****
   94913 Wed May 20 12:01:44 2015
new/usr/src/cmd/make/bin/doname.cc
make: restore a couple of blocks of code from DISTRIBUTED that should have been
****
______***unchanged_portion_omitted_***

```
/*
 * 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
check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
{
        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 if ((!parallel_ok(dependency->name, false)) &&
                           (dependencies_running)) {
                        /*
                         * If we can't execute the current dependency in
                         * parallel, hold off the dependency processing
                         * to preserve the order of the dependencies.
                         */
                        break;
#endif /* ! codereview */
                } 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;
                        }
                }
        }
        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;
                }
        }
        /*
         * 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;
}

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

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

                /* The make macro $@ is bound to the target name once per */
                /* invocation of dynamic_dependencies() */
```

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

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

                first_member = NULL;
                /* We use the original dependency cell for the first */
                /* dependency from the expansion */
                reuse_cell = true;
                /* We also have to deal with dependencies that expand to */
                /* lib.a(members) notation */
                for (p = start; *p != (int) nul_char; p++) {
                        if ((*p == (int) parenleft_char)) {
                                lib = GETNAME(start, p - start);
                                lib->is_member = true;
                                first_member = dependency;
                                start = p + 1;
                                while (iswspace(*start)) {
                                        start++;
                                }
                                break;
                        }
                }
                do {
                    /* First skip whitespace */
                        for (p = start; *p != (int) nul_char; p++) {
                                if ((*p == (int) nul_char) ||
                                    iswspace(*p) ||
                                    (*p == (int) parenright_char)) {
                                        break;
                                }
                        }
                        /* Enter dependency from expansion */
                        if (p != start) {
                                /* Create new dependency cell if */
                                /* this is not the first dependency */
                                /* picked from the expansion */
                                if (!reuse_cell) {
                                        new_depe = ALLOC(Dependency);
                                        new_depe->next = dependency->next;
                                        new_depe->automatic = false;
                                        new_depe->stale = false;
                                        new_depe->built = false;
                                        dependency->next = new_depe;
                                        dependency = new_depe;
                                }
                                reuse_cell = false;
                                /* Internalize the dependency name */
                                // tolik. Fix for bug 4110429: inconsistent expa
                                // include "//" and "/./"
                                //dependency->name = GETNAME(start, p - start);
                                dependency->name = normalize_name(start, p - sta
                                if ((debug_level > 0) &&
                                    (first_member == NULL)) {
                                        (void) printf(catgets(catd, 1, 25, "%*sD
                                                      recursion_level,
                                                      "",
                                                      dependency->name->string_m
                                                      true_target->string_mb);
                                }
                                for (start = p; iswspace(*start); start++);
                                p = start;
                        }
                } while ((*p != (int) nul_char) &&
                         (*p != (int) parenright_char));
                /* If the expansion was of lib.a(members) format we now */
                /* enter the proper member cells */
                if (first_member != NULL) {
                        /* Scan the new dependencies and transform them from */
                        /* "foo" to "lib.a(foo)" */
                        for (; 1; first_member = first_member->next) {
                                /* Build "lib.a(foo)" name */
                                INIT_STRING_FROM_STACK(string, buffer);
                                APPEND_NAME(lib,
                                              &string,
                                              (int) lib->hash.length);
                                append_char((int) parenleft_char, &string);
                                APPEND_NAME(first_member->name,
                                              &string,
                                              FIND_LENGTH);
                                append_char((int) parenright_char, &string);
                                member = first_member->name;
                                /* Replace "foo" with "lib.a(foo)" */
                                first_member->name =
                                  GETNAME(string.buffer.start, FIND_LENGTH);
                                if (string.free_after_use) {
                                        retmem(string.buffer.start);
                                }
                                if (debug_level > 0) {
                                        (void) printf(catgets(catd, 1, 26, "%*sD
                                                      recursion_level,
                                                      "",
                                                      first_member->name->
                                                      string_mb,
                                                      true_target->string_mb);
                                }
                                first_member->name->is_member = lib->is_member;
                                /* Add member property to member */
                                prop = maybe_append_prop(first_member->name,
                                                         member_prop);
                                prop->body.member.library = lib;
                                prop->body.member.entry = NULL;
                                prop->body.member.member = member;
                                if (first_member == dependency) {
                                        break;
                                }
                        }
                }
        }
        Wstring wcb;
        /* Then scan all the dependencies again. This time we want to expand */
        /* shell file wildcards */
```

```
        for (remove = &line->body.line.dependencies, dependency = *remove;
             dependency != NULL;
             dependency = *remove) {
                if (dependency->name == NULL) {
                        dependency = *remove = (*remove)->next;
                        continue;
                }
                /* If dependency name string contains shell wildcards */
                /* replace the name with the expansion */
                if (dependency->name->wildcard) {
                        wcb.init(dependency->name);
                        if ((start = (wchar_t *) wschr(wcb.get_string(),
                                                   (int) parenleft_char)) != NULL) {
                                /* lib(*) type pattern */
                                library = buffer;
                                (void) wsncpy(buffer,
                                              wcb.get_string(),
                                              start - wcb.get_string());
                                buffer[start-wcb.get_string()] =
                                  (int) nul_char;
                                (void) wsncpy(pattern,
                                              start + 1,
(int) (dependency->name->hash.length-(start-wcb.get_string())-2));
                                pattern[dependency->name->hash.length -
                                        (start-wcb.get_string()) - 2] =
                                          (int) nul_char;
                        } else {
                                library = NULL;
                                (void) wsncpy(pattern,
                                              wcb.get_string(),
                                              (int) dependency->name->hash.lengt
                                pattern[dependency->name->hash.length] =
                                  (int) nul_char;
                        }
                        start = (wchar_t *) wsrchr(pattern, (int) slash_char);
                        if (start == NULL) {
                                directory = dot;
                                p = pattern;
                        } else {
                                directory = GETNAME(pattern, start-pattern);
                                p = start+1;
                        }
                        /* The expansion is handled by the read_dir() routine*/
                        if (read_dir(directory, p, line, library)) {
                                *remove = (*remove)->next;
                        } else {
                                remove = &dependency->next;
                        }
                } else {
                        remove = &dependency->next;
                }
        }

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

/*
 * 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) {
                        result = execute_parallel(line, true);
                }
                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) {
                        result = execute_parallel(line, true);
                }
        } else {
                /*
                 * If this is not a touch run, we need to execute the
                 * proper command(s) for the target.
                 */
                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) {
                                        result = execute_parallel(line, true, ta
                                } 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) {
                                                result = execute_parallel(line,
                                        } else {
                                                current_target = NULL;
                                                current_line = NULL;
                                                target->parallel = false;
                                                line->body.line.command_used =
                                                                        NULL;
                                                return build_serial;
                                        }
                                }
                        }
                } else {
                        result = execute_parallel(line, true, target->localhost)
                }
        }
        temp_file_name = NULL;
        if (report_dependencies_level == 0){
                update_target(line, result);
        }
        current_target = NULL;
        current_line = NULL;
        return result;
}

/*
 *      execute_serial(line)
 *
 *      Runs thru the command line for the target and
 *      executes the rules one by one.
 *
 *      Return value:
 *                              The result of the command build
```

```
 *
 *      Parameters:
 *              line            The command to execute
 *
 *      Static variables used:
 *
 *      Global variables used:
 *              continue_after_error -k flag
 *              do_not_exec_rule -n flag
 *              report_dependencies -P flag
 *              silent          Don't echo commands before executing
 *              temp_file_name  Temp file for auto dependencies
 *              vpath_defined   If true, translate path for command
 */
Doname
execute_serial(Property line)
{
        int                     child_pid = 0;
        Boolean                 printed_serial;
        Doname                  result = build_ok;
        Cmd_line                rule, cmd_tail, command = NULL;
        char                    mbstring[MAXPATHLEN];
        int                     filed;
        Name                    target = line->body.line.target;

        SEND_MTOOL_MSG(
                if (!sent_rsrc_info_msg) {
                        if (userName[0] == '\0') {
                                avo_get_user(userName, NULL);
                        }
                        if (hostName[0] == '\0') {
                                strcpy(hostName, avo_hostname());
                        }
                        send_rsrc_info_msg(1, hostName, userName);
                        sent_rsrc_info_msg = 1;
                }
                send_job_start_msg(line);
                job_result_msg = new Avo_MToolJobResultMsg();
        );

        target->has_recursive_dependency = false;
        // We have to create a copy of the rules chain for processing because
        // the original one can be destroyed during .make.state file rereading.
        for (rule = line->body.line.command_used;
             rule != NULL;
             rule = rule->next) {
                if (command == NULL) {
                        command = cmd_tail = ALLOC(Cmd_line);
                } else {
                        cmd_tail->next = ALLOC(Cmd_line);
                        cmd_tail = cmd_tail->next;
                }
                *cmd_tail = *rule;
        }
        if (command) {
                cmd_tail->next = NULL;
        }
        for (rule = command; rule != NULL; rule = rule->next) {
                if (posix && (touch || quest) && !rule->always_exec) {
                        continue;
                }
                if (vpath_defined) {
                        rule->command_line =
                          vpath_translation(rule->command_line);
                }
                /* Echo command line, maybe. */
                if ((rule->command_line->hash.length > 0) &&
                    !silent &&
                    (!rule->silent || do_not_exec_rule) &&
                    (report_dependencies_level == 0)) {
                        (void) printf("%s\n", rule->command_line->string_mb);
                        SEND_MTOOL_MSG(
                                job_result_msg->appendOutput(AVO_STRDUP(rule->co
                        );
                }
                if (rule->command_line->hash.length > 0) {
                        SEND_MTOOL_MSG(
                                (void) sprintf(mbstring,
                                                NOCATGETS("%s/make.stdout.%d.%d.
                                                tmpdir,
                                                getpid(),
                                                file_number++);

                                int tmp_fd = mkstemp(mbstring);
                                if(tmp_fd) {
                                        (void) close(tmp_fd);
                                }

                                stdout_file = strdup(mbstring);
                                stderr_file = NULL;
                                child_pid = pollResults(stdout_file,
                                                        (char *)NULL,
                                                        (char *)NULL);
                        );
                        /* Do assignment if command line prefixed with "=" */
                        if (rule->assign) {
                                result = build_ok;
                                do_assign(rule->command_line, target);
                        } else if (report_dependencies_level == 0) {
                                /* Execute command line. */
                                setvar_envvar();
                                result = dosys(rule->command_line,
                                               (Boolean) rule->ignore_error,
                                               (Boolean) rule->make_refd,
                                               /* ds 98.04.23 bug #4085164. make
                                               false,
                                               /* BOOLEAN(rule->silent &&
                                                       rule->ignore_error), */
                                               (Boolean) rule->always_exec,
                                               target,
                                               send_mtool_msgs);
                                check_state(temp_file_name);
                        }
                        SEND_MTOOL_MSG(
                                append_job_result_msg(job_result_msg);
                                if (child_pid > 0) {
                                        kill(child_pid, SIGUSR1);
                                        while (!((waitpid(child_pid, 0, 0) == -1
                                                && (errno == ECHILD)));
                                }
                                child_pid = 0;
                                (void) unlink(stdout_file);
                                retmem_mb(stdout_file);
                                stdout_file = NULL;
                        );
                } else {
                        result = build_ok;
                }
                if (result == build_failed) {
                        if (silent || rule->silent) {
                                (void) printf(catgets(catd, 1, 242, "The followi
                                              rule->command_line->string_mb);
```

```
                                SEND_MTOOL_MSG(
                                        job_result_msg->appendOutput(AVO_STRDUP(
                                        job_result_msg->appendOutput(AVO_STRDUP(
                                );
                        }
                        if (!rule->ignore_error && !ignore_errors) {
                                if (!continue_after_error) {
                                        SEND_MTOOL_MSG(
                                                job_result_msg->setResult(job_ms
                                                xdr_msg = (RWCollectable*)
                                                        job_result_msg;
                                                xdr(&xdrs, xdr_msg);
                                                (void) fflush(mtool_msgs_fp);
                                                delete job_result_msg;
                                        );
                                        fatal(catgets(catd, 1, 244, "Command fai
                                              target->string_mb);
                                }
                                /*
                                 * Make sure a failing command is not
                                 * saved in .make.state.
                                 */
                                line->body.line.command_used = NULL;
                                break;
                        } else {
                                result = build_ok;
                        }
                }
        }
        for (rule = command; rule != NULL; rule = cmd_tail) {
                cmd_tail = rule->next;
                free(rule);
        }
        command = NULL;
        SEND_MTOOL_MSG(
                job_result_msg->setResult(job_msg_id, (result == build_ok) ? 0 :
                xdr_msg = (RWCollectable*) job_result_msg;
                xdr(&xdrs, xdr_msg);
                (void) fflush(mtool_msgs_fp);

                delete job_result_msg;
        );
        if (temp_file_name != NULL) {
                free_name(temp_file_name);
        }
        temp_file_name = NULL;

        Property spro = get_prop(sunpro_dependencies->prop, macro_prop);
        if(spro != NULL) {
                Name val = spro->body.macro.value;
                if(val != NULL) {
                        free_name(val);
                        spro->body.macro.value = NULL;
                }
        }
        spro = get_prop(sunpro_dependencies->prop, env_mem_prop);
        if(spro) {
                char *val = spro->body.env_mem.value;
                if(val != NULL) {
                        /*
                         * Do not return memory allocated for SUNPRO_DEPENDENCIE
                         * It will be returned in setvar_daemon() in macro.cc
                         */
                        //      retmem_mb(val);
                        spro->body.env_mem.value = NULL;
                }
        }

        return result;
}



/*
 *      vpath_translation(cmd)
 *
 *      Translates one command line by
 *      checking each word. If the word has an alias it is translated.
 *
 *      Return value:
 *                              The translated command
 *
 *      Parameters:
 *              cmd             Command to translate
 *
 *      Global variables used:
 */
Name
vpath_translation(register Name cmd)
{
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        String_rec              new_cmd;
        wchar_t                 *p;
        wchar_t                 *start;

        if (!vpath_defined || (cmd == NULL) || (cmd->hash.length == 0)) {
                return cmd;
        }
        INIT_STRING_FROM_STACK(new_cmd, buffer);

        Wstring wcb(cmd);
        p = wcb.get_string();

        while (*p != (int) nul_char) {
                while (iswspace(*p) && (*p != (int) nul_char)) {
                        append_char(*p++, &new_cmd);
                }
                start = p;
                while (!iswspace(*p) && (*p != (int) nul_char)) {
                        p++;
                }
                cmd = GETNAME(start, p - start);
                if (cmd->has_vpath_alias_prop) {
                        cmd = get_prop(cmd->prop, vpath_alias_prop)->
                                                body.vpath_alias.alias;
                        APPEND_NAME(cmd,
                                        &new_cmd,
                                        (int) cmd->hash.length);
                } else {
                        append_string(start, &new_cmd, p - start);
                }
        }
        cmd = GETNAME(new_cmd.buffer.start, FIND_LENGTH);
        if (new_cmd.free_after_use) {
                retmem(new_cmd.buffer.start);
        }
        return cmd;
}

/*
 *      check_state(temp_file_name)
 *
```

```
 *      Reads and checks the state changed by the previously executed command.
 *
 *      Parameters:
 *              temp_file_name  The auto dependency temp file
 *
 *      Global variables used:
 */
void
check_state(Name temp_file_name)
{
        if (!keep_state) {
                return;
        }

        /*
         * Then read the temp file that now might
         * contain dependency reports from utilities
         */
        read_dependency_file(temp_file_name);

        /*
         * And reread .make.state if it
         * changed (the command ran recursive makes)
         */
        check_read_state_file();
        if (temp_file_name != NULL) {
                (void) unlink(temp_file_name->string_mb);
        }
}

/*
 *      read_dependency_file(filename)
 *
 *      Read the temp file used for reporting dependencies to make
 *
 *      Parameters:
 *              filename        The name of the file with the state info
 *
 *      Global variables used:
 *              makefile_type   The type of makefile being read
 *              read_trace_level Debug flag
 *              temp_file_number The always increasing number for unique files
 *              trace_reader    Debug flag
 */
static void
read_dependency_file(register Name filename)
{
        register Makefile_type  save_makefile_type;

        if (filename == NULL) {
                return;
        }
        filename->stat.time = file_no_time;
        if (exists(filename) > file_doesnt_exist) {
                save_makefile_type = makefile_type;
                makefile_type = reading_cpp_file;
                if (read_trace_level > 1) {
                        trace_reader = true;
                }
                temp_file_number++;
                (void) read_simple_file(filename,
                                        false,
                                        false,
                                        false,
                                        false,
                                        false,
                                        false);
                trace_reader = false;
                makefile_type = save_makefile_type;
        }
}

/*
 *      check_read_state_file()
 *
 *      Check if .make.state has changed
 *      If it has we reread it
 *
 *      Parameters:
 *
 *      Global variables used:
 *              make_state      Make state file name
 *              makefile_type   Type of makefile being read
 *              read_trace_level Debug flag
 *              trace_reader    Debug flag
 */
static void
check_read_state_file(void)
{
        timestruc_t             previous = make_state->stat.time;
        register Makefile_type  save_makefile_type;
        register Property       makefile;

        make_state->stat.time = file_no_time;
        if ((exists(make_state) == file_doesnt_exist) ||
            (make_state->stat.time == previous)) {
                return;
        }
        save_makefile_type = makefile_type;
        makefile_type = rereading_statefile;
        /* Make sure we clear the old cached contents of .make.state */
        makefile = maybe_append_prop(make_state, makefile_prop);
        if (makefile->body.makefile.contents != NULL) {
                retmem(makefile->body.makefile.contents);
                makefile->body.makefile.contents = NULL;
        }
        if (read_trace_level > 1) {
                trace_reader = true;
        }
        temp_file_number++;
        (void) read_simple_file(make_state,
                                false,
                                false,
                                false,
                                false,
                                false,
                                true);
        trace_reader = false;
        makefile_type = save_makefile_type;
}

/*
 *      do_assign(line, target)
 *
 *      Handles runtime assignments for command lines prefixed with "=".
 *
 *      Parameters:
 *              line            The command that contains an assignment
 *              target          The Name of the target, used for error reports
 *
 *      Global variables used:
 *              assign_done     Set to indicate doname needs to reprocess
```

```
 */
static void
do_assign(register Name line, register Name target)
{
        Wstring wcb(line);
        register wchar_t        *string = wcb.get_string();
        register wchar_t        *equal;
        register Name           name;
        register Boolean        append = false;

        /*
         * If any runtime assignments are done, doname() must reprocess all
         * targets in the future since the macro values used to build the
         * command lines for the targets might have changed.
         */
        assign_done = true;
        /* Skip white space. */
        while (iswspace(*string)) {
                string++;
        }
        equal = string;
        /* Find "+=" or "=". */
        while (!iswspace(*equal) &&
               (*equal != (int) plus_char) &&
               (*equal != (int) equal_char)) {
                equal++;
        }
        /* Internalize macro name. */
        name = GETNAME(string, equal - string);
        /* Skip over "+=" "=". */
        while (!((*equal == (int) nul_char) ||
                 (*equal == (int) equal_char) ||
                 (*equal == (int) plus_char))) {
                equal++;
        }
        switch (*equal) {
        case nul_char:
                fatal(catgets(catd, 1, 31, "= expected in rule ‘%s’ for target ‘
                      line->string_mb,
                      target->string_mb);
        case plus_char:
                append = true;
                equal++;
                break;
        }
        equal++;
        /* Skip over whitespace in front of value. */
        while (iswspace(*equal)) {
                equal++;
        }
        /* Enter new macro value. */
        enter_equal(name,
                    GETNAME(equal, wcb.get_string() + line->hash.length - equal)
                    append);
}

/*
 *      build_command_strings(target, line)
 *
 *      Builds the command string to used when
 *      building a target. If the string is different from the previous one
 *      is_out_of_date is set.
 *
 *      Parameters:
 *              target          Target to build commands for
 *              line            Where to stuff result
 *
 *      Global variables used:
 *              c_at            The Name "@", used to set macro value
 *              command_changed Set if command is different from old
 *              debug_level     Should we trace activities?
 *              do_not_exec_rule Always echo when running -n
 *              empty_name      The Name "", used for empty rule
 *              funny           Semantics of characters
 *              ignore_errors   Used to init field for line
 *              is_conditional  Set to false befor evaling macro, checked
 *                              after expanding macros
 *              keep_state      Indicates that .KEEP_STATE is on
 *              make_word_mentioned Set by macro eval, inits field for cmd
 *              query           The Name "?", used to set macro value
 *              query_mentioned Set by macro eval, inits field for cmd
 *              recursion_level Used for tracing
 *              silent          Used to init field for line
 */
static void
build_command_strings(Name target, register Property line)
{
        String_rec              command_line;
        register Cmd_line       command_template = line->body.line.command_templ
        register Cmd_line       *insert = &line->body.line.command_used;
        register Cmd_line       used = *insert;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        wchar_t                 *start;
        Name                    new_command_line;
        register Boolean        new_command_longer = false;
        register Boolean        ignore_all_command_dependency = true;
        Property                member;
        static Name             less_name;
        static Name             percent_name;
        static Name             star;
        Name                    tmp_name;

        if (less_name == NULL) {
                MBSTOWCS(wcs_buffer, "<");
                less_name = GETNAME(wcs_buffer, FIND_LENGTH);
                MBSTOWCS(wcs_buffer, "%");
                percent_name = GETNAME(wcs_buffer, FIND_LENGTH);
                MBSTOWCS(wcs_buffer, "*");
                star = GETNAME(wcs_buffer, FIND_LENGTH);
        }

        /* We have to check if a target depends on conditional macros */
        /* Targets that do must be reprocessed by doname() each time around */
        /* since the macro values used when building the target might have */
        /* changed */
        conditional_macro_used = false;
        /* If we are building a lib.a(member) target $@ should be bound */
        /* to lib.a */
        if (target->is_member &&
            ((member = get_prop(target->prop, member_prop)) != NULL)) {
                target = member->body.member.library;
        }
        /* If we are building a "::" help target $@ should be bound to */
        /* the real target name */
        /* A lib.a(member) target is never :: */
        if (target->has_target_prop) {
                target = get_prop(target->prop, target_prop)->
                  body.target.target;
        }
        /* Bind the magic macros that make supplies */
        tmp_name = target;
        if(tmp_name != NULL) {
```

```
                if (tmp_name->has_vpath_alias_prop) {
                        tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
                                        body.vpath_alias.alias;
                }
        }
        (void) SETVAR(c_at, tmp_name, false);

        tmp_name = line->body.line.star;
        if(tmp_name != NULL) {
                if (tmp_name->has_vpath_alias_prop) {
                        tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
                                        body.vpath_alias.alias;
                }
        }
        (void) SETVAR(star, tmp_name, false);

        tmp_name = line->body.line.less;
        if(tmp_name != NULL) {
                if (tmp_name->has_vpath_alias_prop) {
                        tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
                                        body.vpath_alias.alias;
                }
        }
        (void) SETVAR(less_name, tmp_name, false);

        tmp_name = line->body.line.percent;
        if(tmp_name != NULL) {
                if (tmp_name->has_vpath_alias_prop) {
                        tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
                                        body.vpath_alias.alias;
                }
        }
        (void) SETVAR(percent_name, tmp_name, false);

        /* $? is seldom used and it is expensive to build */
        /* so we store the list form and build the string on demand */
        Chain query_list = NULL;
        Chain *query_list_tail = &query_list;

        for (Chain ch = line->body.line.query; ch != NULL; ch = ch->next) {
                *query_list_tail = ALLOC(Chain);
                (*query_list_tail)->name = ch->name;
                if ((*query_list_tail)->name->has_vpath_alias_prop) {
                        (*query_list_tail)->name =
                                get_prop((*query_list_tail)->name->prop,
                                        vpath_alias_prop)->body.vpath_alias.alia
                }
                (*query_list_tail)->next = NULL;
                query_list_tail = &(*query_list_tail)->next;
        }
        (void) setvar_daemon(query,
                             (Name) query_list,
                             false,
                             chain_daemon,
                             false,
                             debug_level);

        /* build $^ */
        Chain hat_list = NULL;
        Chain *hat_list_tail = &hat_list;

        for (Dependency dependency = line->body.line.dependencies;
                dependency != NULL;
                dependency = dependency->next) {
                /* skip automatic dependencies */
                if (!dependency->automatic) {
                        if ((dependency->name != force) &&
                                (dependency->stale == false)) {
                                *hat_list_tail = ALLOC(Chain);

                                if (dependency->name->is_member &&
                                        (get_prop(dependency->name->prop, member
                                        (*hat_list_tail)->name =
                                                        get_prop(dependency->nam
                                                                member_prop)->bo
                                } else {
                                        (*hat_list_tail)->name = dependency->nam
                                }

                                if((*hat_list_tail)->name != NULL) {
                                        if ((*hat_list_tail)->name->has_vpath_al
                                                (*hat_list_tail)->name =
                                                        get_prop((*hat_list_tail
                                                                vpath_alias_prop
                                        }
                                }

                                (*hat_list_tail)->next = NULL;
                                hat_list_tail = &(*hat_list_tail)->next;
                        }
                }
        }
        (void) setvar_daemon(hat,
                             (Name) hat_list,
                             false,
                             chain_daemon,
                             false,
                             debug_level);

/* We have two command sequences we need to handle */
/* The old one that we probably read from .make.state */
/* and the new one we are building that will replace the old one */
/* Even when KEEP_STATE is not on we build a new command sequence and store */
/* it in the line prop. This command sequence is then executed by */
/* run_command(). If KEEP_STATE is on it is also later written to */
/* .make.state. The routine replaces the old command line by line with the */
/* new one trying to reuse Cmd_lines */

        /* If there is no old command_used we have to start creating */
        /* Cmd_lines to keep the new cmd in */
        if (used == NULL) {
                new_command_longer = true;
                *insert = used = ALLOC(Cmd_line);
                used->next = NULL;
                used->command_line = NULL;
                insert = &used->next;
        }
        /* Run thru the template for the new command and build the expanded */
        /* new command lines */
        for (;
             command_template != NULL;
             command_template = command_template->next, insert = &used->next, us
                /* If there is no old command_used Cmd_line we need to */
                /* create one and say that cmd consistency failed */
                if (used == NULL) {
                        new_command_longer = true;
                        *insert = used = ALLOC(Cmd_line);
                        used->next = NULL;
                        used->command_line = empty_name;
                }
                /* Prepare the Cmd_line for the processing */
                /* The command line prefixes "@-=?" are stripped and that */
```

```
                /* information is saved in the Cmd_line */
                used->assign = false;
                used->ignore_error = ignore_errors;
                used->silent = silent;
                used->always_exec = false;
                /* Expand the macros in the command line */
                INIT_STRING_FROM_STACK(command_line, buffer);
                make_word_mentioned =
                  query_mentioned =
                    false;
                expand_value(command_template->command_line, &command_line, true
                /* If the macro $(MAKE) is mentioned in the command */
                /* "make -n" runs actually execute the command */
                used->make_refd = make_word_mentioned;
                used->ignore_command_dependency = query_mentioned;
                /* Strip the prefixes */
                start = command_line.buffer.start;
                for (;
                     iswspace(*start) ||
                     (get_char_semantics_value(*start) & (int) command_prefix_se
                     start++) {
                        switch (*start) {
                        case question_char:
                                used->ignore_command_dependency = true;
                                break;
                        case exclam_char:
                                used->ignore_command_dependency = false;
                                break;
                        case equal_char:
                                used->assign = true;
                                break;
                        case hyphen_char:
                                used->ignore_error = true;
                                break;
                        case at_char:
                                if (!do_not_exec_rule) {
                                        used->silent = true;
                                }
                                break;
                        case plus_char:
                                if(posix) {
                                  used->always_exec  = true;
                                }
                                break;
                        }
                }
                /* If all command lines of the template are prefixed with "?"*/
                /* the VIRTUAL_ROOT is not used for cmd consistency checks */
                if (!used->ignore_command_dependency) {
                        ignore_all_command_dependency = false;
                }
                /* Internalize the expanded and stripped command line */
                new_command_line = GETNAME(start, FIND_LENGTH);
                if ((used->command_line == NULL) &&
                    (line->body.line.sccs_command)) {
                        used->command_line = new_command_line;
                        new_command_longer = false;
                }
                /* Compare it with the old one for command consistency */
                if (used->command_line != new_command_line) {
                        Name vpath_translated = vpath_translation(new_command_li
                        if (keep_state &&
                            !used->ignore_command_dependency && (vpath_translate
                                if (debug_level > 0) {
                                        if (used->command_line != NULL
                                            && *used->command_line->string_mb !=
                                            '\0') {
                                                (void) printf(catgets(catd, 1, 3
                                                              recursion_level,
                                                              "",
                                                              target->string_mb,
                                                              vpath_translated->
                                                              recursion_level,
                                                              "",
                                                              used->
                                                              command_line->
                                                              string_mb);
                                        } else {
                                                (void) printf(catgets(catd, 1, 3
                                                              recursion_level,
                                                              "",
                                                              target->string_mb,
                                                              vpath_translated->
                                                              recursion_level,
                                                              "");
                                        }
                                }
                                command_changed = true;
                                line->body.line.is_out_of_date = true;
                        }
                        used->command_line = new_command_line;
                }
                if (command_line.free_after_use) {
                        retmem(command_line.buffer.start);
                }
        }
        /* Check if the old command is longer than the new for */
        /* command consistency */
        if (used != NULL) {
                *insert = NULL;
                if (keep_state &&
                    !ignore_all_command_dependency) {
                        if (debug_level > 0) {
                                (void) printf(catgets(catd, 1, 34, "%*sBuilding
                                              recursion_level,
                                              "",
                                              target->string_mb);
                        }
                        command_changed = true;
                        line->body.line.is_out_of_date = true;
                }
        }
        /* Check if the new command is longer than the old command for */
        /* command consistency */
        if (new_command_longer &&
            !ignore_all_command_dependency &&
            keep_state) {
                if (debug_level > 0) {
                        (void) printf(catgets(catd, 1, 35, "%*sBuilding %s becau
                                      recursion_level,
                                      "",
                                      target->string_mb);
                }
                command_changed = true;
                line->body.line.is_out_of_date = true;
        }
        /* Unbind the magic macros */
        (void) SETVAR(c_at, (Name) NULL, false);
        (void) SETVAR(star, (Name) NULL, false);
        (void) SETVAR(less_name, (Name) NULL, false);
        (void) SETVAR(percent_name, (Name) NULL, false);
        (void) SETVAR(query, (Name) NULL, false);
```

```
        if (query_list != NULL) {
                delete_query_chain(query_list);
        }
        (void) SETVAR(hat, (Name) NULL, false);
        if (hat_list != NULL) {
                delete_query_chain(hat_list);
        }

        if (conditional_macro_used) {
                target->conditional_macro_list = cond_macro_list;
                cond_macro_list = NULL;
                target->depends_on_conditional = true;
        }
}

/*
 *      touch_command(line, target, result)
 *
 *      If this is an "make -t" run we do this.
 *      We touch all targets in the target group ("foo + fie:") if any.
 *
 *      Return value:
 *                              Indicates if the command failed or not
 *
 *      Parameters:
 *              line            The command line to update
 *              target          The target we are touching
 *              result          Initial value for the result we return
 *
 *      Global variables used:
 *              do_not_exec_rule Indicates that -n is on
 *              silent          Do not echo commands
 */
static Doname
touch_command(register Property line, register Name target, Doname result)
{
        Name                    name;
        register Chain          target_group;
        String_rec              touch_string;
        wchar_t                 buffer[MAXPATHLEN];
        Name                    touch_cmd;
        Cmd_line                rule;


        SEND_MTOOL_MSG(
                if (!sent_rsrc_info_msg) {
                        if (userName[0] == '\0') {
                                avo_get_user(userName, NULL);
                        }
                        if (hostName[0] == '\0') {
                                strcpy(hostName, avo_hostname());
                        }
                        send_rsrc_info_msg(1, hostName, userName);
                        sent_rsrc_info_msg = 1;
                }
                send_job_start_msg(line);
                job_result_msg = new Avo_MToolJobResultMsg();
        );
        for (name = target, target_group = NULL; name != NULL;) {
                if (!name->is_member) {
                        /*
                         * Build a touch command that can be passed
                         * to dosys(). If KEEP_STATE is on, "make -t"
                         * will save the proper command, not the
                         * "touch" in .make.state.
                         */
                        INIT_STRING_FROM_STACK(touch_string, buffer);
                        MBSTOWCS(wcs_buffer, NOCATGETS("touch "));
                        append_string(wcs_buffer, &touch_string, FIND_LENGTH);
                        touch_cmd = name;
                        if (name->has_vpath_alias_prop) {
                                touch_cmd = get_prop(name->prop,
                                                 vpath_alias_prop)->
                                                   body.vpath_alias.alias;
                        }
                        APPEND_NAME(touch_cmd,
                                      &touch_string,
                                      FIND_LENGTH);
                        touch_cmd = GETNAME(touch_string.buffer.start,
                                            FIND_LENGTH);
                        if (touch_string.free_after_use) {
                                retmem(touch_string.buffer.start);
                        }
                        if (!silent ||
                            do_not_exec_rule &&
                            (target_group == NULL)) {
                                (void) printf("%s\n", touch_cmd->string_mb);
                                SEND_MTOOL_MSG(
                                        job_result_msg->appendOutput(AVO_STRDUP(
                                );
                        }
                        /* Run the touch command, or simulate it */
                        if (!do_not_exec_rule) {

                                SEND_MTOOL_MSG(
                                        (void) sprintf(mbstring,
                                                        NOCATGETS("%s/make.stdou
                                                        tmpdir,
                                                        getpid(),
                                                        file_number++);

                                        int tmp_fd = mkstemp(mbstring);
                                        if(tmp_fd) {
                                                (void) close(tmp_fd);
                                        }

                                        stdout_file = strdup(mbstring);
                                        stderr_file = NULL;
                                        child_pid = pollResults(stdout_file,
                                                                (char *)NULL,
                                                                (char *)NULL);
                                );

                                result = dosys(touch_cmd,
                                               false,
                                               false,
                                               false,
                                               false,
                                               name,
                                               send_mtool_msgs);

                                SEND_MTOOL_MSG(
                                        append_job_result_msg(job_result_msg);
                                        if (child_pid > 0) {
                                                kill(child_pid, SIGUSR1);
                                                while (!((waitpid(child_pid, 0,
                                                        && (errno == ECHILD)));
                                        }
                                        child_pid = 0;
                                        (void) unlink(stdout_file);
                                        retmem_mb(stdout_file);
                                        stdout_file = NULL;
```

```
                                );

                        } else {
                                result = build_ok;
                        }
                } else {
                        result = build_ok;
                }
                if (target_group == NULL) {
                        target_group = line->body.line.target_group;
                } else {
                        target_group = target_group->next;
                }
                if (target_group != NULL) {
                        name = target_group->name;
                } else {
                        name = NULL;
                }
        }
        SEND_MTOOL_MSG(
                job_result_msg->setResult(job_msg_id, (result == build_ok) ? 0 :
                xdr_msg = (RWCollectable*) job_result_msg;
                xdr(&xdrs, xdr_msg);
                (void) fflush(mtool_msgs_fp);
                delete job_result_msg;
        );
        return result;
}

/*
 *      update_target(line, result)
 *
 *      updates the status of a target after executing its commands.
 *
 *      Parameters:
 *              line            The command line block to update
 *              result          Indicates that build is OK so can update
 *
 *      Global variables used:
 *              do_not_exec_rule Indicates that -n is on
 *              touch           Fake the new timestamp if we are just touching
 */
void
update_target(Property line, Doname result)
{
        Name                    target;
        Chain                   target_group;
        Property                line2;
        timestruc_t             old_stat_time;
        Property                member;

        /*
         * [tolik] Additional fix for bug 1063790. It was fixed
         * for serial make long ago, but DMake dumps core when
         * target is a symlink and sccs file is newer then target.
         * In this case, finish_children() calls update_target()
         * with line==NULL.
         */
        if(line == NULL) {
                /* XXX. Should we do anything here? */
                return;
        }

        target = line->body.line.target;

        if ((result == build_ok) && (line->body.line.command_used != NULL)) {
                if (do_not_exec_rule ||
                    touch ||
                    (target->is_member &&
                     (line->body.line.command_template != NULL) &&
                     (line->body.line.command_template->command_line->string_mb[
                     (line->body.line.command_template->next == NULL))) {
                        /* If we are simulating execution we need to fake a */
                        /* new timestamp for the target we didnt build */
                        target->stat.time = file_max_time;
                } else {
                        /*
                         * If we really built the target we read the new
                         * timestamp.
                         * Fix for bug #1110906: if .c file is newer than
                         * the corresponding .o file which is in an archive
                         * file, make will compile the .c file but it won't
                         * update the object in the .a file.
                         */
                        old_stat_time = target->stat.time;
                        target->stat.time = file_no_time;
                        (void) exists(target);
                        if ((target->is_member) &&
                            (target->stat.time == old_stat_time)) {
                                member = get_prop(target->prop, member_prop);
                                if (member != NULL) {
                                        target->stat.time = member->body.member.
                                        target->stat.time.tv_sec++;
                                }
                        }
                }
                /* If the target is part of a group we need to propagate the */
                /* result of the run to all members */
                for (target_group = line->body.line.target_group;
                     target_group != NULL;
                     target_group = target_group->next) {
                        target_group->name->stat.time = target->stat.time;
                        line2 = maybe_append_prop(target_group->name,
                                                  line_prop);
                        line2->body.line.command_used =
                          line->body.line.command_used;
                        line2->body.line.target = target_group->name;
                }
        }
        target->has_built = true;
}

/*
 *      sccs_get(target, command)
 *
 *      Figures out if it possible to sccs get a file
 *      and builds the command to do it if it is.
 *
 *      Return value:
 *                              Indicates if sccs get failed or not
 *
 *      Parameters:
 *              target          Target to get
 *              command         Where to deposit command to use
 *
 *      Global variables used:
 *              debug_level     Should we trace activities?
 *              recursion_level Used for tracing
 *              sccs_get_rule   The rule to used for sccs getting
 */
static Doname
sccs_get(register Name target, register Property *command)
```

```
{
        register int            result;
        char                    link[MAXPATHLEN];
        String_rec              string;
        wchar_t                 name[MAXPATHLEN];
        register wchar_t        *p;
        timestruc_t             sccs_time;
        register Property       line;
        int                     sym_link_depth = 0;

        /* For sccs, we need to chase symlinks. */
        while (target->stat.is_sym_link) {
                if (sym_link_depth++ > 90) {
                        fatal(catgets(catd, 1, 95, "Can't read symbolic link '%s
                              target->string_mb);
                }
                /* Read the value of the link. */
                result = readlink_vroot(target->string_mb,
                                        link,
                                        sizeof(link),
                                        NULL,
                                        VROOT_DEFAULT);
                if (result == -1) {
                        fatal(catgets(catd, 1, 36, "Can't read symbolic link '%s
                              target->string_mb, errmsg(errno));
                }
                link[result] = 0;
                /* Use the value to build the proper filename. */
                INIT_STRING_FROM_STACK(string, name);

                Wstring wcb(target);
                if ((link[0] != slash_char) &&
                    ((p = (wchar_t *) wsrchr(wcb.get_string(), slash_char)) != N
                        append_string(wcb.get_string(), &string, p - wcb.get_str
                }
                append_string(link, &string, result);
                /* Replace the old name with the translated name. */
                target = normalize_name(string.buffer.start, string.text.p - str
                (void) exists(target);
                if (string.free_after_use) {
                        retmem(string.buffer.start);
                }
        }

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

        if ((!target->has_complained &&
            (sccs_time != file_doesnt_exist) &&
            (sccs_get_rule != NULL))) {
                /* only checking */
                if (command == NULL) {
                        return build_ok;
                }
                /*
                 * We provide a command line for the target. The line is a
                 * "sccs get" command from default.mk.
                 */
                line = maybe_append_prop(target, line_prop);
                *command = line;
                if (sccs_time > target->stat.time) {
                        /*
                         * And only if the plain file is out of date do we
                         * request execution of the command.
                         */
                        line->body.line.is_out_of_date = true;
                        if (debug_level > 0) {
                                (void) printf(catgets(catd, 1, 37, "%*sSccs gett
                                              recursion_level,
                                              "",
                                              target->string_mb);
                        }
                }
                line->body.line.sccs_command = true;
                line->body.line.command_template = sccs_get_rule;
                if(!svr4 && (!allrules_read || posix)) {
                   if((target->prop) &&
                      (target->prop->body.sccs.file) &&
                      (target->prop->body.sccs.file->string_mb)) {
                      if((strlen(target->prop->body.sccs.file->string_mb) ==
                        strlen(target->string_mb) + 2) &&
                        (target->prop->body.sccs.file->string_mb[0] == ’s’) &&
                        (target->prop->body.sccs.file->string_mb[1] == ’.’)) {

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

/*
 *      read_directory_of_file(file)
 *
```

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

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

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

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

        /*
         * If the filename contains a "/" we have to extract the path
         * Else the path defaults to ".".
         */
        if (p != NULL) {
                /*
                 * Check some popular directories first to possibly
                 * save time. Compare string length first to gain speed.
                 */
                if ((usr_include->hash.length == length) &&
                    IS_WEQUALN(usr_include_buf,
                               wcb,
                               length)) {
                        directory = usr_include;
                } else if ((usr_include_sys->hash.length == length) &&
                           IS_WEQUALN(usr_include_sys_buf,
                                      wcb,
                                      length)) {
                        directory = usr_include_sys;
                } else {
                        directory = GETNAME(wcb, length);
                }
        }
        (void) read_dir(directory,
                        (wchar_t *) NULL,
                        (Property) NULL,
                        (wchar_t *) NULL);
}

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

        Wstring wcb(target);
        Wstring wcb1;

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

/*
 *      set_locals(target, old_locals)
 *
 *      Sets any conditional macros for the target.
 *      Each target carries a possibly empty set of conditional properties.
 *
 *      Parameters:
 *              target          The target to set conditional macros for
 *              old_locals      Space to store old values in
 *
 *      Global variables used:
 *              debug_level     Should we trace activity?
```

```
 *              is_conditional  We need to preserve this value
 *              recursion_level Used for tracing
 */
void
set_locals(register Name target, register Property old_locals)
{
        register Property       conditional;
        register int            i;
        register Boolean        saved_conditional_macro_used;
        Chain                   cond_name;
        Chain                   cond_chain;

        if (target->dont_activate_cond_values) {
                return;
        }

        saved_conditional_macro_used = conditional_macro_used;

        /* Scan the list of conditional properties and apply each one */
        for (conditional = get_prop(target->prop, conditional_prop), i = 0;
             conditional != NULL;
             conditional = get_prop(conditional->next, conditional_prop),
             i++) {
                /* Save the old value */
                old_locals[i].body.macro =
                  maybe_append_prop(conditional->body.conditional.name,
                                    macro_prop)->body.macro;
                if (debug_level > 1) {
                        (void) printf(catgets(catd, 1, 38, "%*sActivating condit
                                      recursion_level,
                                      "");
                }
                /* Set the conditional value. Macros are expanded when the */
                /* macro is refd as usual */
                if ((conditional->body.conditional.name != virtual_root) ||
                    (conditional->body.conditional.value != virtual_root)) {
                        (void) SETVAR(conditional->body.conditional.name,
                                      conditional->body.conditional.value,
                                      (Boolean) conditional->body.conditional.ap
                }
                cond_name = ALLOC(Chain);
                cond_name->name = conditional->body.conditional.name;
        }
        /* Put this target on the front of the chain of conditional targets */
        cond_chain = ALLOC(Chain);
        cond_chain->name = target;
        cond_chain->next = conditional_targets;
        conditional_targets = cond_chain;
        conditional_macro_used = saved_conditional_macro_used;
}

/*
 *      reset_locals(target, old_locals, conditional, index)
 *
 *      Removes any conditional macros for the target.
 *
 *      Parameters:
 *              target          The target we are retoring values for
 *              old_locals      The values to restore
 *              conditional     The first conditional block for the target
 *              index           into the old_locals vector
 *      Global variables used:
 *              debug_level     Should we trace activities?
 *              recursion_level Used for tracing
 */
void
```

```
reset_locals(register Name target, register Property old_locals, register Proper
{
        register Property       this_conditional;
        Chain                   cond_chain;

        if (target->dont_activate_cond_values) {
                return;
        }

        /* Scan the list of conditional properties and restore the old value */
        /* to each one Reverse the order relative to when we assigned macros */
        this_conditional = get_prop(conditional->next, conditional_prop);
        if (this_conditional != NULL) {
                reset_locals(target, old_locals, this_conditional, index+1);
        } else {
                /* Remove conditional target from chain */
                if (conditional_targets == NULL ||
                    conditional_targets->name != target) {
                        warning(catgets(catd, 1, 39, "Internal error: reset targ
                } else {
                        cond_chain = conditional_targets->next;
                        retmem_mb((caddr_t) conditional_targets);
                        conditional_targets = cond_chain;
                }
        }
        get_prop(conditional->body.conditional.name->prop,
                 macro_prop)->body.macro = old_locals[index].body.macro;
        if (conditional->body.conditional.name == virtual_root) {
                (void) SETVAR(virtual_root, getvar(virtual_root), false);
        }
        if (debug_level > 1) {
                if (old_locals[index].body.macro.value != NULL) {
                        (void) printf(catgets(catd, 1, 40, "%*sdeactivating cond
                                      recursion_level,
                                      "",
                                      conditional->body.conditional.name->
                                      string_mb,
                                      old_locals[index].body.macro.value->
                                      string_mb);
                } else {
                        (void) printf(catgets(catd, 1, 41, "%*sdeactivating cond
                                      recursion_level,
                                      "",
                                      conditional->body.conditional.name->
                                      string_mb);
                }
        }
}

/*
 *      check_auto_dependencies(target, auto_count, automatics)
 *
 *      Returns true if the target now has a dependency
 *      it didn’t previously have (saved on automatics).
 *
 *      Return value:
 *                              true if new dependency found
 *
 *      Parameters:
 *              target          Target we check
 *              auto_count      Number of old automatic vars
 *              automatics      Saved old automatics
 *
 *      Global variables used:
 *              keep_state      Indicates that .KEEP_STATE is on
 */
```

```
Boolean
check_auto_dependencies(Name target, int auto_count, Name *automatics)
{
        Name            *p;
        int             n;
        Property        line;
        Dependency      dependency;

        if (keep_state) {
                if ((line = get_prop(target->prop, line_prop)) == NULL) {
                        return false;
                }
                /* Go thru new list of automatic depes */
                for (dependency = line->body.line.dependencies;
                     dependency != NULL;
                     dependency = dependency->next) {
                        /* And make sure that each one existed before we */
                        /* built the target */
                        if (dependency->automatic && !dependency->stale) {
                                for (n = auto_count, p = automatics;
                                     n > 0;
                                     n--) {
                                        if (*p++ == dependency->name) {
                                                /* If we can find it on the */
                                                /* saved list of autos we */
                                                /* are OK  */
                                                goto not_new;
                                        }
                                }
                                /* But if we scan over the old list */
                                /* of auto. without finding it it is */
                                /* new and we must check it */
                                return true;
                        }
                not_new:;
                }
                return false;
        } else {
                return false;
        }
}


// Recursively delete each of the Chain struct on the chain.

static void
delete_query_chain(Chain ch)
{
        if (ch == NULL) {
                return;
        } else {
                delete_query_chain(ch->next);
                retmem_mb((char *) ch);
        }
}

Doname
target_can_be_built(register Name target) {
        Doname          result = build_dont_know;
        Name            true_target = target;
        Property        line;

        if (target == wait_name) {
                return(build_ok);
        }
        /*
```

```
         * 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;
        }

        (void) exists(true_target);

        if (true_target->state == build_running) {
                return(build_running);
        }
        if (true_target->stat.time != file_doesnt_exist) {
                result = build_ok;
        }

        /* get line property for the target */
        line = get_prop(true_target->prop, line_prop);

        /* first check for explicit rule */
        if (line != NULL && line->body.line.command_template != NULL) {
                result = build_ok;
        }
        /* try to find pattern rule */
        if (result == build_dont_know) {
                result = find_percent_rule(target, NULL, false);
        }

        /* try to find double suffix rule */
        if (result == build_dont_know) {
                if (target->is_member) {
                        Property member = get_prop(target->prop, member_prop);
                        if (member != NULL && member->body.member.member != NULL
                                result = find_ar_suffix_rule(target, member->bod
                        } else {
                                result = find_double_suffix_rule(target, NULL, f
                        }
                } else {
                        result = find_double_suffix_rule(target, NULL, false);
                }
        }

        /* try to find suffix rule */
        if ((result == build_dont_know) && second_pass) {
                result = find_suffix_rule(target, target, empty_name, NULL, fals
        }

        /* check for sccs */
        if (result == build_dont_know) {
                result = sccs_get(target, NULL);
        }

        /* try to find dyn target */
        if (result == build_dont_know) {
                Name dtarg = find_dyntarget(target);
                if (dtarg != NULL) {
                        result = target_can_be_built(dtarg);
                }
        }

        /* check whether target was mentioned in makefile */
        if (result == build_dont_know) {
                if (target->colons != no_colon) {
                        result = build_ok;
                }
```

```
        }

        /* result */
        return result;
}
```

****
37160 Wed May 20 12:01:46 2015
new/usr/src/cmd/make/lib/mksh/macro.cc
make: restore a couple of blocks of code from DISTRIBUTED that should have been
****
______unchanged_portion_omitted_

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

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


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

```
                        buffer[0] = 0;
                        append_string(val_string, &destination, length);
                        if (strip_trailing_spaces) {
                                while ((length > 0) &&
                                       iswspace(destination.buffer.start[length-
                                        destination.buffer.start[--length] = 0;
                                }
                        }
                        value = GETNAME(destination.buffer.start, FIND_LENGTH);
                }
        }

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

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

        /* Debugging trace */
        if (debug_level > 1) {
                if (value != NULL) {
                        switch (daemon) {
                        case chain_daemon:
                                (void) printf("%s =", name->string_mb);
                                for (chain = (Chain) value;
                                     chain != NULL;
                                     chain = chain->next) {
                                        (void) printf(" %s", chain->name->string
                                }
                                (void) printf("\n");
                                break;
                        case no_daemon:
                                (void) printf("%s= %s\n",
                                              name->string_mb,
                                              value->string_mb);
                                break;
                        }
                } else {
                        (void) printf("%s =\n", name->string_mb);
                }
        }
        /* Set the new values in the macro property block */
/**/
        if(macro_apx != NULL) {
                macro_apx->body.macro_appendix.value = value;
                INIT_STRING_FROM_STACK(destination, buffer);
                buffer[0] = 0;
                if (value != NULL) {
                        APPEND_NAME(value,
                                      &destination,
                                      (int) value->hash.length);
                        if (macro_apx->body.macro_appendix.value_to_append != NU
                                MBTOWC(wcs_buffer, " ");
                                append_char(wcs_buffer[0], &destination);
                        }
                }
                if (macro_apx->body.macro_appendix.value_to_append != NULL) {
                        APPEND_NAME(macro_apx->body.macro_appendix.value_to_appe
                                      &destination,
                                      (int) macro_apx->body.macro_appendix.value
                }
                value = GETNAME(destination.buffer.start, FIND_LENGTH);
                if (destination.free_after_use) {
                        retmem(destination.buffer.start);
                }
        }
/**/
        macro->body.macro.value = value;
        macro->body.macro.daemon = daemon;
        /*
         * If the user changes the VIRTUAL_ROOT, we need to flush
         * the vroot package cache.
         */
        if (name == path_name) {
                flush_path_cache();
        }
        if (name == virtual_root) {
                flush_vroot_cache();
        }
        /* If this sets the VPATH we remember that */
        if ((name == vpath_name) &&
            (value != NULL) &&
            (value->hash.length > 0)) {
                vpath_defined = true;
        }
        /*
         * For environment variables we also set the
         * environment value each time.
         */
        if (macro->body.macro.exported) {
                static char     *env;

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

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

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

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

                wchar_t * wcb_ha = ha_str.get_string();
                wchar_t * wcb_ta = ta_str.get_string();
                wchar_t * wcb_vr = vr_str.get_string();
                length = 32 +
                  wslen(wcb_ha) +
                    wslen(wcb_ta) +
                      wslen(wcb_vr);
                old_vr = wcb_vr;
                MBSTOWCS(wcs_buffer, NOCATGETS("/usr/arch/"));
                if (IS_WEQUALN(old_vr,
                               wcs_buffer,
                               wslen(wcs_buffer))) {
                        old_vr = (wchar_t *) wschr(old_vr, (int) colon_char) + 1
                }
                if ( (ha == ta) || (wslen(wcb_ta) == 0) ) {
                        new_value = old_vr;
                } else {
                        new_value = ALLOC_WC(length);
                        new_value_allocated = true;
                        WCSTOMBS(mbs_buffer, old_vr);
                        (void) wsprintf(new_value,
                                        NOCATGETS("/usr/arch/%s/%s:%s"),
                                        ha->string_mb + 1,
                                        ta->string_mb + 1,
                                        mbs_buffer);
                }
                if (new_value[0] != 0) {
                        (void) setvar_daemon(virtual_root,
                                             GETNAME(new_value, FIND_LENGTH),
                                             false,
                                             no_daemon,
                                             true,
                                             debug_level);
                }
                if (new_value_allocated) {
                        retmem(new_value);
                }
        }
        return macro;
}
```

______***unchanged_portion_omitted_***