```
**********************************************************
     605 Wed May 20 11:30:07 2015
new/usr/src/cmd/make/Makefile.com
make: unifdef for SUNOS4_AND_AFTER (defined)
**********************************************************
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

# Copyright 2015, Richard Lowe.

MAKE_INCLUDE= $(SRC)/cmd/make/include
MAKE_DEFS= -DSYSV -DINTER
MAKE_DEFS= -DSYSV -DINTER -DSUNOS4_AND_AFTER
$(RELEASE_BUILD)MAKE_DEFS += -DNDEBUG
CFLAGS += $(CCVERBOSE)
CPPFLAGS += -I$(MAKE_INCLUDE) $(MAKE_DEFS)
```

```
**********************************************************
    3177 Wed May 20 11:30:08 2015
new/usr/src/cmd/make/bin/depvar.cc
make: unifdef for SUNOS4_AND_AFTER (defined)
**********************************************************
______unchanged_portion_omitted_

typedef struct _Depvar  *Depvar;

static  Depvar          depvar_list;
static  Depvar          *bpatch = &depvar_list;
static  Boolean         variant_deps;

/*
 * Add a name to the list.
 */

void
depvar_add_to_list(Name name, Boolean cmdline)
{
        Depvar          dv;

#ifdef SUNOS4_AND_AFTER
        dv = ALLOC(Depvar);
#else
        dv = (Depvar) Malloc(sizeof(struct _Depvar));
#endif
        dv->name = name;
        dv->next = NULL;
        dv->cmdline = cmdline;
        *bpatch = dv;
        bpatch = &dv->next;
}

/*
 * The macro ‘name’ has been used in either the left-hand or
 * right-hand side of a dependency.  See if it is in the
 * list.  Two things are looked for.  Names given as args
 * to the -V list are checked so as to set the same/differ
 * output for the -P option.  Names given as macro=value
 * command-line args are checked and, if found, an NSE
 * warning is produced.
 */
void
depvar_dep_macro_used(Name name)
{
        Depvar          dv;

        for (dv = depvar_list; dv != NULL; dv = dv->next) {
                if (name == dv->name) {
#ifdef NSE
#ifdef SUNOS4_AND_AFTER
                        if (dv->cmdline) {
#else
                        if (is_true(dv->cmdline)) {
#endif
                                nse_dep_cmdmacro(dv->name->string);
                        }
#endif
                        variant_deps = true;
                        break;
                }
        }
}

#ifdef NSE
```

```
/*
 * The macro ‘name’ has been used in either the argument
 * to a cd before a recursive make.  See if it was
 * defined on the command-line and, if so, complain.
 */
void
depvar_rule_macro_used(Name name)
{
        Depvar          dv;

        for (dv = depvar_list; dv != NULL; dv = dv->next) {
                if (name == dv->name) {
#ifdef SUNOS4_AND_AFTER
                        if (dv->cmdline) {
#else
                        if (is_true(dv->cmdline)) {
#endif
                                nse_rule_cmdmacro(dv->name->string);
                        }
                        break;
                }
        }
}
______unchanged_portion_omitted_
```

```
**********************************************************
   12541 Wed May 20 11:30:08 2015
new/usr/src/cmd/make/bin/nse.cc
make: unifdef for SUNOS4_AND_AFTER (defined)
**********************************************************
______unchanged_portion_omitted_
static Nse_suffix       sufx_hdr;
static int              our_exit_status;

static void             nse_warning(void);
static Boolean          nse_gettoken(wchar_t **, wchar_t *);

/*
 * Given a command that has just recursed to a sub make
 * try to determine if it cd'ed to a directory that was
 * defined by a make variable imported from the shell
 * environment or a variable with backquotes in it.
 * This routine will find something like:
 *      cd $(DIR); $(MAKE)
 * where DIR is imported from the shell environment.
 * However it well not find:
 *      CD = cd
 *              $(CD) $(DIR); $(MAKE)
 * or
 *      CD = cd $(DIR)
 *              $(CD); $(MAKE)
 *
 * This routine also checks for recursion to the same
 * directory.
 */
void
nse_check_cd(Property prop)
{
        wchar_t         tok[512];
        wchar_t         *p;
        wchar_t         *our_template;
        int             len;
        Boolean         cd;
#ifdef SUNOS4_AND_AFTER
        String_rec      string;
#else
        String          string;
#endif
        Name            name;
        Name            target;
        struct  Line    *line;
        struct  Recursive *r;
        Property        recurse;
        wchar_t         strbuf[STRING_BUFFER_LENGTH];
        wchar_t         tmpbuf[STRING_BUFFER_LENGTH];

#ifdef LTEST
        printf("In nse_check_cd, nse = %d, nse_did_recursion = %d\n", nse, nse_d
#endif
#ifdef SUNOS4_AND_AFTER
        if (!nse_did_recursion || !nse) {
#else
        if (is_false(nse_did_recursion) || is_false(flag.nse)) {
#endif
#ifdef LTEST
                printf ("returning,  nse = %d,  nse_did_recursion = %d\n", nse,
#endif
                return;
        }
        line = &prop->body.line;
#ifdef LTEST
```

```
        printf("string = %s\n", line->command_template->command_line->string_mb)
#endif

        wscpy(tmpbuf, line->command_template->command_line->string);
        our_template = tmpbuf;
        cd = false;
        while (nse_gettoken(&our_template, tok)) {
#ifdef LTEST
                printf("in gettoken loop\n");
#endif
#ifdef SUNOS4_AND_AFTER
                if (IS_WEQUAL(tok, (wchar_t *) "cd")) {
#else
                if (is_equal(tok, "cd")) {
#endif
                        cd = true;
                } else if (cd && tok[0] == '$') {
                        nse_backquote_seen = NULL;
                        nse_shell_var_used = NULL;
                        nse_watch_vars = true;
#ifdef SUNOS4_AND_AFTER
                        INIT_STRING_FROM_STACK(string, strbuf);
                        name = GETNAME(tok, FIND_LENGTH);
#else
                        init_string_from_stack(string, strbuf);
                        name = getname(tok, FIND_LENGTH);
#endif
                        expand_value(name, &string, false);
                        nse_watch_vars = false;

#ifdef LTEST
                        printf("cd = %d, tok = $\n", cd);
#endif
                        /*
                         * Try to trim tok to just
                         * the variable.
                         */
                        if (nse_shell_var_used != NULL) {
                                nse_warning();
                                fprintf(stderr, "\tMake invoked recursively by c
                                    nse_shell_var_used->string_mb,
                                    line->command_template->command_line->string
                        }
                        if (nse_backquote_seen != NULL) {
                                nse_warning();
                                fprintf(stderr, "\tMake invoked recursively by c
                                    nse_backquote_seen->string_mb,
                                    line->command_template->command_line->string
                        }
                        cd = false;
                } else if (cd && nse_backquotes(tok)) {
                        nse_warning();
                        fprintf(stderr, "\tMake invoked recursively by cd'ing to
                            line->command_template->command_line->string_mb);
                        cd = false;
                } else {
                        cd = false;
                }
        }

        /*
         * Now check for recursion to ".".
         */
        if (primary_makefile != NULL) {
                target = prop->body.line.target;
                recurse = get_prop(target->prop, recursive_prop);
                while (recurse != NULL) {
                        r = &recurse->body.recursive;
#ifdef SUNOS4_AND_AFTER
                        if (IS_WEQUAL(r->directory->string, (wchar_t *) ".") &&
                            !IS_WEQUAL(r->makefiles->name->string,
                            primary_makefile->string)) {
#else
                        if (is_equal(r->directory->string, ".") &&
                            !is_equal(r->makefiles->name->string,
                            primary_makefile->string)) {
#endif
                                nse_warning();
                                fprintf(stderr, "\tRecursion to makefile ‘%s’ in
                                    r->makefiles->name->string_mb,
                                    line->command_template->command_line->string
                        }
                        recurse = get_prop(recurse->next, recursive_prop);
                }
        }
}

/*
 * Print an NSE make warning line.
 * If the -P flag was given then consider this a fatal
 * error, otherwise, just a warning.
 */
static void
nse_warning(void)
{
#ifdef SUNOS4_AND_AFTER
        if (report_dependencies_level > 0) {
#else
        if (is_true(flag.report_dependencies)) {
#endif
                our_exit_status = 1;
        }
        if (primary_makefile != NULL) {
                fprintf(stderr, "make: NSE warning from makefile %s/%s:\n",
                        get_current_path(), primary_makefile->string_mb);
        } else {
                fprintf(stderr, "make: NSE warning from directory %s:\n",
                        get_current_path());
        }
}
```

______***unchanged_portion_omitted_***

```
/*
 * Given a dependency and a target, see if the dependency
 * is an SCCS file.  Check for the last component of its name
 * beginning with "s." and the component before that being "SCCS".
 * The NSE does not consider a source file to be derived from
 * an SCCS file.
 */
void
nse_check_sccs(wchar_t *targ, wchar_t *dep)
{
        wchar_t         *slash;
        wchar_t         *p;

#ifdef SUNOS4_AND_AFTER
        if (!nse) {
#else
        if (is_false(flag.nse)) {
#endif
                return;
        }
```

```
#ifdef SUNOS4_AND_AFTER
        slash = wsrchr(dep, (int) slash_char);
#else
        slash = rindex(dep, ’/’);
#endif
        if (slash == NULL) {
                return;
        }
        if (slash[1] != ’s’ || slash[2] != ’.’) {
                return;
        }

        /*
         * Find the next to last filename component.
         */
        for (p = slash - 1; p >= dep; p--) {
                if (*p == ’/’) {
                        break;
                }
        }
        p++;
#ifdef SUNOS4_AND_AFTER
        MBSTOWCS(wcs_buffer, "SCCS/");
        if (IS_WEQUALN(p, wcs_buffer, wslen(wcs_buffer))) {
#else
        if (is_equaln(p, "SCCS/", 5)) {
#endif
                nse_warning();
                WCSTOMBS(mbs_buffer, targ);
                WCSTOMBS(mbs_buffer2, dep);
                fprintf(stderr, "\tFile ‘%s’ depends upon SCCS file ‘%s’\n",
                        mbs_buffer, mbs_buffer2);
        }
        return;
}

/*
 * Given a filename check to see if it has 2 backquotes in it.
 * Complain about this because the shell expands the backquotes
 * but make does not so the files always appear to be out of date.
 */
void
nse_check_file_backquotes(wchar_t *file)
{
#ifdef SUNOS4_AND_AFTER
        if (!nse) {
#else
        if (is_false(flag.nse)) {
#endif
                return;
        }
        if (nse_backquotes(file)) {
                nse_warning();
                WCSTOMBS(mbs_buffer, file);
                fprintf(stderr, "\tFilename \"%s\" has backquotes in it\n",
                        mbs_buffer);
        }
}

/*
 * Return true if the string has two backquotes in it.
 */
Boolean
nse_backquotes(wchar_t *str)
{
        wchar_t         *bq;
```

```
#ifdef SUNOS4_AND_AFTER
        bq = wschr(str, (int) backquote_char);
        if (bq) {
                bq = wschr(&bq[1], (int) backquote_char);
#else
        bq = index(str, ’‘’);
        if (bq) {
                bq = index(&bq[1], ’‘’);
#endif
                if (bq) {
                        return true;
                }
        }
        return false;
}

/*
 * A macro that was defined on the command-line was found to affect the
 * set of dependencies.  The NSE "target explode" will not know about
 * this and will not get the same set of dependencies.
 */
void
nse_dep_cmdmacro(wchar_t *macro)
{
#ifdef SUNOS4_AND_AFTER
        if (!nse) {
#else
        if (is_false(flag.nse)) {
#endif
                return;
        }
        nse_warning();
        WCSTOMBS(mbs_buffer, macro);
        fprintf(stderr, "\tVariable ‘%s’ is defined on the command-line and\n\ta
                mbs_buffer);
}

/*
 * A macro that was defined on the command-line was found to
 * be part of the argument to a cd before a recursive make.
 * This make cause the make to recurse to different places
 * depending upon how it is invoked.
 */
void
nse_rule_cmdmacro(wchar_t *macro)
{
#ifdef SUNOS4_AND_AFTER
        if (!nse) {
#else
        if (is_false(flag.nse)) {
#endif
                return;
        }
        nse_warning();
        WCSTOMBS(mbs_buffer, macro);
        fprintf(stderr, "\tMake invoked recursively by cd’ing to a directory\n\t
                mbs_buffer);
}

/*
 * A dependency has been found with a wildcard in it.
 * This causes the NSE problems because the set of dependencies
 * can change without changing the Makefile.
 */
void
```

```
nse_wildcard(wchar_t *targ, wchar_t *dep)
{
#ifdef SUNOS4_AND_AFTER
        if (!nse) {
#else
        if (is_false(flag.nse)) {
#endif
                return;
        }
        nse_warning();
        WCSTOMBS(mbs_buffer, targ);
        WCSTOMBS(mbs_buffer2, dep);
        fprintf(stderr, "\tFile ‘%s’ has a wildcard in dependency ‘%s’\n",
                mbs_buffer, mbs_buffer2);
}

/*
 * Read in the list of suffixes that are interpreted as source
 * files.
 */
void
nse_init_source_suffixes(void)
{
        FILE            *fp;
        wchar_t         suffix[100];
        Nse_suffix      sufx;
        Nse_suffix      *bpatch;

        fp = fopen(TARG_SUFX, "r");
        if (fp == NULL) {
                return;
        }
        bpatch = &sufx_hdr;
        while (fscanf(fp, "%s %*s", suffix) == 1) {
#ifdef SUNOS4_AND_AFTER
                sufx = ALLOC(Nse_suffix);
                sufx->suffix = wscpy(ALLOC_WC(wslen(suffix) + 1), suffix);
#else
                sufx = alloc(Nse_suffix);
                sufx->suffix = strcpy(malloc(strlen(suffix) + 1), suffix);
#endif
                sufx->next = NULL;
                *bpatch = sufx;
                bpatch = &sufx->next;
        }
        fclose(fp);
}

/*
 * Check if a derived file (something with a dependency) appears
 * to be a source file (by its suffix) but has no rule to build it.
 * If so, complain.
 *
 * This generally arises from the old-style of make-depend that
 * produces:
 *      foo.c:  foo.h
 */
void
nse_check_derived_src(Name target, wchar_t *dep, Cmd_line command_template)
{
        Nse_suffix      sufx;
        wchar_t         *suffix;
        wchar_t         *depsufx;

#ifdef SUNOS4_AND_AFTER
        if (!nse) {
```

```
#else
        if (is_false(flag.nse)) {
#endif
                return;
        }
#ifdef SUNOS4_AND_AFTER
        if (target->stat.is_derived_src) {
#else
        if (is_true(target->stat.is_derived_src)) {
#endif
                return;
        }
        if (command_template != NULL) {
                return;
        }
#ifdef SUNOS4_AND_AFTER
        suffix = wsrchr(target->string, (int) period_char );
#else
        suffix = rindex(target->string, '.');
#endif
        if (suffix != NULL) {
                for (sufx = sufx_hdr; sufx != NULL; sufx = sufx->next) {
#ifdef SUNOS4_AND_AFTER
                        if (IS_WEQUAL(sufx->suffix, suffix)) {
#else
                        if (is_equal(sufx->suffix, suffix)) {
#endif
                                nse_warning();
                                WCSTOMBS(mbs_buffer, dep);
                                fprintf(stderr, "\tProbable source file '%s' app
                                        target->string_mb, mbs_buffer);
                                break;
                        }
                }
        }
}

/*
 * See if a target is a potential source file and has no
 * dependencies and no rule but shows up on the right-hand
 * side.  This tends to occur from old "make depend" output.
 */
void
nse_check_no_deps_no_rule(Name target, Property line, Property command)
{
        Nse_suffix      sufx;
        wchar_t         *suffix;

#ifdef SUNOS4_AND_AFTER
        if (!nse) {
#else
        if (is_false(flag.nse)) {
#endif
                return;
        }
#ifdef SUNOS4_AND_AFTER
        if (target->stat.is_derived_src) {
#else
        if (is_true(target->stat.is_derived_src)) {
#endif
                return;
        }
        if (line != NULL && line->body.line.dependencies != NULL) {
                return;
        }
#ifdef SUNOS4_AND_AFTER
```

```
        if (command->body.line.sccs_command) {
#else
        if (is_true(command->body.line.sccs_command)) {
#endif
                return;
        }
#ifdef SUNOS4_AND_AFTER
        suffix = wsrchr(target->string, (int) period_char);
#else
        suffix = rindex(target->string, '.');
#endif
        if (suffix != NULL) {
                for (sufx = sufx_hdr; sufx != NULL; sufx = sufx->next) {
#ifdef SUNOS4_AND_AFTER
                        if (IS_WEQUAL(sufx->suffix, suffix)) {
#else
                        if (is_equal(sufx->suffix, suffix)) {
#endif
                                if (command->body.line.command_template == NULL)
                                        nse_warning();
                                        fprintf(stderr, "\tProbable source file
                                                target->string_mb);
                                }
                        }
                }
        }
}

/*
 * Detected a situation where a recursive make derived a file
 * without using a makefile.
 */
void
nse_no_makefile(Name target)
{
#ifdef SUNOS4_AND_AFTER
        if (!nse) {
#else
        if (is_false(flag.nse)) {
#endif
                return;
        }
        nse_warning();
        fprintf(stderr, "Recursive make to derive %s did not use a makefile\n",
                target->string_mb);
}
```

______***unchanged_portion_omitted_***

****
8831 Wed May 20 11:30:08 2015
new/usr/src/cmd/make/bin/nse_printdep.cc
make: unifdef for SUNOS4_AND_AFTER (defined)
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2003 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Included files
 */
#include <mk/defs.h>
#include <mksh/misc.h>          /* get_prop() */

/*
 * File table of contents
 */
void   print_dependencies(register Name target, register Property line);
static void     print_deps(register Name target, register Property line);
static void     print_more_deps(Name target, Name name);
static void     print_filename(Name name);
static Boolean  should_print_dep(Property line);
static void     print_forest(Name target);
static void     print_deplist(Dependency head);
void            print_value(register Name value, Daemon daemon);
static void     print_rule(register Name target);
static  void    print_rec_info(Name target);
static Boolean  is_out_of_date(Property line);
extern void depvar_print_results (void);
extern int printf (const char *, ...);
extern int _flsbuf (unsigned int, FILE *);

/*
 *      print_dependencies(target, line)
 *
 *      Print all the dependencies of a target. First print all the Makefiles.
 *      Then print all the dependencies. Finally, print all the .INIT
 *      dependencies.
 *
 *      Parameters:
 *              target          The target we print dependencies for
 *              line            We get the dependency list from here
 *
 *      Global variables used:
```

```
 *              done            The Name ".DONE"
 *              init            The Name ".INIT"
 *              makefiles_used  List of all makefiles read
 */
void
print_dependencies(register Name target, register Property line)
{
        Dependency      dp;
        static Boolean  makefiles_printed = false;

#ifdef SUNOS4_AND_AFTER
        if (target_variants) {
#else
        if (is_true(flag.target_variants)) {
#endif
                depvar_print_results();
        }

        if (!makefiles_printed) {
                /*
                 * Search the makefile list for the primary makefile,
                 * then print it and its inclusions.  After that go back
                 * and print the default.mk file and its inclusions.
                 */
                for (dp = makefiles_used; dp != NULL; dp = dp->next) {
                        if (dp->name == primary_makefile) {
                                break;
                        }
                }
                if (dp) {
                        print_deplist(dp);
                        for (dp = makefiles_used; dp != NULL; dp = dp->next) {
                                if (dp->name == primary_makefile) {
                                        break;
                                }
                                (void)printf(" %s", dp->name->string_mb);
                        }
                }
                (void) printf("\n");
                makefiles_printed = true;
        }
        print_deps(target, line);
#ifdef SUNOS4_AND_AFTER
/*
        print_more_deps(target, init);
        print_more_deps(target, done);
 */
        if (target_variants) {
#else
        print_more_deps(target, cached_names.init);
        print_more_deps(target, cached_names.done);
        if (is_true(flag.target_variants)) {
#endif
                print_forest(target);
        }
}
```

______***unchanged_portion_omitted_***

```
/*
 *      print_deps(target, line, go_recursive)
 *
 *      Print a regular dependency list.  Append to this information which
 *      indicates whether or not the target is recursive.
 *
 *      Parameters:
 *              target          target to print dependencies for
 *              line            We get the dependency list from here
 *              go_recursive    Should we show all dependencies recursively?
 *
 *      Global variables used:
 *              recursive_name  The Name ".RECURSIVE", printed
 */
static void
print_deps(register Name target, register Property line)
{
        register Dependency     dep;

#ifdef SUNOS4_AND_AFTER
        if ((target->dependency_printed) ||
            (target == force)) {
#else
        if (is_true(target->dependency_printed)) {
#endif
                return;
        }
        target->dependency_printed = true;

        /* only print entries that are actually derived and are not leaf
         * files and are not the result of sccs get.
         */
        if (should_print_dep(line)) {
#ifdef NSE
                nse_check_no_deps_no_rule(target, line, line);
#endif
                if ((report_dependencies_level == 2) ||
                    (report_dependencies_level == 4)) {
                        if (is_out_of_date(line)) {
                                (void) printf("1 ");
                        } else {
                                (void) printf("0 ");
                        }
                }
                print_filename(target);
                (void) printf(":\t");
                print_deplist(line->body.line.dependencies);
                print_rec_info(target);
                (void) printf("\n");
                for (dep = line->body.line.dependencies;
                     dep != NULL;
                     dep = dep->next) {
                        print_deps(dep->name,
                                   get_prop(dep->name->prop, line_prop));
                }
        }
}
```

______***unchanged_portion_omitted_***

```
/*
 *      should_print_dep(line)
 *
 *      Test if we should print the dependencies of this target.
 *      The line must exist and either have children dependencies
 *      or have a command that is not an SCCS command.
 *
 *      Return value:
 *                              true if the dependencies should be printed
 *
 *      Parameters:
 *              line            We get the dependency list from here
 *
 *      Global variables used:
 */
```

```
static Boolean
should_print_dep(Property line)
{
        if (line == NULL) {
                return false;
        }
        if (line->body.line.dependencies != NULL) {
                return true;
        }
#ifdef SUNOS4_AND_AFTER
        if (line->body.line.sccs_command) {
#else
        if (is_true(line->body.line.sccs_command)) {
#endif
                return false;
        }
        return true;
}

/*
 * Print out the root nodes of all the dependency trees
 * in this makefile.
 */
static void
print_forest(Name target)
{
        Name_set::iterator np, e;
        Property        line;

        for (np = hashtab.begin(), e = hashtab.end(); np != e; np++) {
#ifdef SUNOS4_AND_AFTER
                        if (np->is_target && !np->has_parent && np != target) {
#else
                        if (is_true(np->is_target) &&
                            is_false(np->has_parent) &&
                            np != target) {
#endif
                                (void) doname_check(np, true, false, false);
                                line = get_prop(np->prop, line_prop);
                                printf("-\n");
                                print_deps(np, line);
                        }
        }
}

#ifndef SUNOS4_AND_AFTER
printdesc()
{
        Name_set::iterator      p, e;
        register Property       prop;
        register Dependency     dep;
        register Cmd_line       rule;
        Percent                 percent, percent_depe;

        /* Default target */
        if (default_target_to_build != NULL) {
                print_rule(default_target_to_build);
                default_target_to_build->dependency_printed= true;
        };
        (void)printf("\n");

        /* .AR_REPLACE */
        if (ar_replace_rule != NULL) {
                (void)printf("%s:\n", cached_names.ar_replace->string_mb);
                for (rule= ar_replace_rule; rule != NULL; rule= rule->next)
                        (void)printf("\t%s\n", rule->command_line->string_mb);
        };

        /* .DEFAULT */
        if (default_rule != NULL) {
                (void)printf("%s:\n", cached_names.default_rule->string_mb);
                for (rule= default_rule; rule != NULL; rule= rule->next)
                        (void)printf("\t%s\n", rule->command_line->string_mb);
        };

        /* .IGNORE */
        if (is_true(flag.ignore_errors))
                (void)printf("%s:\n", cached_names.ignore->string_mb);

        /* .KEEP_STATE: */
        if (is_true(flag.keep_state))
                (void)printf("%s:\n\n", cached_names.dot_keep_state->string_mb);

        /* .PRECIOUS */
        (void)printf("%s: ", cached_names.precious->string_mb);
        for (p = hashtab.begin(), e = hashtab.end(); p != e; p++)
                        if (is_true(p->stat.is_precious | all_precious))
                                (void)printf("%s ", p->string_mb);
        (void)printf("\n");

        /* .SCCS_GET */
        if (sccs_get_rule != NULL) {
                (void)printf("%s:\n", cached_names.sccs_get->string_mb);
                for (rule= sccs_get_rule; rule != NULL; rule= rule->next)
                        (void)printf("\t%s\n", rule->command_line->string_mb);
        };

        /* .SILENT */
        if (is_true(flag.silent))
                (void)printf("%s:\n", cached_names.silent->string_mb);

        /* .SUFFIXES: */
        (void)printf("%s: ", cached_names.suffixes->string_mb);
        for (dep= suffixes; dep != NULL; dep= dep->next) {
                (void)printf("%s ", dep->name->string_mb);
                build_suffix_list(dep->name);
        };
        (void)printf("\n\n");

        /* % rules */
        for (percent= percent_list; percent != NULL; percent= percent->next) {
                (void) printf("%s:", percent->name->string_mb);

                for (percent_depe= percent->dependencies; percent_depe != NULL;
                        (void) printf(" %s", percent_depe->name->string_mb);

                (void) printf("\n");

                for (rule= percent->command_template; rule != NULL; rule= rule->
                        (void)printf("\t%s\n", rule->command_line->string_mb);
        };

        /* Suffix rules */
        for (p = hashtab.begin(), e = hashtab.end(); p != e; p++)
                        if (is_false(p->dependency_printed) && (p->string[0] ==
                                print_rule(p);
                                p->dependency_printed= true;
                        };

        /* Macro assignments */
        for (p = hashtab.begin(), e = hashtab.end(); p != e; p++)
                        if (((prop= get_prop(p->prop, macro_prop)) != NULL) &&
                            (prop->body.macro.value != NULL)) {
                                (void)printf("%s", p->string_mb);
                                print_value(prop->body.macro.value,
                                            prop->body.macro.daemon);
                        };
        (void)printf("\n");

        /* Delays */
        for (p = hashtab.begin(), e = hashtab.end(); p != e; p++)
                        for (prop= get_prop(p->prop, conditional_prop);
                             prop != NULL;
                             prop= get_prop(prop->next, conditional_prop)) {
                                (void)printf("%s := %s",
                                             p->string_mb,
                                             prop->body.conditional.name->string
                                print_value(prop->body.conditional.value, no_dae
                        };
        (void)printf("\n");

        /* All other dependencies */
        for (p = hashtab.begin(), e = hashtab.end(); p != e; p++)
                        if (is_false(p->dependency_printed) && (p->colons != no_
                                print_rule(p);
        (void)printf("\n");
        exit(0);
}
#endif
```

```
/*
 *      This is a set  of routines for dumping the internal make state
 *      Used for the -p option
 */
void
print_value(register Name value, Daemon daemon)

#ifdef SUNOS4_AND_AFTER

#else

#endif
{
        Chain                   cp;

        if (value == NULL)
                (void)printf("=\n");
        else
                switch (daemon) {
                    case no_daemon:
                        (void)printf("= %s\n", value->string_mb);
                        break;
                    case chain_daemon:
                        for (cp= (Chain) value; cp != NULL; cp= cp->next)
                                (void)printf(cp->next == NULL ? "%s" : "%s ",
                                        cp->name->string_mb);
                        (void)printf("\n");
                        break;
                };
}
```
______***unchanged_portion_omitted_***