stderr);
        if (fatal_in_progress) {
                exit_status = 1;
                exit(1);
        }
        fatal_in_progress = true;
#ifdef TEAMWARE_MAKE_CMN
        /* Let all parallel children finish */
        if ((dmake_mode_type == parallel_mode) &&
            (parallel_process_cnt > 0)) {
                (void) fprintf(stderr,
                               catgets(catd, 1, 157, "Waiting for %d %s to finis
                               parallel_process_cnt,
                               parallel_process_cnt == 1 ?
                               catgets(catd, 1, 158, "job") : catgets(catd, 1, 1
                (void) fflush(stderr);
        }

        while (parallel_process_cnt > 0) {
                await_parallel(true);
                finish_children(false);
        }
#endif

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

        exit_status = 1;
        exit(1);
}
```

______unchanged_portion_omitted_

```
**********************************************************
   46703 Wed May 20 11:58:29 2015
new/usr/src/cmd/make/bin/parallel.cc
make: unifdef for TEAMWARE_MAKE_CMN (defined)
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

#ifdef TEAMWARE_MAKE_CMN

/*
 *      parallel.cc
 *
 *      Deal with the parallel processing
 */

/*
 * Included files
 */
#include <errno.h>              /* errno */
#include <fcntl.h>
#include <mk/defs.h>
#include <mksh/dosys.h>         /* redirect_io() */
#include <mksh/macro.h>         /* expand_value() */
#include <mksh/misc.h>          /* getmem() */
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netdb.h>



/*
 * Defined macros
 */
#define MAXRULES                100

/*
 * This const should be in avo_dms/include/AvoDmakeCommand.h
 */
const int local_host_mask = 0x20;
```

```
/*
 * typedefs & structs
 */


/*
 * Static variables
 */
#ifdef TEAMWARE_MAKE_CMN
static  Boolean         just_did_subtree = false;
static  char            local_host[MAXNAMELEN] = "";
static  char            user_name[MAXNAMELEN] = "";
#endif
static  int             pmake_max_jobs = 0;
static  pid_t           process_running = -1;
static  Running         *running_tail = &running_list;
static  Name            subtree_conflict;
static  Name            subtree_conflict2;


/*
 * File table of contents
 */
static  void            delete_running_struct(Running rp);
static  Boolean         dependency_conflict(Name target);
static  Doname          distribute_process(char **commands, Property line);
static  void            doname_subtree(Name target, Boolean do_get, Boolean impl
static  void            dump_out_file(char *filename, Boolean err);
static  void            finish_doname(Running rp);
static  void            maybe_reread_make_state(void);
static  void            process_next(void);
static  void            reset_conditionals(int cnt, Name *targets, Property *loc
static  pid_t           run_rule_commands(char *host, char **commands);
static  Property        *set_conditionals(int cnt, Name *targets);
static  void            store_conditionals(Running rp);


/*
 *      execute_parallel(line, waitflg)
 *
 *      DMake 2.x:
 *      parallel mode: spawns a parallel process to execute the command group.
 *      distributed mode: sends the command group down the pipe to rxm.
 *
 *      Return value:
 *                              The result of the execution
 *
 *      Parameters:
 *              line            The command group to execute
 */
Doname
execute_parallel(Property line, Boolean waitflg, Boolean local)
{
        int                     argcnt;
        int                     cmd_options = 0;
        char                    *commands[MAXRULES + 5];
        char                    *cp;
        Name                    dmake_name;
        Name                    dmake_value;
        int                     ignore;
        Name                    make_machines_name;
        char                    **p;
        Property                prop;
        Doname                  result = build_ok;
        Cmd_line                rule;
        Boolean                 silent_flag;
        Name                    target = line->body.line.target;
        Boolean                 wrote_state_file = false;

        if ((pmake_max_jobs == 0) &&
            (dmake_mode_type == parallel_mode)) {
                if (local_host[0] == '\0') {
                        (void) gethostname(local_host, MAXNAMELEN);
                }
                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
                dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
                if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
                    ((dmake_value = prop->body.macro.value) != NULL)) {
                        pmake_max_jobs = atoi(dmake_value->string_mb);
                        if (pmake_max_jobs <= 0) {
                                warning(catgets(catd, 1, 308, "DMAKE_MAX_JOBS ca
                                warning(catgets(catd, 1, 309, "setting DMAKE_MAX
                                pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
                        }
                } else {
                        /*
                         * For backwards compatibility w/ PMake 1.x, when
                         * DMake 2.x is being run in parallel mode, DMake
                         * should parse the PMake startup file
                         * $(HOME)/.make.machines to get the pmake_max_jobs.
                         */
                        MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
                        dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
                        if (((prop = get_prop(dmake_name->prop, macro_prop)) !=
                            ((dmake_value = prop->body.macro.value) != NULL)) {
                                make_machines_name = dmake_value;
                        } else {
                                make_machines_name = NULL;
                        }
                        if ((pmake_max_jobs = read_make_machines(make_machines_n
                                pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
                        }
                }
        }

        if ((dmake_mode_type == serial_mode) ||
            ((dmake_mode_type == parallel_mode) && (waitflg))) {
                return (execute_serial(line));
        }

        {
                p = commands;
        }

        argcnt = 0;
        for (rule = line->body.line.command_used;
             rule != NULL;
             rule = rule->next) {
                if (posix && (touch || quest) && !rule->always_exec) {
                        continue;
                }
                if (vpath_defined) {
                        rule->command_line =
                          vpath_translation(rule->command_line);
                }
                if (dmake_mode_type == distributed_mode) {
                        cmd_options = 0;
                        if(local) {
                                cmd_options |= local_host_mask;
                        }
```

```
                } else {
                        silent_flag = false;
                        ignore = 0;
                }
                if (rule->command_line->hash.length > 0) {
                        if (++argcnt == MAXRULES) {
                                if (dmake_mode_type == distributed_mode) {
                                        /* XXX - tell rxm to execute on local ho
                                        /* I WAS HERE!!! */
                                } else {
                                        /* Too many rules, run serially instead.
                                        return build_serial;
                                }
                        }
                        {
                                if (rule->silent && !silent) {
                                        silent_flag = true;
                                }
                                if (rule->ignore_error) {
                                        ignore++;
                                }
                                /* XXX - need to add support for + prefix */
                                if (silent_flag || ignore) {
                                        *p = getmem((silent_flag ? 1 : 0) +
                                                    ignore +
                                                    (strlen(rule->
                                                           command_line->
                                                           string_mb)) +
                                                    1);
                                        cp = *p++;
                                        if (silent_flag) {
                                                *cp++ = (int) at_char;
                                        }
                                        if (ignore) {
                                                *cp++ = (int) hyphen_char;
                                        }
                                        (void) strcpy(cp, rule->command_line->st
                                } else {
                                        *p++ = rule->command_line->string_mb;
                                }
                        }
                }
        }
        if ((argcnt == 0) ||
            (report_dependencies_level > 0)) {
                return build_ok;
        }
        {
                *p = NULL;

                Doname res = distribute_process(commands, line);
                if (res == build_running) {
                        parallel_process_cnt++;
                }

                /*
                 * Return only those memory that were specially allocated
                 * for part of commands.
                 */
                for (int i = 0; commands[i] != NULL; i++) {
                        if ((commands[i][0] == (int) at_char) ||
                            (commands[i][0] == (int) hyphen_char)) {
                                retmem_mb(commands[i]);
                        }
                }
                return res;
        }
}


#ifdef TEAMWARE_MAKE_CMN
#define MAXJOBS_ADJUST_RFE4694000

#ifdef MAXJOBS_ADJUST_RFE4694000

#include <unistd.h>     /* sysconf(_SC_NPROCESSORS_ONLN) */
#include <sys/ipc.h>            /* ftok() */
#include <sys/shm.h>            /* shmget(), shmat(), shmdt(), shmctl() */
#include <semaphore.h>          /* sem_init(), sem_trywait(), sem_post(), sem_de
#include <sys/loadavg.h>        /* getloadavg() */

/*
 *      adjust_pmake_max_jobs (int pmake_max_jobs)
 *
 *      Parameters:
 *              pmake_max_jobs  - max jobs limit set by user
 *
 *      External functions used:
 *              sysconf()
 *              getloadavg()
 */
static int
adjust_pmake_max_jobs (int pmake_max_jobs)
{
        static int      ncpu = 0;
        double          loadavg[3];
        int             adjustment;
        int             adjusted_max_jobs;

        if (ncpu <= 0) {
                if ((ncpu = sysconf(_SC_NPROCESSORS_ONLN)) <= 0) {
                        ncpu = 1;
                }
        }
        if (getloadavg(loadavg, 3) != 3) return(pmake_max_jobs);
        adjustment = ((int)loadavg[LOADAVG_1MIN]);
        if (adjustment < 2) return(pmake_max_jobs);
        if (ncpu > 1) {
                adjustment = adjustment / ncpu;
        }
        adjusted_max_jobs = pmake_max_jobs - adjustment;
        if (adjusted_max_jobs < 1) adjusted_max_jobs = 1;
        return(adjusted_max_jobs);
}
```

______***unchanged_portion_omitted_***

```
#endif /* MAXJOBS_ADJUST_RFE4694000 */
#endif /* TEAMWARE_MAKE_CMN */

/*
 *      distribute_process(char **commands, Property line)
 *
 *      Parameters:
 *              commands        argv vector of commands to execute
 *
 *      Return value:
 *                              The result of the execution
 *
 *      Static variables used:
 *              process_running Set to the pid of the process set running
 * #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
 *              job_adjust_mode Current job adjust mode
```

```
 * #endif
 */
static Doname
distribute_process(char **commands, Property line)
{
        static unsigned file_number = 0;
        wchar_t         string[MAXPATHLEN];
        char            mbstring[MAXPATHLEN];
        int             filed;
        int             res;
        int             tmp_index;
        char            *tmp_index_str_ptr;

#if !defined (TEAMWARE_MAKE_CMN) || !defined (MAXJOBS_ADJUST_RFE4694000)
        while (parallel_process_cnt >= pmake_max_jobs) {
                await_parallel(false);
                finish_children(true);
        }
#else /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
        /* initialize adjust mode, if not initialized */
        if (job_adjust_mode == ADJUST_UNKNOWN) {
                job_adjust_init();
        }

        /* actions depend on adjust mode */
        switch (job_adjust_mode) {
        case ADJUST_M1:
                while (parallel_process_cnt >= adjust_pmake_max_jobs (pmake_max_
                        await_parallel(false);
                        finish_children(true);
                }
                break;
        case ADJUST_M2:
                if ((res = m2_acquire_job()) == 0) {
                        if (parallel_process_cnt > 0) {
                                await_parallel(false);
                                finish_children(true);

                                if ((res = m2_acquire_job()) == 0) {
                                        return build_serial;
                                }
                        } else {
                                return build_serial;
                        }
                }
                if (res < 0) {
                        /* job adjustment error */
                        job_adjust_error();

                        /* no adjustment */
                        while (parallel_process_cnt >= pmake_max_jobs) {
                                await_parallel(false);
                                finish_children(true);
                        }
                }
                break;
        default:
                while (parallel_process_cnt >= pmake_max_jobs) {
                        await_parallel(false);
                        finish_children(true);
                }
        }
#endif /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
        setvar_envvar();
        /*
         * Tell the user what DMake is doing.
         */
        if (!silent && output_mode != txt2_mode) {
                /*
                 * Print local_host --> x job(s).
                 */
                (void) fprintf(stdout,
                               catgets(catd, 1, 325, "%s --> %d %s\n"),
                               local_host,
                               parallel_process_cnt + 1,
                               (parallel_process_cnt == 0) ? catgets(catd, 1, 12

                /* Print command line(s). */
                tmp_index = 0;
                while (commands[tmp_index] != NULL) {
                    /* No @ char. */
                    /* XXX - need to add [2] when + prefix is added */
                    if ((commands[tmp_index][0] != (int) at_char) &&
                        (commands[tmp_index][1] != (int) at_char)) {
                        tmp_index_str_ptr = commands[tmp_index];
                        if (*tmp_index_str_ptr == (int) hyphen_char) {
                                tmp_index_str_ptr++;
                        }
                        (void) fprintf(stdout, "%s\n", tmp_index_str_ptr);
                    }
                    tmp_index++;
                }
                (void) fflush(stdout);
        }

        (void) sprintf(mbstring,
                        NOCATGETS("%s/dmake.stdout.%d.%d.XXXXXX"),
                        tmpdir,
                        getpid(),
                        file_number++);

        mktemp(mbstring);

        stdout_file = strdup(mbstring);
        stderr_file = NULL;

#if defined (TEAMWARE_MAKE_CMN)
        if (!out_err_same) {
                (void) sprintf(mbstring,
                                NOCATGETS("%s/dmake.stderr.%d.%d.XXXXXX"),
                                tmpdir,
                                getpid(),
                                file_number++);

                mktemp(mbstring);

                stderr_file = strdup(mbstring);
        }
#endif

        process_running = run_rule_commands(local_host, commands);

        return build_running;
}
```

______***unchanged_portion_omitted_***

```
#endif
```

```
**********************************************************
   11199 Wed May 20 11:58:30 2015
new/usr/src/cmd/make/bin/pmake.cc
make: unifdef for TEAMWARE_MAKE_CMN (defined)
**********************************************************
/*
 * 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.
 */

#ifdef TEAMWARE_MAKE_CMN

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

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */

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

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


        MBSTOWCS(wcs_buffer, NOCATGETS("MAKE_MACHINES"));
        MAKE_MACHINES = GETNAME(wcs_buffer, FIND_LENGTH);
        /* Did the user specify a .make.machines file on the command line? */
        default_make_machines = false;
        if (make_machines_name == NULL) {
                /* Try reading the default .make.machines file, in $(HOME). */
                homedir = getenv(NOCATGETS("HOME"));
                if ((homedir != NULL) && (strlen(homedir) < (sizeof(mb_make_mach
                        sprintf(mb_make_machines_path,
                         NOCATGETS("%s/.make.machines"), homedir);
                        MBSTOWCS(make_machines_path, mb_make_machines_path);
                        make_machines_name = GETNAME(make_machines_path, FIND_LE
                        default_make_machines = true;
                }
                if (make_machines_name == NULL) {
                        /*
                         * No $(HOME)/.make.machines file.
                         * Return 0 for PMake max jobs.
                         */
                        return(0);
                }
        }
/*
        make_machines_list_mb = getenv(MAKE_MACHINES->string_mb);
 */
        /* Open the .make.machines file. */
        if ((make_machines_file = fopen(make_machines_name->string_mb, "r")) ==
                if (!default_make_machines) {
                        /* Error opening .make.machines file. */
                        fatal(catgets(catd, 1, 314, "Open of %s failed: %s"),
                              make_machines_name->string_mb,
                              errmsg(errno));
                } else {
                        /*
                         * No $(HOME)/.make.machines file.
                         * Return 0 for PMake max jobs.
                         */
                        return(0);
                }
        /* Stat the .make.machines file to get the size of the file.  */
        } else if (fstat(fileno(make_machines_file), &make_machines_buf) < 0) {
                /* Error stat'ing .make.machines file. */
                fatal(catgets(catd, 1, 315, "Stat of %s failed: %s"),
                      make_machines_name->string_mb,
                      errmsg(errno));
        } else {
                /* Allocate memory for "MAKE_MACHINES=<contents of .m.m>" */
                make_machines_list_mb =
                  (char *) getmem((int) (strlen(MAKE_MACHINES->string_mb) +
                                         2 +
                                         make_machines_buf.st_size));
                sprintf(make_machines_list_mb,
                        "%s=",
                        MAKE_MACHINES->string_mb);
                /* Read in the .make.machines file. */
                if (fread(make_machines_list_mb + strlen(MAKE_MACHINES->string_m
                          sizeof(char),
                          (int) make_machines_buf.st_size,
                          make_machines_file) != make_machines_buf.st_size) {
                        /*
                         * Error reading .make.machines file.
                         * Return 0 for PMake max jobs.
                         */
                        warning(catgets(catd, 1, 316, "Unable to read %s"),
                                make_machines_name->string_mb);
                        (void) fclose(make_machines_file);
                        retmem_mb((caddr_t) make_machines_list_mb);
                        return(0);
                } else {
                        (void) fclose(make_machines_file);
                        /* putenv "MAKE_MACHINES=<contents of .m.m>" */
                        *(make_machines_list_mb +
                          strlen(MAKE_MACHINES->string_mb) +
                          1 +
                          make_machines_buf.st_size) = (int) nul_char;
                        if (putenv(make_machines_list_mb) != 0) {
                                warning(catgets(catd, 1, 317, "Couldn't put cont
                                        make_machines_name->string_mb);
                        } else {
                                make_machines_list_mb += strlen(MAKE_MACHINES->s
                                make_machines_list = ALLOC_WC(strlen(make_machin
                                (void) mbstowcs(make_machines_list,
                                                make_machines_list_mb,
                                                (strlen(make_machines_list_mb) +
                        }
                }
        }

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

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

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

                /*
                 * If we haven't reached the end of file, process the
                 * machine name.
                 */
                if (*ms) {
                        /*
                         * If invalid machine name decrement counter
                         * and skip line.
                         */
                        mp = ms;
                        SKIPWORD(ms);
                        c = *ms;
                        *ms++ = '\0'; /* Append null to machine name. */
                        /*
                         * If this was the beginning of a comment
                         * (we overwrote a # sign) and it's not
                         * end of line yet, shift the # sign.
                         */
                        if ((c == '#') && (*ms != '\n') && (*ms)) {
                                *ms = '#';
                        }
                        WCSTOMBS(mbs_buffer, mp);
                        /*
                         * Print "Ignoring unknown host" if:
                         * 1) hostname is longer than MAX_HOSTNAMELEN, or
                         * 2) hostname is unknown
                         */
                        if ((wslen(mp) > MAX_HOSTNAMELEN) ||
                            ((hp = gethostbyname(mbs_buffer)) == NULL)) {
                                warning(catgets(catd, 1, 318, "Ignoring unknown
                                        mbs_buffer);
                                SKIPTOEND(ms);
                                /* Increment ptr if not end of file. */
                                if (*ms) {
                                        ms++;
                                }
                        } else {
                                /* Compare current hostname with local_host. */
                                if (wslen(mp) == local_host_wslen &&
                                    IS_WEQUALN(mp, local_host, local_host_wslen)
                                        /*
                                         * Bingo, local_host is in .make.machine
                                         * Continue reading.
                                         */
                                        pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
                                /* Compare current hostname with full_host. */
                                } else if (wslen(mp) == full_host_wslen &&
                                                    IS_WEQUALN(mp, full_host, full_host_w
                                                /*
                                                 * Bingo, full_host is in .make.machines
                                                 * Continue reading.
                                                 */
                                                pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
                                        } else {
                                                if (c != '\n') {
                                                    SKIPTOEND(ms);
                                                    if (*ms) {
                                                        ms++;
                                                    }
                                                }
                                                continue;
                                        }
                                        /* If we get here, local_host is in .make.machin
                                        if (c != '\n')  {
                                                /* Now look for keyword 'max'. */
                                                MBSTOWCS(wcs_buffer, NOCATGETS("max"));
                                                SKIPSPACE(ms);
                                                while ((*ms != '\n') && (*ms)) {
                                                        if (*ms == '#') {
                                                                pskip_comment(&ms);
                                                        } else if (IS_WEQUALN(ms, wcs_bu
                                                                /* Skip "max". */
                                                                ms += 3;
                                                                pmake_max_jobs = get_max
                                                                SKIPSPACE(ms);
                                                        } else {
                                                                warning(catgets(catd, 1,
                                                                SKIPTOEND(ms);
                                                                break;
                                                        }
                                                }
                                        }
                                        break; /* out of outermost for() loop. */
                                }
                        }
                }
                retmem(make_machines_list);
                return(pmake_max_jobs);
}
```

______***unchanged_portion_omitted_***

```
#endif
```

****
   14099 Wed May 20 11:58:30 2015
new/usr/src/cmd/make/include/mk/defs.h
make: unifdef for TEAMWARE_MAKE_CMN (defined)
****
______*unchanged_portion_omitted_*

```
struct _Running {
        struct _Running         *next;
        Doname                  state;
        struct _Name            *target;
        struct _Name            *true_target;
        struct _Property        *command;
        struct _Name            *sprodep_value;
        char                    *sprodep_env;
        int                     recursion_level;
        Boolean                 do_get;
        Boolean                 implicit;
        Boolean                 redo;
        int                     auto_count;
        struct _Name            **automatics;
        pid_t                   pid;
#ifdef TEAMWARE_MAKE_CMN
        int                     job_msg_id;
#else
        int                     host;
#endif
        char                    *stdout_file;
        char                    *stderr_file;
        struct _Name            *temp_file;
        int                     conditional_cnt;
        struct _Name            **conditional_targets;
#ifdef TEAMWARE_MAKE_CMN
        Boolean                 make_refd;
#endif
};
```
______*unchanged_portion_omitted_*

```
/*
 * Typedefs for all structs
 */
typedef struct _Cmd_line        *Cmd_line, Cmd_line_rec;
typedef struct _Dependency      *Dependency, Dependency_rec;
typedef struct _Macro           *Macro, Macro_rec;
typedef struct _Name_vector     *Name_vector, Name_vector_rec;
typedef struct _Percent         *Percent, Percent_rec;
typedef struct _Dyntarget       *Dyntarget;
typedef struct _Recursive_make  *Recursive_make, Recursive_make_rec;
typedef struct _Running         *Running, Running_rec;


/*
 *      extern declarations for all global variables.
 *      The actual declarations are in globals.cc
 */
extern  Boolean         allrules_read;
extern  Name            posix_name;
extern  Name            svr4_name;
extern  Boolean         sdot_target;
extern  Boolean         all_parallel;
extern  Boolean         assign_done;
extern  Boolean         build_failed_seen;
extern  Name            built_last_make_run;
extern  Name            c_at;
extern  Boolean         command_changed;
```

```
extern  Boolean         commands_done;
extern  Chain           conditional_targets;
extern  Name            conditionals;
extern  Boolean         continue_after_error;
extern  Property        current_line;
extern  Name            current_make_version;
extern  Name            current_target;
extern  short           debug_level;
extern  Cmd_line        default_rule;
extern  Name            default_rule_name;
extern  Name            default_target_to_build;
extern  Boolean         depinfo_already_read;
extern  Name            dmake_group;
extern  Name            dmake_max_jobs;
extern  Name            dmake_mode;
extern  DMake_mode      dmake_mode_type;
extern  Name            dmake_output_mode;
extern  DMake_output_mode       output_mode;
extern  Name            dmake_odir;
extern  Name            dmake_rcfile;
extern  Name            done;
extern  Name            dot;
extern  Name            dot_keep_state;
extern  Name            dot_keep_state_file;
extern  Name            empty_name;
extern  Boolean         fatal_in_progress;
extern  int             file_number;
extern  Name            force;
extern  Name            ignore_name;
extern  Boolean         ignore_errors;
extern  Boolean         ignore_errors_all;
extern  Name            init;
extern  int             job_msg_id;
extern  Boolean         keep_state;
extern  Name            make_state;
#ifdef TEAMWARE_MAKE_CMN
extern  timestruc_t     make_state_before;
#endif
extern  Boolean         make_state_locked;
extern  Dependency      makefiles_used;
extern  Name            makeflags;
extern  Name            make_version;
extern  char            mbs_buffer2[];
extern  char            *mbs_ptr;
extern  char            *mbs_ptr2;
extern  Boolean         no_action_was_taken;
extern  int             mtool_msgs_fd;
extern  Boolean         no_parallel;
extern  Name            no_parallel_name;
extern  Name            not_auto;
extern  Boolean         only_parallel;
extern  Boolean         parallel;
extern  Name            parallel_name;
extern  Name            localhost_name;
extern  int             parallel_process_cnt;
extern  Percent         percent_list;
extern  Dyntarget       dyntarget_list;
extern  Name            plus;
extern  Name            pmake_machinesfile;
extern  Name            precious;
extern  Name            primary_makefile;
extern  Boolean         quest;
extern  short           read_trace_level;
extern  Boolean         reading_dependencies;
extern  int             recursion_level;
extern  Name            recursive_name;
extern  short           report_dependencies_level;
extern  Boolean         report_pwd;
extern  Boolean         rewrite_statefile;
extern  Running         running_list;
extern  char            *sccs_dir_path;
extern  Name            sccs_get_name;
extern  Name            sccs_get_posix_name;
extern  Cmd_line        sccs_get_rule;
extern  Cmd_line        sccs_get_org_rule;
extern  Cmd_line        sccs_get_posix_rule;
extern  Name            get_name;
extern  Name            get_posix_name;
extern  Cmd_line        get_rule;
extern  Cmd_line        get_posix_rule;
extern  Boolean         send_mtool_msgs;
extern  Boolean         all_precious;
extern  Boolean         report_cwd;
extern  Boolean         silent_all;
extern  Boolean         silent;
extern  Name            silent_name;
extern  char            *stderr_file;
extern  char            *stdout_file;
extern  Boolean         stdout_stderr_same;
extern  Dependency      suffixes;
extern  Name            suffixes_name;
extern  Name            sunpro_dependencies;
extern  Boolean         target_variants;
extern  const char      *tmpdir;
extern  const char      *temp_file_directory;
extern  Name            temp_file_name;
extern  short           temp_file_number;
extern  wchar_t         *top_level_target;
extern  Boolean         touch;
extern  Boolean         trace_reader;
extern  Boolean         build_unconditional;
extern  pathpt          vroot_path;
extern  Name            wait_name;
extern  wchar_t         wcs_buffer2[];
extern  wchar_t         *wcs_ptr;
extern  wchar_t         *wcs_ptr2;
extern  nl_catd         catd;
extern  long int        hostid;

/*
 * Declarations of system defined variables
 */
/* On linux this variable is defined in 'signal.h' */
extern  char            *sys_siglist[];

/*
 * Declarations of system supplied functions
 */
extern  int             file_lock(char *, char *, int *, int);

/*
 * Declarations of functions declared and used by make
 */
extern  void            add_pending(Name target, int recursion_level, Boolean do
extern  void            add_running(Name target, Name true_target, Property comm
extern  void            add_serial(Name target, int recursion_level, Boolean do_
extern  void            add_subtree(Name target, int recursion_level, Boolean do
extern  void            append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar
#ifdef TEAMWARE_MAKE_CMN
extern  void            await_parallel(Boolean waitflg);
#endif
extern  void            build_suffix_list(Name target_suffix);
```

```
extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
extern  void            check_state(Name temp_file_name);
extern  void            cond_macros_into_string(Name np, String_rec *buffer);
extern  void            construct_target_string();
extern  void            create_xdrs_ptr(void);
extern  void            depvar_add_to_list (Name name, Boolean cmdline);
extern  Doname          doname(register Name target, register Boolean do_get, re
extern  Doname          doname_check(register Name target, register Boolean do_g
extern  Doname          doname_parallel(Name target, Boolean do_get, Boolean imp
extern  Doname          dosys(register Name command, register Boolean ignore_err
extern  void            dump_make_state(void);
extern  void            dump_target_list(void);
extern  void            enter_conditional(register Name target, Name name, Name
extern  void            enter_dependencies(register Name target, Chain target_gr
extern  void            enter_dependency(Property line, register Name depe, Bool
extern  void            enter_equal(Name name, Name value, register Boolean appe
extern  Percent         enter_percent(register Name target, Chain target_group,
extern  Dyntarget       enter_dyntarget(register Name target);
extern  Name_vector     enter_name(String string, Boolean tail_present, register
extern  Boolean         exec_vp(register char *name, register char **argv, char
extern  Doname          execute_parallel(Property line, Boolean waitflg, Boolean
extern  Doname          execute_serial(Property line);
extern  timestruc_t&    exists(register Name target);
extern  void            fatal(const char *, ...);
extern  void            fatal_reader(char *, ...);
extern  Doname          find_ar_suffix_rule(register Name target, Name true_targ
extern  Doname          find_double_suffix_rule(register Name target, Property *
extern  Doname          find_percent_rule(register Name target, Property *comman
extern  int             find_run_directory (char *cmd, char *cwd, char *dir, cha
extern  Doname          find_suffix_rule(Name target, Name target_body, Name tar
extern  Chain           find_target_groups(register Name_vector target_list, reg
extern  void            finish_children(Boolean docheck);
extern  void            finish_running(void);
extern  void            free_chain(Name_vector ptr);
extern  void            gather_recursive_deps(void);
extern  char            *get_current_path(void);
extern  int             get_job_msg_id(void);
extern  FILE            *get_mtool_msgs_fp(void);
extern  wchar_t         *getmem_wc(register int size);
/* On linux getwd(char *) is defined in 'unistd.h' */
#ifdef __cplusplus
extern "C" {
#endif
extern  char            *getwd(char *);
#ifdef __cplusplus
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   22598 Wed May 20 11:58:32 2015
new/usr/src/cmd/make/include/mksh/defs.h
make: unifdef for TEAMWARE_MAKE_CMN (defined)
**********************************************************
#ifndef _MKSH_DEFS_H
#define _MKSH_DEFS_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 * This is not "#ifdef TEAMWARE_MAKE_CMN" because we're currently
 * using the TW fake i18n headers and libraries to build both
 * SMake and PMake on SPARC/S1 and x86/S2.
 */

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

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


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

#define false   BOOLEAN_false
#define true    BOOLEAN_true

typedef enum {
        false =         0,
        true =          1,
        failed =        0,
        succeeded =     1
```

```
} Boolean;
______unchanged_portion_omitted_

/*
 *      extern declarations for all global variables.
 *      The actual declarations are in globals.cc
 */
extern char             char_semantics[];
extern wchar_t          char_semantics_char[];
extern Macro_list       cond_macro_list;
extern Boolean          conditional_macro_used;
extern Boolean          do_not_exec_rule;               /* '-n' */
extern Boolean          dollarget_seen;
extern Boolean          dollarless_flag;
extern Name             dollarless_value;
extern char             **environ;
extern Envvar           envvar;
extern int              exit_status;
extern wchar_t          *file_being_read;
/* Variable gnu_style=true if env. var. SUN_MAKE_COMPAT_MODE=GNU (RFE 4866328) *
extern Boolean          gnu_style;
extern Name_set         hashtab;
extern Name             host_arch;
extern Name             host_mach;
extern int              line_number;
extern char             *make_state_lockfile;
extern Boolean          make_word_mentioned;
extern Makefile_type    makefile_type;
extern char             mbs_buffer[];
extern Name             path_name;
extern Boolean          posix;
extern Name             query;
extern Boolean          query_mentioned;
extern Name             hat;
extern Boolean          reading_environment;
extern Name             shell_name;
extern Boolean          svr4;
extern Name             target_arch;
extern Name             target_mach;
extern Boolean          tilde_rule;
extern wchar_t          wcs_buffer[];
extern Boolean          working_on_targets;
extern Name             virtual_root;
extern Boolean          vpath_defined;
extern Name             vpath_name;
extern Boolean          make_state_locked;
#if defined (TEAMWARE_MAKE_CMN)
extern Boolean          out_err_same;
#endif
extern pid_t            childPid;
extern nl_catd          libmksh_catd;

/*
 * RFE 1257407: make does not use fine granularity time info available from stat
 * High resolution time comparison.
 */

inline int
operator==(const timestruc_t &t1, const timestruc_t &t2) {
        return ((t1.tv_sec == t2.tv_sec) && (t1.tv_nsec == t2.tv_nsec));
}
______unchanged_portion_omitted_
```

****
2971 Wed May 20 11:58:32 2015
new/usr/src/cmd/make/lib/mksh/globals.cc
make: unifdef for TEAMWARE_MAKE_CMN (defined)
****
______unchanged_portion_omitted_

```
Macro_list      cond_macro_list;
Boolean         conditional_macro_used;
Boolean         do_not_exec_rule;               /* '-n' */
Boolean         dollarget_seen;
Boolean         dollarless_flag;
Name            dollarless_value;
Envvar          envvar;
#ifdef lint
char            **environ;
#endif
int             exit_status;
wchar_t         *file_being_read;
/* Variable gnu_style=true if env. var. SUN_MAKE_COMPAT_MODE=GNU (RFE 4866328) *
Boolean         gnu_style = false;
Name_set        hashtab;
Name            host_arch;
Name            host_mach;
int             line_number;
char            *make_state_lockfile;
Boolean         make_word_mentioned;
Makefile_type   makefile_type = reading_nothing;
char            mbs_buffer[(MAXPATHLEN * MB_LEN_MAX)];
Name            path_name;
Boolean         posix = true;
Name            hat;
Name            query;
Boolean         query_mentioned;
Boolean         reading_environment;
Name            shell_name;
Boolean         svr4 = false;
Name            target_arch;
Name            target_mach;
Boolean         tilde_rule;
Name            virtual_root;
Boolean         vpath_defined;
Name            vpath_name;
wchar_t         wcs_buffer[MAXPATHLEN];
Boolean         working_on_targets;
#if defined (TEAMWARE_MAKE_CMN)
Boolean         out_err_same;
#endif
pid_t           childPid = -1;  // This variable is used for killing child's pro
                                // Such as qrsh, running command, etc.

/*
 * timestamps defined in defs.h
 */
const timestruc_t file_no_time          = { -1, 0 };
const timestruc_t file_doesnt_exist     = { 0, 0 };
const timestruc_t file_is_dir           = { 1, 0 };
const timestruc_t file_min_time         = { 2, 0 };
const timestruc_t file_max_time         = { INT_MAX, 0 };
```