```
**********************************************************
    1016 Wed May 20 11:04:05 2015
new/usr/src/cmd/make/Makefile
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.21 06/12/12
#

TOP     = ..

SUBDIRS=        lib \
                bin \
                localize \
                man

include ${TOP}/rules/recurse.mk
#endif /* ! codereview */
```

```
**********************************************************
    1034 Wed May 20 11:04:06 2015
new/usr/src/cmd/make/bin/Makefile
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.16 06/12/12
#

TOP =           ../..

# SUBDIRS =     make maketool rxm rxs
SUBDIRS =       make  rxm rxs

include $(TOP)/rules/recurse.mk
#endif /* ! codereview */
```

```
**********************************************************
    1406 Wed May 20 11:04:06 2015
new/usr/src/cmd/make/bin/make/Common.mk
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
  1 #
  2 # CDDL HEADER START
  3 #
  4 # The contents of this file are subject to the terms of the
  5 # Common Development and Distribution License (the "License").
  6 # You may not use this file except in compliance with the License.
  7 #
  8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  9 # or http://www.opensolaris.org/os/licensing.
 10 # See the License for the specific language governing permissions
 11 # and limitations under the License.
 12 #
 13 # When distributing Covered Code, include this CDDL HEADER in each
 14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 15 # If applicable, add the following below this CDDL HEADER, with the
 16 # fields enclosed by brackets "[]" replaced with your own identifying
 17 # information: Portions Copyright [yyyy] [name of copyright owner]
 18 #
 19 # CDDL HEADER END
 20 #
 21 # Copyright 1996 Sun Microsystems, Inc. All rights reserved.
 22 # Use is subject to license terms.
 23 #
 24 # @(#)Common.mk 1.3 06/12/12
 25 #

 27 TOP             = ../../../../..
 28 include $(TOP)/rules/master.mk

 30 PKG_TOP         = $(TOP)/Make

 32 LIBBSD_DIR      = $(PKG_TOP)/libbsd
 33 LIBBSD          = $(LIBBSD_DIR)/$(VARIANT)/libbsd.a

 35 LIBMKSDMSI18N_DIR = $(PKG_TOP)/libmksdmsi18n
 36 LIBMKSDMSI18N   = $(LIBMKSDMSI18N_DIR)/$(VARIANT)/libmksdmsi18n.a

 38 LIBMKSH_DIR     = $(PKG_TOP)/libmksh
 39 LIBMKSH         = $(LIBMKSH_DIR)/$(VARIANT)/libmksh.a

 41 LIBVROOT_DIR    = $(PKG_TOP)/libvroot
 42 LIBVROOT        = $(LIBVROOT_DIR)/$(VARIANT)/libvroot.a

 44 SRC             = ../src
 45 include $(TOP)/rules/derived.mk

 47 #endif /* ! codereview */
```

****
1043 Wed May 20 11:04:06 2015
new/usr/src/cmd/make/bin/make/Makefile
make: initial Sun make source, disconnected from the build
****

```
#
# 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 1997 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.7 06/12/12
#

TOP =                   ../../..

#SUBDIRS =              dmake localize
SUBDIRS =               smake make.xpg4 make.svr4

include $(TOP)/rules/recurse.mk
#endif /* ! codereview */
```

```
**********************************************************
   24950 Wed May 20 11:04:06 2015
new/usr/src/cmd/make/bin/make/common/ar.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)ar.cc 1.28 06/12/12
 */

#pragma ident   "@(#)ar.cc      1.28    06/12/12"

/*
 *      ar.c
 *
 *      Deal with the lib.a(member.o) and lib.a((entry-point)) notations
 *
 * Look inside archives for notations a(b) and a((b))
 *      a(b)    is file member   b  in archive a
 *      a((b))  is entry point   b  in object archive a
 *
 * For 6.0, create a make which can understand all archive
 * formats.  This is kind of tricky, and <ar.h> isnt any help.
 */

/*
 * Included files
 */
#include <avo/avo_alloca.h>             /* alloca() */
#include <ar.h>
#include <errno.h>              /* errno */
#include <fcntl.h>              /* open() */
#include <mk/defs.h>
#include <mksh/misc.h>          /* retmem_mb() */

#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
struct ranlib {
        union {
                off_t   ran_strx;       /* string table index of */
                char    *ran_name;      /* symbol defined by */
        }       ran_un;
        off_t   ran_off;                /* library member at this offset */
};
```



```
#else
#include <ranlib.h>
#endif

#if defined(linux)
#include <ctype.h>              /* isspace */
#else
#include <unistd.h>             /* close() */
#endif


/*
 * Defined macros
 */
#ifndef S5EMUL
#undef BITSPERBYTE
#define BITSPERBYTE     8
#endif

/*
 * Defines for all the different archive formats.  See next comment
 * block for justification for not using <ar.h>s versions.
 */
#define AR_5_MAGIC              "<ar>"          /* 5.0 format magic string */
#define AR_5_MAGIC_LENGTH       4               /* 5.0 format string length */

#define AR_PORT_MAGIC           "!<arch>\n"     /* Port. (6.0) magic string */
#define AR_PORT_MAGIC_LENGTH    8               /* Port. (6.0) string length */
#define AR_PORT_END_MAGIC       "`\n"           /* Port. (6.0) end of header */
#define AR_PORT_WORD            4               /* Port. (6.0) 'word' length */

/*
 * typedefs & structs
 */
/*
 * These are the archive file headers for the formats.  Note
 * that it really doesnt matter if these structures are defined
 * here.  They are correct as of the respective archive format
 * releases.  If the archive format is changed, then since backwards
 * compatability is the desired behavior, a new structure is added
 * to the list.
 */
typedef struct {        /* 5.0 ar header format: vax family; 3b family */
        char                    ar_magic[AR_5_MAGIC_LENGTH];    /* AR_5_MAGIC*/
        char                    ar_name[16];    /* Space terminated */
        char                    ar_date[AR_PORT_WORD];  /* sgetl() accessed */
        char                    ar_syms[AR_PORT_WORD];  /* sgetl() accessed */
}                       Arh_5;

typedef struct {        /* 5.0 ar symbol format: vax family; 3b family */
        char                    sym_name[8];    /* Space terminated */
        char                    sym_ptr[AR_PORT_WORD];  /* sgetl() accessed */
}                       Ars_5;

typedef struct {        /* 5.0 ar member format: vax family; 3b family */
        char                    arf_name[16];   /* Space terminated */
        char                    arf_date[AR_PORT_WORD]; /* sgetl() accessed */
        char                    arf_uid[AR_PORT_WORD];  /* sgetl() accessed */
        char                    arf_gid[AR_PORT_WORD];  /* sgetl() accessed */
        char                    arf_mode[AR_PORT_WORD]; /* sgetl() accessed */
        char                    arf_size[AR_PORT_WORD]; /* sgetl() accessed */
}                       Arf_5;

typedef struct {        /* Portable (6.0) ar format: vax family; 3b family */
        char                    ar_name[16];    /* Space terminated */
        /* left-adjusted fields; decimal ascii; blank filled */
```

```
128         char                    ar_date[12];
129         char                    ar_uid[6];
130         char                    ar_gid[6];
131         char                    ar_mode[8];     /* octal ascii */
132         char                    ar_size[10];
133         /* special end-of-header string (AR_PORT_END_MAGIC) */
134         char                    ar_fmag[2];
135 }                       Ar_port;

137 enum ar_type {
138                 AR_5,
139                 AR_PORT
140 };

142 typedef unsigned int ar_port_word; // must be 4-bytes long

144 typedef struct {
145         FILE                    *fd;
146         /* to distiguish ar format */
147         enum ar_type            type;
148         /* where first ar member header is at */
149         long                    first_ar_mem;
150         /* where the symbol lookup starts */
151         long                    sym_begin;
152         /* the number of symbols available */
153         long                    num_symbols;
154         /* length of symbol directory file */
155         long                    sym_size;
156         Arh_5                   arh_5;
157         Ars_5                   ars_5;
158         Arf_5                   arf_5;
159         Ar_port                 ar_port;
160 }                       Ar;

162 /*
163  * Static variables
164  */

166 /*
167  * File table of contents
168  */
169 extern  timestruc_t&    read_archive(register Name target);
170 static  Boolean         open_archive(char *filename, register Ar *arp);
171 static  void            close_archive(register Ar *arp);
172 static  Boolean         read_archive_dir(register Ar *arp, Name library, char **
173 static  void            translate_entry(register Ar *arp, Name target, register
174 static  long            sgetl(char *);

176 /*
177  *      read_archive(target)
178  *
179  *      Read the contents of an ar file.
180  *
181  *      Return value:
182  *                              The time the member was created
183  *
184  *      Parameters:
185  *              target          The member to find time for
186  *
187  *      Global variables used:
188  *              empty_name      The Name ""
189  */

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



```
194 timestruc_t&
195 read_archive(register Name target)
196 {
197         register Property       member;
198         wchar_t                 *slash;
199         String_rec              true_member_name;
200         wchar_t                 buffer[STRING_BUFFER_LENGTH];
201         register Name           true_member = NULL;
202         Ar                      ar;
203         char                    *long_names_table = NULL; /* Table of long
204                                                             member names */

206         member = get_prop(target->prop, member_prop);
207         /*
208          * Check if the member has directory component.
209          * If so, remove the dir and see if we know the date.
210          */
211         if (member->body.member.member != NULL) {
212                 Wstring member_string(member->body.member.member);
213                 wchar_t * wcb = member_string.get_string();
214                 if((slash = (wchar_t *) wsrchr(wcb, (int) slash_char)) != NULL)
215                         INIT_STRING_FROM_STACK(true_member_name, buffer);
216                         append_string(member->body.member.library->string_mb,
217                                       &true_member_name,
218                                       FIND_LENGTH);
219                         append_char((int) parenleft_char, &true_member_name);
220                         append_string(slash + 1, &true_member_name, FIND_LENGTH)
221                         append_char((int) parenright_char, &true_member_name);
222                         true_member = GETNAME(true_member_name.buffer.start,
223                                               FIND_LENGTH);
224                         if (true_member->stat.time != file_no_time) {
225                                 target->stat.time = true_member->stat.time;
226                                 return target->stat.time;
227                         }
228                 }
229         }
230         if (open_archive(member->body.member.library->string_mb, &ar) == failed)
231                 if (errno == ENOENT) {
232                         target->stat.stat_errno = ENOENT;
233                         close_archive(&ar);
234                         if (member->body.member.member == NULL) {
235                                 member->body.member.member = empty_name;
236                         }
237                         return target->stat.time = file_doesnt_exist;
238                 } else {
239                         fatal(catgets(catd, 1, 1, "Can't access archive '%s': %s
240                               member->body.member.library->string_mb,
241                               errmsg(errno));
242                 }
243         }
244         if (target->stat.time == file_no_time) {
245                 if (read_archive_dir(&ar, member->body.member.library,
246                                      &long_names_table)
247                     == failed){
248                         fatal(catgets(catd, 1, 2, "Can't access archive '%s': %s
249                               member->body.member.library->string_mb,
250                               errmsg(errno));
251                 }
252         }
253         if (member->body.member.entry != NULL) {
254                 translate_entry(&ar, target, member,&long_names_table);
255         }
256         close_archive(&ar);
257         if (long_names_table) {
258                 retmem_mb(long_names_table);
259         }
```

```
260         if (true_member != NULL) {
261                 target->stat.time = true_member->stat.time;
262         }
263         if (target->stat.time == file_no_time) {
264                 target->stat.time = file_doesnt_exist;
265         }
266         return target->stat.time;
267 }

269 /*
270  *      open_archive(filename, arp)
271  *
272  *      Return value:
273  *                              Indicates if open failed or not
274  *
275  *      Parameters:
276  *              filename        The name of the archive we need to read
277  *              arp             Pointer to ar file description block
278  *
279  *      Global variables used:
280  */
281 static Boolean
282 open_archive(char *filename, register Ar *arp)
283 {
284         int                     fd;
285         char                    mag_5[AR_5_MAGIC_LENGTH];
286         char                    mag_port[AR_PORT_MAGIC_LENGTH];
287         char                    buffer[4];

289         arp->fd = NULL;
290         fd = open_vroot(filename, O_RDONLY, 0, NULL, VROOT_DEFAULT);
291         if ((fd < 0) || ((arp->fd = fdopen(fd, "r")) == NULL)) {
292                 return failed;
293         }
294         (void) fcntl(fileno(arp->fd), F_SETFD, 1);

296 #if !defined(SUN5_0) && !defined(linux) //XXX
297         /* Read enough of the archive to distinguish between the formats */
298         if (fread(mag_5, AR_5_MAGIC_LENGTH, 1, arp->fd) != 1) {
299                 return failed;
300         }
301         if (IS_EQUALN(mag_5, AR_5_MAGIC, AR_5_MAGIC_LENGTH)) {
302                 arp->type = AR_5;
303                 /* Must read in header to set necessary info */
304                 if (fseek(arp->fd, 0L, 0) != 0 ||
305                     fread((char *) &arp->arh_5, sizeof (Arh_5), 1, arp->fd) !=
306                                                                         1) {
307                         return failed;
308                 }
309                 arp->sym_begin = ftell(arp->fd);
310                 arp->num_symbols = sgetl(arp->arh_5.ar_syms);
311                 arp->first_ar_mem = arp->sym_begin +
312                                     sizeof (Ars_5) * arp->num_symbols;
313                 arp->sym_size = 0L;
314                 return succeeded;
315         }
316         if (fseek(arp->fd, 0L, 0) != 0) {
317                 return failed;
318         }
319 #endif
320         if (fread(mag_port, AR_PORT_MAGIC_LENGTH, 1, arp->fd) != 1) {
321                 return failed;
322         }
323         if (IS_EQUALN(mag_port, AR_PORT_MAGIC, AR_PORT_MAGIC_LENGTH)) {
324                 arp->type = AR_PORT;
325                 /*
```



```
326                  * Read in first member header to find out if there is
327                  * a symbol definition table.
328                  */

330                 int ret = read_member_header(&arp->ar_port, arp->fd, filename);
331                 if (ret == failed) {
332                         return failed;
333                 } else if(ret == -1) {
334                         /* There is no member header - empty archive */
335                         arp->sym_size = arp->num_symbols = arp->sym_begin = 0L;
336                         arp->first_ar_mem = ftell(arp->fd);
337                         return succeeded;
338                 }
339                 /*
340                  * The following values are the default if there is
341                  * no symbol directory and long member names.
342                  */
343                 arp->sym_size = arp->num_symbols = arp->sym_begin = 0L;
344                 arp->first_ar_mem = ftell(arp->fd) - (long) sizeof (Ar_port);

346                 /*
347                  * Do we have a symbol table? A symbol table is always
348                  * the first member in an archive. In 4.1.x it has the
349                  * name __.SYMDEF, in SVr4, it has the name "/       "
350                  */
351 /*
352 #ifdef SUN5_0
353                 MBSTOWCS(wcs_buffer, NOCATGETS("/               "));
354                 if (IS_WEQUALN(arp->ar_port.ar_name, wcs_buffer, 16)) {
355 #else
356                 MBSTOWCS(wcs_buffer, NOCATGETS("__.SYMDEF       "));
357                 if (IS_WEQUALN(arp->ar_port.ar_name, wcs_buffer, 16)) {
358 #endif
359  */
360 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
361                 if (IS_EQUALN(arp->ar_port.ar_name,
362                                 NOCATGETS("/               "),
363                                 16)) {
364 #else
365                 if (IS_EQUALN(arp->ar_port.ar_name,
366                                 NOCATGETS("__.SYMDEF       "),
367                                 16)) {
368 #endif
369                         if (sscanf(arp->ar_port.ar_size,
370                                    "%ld",
371                                    &arp->sym_size) != 1) {
372                                 return failed;
373                         }
374                         arp->sym_size += (arp->sym_size & 1); /* round up */
375                         if (fread(buffer, sizeof buffer, 1, arp->fd) != 1) {
376                                 return failed;
377                         }
378                         arp->num_symbols = sgetl(buffer);
379                         arp->sym_begin = ftell(arp->fd);
380                         arp->first_ar_mem = arp->sym_begin +
381                                             arp->sym_size - sizeof buffer;
382                 }
383                 return succeeded;
384         }
385         fatal(catgets(catd, 1, 3, "‘%s’ is not an archive"), filename);
386         /* NOTREACHED */
387         return failed;
388 }


391 /*
```

```
 *      close_archive(arp)
 *
 *      Parameters:
 *              arp             Pointer to ar file description block
 *
 *      Global variables used:
 */
static void
close_archive(register Ar *arp)
{
        if (arp->fd != NULL) {
                (void) fclose(arp->fd);
        }
}

/*
 *      read_archive_dir(arp, library, long_names_table)
 *
 *      Reads the directory of an archive and enters all
 *      the members into the make symboltable in lib(member) format
 *      with their dates.
 *
 *      Parameters:
 *              arp             Pointer to ar file description block
 *              library         Name of lib to enter members for.
 *                              Used to form "lib(member)" string.
 *              long_names_table table that contains list of members
 *                              with names > 15 characters long
 *
 *      Global variables used:
 */
static Boolean
#if defined(SUN5_0) || defined(linux) //XXX
read_archive_dir(register Ar *arp, Name library, char **long_names_table)
#else
read_archive_dir(register Ar *arp, Name library, char **)
#endif
{
        wchar_t                 *name_string;
        wchar_t                 *member_string;
        register long           len;
        register wchar_t        *p;
        register char           *q;
        register Name           name;
        Property                member;
        long                    ptr;
        long                    date;

#if defined(SUN5_0) || defined(linux) //XXX
        int                     offset;

        /*
         * If any of the members has a name > 15 chars,
         * it will be found here.
         */
        if (process_long_names_member(arp, long_names_table, library->string_mb)
                return failed;
        }
#endif
        name_string = ALLOC_WC((int) (library->hash.length +
                                      (int) ar_member_name_len * 2));
        (void) mbstowcs(name_string, library->string_mb, (int) library->hash.len
        member_string = name_string + library->hash.length;
        *member_string++ = (int) parenleft_char;

        if (fseek(arp->fd, arp->first_ar_mem, 0) != 0) {
```



```
                goto read_error;
        }
        /* Read the directory using the appropriate format */
        switch (arp->type) {
        case AR_5:
            for (;;) {
                if (fread((char *) &arp->arf_5, sizeof arp->arf_5, 1, arp->fd)
                    != 1) {
                        if (feof(arp->fd)) {
                                return succeeded;
                        }
                        break;
                }
                len = sizeof arp->arf_5.arf_name;
                for (p = member_string, q = arp->arf_5.arf_name;
                     (len > 0) && (*q != (int) nul_char) && !isspace(*q);
                     ) {
                        MBTOWC(p, q);
                        p++;
                        q++;
                }
                *p++ = (int) parenright_char;
                *p = (int) nul_char;
                name = GETNAME(name_string, FIND_LENGTH);
                /*
                 * [tolik] Fix for dmake bug 1234018.
                 * If name->stat.time is already set, then it should not
                 * be changed. (D)make propogates time stamp for one
                 * member, and when it calls exists() for another member,
                 * the first one may be changed.
                 */
                if(name->stat.time == file_no_time) {
                        name->stat.time.tv_sec = sgetl(arp->arf_5.arf_date);
                        name->stat.time.tv_nsec = LONG_MAX;
                }
                name->is_member = library->is_member;
                member = maybe_append_prop(name, member_prop);
                member->body.member.library = library;
                *--p = (int) nul_char;
                if (member->body.member.member == NULL) {
                        member->body.member.member =
                          GETNAME(member_string, FIND_LENGTH);
                }
                ptr = sgetl(arp->arf_5.arf_size);
                ptr += (ptr & 1);
                if (fseek(arp->fd, ptr, 1) != 0) {
                        goto read_error;
                }
            }
            break;
        case AR_PORT:
            for (;;) {
                    if ((fread((char *) &arp->ar_port,
                               sizeof arp->ar_port,
                               1,
                               arp->fd) != 1) ||
                        !IS_EQUALN(arp->ar_port.ar_fmag,
                                   AR_PORT_END_MAGIC,
                                   sizeof arp->ar_port.ar_fmag)) {
                            if (feof(arp->fd)) {
                                 return succeeded;
                            }
                            fatal(
                                catgets(catd, 1, 28, "Read error in archive `%s'
                                library->string_mb,
                                ftell(arp->fd)
```

```
524                             );
525                     }
526 #if defined(SUN5_0) || defined(linux) //XXX
527                     /* If it's a long name, retrieve it from long name table */
528                     if (arp->ar_port.ar_name[0] == '/') {
529                             /*
530                              * "len" is used for hashing the string.
531                              * We're using "ar_member_name_len" instead of
532                              * the actual name length since it's the longest
533                              * string the "ar" command can handle at this
534                              * point.
535                              */
536                             len = ar_member_name_len;
537                             sscanf(arp->ar_port.ar_name + 1,
538                                    "%ld",
539                                    &offset);
540                             q = *long_names_table + offset;
541                     } else {
542                             q = arp->ar_port.ar_name;
543                             len = sizeof arp->ar_port.ar_name;
544                     }
545 #else
546                     q = arp->ar_port.ar_name;
547                     len = sizeof arp->ar_port.ar_name;
548 #endif
549
550                     for (p = member_string;
551                          (len > 0) &&
552                          (*q != (int) nul_char) &&
553                          !isspace(*q) &&
554                          (*q != (int) slash_char);
555                          ) {
556                             MBTOWC(p, q);
557                             p++;
558                             q++;
559                     }
560                     *p++ = (int) parenright_char;
561                     *p = (int) nul_char;
562                     name = GETNAME(name_string, FIND_LENGTH);
563                     name->is_member = library->is_member;
564                     member = maybe_append_prop(name, member_prop);
565                     member->body.member.library = library;
566                     *--p = (int) nul_char;
567                     if (member->body.member.member == NULL) {
568                             member->body.member.member =
569                               GETNAME(member_string, FIND_LENGTH);
570                     }
571                     if (sscanf(arp->ar_port.ar_date, "%ld", &date) != 1) {
572                             WCSTOMBS(mbs_buffer, name_string);
573                             fatal(catgets(catd, 1, 4, "Bad date field for member
574                                   mbs_buffer,
575                                   library->string_mb);
576                     }
577                     /*
578                      * [tolik] Fix for dmake bug 1234018.
579                      */
580                     if(name->stat.time == file_no_time) {
581                         name->stat.time.tv_sec = date;
582                         name->stat.time.tv_nsec = LONG_MAX;
583                     }
584                     if (sscanf(arp->ar_port.ar_size, "%ld", &ptr) != 1) {
585                             WCSTOMBS(mbs_buffer, name_string);
586                             fatal(catgets(catd, 1, 5, "Bad size field for member
587                                   mbs_buffer,
588                                   library->string_mb);
589                     }
```



```
590                     ptr += (ptr & 1);
591                     if (fseek(arp->fd, ptr, 1) != 0) {
592                             goto read_error;
593                     }
594             }
595             break;
596         }
597
598         /* Only here if fread() [or IS_EQUALN()] failed and not at EOF */
599 read_error:
600         fatal(catgets(catd, 1, 6, "Read error in archive '%s': %s"),
601               library->string_mb,
602               errmsg(errno));
603             /* NOTREACHED */
604 }
605
606
607 /*
608  *      process_long_names_member(arp)
609  *
610  *      If the archive contains members with names longer
611  *      than 15 characters, then it has a special member
612  *      with the name "//        " that contains a table
613  *      of null-terminated long names. This member
614  *      is always the first member, after the symbol table
615  *      if it exists.
616  *
617  *      Parameters:
618  *              arp             Pointer to ar file description block
619  *
620  *      Global variables used:
621  */
622 int
623 process_long_names_member(register Ar *arp, char **long_names_table, char *filen
624 {
625         Ar_port                 *ar_member_header;
626         int                     table_size;
627
628         if (fseek(arp->fd, arp->first_ar_mem, 0) != 0) {
629                 return failed;
630         }
631         if ((ar_member_header =
632              (Ar_port *) alloca((int) sizeof(Ar_port))) == NULL){
633                 perror(catgets(catd, 1, 7, "memory allocation failure"));
634                 return failed;
635         }
636         int ret = read_member_header(ar_member_header, arp->fd, filename);
637         if (ret == failed) {
638                 return failed;
639         } else if(ret == -1) {
640                 /* There is no member header - empty archive */
641                 return succeeded;
642         }
643         /* Do we have special member containing long names? */
644         if (IS_EQUALN(ar_member_header->ar_name,
645                       NOCATGETS("//              "),
646                       16)){
647                 if (sscanf(ar_member_header->ar_size,
648                            "%ld",
649                            &table_size) != 1) {
650                         return failed;
651                 }
652                 *long_names_table = (char *) malloc(table_size);
653                 /* Read the list of long member names into the table */
654                 if (fread(*long_names_table, table_size, 1, arp->fd) != 1) {
655                         return failed;
```

```
656                 }
657                 arp->first_ar_mem = ftell(arp->fd);
658         }
659         return succeeded;
660 }

662 /*
663  *      translate_entry(arp, target, member)
664  *
665  *      Finds the member for one lib.a((entry))
666  *
667  *      Parameters:
668  *              arp             Pointer to ar file description block
669  *              target          Target to find member name for
670  *              member          Property to fill in with info
671  *
672  *      Global variables used:
673  */
674 static void
675 translate_entry(register Ar *arp, Name target, register Property member, char **
676 {
677         register int            len;
678         register int            i;
679         wchar_t                 *member_string;
680         ar_port_word            *offs;
681         int                     strtablen;
682         char                    *syms;          /* string table */
683         char                    *csym;          /* string table */
684         ar_port_word            *offend;        /* end of offsets table */
685         int                     date;
686         register wchar_t        *ap;
687         register char           *hp;
688         int                     maxs;
689         int                     offset;
690         char            buffer[4];

692         if (arp->sym_begin == 0L || arp->num_symbols == 0L) {
693                 fatal(catgets(catd, 1, 8, "Cannot find symbol `%s' in archive `%
694                       member->body.member.entry->string_mb,
695                       member->body.member.library->string_mb);
696         }

698         if (fseek(arp->fd, arp->sym_begin, 0) != 0) {
699                 goto read_error;
700         }
701         member_string = ALLOC_WC((int) ((int) ar_member_name_len * 2));

703         switch (arp->type) {
704         case AR_5:
705                 if ((len = member->body.member.entry->hash.length) > 8) {
706                         len = 8;
707                 }
708                 for (i = 0; i < arp->num_symbols; i++) {
709                         if (fread((char *) &arp->ars_5,
710                                   sizeof arp->ars_5,
711                                   1,
712                                   arp->fd) != 1) {
713                                 goto read_error;
714                         }
715                         if (IS_EQUALN(arp->ars_5.sym_name,
716                                       member->body.member.entry->string_mb,
717                                       len)) {
718                                 if ((fseek(arp->fd,
719                                            sgetl(arp->ars_5.sym_ptr),
720                                            0) != 0) ||
721                                     (fread((char *) &arp->arf_5,
```



```
722                                            sizeof arp->arf_5,
723                                            1,
724                                            arp->fd) != 1)) {
725                                         goto read_error;
726                                 }
727                                 MBSTOWCS(wcs_buffer, arp->arf_5.arf_name);
728                                 (void) wsncpy(member_string,
729                                               wcs_buffer,
730                                               wslen(wcs_buffer));
731                                 member_string[sizeof(arp->arf_5.arf_name)] =
732                                                                 (int) nul_char;
733                                 member->body.member.member =
734                                         GETNAME(member_string, FIND_LENGTH);
735                                 target->stat.time.tv_sec = sgetl(arp->arf_5.arf_
736                                 target->stat.time.tv_nsec = LONG_MAX;
737                                 return;
738                         }
739                 }
740                 break;
741         case AR_PORT:
742                 offs = (ar_port_word *) alloca((int) (arp->num_symbols * AR_PORT
743                 if (fread((char *) offs,
744                           AR_PORT_WORD,
745                           (int) arp->num_symbols,
746                           arp->fd) != arp->num_symbols) {
747                         goto read_error;
748                 }

750                 for(i=0;i<arp->num_symbols;i++) {
751                         *((int*)buffer)=offs[i];
752                         offs[i]=(ar_port_word)sgetl(buffer);
753                 }

755                 strtablen=arp->sym_size-4-(int) (arp->num_symbols * AR_PORT_WORD
756                 syms = (char *) alloca(strtablen);
757                 if (fread(syms,
758                           sizeof (char),
759                           strtablen,
760                           arp->fd) != strtablen) {
761                         goto read_error;
762                 }
763                 offend = &offs[arp->num_symbols];
764                 while (offs < offend) {
765                         maxs = strlen(member->body.member.entry->string_mb);
766                         if(strlen(syms) > maxs)
767                                 maxs = strlen(syms);
768                         if (IS_EQUALN(syms,
769                                       member->body.member.entry->string_mb,
770                                       maxs)) {
771                                 if (fseek(arp->fd,
772                                           (long) *offs,
773                                           0) != 0) {
774                                         goto read_error;
775                                 }
776                                 if ((fread((char *) &arp->ar_port,
777                                            sizeof arp->ar_port,
778                                            1,
779                                            arp->fd) != 1) ||
780                                     !IS_EQUALN(arp->ar_port.ar_fmag,
781                                                AR_PORT_END_MAGIC,
782                                                sizeof arp->ar_port.ar_fmag)) {
783                                         goto read_error;
784                                 }
785                                 if (sscanf(arp->ar_port.ar_date,
786                                            "%ld",
787                                            &date) != 1) {
```

```
788                                     fatal(catgets(catd, 1, 9, "Bad date fiel
789                                           arp->ar_port.ar_name,
790                                           target->string_mb);
791                             }
792 #if defined(SUN5_0) || defined(linux) //XXX
793                     /* If it's a long name, retrieve it from long name table */
794                     if (arp->ar_port.ar_name[0] == '/') {
795                             sscanf(arp->ar_port.ar_name + 1,
796                                    "%ld",
797                                    &offset);
798                             len = ar_member_name_len;
799                             hp = *long_names_table + offset;
800                     } else {
801                             len = sizeof arp->ar_port.ar_name;
802                             hp = arp->ar_port.ar_name;
803                     }
804 #else
805                     hp = arp->ar_port.ar_name;
806 #endif
807                                 ap = member_string;
808                                 while (*hp &&
809                                        (*hp != (int) slash_char) &&
810                                        (ap < &member_string[len])) {
811                                         MBTOWC(ap, hp);
812                                         ap++;
813                                         hp++;
814                                 }
815                                 *ap = (int) nul_char;
816                                 member->body.member.member =
817                                         GETNAME(member_string, FIND_LENGTH);
818                                 target->stat.time.tv_sec = date;
819                                 target->stat.time.tv_nsec = LONG_MAX;
820                                 return;
821                         }
822                         offs++;
823                         while(*syms!='\0') syms++;
824                         syms++;
825                 }
826         }
827         fatal(catgets(catd, 1, 10, "Cannot find symbol '%s' in archive '%s'"),
828               member->body.member.entry->string_mb,
829               member->body.member.library->string_mb);
830         /*NOTREACHED*/

832 read_error:
833         if (ferror(arp->fd)) {
834                 fatal(catgets(catd, 1, 11, "Read error in archive '%s': %s"),
835                       member->body.member.library->string_mb,
836                       errmsg(errno));
837         } else {
838                 fatal(catgets(catd, 1, 12, "Read error in archive '%s': Prematur
839                       member->body.member.library->string_mb);
840         }
841 }

843 /*
844  *      sgetl(buffer)
845  *
846  *      The intent here is to provide a means to make the value of
847  *      bytes in an io-buffer correspond to the value of a long
848  *      in the memory while doing the io a long at a time.
849  *      Files written and read in this way are machine-independent.
850  *
851  *      Return value:
852  *                              Long int read from buffer
853  *      Parameters:
```



```
854  *              buffer          buffer we need to read long int from
855  *
856  *      Global variables used:
857  */
858 static long
859 sgetl(register char *buffer)
860 {
861         register long           w = 0;
862         register int            i = BITSPERBYTE * AR_PORT_WORD;

864         while ((i -= BITSPERBYTE) >= 0) {
865                 w |= (long) ((unsigned char) *buffer++) << i;
866         }
867         return w;
868 }


871 /*
872  *      read_member_header(header, fd, filename)
873  *
874  *      reads the member header for the 4.1.x and SVr4 archives.
875  *
876  *      Return value:
877  *                              fails if read error or member
878  *                              header is not the right format
879  *      Parameters:
880  *              header          There's one before each archive member
881  *              fd              file descriptor for the archive file.
882  *
883  *      Global variables used:
884  */
885 int
886 read_member_header(Ar_port *header, FILE *fd, char* filename)
887 {
888         int num = fread((char *) header, sizeof (Ar_port), 1, fd);
889         if (num != 1 && feof(fd)) {
890                 /* There is no member header - empty archive */
891                 return -1;
892         }
893         if ((num != 1) ||
894             !IS_EQUALN(
895                 AR_PORT_END_MAGIC,
896                 header->ar_fmag,
897                 sizeof (header->ar_fmag)
898             )
899         ) {
900                 fatal(
901                         catgets(catd, 1, 28, "Read error in archive '%s': invali
902                         filename,
903                         ftell(fd)
904                 );
905         }
906         return succeeded;
907 }

909 #endif /* ! codereview */
```

```
**********************************************************
    4834 Wed May 20 11:04:06 2015
new/usr/src/cmd/make/bin/make/common/default.mk.file
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1993 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)default.mk.file 1.4 06/12/12
#

SUFFIXES = .o .c .c~ .cc .cc~ .s .s~ .S .S~ .ln .f .f~ .F .F~ .l .l~ \
        .mod .mod~ .sym .def .def~ .p .p~ .r .r~ .y .y~ .h .h~ .sh .sh~ \
        .cps .cps~
.SUFFIXES: $(SUFFIXES)

# OUTPUT_OPTION should be defined to "-o $@" when
# the default rules are used for non-local files.
OUTPUT_OPTION=

#       C language section.
CC=cc
CFLAGS=
CPPFLAGS=
LINT=lint
LINTFLAGS=
COMPILE.c=$(CC) $(CFLAGS) $(CPPFLAGS) -target $(TARGET_ARCH:-%=%) -c
LINK.c=$(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS) -target $(TARGET_ARCH:-%=%)
LINT.c=$(LINT) $(LINTFLAGS) $(CPPFLAGS) $(TARGET_ARCH)
.c:
        $(LINK.c) -o $@ $< $(LDLIBS)
.c.ln:
        $(LINT.c) $(OUTPUT_OPTION) -i $<
.c.o:
        $(COMPILE.c) $(OUTPUT_OPTION) $<
.c.a:
        $(COMPILE.c) -o $% $<
        $(AR) $(ARFLAGS) $@ $%
        $(RM) $%

#       C language section. yacc.
YACC=yacc
YFLAGS=
YACC.y=$(YACC) $(YFLAGS)
.y:
        $(YACC.y) $<
```



```
        $(LINK.c) -o $@ y.tab.c $(LDLIBS)
        $(RM) y.tab.c
.y.c:
        $(YACC.y) $<
        mv y.tab.c $@
.y.ln:
        $(YACC.y) $<
        $(LINT.c) -o $@ -i y.tab.c
        $(RM) y.tab.c
.y.o:
        $(YACC.y) $<
        $(COMPILE.c) -o $@ y.tab.c
        $(RM) y.tab.c

#       C language section. lex.
LEX=lex
LFLAGS=
LEX.l=$(LEX) $(LFLAGS) -t
.l:
        $(RM) $*.c
        $(LEX.l) $< > $*.c
        $(LINK.c) -o $@ $*.c -ll $(LDLIBS)
        $(RM) $*.c
.l.c :
        $(RM) $@
        $(LEX.l) $< > $@
.l.ln:
        $(RM) $*.c
        $(LEX.l) $< > $*.c
        $(LINT.c) -o $@ -i $*.c
        $(RM) $*.c
.l.o:
        $(RM) $*.c
        $(LEX.l) $< > $*.c
        $(COMPILE.c) -o $@ $*.c
        $(RM) $*.c

#       C++ language section.
CCC=CC
CCFLAGS=
COMPILE.cc=$(CCC) $(CCFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
LINK.cc=$(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)
.cc:
        $(LINK.cc) -o $@ $< $(LDLIBS)
.cc.o:
        $(COMPILE.cc) $(OUTPUT_OPTION) $<
.cc.a:
        $(COMPILE.cc) -o $% $<
        $(AR) $(ARFLAGS) $@ $%
        $(RM) $%

#       FORTRAN section.
FC=f77
FFLAGS=
COMPILE.f=$(FC) $(FFLAGS) $(TARGET_ARCH) -c
LINK.f=$(FC) $(FFLAGS) $(LDFLAGS) $(TARGET_ARCH)
COMPILE.F=$(FC) $(FFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
LINK.F=$(FC) $(FFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)
.f:
        $(LINK.f) -o $@ $< $(LDLIBS)
.f.o:
        $(COMPILE.f) $(OUTPUT_OPTION) $<
.f.a:
        $(COMPILE.f) -o $% $<
        $(AR) $(ARFLAGS) $@ $%
        $(RM) $%
```

```
.F:
	$(LINK.F) -o $@ $< $(LDLIBS)
.F.o:
	$(COMPILE.F) $(OUTPUT_OPTION) $<
.F.a:
	$(COMPILE.F) -o $% $<
	$(AR) $(ARFLAGS) $@ $%
	$(RM) $%

#	FORTRAN section. ratfor.
RFLAGS=
COMPILE.r=$(FC) $(FFLAGS) $(RFLAGS) $(TARGET_ARCH) -c
LINK.r=$(FC) $(FFLAGS) $(RFLAGS) $(LDFLAGS) $(TARGET_ARCH)
.r:
	$(LINK.r) -o $@ $< $(LDLIBS)
.r.o:
	$(COMPILE.r) $(OUTPUT_OPTION) $<
.r.a:
	$(COMPILE.r) -o $% $<
	$(AR) $(ARFLAGS) $@ $%
	$(RM) $%

#	Modula-2 section.
M2C=m2c
M2FLAGS=
MODFLAGS=
DEFFLAGS=
COMPILE.def=$(M2C) $(M2FLAGS) $(DEFFLAGS) $(TARGET_ARCH)
COMPILE.mod=$(M2C) $(M2FLAGS) $(MODFLAGS) $(TARGET_ARCH)
.def.sym:
	$(COMPILE.def) -o $@ $<
.mod:
	$(COMPILE.mod) -o $@ -e $@ $<
.mod.o:
	$(COMPILE.mod) -o $@ $<
.mod.a:
	$(COMPILE.mod) -o $% $<
	$(AR) $(ARFLAGS) $@ $%
	$(RM) $%

#	Pascal section.
PC=pc
PFLAGS=
COMPILE.p=$(PC) $(PFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
LINK.p=$(PC) $(PFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)
.p:
	$(LINK.p) -o $@ $< $(LDLIBS)
.p.o:
	$(COMPILE.p) $(OUTPUT_OPTION) $<
.p.a:
	$(COMPILE.p) -o $% $<
	$(AR) $(ARFLAGS) $@ $%
	$(RM) $%

#	Assembly section.
AS=as
ASFLAGS=
COMPILE.s=$(AS) $(ASFLAGS) $(TARGET_MACH)
COMPILE.S=$(CC) $(ASFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
.s.o:
	$(COMPILE.s) -o $@ $<
.s.a:
	$(COMPILE.s) -o $% $<
	$(AR) $(ARFLAGS) $@ $%
	$(RM) $%
.S.o:
```



```
	$(COMPILE.S) -o $@ $<
.S.a:
	$(COMPILE.S) -o $% $<
	$(AR) $(ARFLAGS) $@ $%
	$(RM) $%

#	Shell section.
.sh:
	$(RM) $@
	cat $< > $@
	chmod +x $@

#	NeWS section
CPS=cps
CPSFLAGS=
.cps.h:
	$(CPS) $(CPSFLAGS) $*.cps

#	Miscellaneous section.
LD=ld
LDFLAGS=
LDLIBS=
MAKE=make
RM=rm -f
AR=ar
ARFLAGS=rv
GET=/usr/sccs/get
GFLAGS=

markfile.o:	markfile
	echo "static char _sccsid[] = \"‘grep @’(#)’ markfile‘\";" > markfile.c
	cc -c markfile.c
	$(RM) markfile.c

SCCSFLAGS=
SCCSGETFLAGS=-s
.SCCS_GET:
	sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $@ -G$@
#endif /* ! codereview */
```

```
**********************************************************
    3480 Wed May 20 11:04:07 2015
new/usr/src/cmd/make/bin/make/common/depvar.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 1995 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)depvar.cc 1.14 06/12/12
 */

#pragma ident   "@(#)depvar.cc  1.14    06/12/12"

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

/*
 * This file deals with "Dependency Variables".
 * The "-V var" command line option is used to indicate
 * that var is a dependency variable.  Used in conjunction with
 * the -P option the user is asking if the named variables affect
 * the dependencies of the given target.
 */

struct _Depvar {
        Name            name;           /* Name of variable */
        struct _Depvar  *next;          /* Linked list */
        Boolean         cmdline;        /* Macro defined on the cmdline? */
};

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);
```

```
128                         }
129                         break;
130                 }
131         }
132 }
133 #endif

135 /*
136  * Print the results.  If any of the Dependency Variables
137  * affected the dependencies then the dependencies potentially
138  * differ because of these variables.
139  */
140 void
141 depvar_print_results(void)
142 {
143         if (variant_deps) {
144                 printf(catgets(catd, 1, 234, "differ\n"));
145         } else {
146                 printf(catgets(catd, 1, 235, "same\n"));
147         }
148 }

150 #endif /* ! codereview */
```

```
**********************************************************
   15498 Wed May 20 11:04:07 2015
new/usr/src/cmd/make/bin/make/common/dist.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)dist.cc 1.35 06/12/12
 */

#pragma ident   "@(#)dist.cc    1.25    96/03/12"

#ifdef DISTRIBUTED
/*
 *      dist.cc
 *
 *      Deal with the distributed processing
 */

#include <avo/err.h>
#include <avo/find_dir.h>
#include <avo/util.h>
#include <dm/Avo_AcknowledgeMsg.h>
#include <dm/Avo_DoJobMsg.h>
#include <dm/Avo_JobResultMsg.h>
#include <mk/defs.h>
#include <mksh/misc.h>          /* getmem() */
#include <rw/pstream.h>
#include <rw/queuecol.h>
#include <rw/xdrstrea.h>
#include <signal.h>
#ifdef linux
#include <sstream>
using namespace std;
#else
#include <strstream.h>
#endif
#include <sys/stat.h>           /* stat() */
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <errno.h>
```



```
/*
 * Defined macros
 */

#define AVO_BLOCK_INTERUPTS sigfillset(&newset) ; \
        sigprocmask(SIG_SETMASK, &newset, &oldset)

#define AVO_UNBLOCK_INTERUPTS \
        sigprocmask(SIG_SETMASK, &oldset, &newset)


/*
 * typedefs & structs
 */

/*
 * Static variables
 */
int             dmake_ifd;
FILE*           dmake_ifp;
XDR             xdrs_in;

int             dmake_ofd;
FILE*           dmake_ofp;
XDR             xdrs_out;

// These instances are required for the RWfactory.
static Avo_JobResultMsg dummyJobResultMsg;
static Avo_AcknowledgeMsg dummyAcknowledgeMsg;
static int firstAcknowledgeReceived = 0;

int             rxmPid = 0;

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

/*
 * void
 * startup_rxm(void)
 *
 * When startup_rxm() is called, read_command_options() and
 * read_files_and_state() have already been read, so DMake
 * will now know what options to pass to rxm.
 *
 * rxm
 *      [ -k ] [ -n ]
 *      [ -c <dmake_rcfile> ]
 *      [ -g <dmake_group> ]
 *      [ -j <dmake_max_jobs> ]
 *      [ -m <dmake_mode> ]
 *      [ -o <dmake_odir> ]
 *     <read_fd> <write_fd>
 *
 * rxm will, among other things, read the rc file.
 *
 */
void
startup_rxm(void)
{
        Name            dmake_name;
        Name            dmake_value;
        Avo_err         *find_run_dir_err;
        int             pipe1[2], pipe2[2];
        Property        prop;
```

```
128         char            *run_dir;
129         char            rxm_command[MAXPATHLEN];
130         int             rxm_debug = 0;

132         int             length;
133         char *          env;

135         firstAcknowledgeReceived = 0;
136         /*
137          * Create two pipes, one for dmake->rxm, and one for rxm->dmake.
138          * pipe1 is dmake->rxm,
139          * pipe2 is rxm->dmake.
140          */
141         if ((pipe(pipe1) < 0) || (pipe(pipe2) < 0)) {
142                 fatal(catgets(catd, 1, 245, "pipe() failed: %s"), errmsg(errno))
143         }

145         set_dmake_env_vars();

147         if ((rxmPid = fork()) < 0) {    /* error */
148                 fatal(catgets(catd, 1, 246, "fork() failed: %s"), errmsg(errno))
149         } else if (rxmPid > 0) {        /* parent, dmake */
150                 dmake_ofd = pipe1[1];   // write side of pipe
151                 if (!(dmake_ofp = fdopen(dmake_ofd, "a"))) {
152                         fatal(catgets(catd, 1, 247, "fdopen() failed: %s"), errm
153                 }
154                 xdrstdio_create(&xdrs_out, dmake_ofp, XDR_ENCODE);

156                 dmake_ifd = pipe2[0];   // read side of pipe
157                 if (!(dmake_ifp = fdopen(dmake_ifd, "r"))) {
158                         fatal(catgets(catd, 1, 248, "fdopen() failed: %s"), errm
159                 }
160                 xdrstdio_create(&xdrs_in, dmake_ifp, XDR_DECODE);

162                 close(pipe1[0]);        // read side
163                 close(pipe2[1]);        // write side
164         } else {                        /* child, rxm */
165                 close(pipe1[1]);        // write side
166                 close(pipe2[0]);        // read side

168                 /* Find the run directory of dmake, for rxm. */
169                 find_run_dir_err = avo_find_run_dir(&run_dir);
170                 if (find_run_dir_err) {
171                         delete find_run_dir_err;
172                         /* Use the path to find rxm. */
173                         (void) sprintf(rxm_command, NOCATGETS("rxm"));
174                 } else {
175                         /* Use the run dir of dmake for rxm. */
176                         (void) sprintf(rxm_command, NOCATGETS("%s/rxm"), run_dir
177                 }

179                 if (continue_after_error) {
180                         (void) strcat(rxm_command, NOCATGETS(" -k"));
181                 }
182                 if (do_not_exec_rule) {
183                         (void) strcat(rxm_command, NOCATGETS(" -n"));
184                 }
185                 if (rxm_debug) {
186                         (void) strcat(rxm_command, NOCATGETS(" -S"));
187                 }
188                 if (send_mtool_msgs) {
189                         (void) sprintf(&rxm_command[strlen(rxm_command)],
190                                        NOCATGETS(" -O %d"),
191                                        mtool_msgs_fd);
192                 }
193                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
```



```
194                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
195                 if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
196                     ((dmake_value = prop->body.macro.value) != NULL)) {
197                         (void) sprintf(&rxm_command[strlen(rxm_command)],
198                                        NOCATGETS(" -c %s"),
199                                        dmake_value->string_mb);
200                 } else {
201                         length = 2 + strlen(NOCATGETS("DMAKE_RCFILE"));
202                         env = getmem(length);
203                         (void) sprintf(env,
204                                        "%s=",
205                                        NOCATGETS("DMAKE_RCFILE"));
206                         (void) putenv(env);
207                 }
208                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
209                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
210                 if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
211                     ((dmake_value = prop->body.macro.value) != NULL)) {
212                         (void) sprintf(&rxm_command[strlen(rxm_command)],
213                                        NOCATGETS(" -g %s"),
214                                        dmake_value->string_mb);
215                 } else {
216                         length = 2 + strlen(NOCATGETS("DMAKE_GROUP"));
217                         env = getmem(length);
218                         (void) sprintf(env,
219                                        "%s=",
220                                        NOCATGETS("DMAKE_GROUP"));
221                         (void) putenv(env);
222                 }
223                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
224                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
225                 if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
226                     ((dmake_value = prop->body.macro.value) != NULL)) {
227                         (void) sprintf(&rxm_command[strlen(rxm_command)],
228                                        NOCATGETS(" -j %s"),
229                                        dmake_value->string_mb);
230                 } else {
231                         length = 2 + strlen(NOCATGETS("DMAKE_MAX_JOBS"));
232                         env = getmem(length);
233                         (void) sprintf(env,
234                                        "%s=",
235                                        NOCATGETS("DMAKE_MAX_JOBS"));
236                         (void) putenv(env);
237                 }
238                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
239                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
240                 if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
241                     ((dmake_value = prop->body.macro.value) != NULL)) {
242                         (void) sprintf(&rxm_command[strlen(rxm_command)],
243                                        NOCATGETS(" -m %s"),
244                                        dmake_value->string_mb);
245                 } else {
246                         length = 2 + strlen(NOCATGETS("DMAKE_MODE"));
247                         env = getmem(length);
248                         (void) sprintf(env,
249                                        "%s=",
250                                        NOCATGETS("DMAKE_MODE"));
251                         (void) putenv(env);
252                 }
253                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
254                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
255                 if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
256                     ((dmake_value = prop->body.macro.value) != NULL)) {
257                         (void) sprintf(&rxm_command[strlen(rxm_command)],
258                                        NOCATGETS(" -o %s"),
259                                        dmake_value->string_mb);
```

```
260                 } else {
261                         length = 2 + strlen(NOCATGETS("DMAKE_ODIR"));
262                         env = getmem(length);
263                         (void) sprintf(env,
264                                        "%s=",
265                                        NOCATGETS("DMAKE_ODIR"));
266                         (void) putenv(env);
267                 }

269                 (void) sprintf(&rxm_command[strlen(rxm_command)],
270                                NOCATGETS(" %d %d"),
271                                pipe1[0], pipe2[1]);
272 #ifdef linux
273                 execl(NOCATGETS("/bin/sh"),
274 #else
275                 execl(NOCATGETS("/usr/bin/sh"),
276 #endif
277                       NOCATGETS("sh"),
278                       NOCATGETS("-c"),
279                       rxm_command,
280                       (char *)NULL);
281                 _exit(127);
282         }
283 }

285 /*
286  * static void
287  * set_dmake_env_vars()
288  *
289  * Sets the DMAKE_* environment variables for rxm and rxs.
290  *      DMAKE_PWD
291  *      DMAKE_NPWD
292  *      DMAKE_UMASK
293  *      DMAKE_SHELL
294  */
295 static void
296 set_dmake_env_vars()
297 {
298         char            *current_netpath;
299         char            *current_path;
300         static char     *env;
301         int             length;
302         char            netpath[MAXPATHLEN];
303         mode_t          um;
304         char            um_buf[MAXPATHLEN];
305         Name            dmake_name;
306         Name            dmake_value;
307         Property        prop;

309 #ifdef REDIRECT_ERR
310         /* Set __DMAKE_REDIRECT_STDERR */
311         length = 2 + strlen(NOCATGETS("__DMAKE_REDIRECT_STDERR")) + 1;
312         env = getmem(length);
313         (void) sprintf(env,
314                        "%s=%s",
315                        NOCATGETS("__DMAKE_REDIRECT_STDERR"),
316                        out_err_same ? NOCATGETS("0") : NOCATGETS("1"));
317         (void) putenv(env);
318 #endif

320         /* Set DMAKE_PWD to the current working directory */
321         current_path = get_current_path();
322         length = 2 + strlen(NOCATGETS("DMAKE_PWD")) + strlen(current_path);
323         env = getmem(length);
324         (void) sprintf(env,
325                        "%s=%s",
```



```
326                        NOCATGETS("DMAKE_PWD"),
327                        current_path);
328         (void) putenv(env);

330         /* Set DMAKE_NPWD to machine:pathname */
331         if (avo_path_to_netpath(current_path, netpath)) {
332                 current_netpath = netpath;
333         } else {
334                 current_netpath = current_path;
335         }
336         length = 2 + strlen(NOCATGETS("DMAKE_NPWD")) + strlen(current_netpath);
337         env = getmem(length);
338         (void) sprintf(env,
339                        "%s=%s",
340                        NOCATGETS("DMAKE_NPWD"),
341                        current_netpath);
342         (void) putenv(env);

344         /* Set DMAKE_UMASK to the value of umask */
345         um = umask(0);
346         umask(um);
347         (void) sprintf(um_buf, NOCATGETS("%ul"), um);
348         length = 2 + strlen(NOCATGETS("DMAKE_UMASK")) + strlen(um_buf);
349         env = getmem(length);
350         (void) sprintf(env,
351                        "%s=%s",
352                        NOCATGETS("DMAKE_UMASK"),
353                        um_buf);
354         (void) putenv(env);

356         if (((prop = get_prop(shell_name->prop, macro_prop)) != NULL) &&
357             ((dmake_value = prop->body.macro.value) != NULL)) {
358                 length = 2 + strlen(NOCATGETS("DMAKE_SHELL")) +
359                              strlen(dmake_value->string_mb);
360                 env = getmem(length);
361                 (void) sprintf(env,
362                                "%s=%s",
363                                NOCATGETS("DMAKE_SHELL"),
364                                dmake_value->string_mb);
365                 (void) putenv(env);
366         }
367         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
368         dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
369         if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
370             ((dmake_value = prop->body.macro.value) != NULL)) {
371                 length = 2 + strlen(NOCATGETS("DMAKE_OUTPUT_MODE")) +
372                              strlen(dmake_value->string_mb);
373                 env = getmem(length);
374                 (void) sprintf(env,
375                                "%s=%s",
376                                NOCATGETS("DMAKE_OUTPUT_MODE"),
377                                dmake_value->string_mb);
378                 (void) putenv(env);
379         }
380 }

382 /*
383  * void
384  * distribute_rxm(Avo_DoJobMsg *dmake_job_msg)
385  *
386  * Write the DMake rule to be distributed down the pipe to rxm.
387  *
388  */
389 void
390 distribute_rxm(Avo_DoJobMsg *dmake_job_msg)
391 {
```

```
392         /* Add all dynamic env vars to the dmake_job_msg. */
393         setvar_envvar(dmake_job_msg);

395         /*
396          * Copying dosys()...
397          * Stat .make.state to see if we'll need to reread it later
398          */
399         make_state->stat.time = file_no_time;
400         (void)exists(make_state);
401         make_state_before = make_state->stat.time;

403         // Wait for the first Acknowledge message from the rxm process
404         // before sending the first message.
405         if (!firstAcknowledgeReceived) {
406                 firstAcknowledgeReceived++;
407                 Avo_AcknowledgeMsg *msg = getAcknowledgeMsg();
408                 if (msg) {
409                         delete msg;
410                 }
411         }

413         RWCollectable *doJobMsg = (RWCollectable *)dmake_job_msg;
414         sigset_t newset;
415         sigset_t oldset;

417         AVO_BLOCK_INTERUPTS;
418         int xdrResult = xdr(&xdrs_out, doJobMsg);

420         if (xdrResult) {
421                 fflush(dmake_ofp);
422                 AVO_UNBLOCK_INTERUPTS;
423         } else {
424                 AVO_UNBLOCK_INTERUPTS;
425                 fatal(catgets(catd, 1, 249, "Couldn't send the job request to rx
426         }

428         delete dmake_job_msg;
429 }

431 // Queue for JobResult messages.
432 static RWSlistCollectablesQueue jobResultQueue;

434 // Queue for Acknowledge messages.
435 static RWSlistCollectablesQueue acknowledgeQueue;

437 // Read a message from the rxm process, and put it into a queue, by
438 // message type. Return the message type.

440 int
441 getRxmMessage(void)
442 {
443         RWCollectable   *msg = (RWCollectable *)0;
444         int             msgType = 0;
445 //      sigset_t        newset;
446 //      sigset_t        oldset;

448         // It seems unnecessarily to block interrupts here because
449         // any nonignored signal means exit for dmake in distributed mode.
450 //      AVO_BLOCK_INTERUPTS;
451         int xdrResult = xdr(&xdrs_in, msg);
452 //      AVO_UNBLOCK_INTERUPTS;

454         if (xdrResult) {
455                 switch(msg->isA()) {
456                 case __AVO_ACKNOWLEDGEMSG:
457                         acknowledgeQueue.append(msg);
```



```
458                         msgType = __AVO_ACKNOWLEDGEMSG;
459                         break;
460                 case __AVO_JOBRESULTMSG:
461                         jobResultQueue.append(msg);
462                         msgType = __AVO_JOBRESULTMSG;
463                         break;
464                 default:
465                         warning(catgets(catd, 1, 291, "Unknown message on rxm in
466                         msgType = 0;
467                         break;
468                 }
469         } else {
470                 if (errno == EINTR) {
471                         fputs(NOCATGETS("dmake: Internal error: xdr() has been i
472                 }
473                 fatal(catgets(catd, 1, 250, "Couldn't receive message from rxm")
474         }

476         return msgType;
477 }

479 // Get a JobResult message from it's queue, and
480 // if the queue is empty, call the getRxmMessage() function until
481 // a JobResult message shows.

483 Avo_JobResultMsg *
484 getJobResultMsg(void)
485 {
486         RWCollectable *msg = 0;

488         if (!(msg = jobResultQueue.get())) {
489                 while (getRxmMessage() != __AVO_JOBRESULTMSG);
490                 msg = jobResultQueue.get();
491         }

493         return (Avo_JobResultMsg *)msg;
494 }

496 // Get an Acknowledge message from it's queue, and
497 // if the queue is empty, call the getRxmMessage() function until
498 // a Acknowledge message shows.

500 Avo_AcknowledgeMsg *
501 getAcknowledgeMsg(void)
502 {
503         RWCollectable *msg = 0;

505         if (!(msg = acknowledgeQueue.get())) {
506                 while (getRxmMessage() != __AVO_ACKNOWLEDGEMSG);
507                 msg = acknowledgeQueue.get();
508         }

510         return (Avo_AcknowledgeMsg *)msg;
511 }


514 /*
515  * Doname
516  * await_dist(Boolean waitflg)
517  *
518  * Waits for distributed children to exit and finishes their processing.
519  * Rxm will send a msg down the pipe when a child is done.
520  *
521  */
522 Doname
523 await_dist(Boolean waitflg)
```

```
524 {
525         Avo_JobResultMsg        *dmake_result_msg;
526         int                     job_msg_id;
527         Doname                  result = build_ok;
528         int                     result_msg_cmd_status;
529         int                     result_msg_job_status;
530         Running                 rp;

532         while (!(dmake_result_msg = getJobResultMsg()));
533         job_msg_id = dmake_result_msg->getId();
534         result_msg_cmd_status = dmake_result_msg->getCmdStatus();
535         result_msg_job_status = dmake_result_msg->getJobStatus();

537         if (waitflg) {
538                 result = (result_msg_cmd_status == 0) ? build_ok : build_failed;

540 #ifdef PRINT_EXIT_STATUS
541                 if (result == build_ok) {
542                         warning(NOCATGETS("I'm in await_dist(), waitflg is true,
543                 } else {
544                         warning(NOCATGETS("I'm in await_dist(), waitflg is true,
545                 }
546 #endif

548         } else {
549                 for (rp = running_list;
550                      (rp != NULL) && (job_msg_id != rp->job_msg_id);
551                      rp = rp->next) {
552                 }
553                 if (rp == NULL) {
554                         fatal(catgets(catd, 1, 251, "Internal error: returned ch
555                 } else {
556                         /* XXX - This may not be correct! */
557                         if (result_msg_job_status == RETURNED) {
558                                 rp->state = build_ok;
559                         } else {
560                                 rp->state = (result_msg_cmd_status == 0) ? build
561                         }
562                         result = rp->state;

564 #ifdef PRINT_EXIT_STATUS
565                         if (result == build_ok) {
566                                 warning(NOCATGETS("I'm in await_dist(), waitflg
567                         } else {
568                                 warning(NOCATGETS("I'm in await_dist(), waitflg
569                         }
570 #endif

572                 }
573                 parallel_process_cnt--;
574         }
575         delete dmake_result_msg;
576         return result;
577 }

579 #endif


582 #endif /* ! codereview */
```

```
**********************************************************
    1004 Wed May 20 11:04:07 2015
new/usr/src/cmd/make/bin/make/common/dmake.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)dmake.cc 1.2 06/12/12
 */

#pragma ident   "@(#)dmake.cc   1.2     06/12/12"
#endif /* ! codereview */
```

```
**********************************************************
  105477 Wed May 20 11:04:07 2015
new/usr/src/cmd/make/bin/make/common/doname.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)doname.cc 1.115 06/12/12
 */

#pragma ident   "@(#)doname.cc  1.115   06/12/12"

/*
 *      doname.c
 *
 *      Figure out which targets are out of date and rebuild them
 */

/*
 * Included files
 */
#include <avo/avo_alloca.h>             /* alloca() */
#if defined(TEAMWARE_MAKE_CMN)
#include <avo/util.h>           /* avo_get_user(), avo_hostname() */
#endif

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
#       include <avo/strings.h> /* AVO_STRDUP() */
#       include <dm/Avo_MToolJobResultMsg.h>
#       include <dm/Avo_MToolJobStartMsg.h>
#       include <dm/Avo_MToolRsrcInfoMsg.h>
#       include <dm/Avo_macro_defs.h> /* AVO_BLOCK_INTERUPTS & AVO_UNBLOCK_INTER
#       include <dmthread/Avo_ServerState.h>
#       include <rw/pstream.h>
#       include <rw/xdrstrea.h>
#endif

#include <fcntl.h>
#include <mk/defs.h>
#include <mksh/i18n.h>          /* get_char_semantics_value() */
#include <mksh/macro.h>         /* getvar(), expand_value() */
#include <mksh/misc.h>          /* getmem() */
#include <poll.h>
```



```
#ifdef PARALLEL
#       include <rx/api.h>
#endif

#include <signal.h>

#ifndef HP_UX
#       include <stropts.h>
#endif

#include <sys/errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>        /* uname() */
#include <sys/wait.h>
#include <unistd.h>             /* close() */

/*
 * Defined macros
 */
#ifndef PARALLEL
#       define LOCALHOST "localhost"
#endif

#define MAXRULES 100

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
#define SEND_MTOOL_MSG(cmds) \
        if (send_mtool_msgs) { \
                cmds \
        }
#else
#define SEND_MTOOL_MSG(cmds)
#endif

// Sleep for .1 seconds between stat()'s
const int       STAT_RETRY_SLEEP_TIME = 100000;

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static char     hostName[MAXNAMELEN] = "";
static char     userName[MAXNAMELEN] = "";

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
        static FILE     *mtool_msgs_fp;
        static XDR      xdrs;
        static int      sent_rsrc_info_msg = 0;
#endif

static int      second_pass = 0;

/*
 * File table of contents
 */
extern  Doname          doname_check(register Name target, register Boolean do_g
extern  Doname          doname(register Name target, register Boolean do_get, re
static  Boolean         check_dependencies(Doname *result, Property line, Boolea
void            dynamic_dependencies(Name target);
static  Doname          run_command(register Property line, Boolean print_machin
extern  Doname          execute_serial(Property line);
```

```
128 extern  Name            vpath_translation(register Name cmd);
129 extern  void            check_state(Name temp_file_name);
130 static  void            read_dependency_file(register Name filename);
131 static  void            check_read_state_file(void);
132 static  void            do_assign(register Name line, register Name target);
133 static  void            build_command_strings(Name target, register Property lin
134 static  Doname          touch_command(register Property line, register Name targ
135 extern  void            update_target(Property line, Doname result);
136 static  Doname          sccs_get(register Name target, register Property *comman
137 extern  void            read_directory_of_file(register Name file);
138 static  void            add_pattern_conditionals(register Name target);
139 extern  void            set_locals(register Name target, register Property old_l
140 extern  void            reset_locals(register Name target, register Property old
141 extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
142 static  void            delete_query_chain(Chain ch);

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


148 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
149         static void             append_job_result_msg(Avo_MToolJobResultMsg *job
150         static int              pollResults(char *outFn, char *errFn, char *host
151         static void             pollResultsAction(char *outFn, char *errFn);
152         static void             rxmGetNextResultsBlock(int fd);
153         static int              us_sleep(unsigned int nusecs);
154         extern "C" void         Avo_PollResultsAction_Sigusr1Handler(int foo);
155 #endif

157 /*
158  * DONE.
159  *
160  *      doname_check(target, do_get, implicit, automatic)
161  *
162  *      Will call doname() and then inspect the return value
163  *
164  *      Return value:
165  *                              Indication if the build failed or not
166  *
167  *      Parameters:
168  *              target          The target to build
169  *              do_get          Passed thru to doname()
170  *              implicit        Passed thru to doname()
171  *              automatic       Are we building a hidden dependency?
172  *
173  *      Global variables used:
174  *              build_failed_seen       Set if -k is on and error occurs
175  *              continue_after_error    Indicates that -k is on
176  *              report_dependencies     No error msg if -P is on
177  */
178 Doname
179 doname_check(register Name target, register Boolean do_get, register Boolean imp
180 {
181         int first_time = 1;
182         (void) fflush(stdout);
183 try_again:
184         switch (doname(target, do_get, implicit, automatic)) {
185         case build_ok:
186                 second_pass = 0;
187                 return build_ok;
188         case build_running:
189                 second_pass = 0;
190                 return build_running;
191         case build_failed:
192                 if (!continue_after_error) {
193                         fatal(catgets(catd, 1, 13, "Target '%s' not remade becau
```



```
194                                 target->string_mb);
195                 }
196                 build_failed_seen = true;
197                 second_pass = 0;
198                 return build_failed;
199         case build_dont_know:
200                 /*
201                  * If we can't figure out how to build an automatic
202                  * (hidden) dependency, we just ignore it.
203                  * We later declare the target to be out of date just in
204                  * case something changed.
205                  * Also, don't complain if just reporting the dependencies
206                  * and not building anything.
207                  */
208                 if (automatic || (report_dependencies_level > 0)) {
209                         second_pass = 0;
210                         return build_dont_know;
211                 }
212                 if(first_time) {
213                         first_time = 0;
214                         second_pass = 1;
215                         goto try_again;
216                 }
217                 second_pass = 0;
218                 if (continue_after_error && !svr4) {
219                         warning(catgets(catd, 1, 14, "Don't know how to make tar
220                                 target->string_mb);
221                         build_failed_seen = true;
222                         return build_failed;
223                 }
224                 fatal(catgets(catd, 1, 15, "Don't know how to make target '%s'")
225                 break;
226         }
227 #ifdef lint
228         return build_failed;
229 #endif
230 }


233 void
234 enter_explicit_rule_from_dynamic_rule(Name target, Name source)
235 {
236         Property line, source_line;
237         Dependency dependency;

239         source_line = get_prop(source->prop, line_prop);
240         line = maybe_append_prop(target, line_prop);
241         line->body.line.sccs_command = false;
242         line->body.line.target = target;
243         if (line->body.line.command_template == NULL) {
244                 line->body.line.command_template = source_line->body.line.comman
245                 for (dependency = source_line->body.line.dependencies;
246                      dependency != NULL;
247                      dependency = dependency->next) {
248                         enter_dependency(line, dependency->name, false);
249                 }
250                 line->body.line.less = target;
251         }
252         line->body.line.percent = NULL;
253 }



257 Name
258 find_dyntarget(Name target)
259 {
```

```
260         Dyntarget               p;
261         int                     i;
262         String_rec              string;
263         wchar_t                 buffer[STRING_BUFFER_LENGTH];
264         wchar_t                 *pp, * bufend;
265         wchar_t                 tbuffer[MAXPATHLEN];
266         Wstring                 wcb(target);

268         for (p = dyntarget_list; p != NULL; p = p->next) {
269                 INIT_STRING_FROM_STACK(string, buffer);
270                 expand_value(p->name, &string, false);
271                 i = 0;
272                 pp = string.buffer.start;
273                 bufend = pp + STRING_BUFFER_LENGTH;
274                 while((*pp != nul_char) && (pp < bufend)) {
275                         if(iswspace(*pp)) {
276                                 tbuffer[i] = nul_char;
277                                 if(i > 0) {
278                                         if (wcb.equal(tbuffer)) {
279                                                 enter_explicit_rule_from_dynamic
280                                                 return(target);
281                                         }
282                                 }
283                                 pp++;
284                                 i = 0;
285                                 continue;
286                         }
287                         tbuffer[i] = *pp;
288                         i++;
289                         pp++;
290                         if(*pp == nul_char) {
291                                 tbuffer[i] = nul_char;
292                                 if(i > 0) {
293                                         if (wcb.equal(tbuffer)) {
294                                                 enter_explicit_rule_from_dynamic
295                                                 return(target);
296                                         }
297                                 }
298                                 break;
299                         }
300                 }
301         }
302         return(NULL);
303 }

305 /*
306  * DONE.
307  *
308  *      doname(target, do_get, implicit)
309  *
310  *      Chases all files the target depends on and builds any that
311  *      are out of date. If the target is out of date it is then rebuilt.
312  *
313  *      Return value:
314  *                              Indiates if build failed or nt
315  *
316  *      Parameters:
317  *              target          Target to build
318  *              do_get          Run sccs get is nessecary
319  *              implicit        doname is trying to find an implicit rule
320  *
321  *      Global variables used:
322  *              assign_done     True if command line assgnment has happened
323  *              commands_done   Preserved for the case that we need local value
324  *              debug_level     Should we trace make's actions?
325  *              default_rule    The rule for ".DEFAULT", used as last resort
```



```
326  *              empty_name      The Name "", used when looking for single sfx
327  *              keep_state      Indicates that .KEEP_STATE is on
328  *              parallel        True if building in parallel
329  *              recursion_level Used for tracing
330  *              report_dependencies make -P is on
331  */
332 Doname
333 doname(register Name target, register Boolean do_get, register Boolean implicit,
334 {
335         Doname                  result = build_dont_know;
336         Chain                   out_of_date_list = NULL;
337 #ifdef TEAMWARE_MAKE_CMN
338         Chain                   target_group;
339 #endif
340         Property                old_locals = NULL;
341         register Property       line;
342         Property                command = NULL;
343         register Dependency     dependency;
344         Name                    less = NULL;
345         Name                    true_target = target;
346         Name                    *automatics = NULL;
347         register int            auto_count;
348         Boolean                 rechecking_target = false;
349         Boolean                 saved_commands_done;
350         Boolean                 restart = false;
351         Boolean                 save_parallel = parallel;
352 #ifdef NSE
353         Boolean                 save_readdep;
354 #endif
355         Boolean                 doing_subtree = false;

357         Boolean                 recheck_conditionals = false;

359         if (target->state == build_running) {
360                 return build_running;
361         }
362         line = get_prop(target->prop, line_prop);
363 #ifdef TEAMWARE_MAKE_CMN
364         if (line != NULL) {
365                 /*
366                  * If this target is a member of target group and one of the
367                  * other members of the group is running, mark this target
368                  * as running.
369                  */
370                 for (target_group = line->body.line.target_group;
371                      target_group != NULL;
372                      target_group = target_group->next) {
373                         if (is_running(target_group->name)) {
374                                 target->state = build_running;
375                                 add_pending(target,
376                                             recursion_level,
377                                             do_get,
378                                             implicit,
379                                             false);
380                                 return build_running;
381                         }
382                 }
383         }
384 #ifdef NSE
385         nse_check_file_backquotes(target->string);
386 #endif
387 #endif
388         /*
389          * If the target is a constructed one for a "::" target,
390          * we need to consider that.
391          */
```

```
392         if (target->has_target_prop) {
393                 true_target = get_prop(target->prop,
394                                        target_prop)->body.target.target;
395                 if (true_target->colon_splits > 0) {
396                         /* Make sure we have a valid time for :: targets */
397                         Property        time;

399                         time = get_prop(true_target->prop, time_prop);
400                         if (time != NULL) {
401                                 true_target->stat.time = time->body.time.time;
402                         }
403                 }
404         }
405         (void) exists(true_target);
406         /*
407          * If the target has been processed, we don't need to do it again,
408          * unless it depends on conditional macros or a delayed assignment,
409          * or it has been done when KEEP_STATE is on.
410          */
411         if (target->state == build_ok) {
412                 if((!keep_state || (!target->depends_on_conditional && !assign_d
413                         return build_ok;
414                 } else {
415                         recheck_conditionals = true;
416                 }
417         }
418         if (target->state == build_subtree) {
419                 /* A dynamic macro subtree is being built */
420                 target->state = build_dont_know;
421                 doing_subtree = true;
422                 if (!target->checking_subtree) {
423                         /*
424                          * This target has been started before and therefore
425                          * not all dependencies have to be built.
426                          */
427                         restart = true;
428                 }
429         } else if (target->state == build_pending) {
430                 target->state = build_dont_know;
431                 restart = true;
432 /*
433 #ifdef TEAMWARE_MAKE_CMN
434         } else if (parallel &&
435                    keep_state &&
436                    (target->conditional_cnt > 0)) {
437             if (!parallel_ok(target, false)) {
438                 add_subtree(target, recursion_level, do_get, implicit);
439                 target->state = build_running;
440                 return build_running;
441             }
442 #endif
443  */
444         }
445         /*
446          * If KEEP_STATE is on, we have to rebuild the target if the
447          * building of it caused new automatic dependencies to be reported.
448          * This is where we restart the build.
449          */
450         if (line != NULL) {
451                 line->body.line.percent = NULL;
452         }
453 recheck_target:
454         /* Init all local variables */
455         result = build_dont_know;
456         out_of_date_list = NULL;
457         command = NULL;
```



```
458         less = NULL;
459         auto_count = 0;
460         if (!restart && line != NULL) {
461                 /*
462                  * If this target has never been built before, mark all
463                  * of the dependencies as never built.
464                  */
465                 for (dependency = line->body.line.dependencies;
466                      dependency != NULL;
467                      dependency = dependency->next) {
468                         dependency->built = false;
469                 }
470         }
471         /* Save the set of automatic depes defined for this target */
472         if (keep_state &&
473             (line != NULL) &&
474             (line->body.line.dependencies != NULL)) {
475                 Name *p;

477                 /*
478                  * First run thru the dependency list to see how many
479                  * autos there are.
480                  */
481                 for (dependency = line->body.line.dependencies;
482                      dependency != NULL;
483                      dependency = dependency->next) {
484                         if (dependency->automatic && !dependency->stale) {
485                                 auto_count++;
486                         }
487                 }
488                 /* Create vector to hold the current autos */
489                 automatics =
490                   (Name *) alloca((int) (auto_count * sizeof (Name)));
491                 /* Copy them */
492                 for (p = automatics, dependency = line->body.line.dependencies;
493                      dependency != NULL;
494                      dependency = dependency->next) {
495                         if (dependency->automatic && !dependency->stale) {
496                                 *p++ = dependency->name;
497                         }
498                 }
499         }
500         if (debug_level > 1) {
501                 (void) printf(NOCATGETS("%*sdoname(%s)\n"),
502                               recursion_level,
503                               "",
504                               target->string_mb);
505         }
506         recursion_level++;
507         /* Avoid infinite loops */
508         if (target->state == build_in_progress) {
509                 warning(catgets(catd, 1, 16, "Infinite loop: Target '%s' depends
510                         target->string_mb);
511                 return build_ok;
512         }
513         target->state = build_in_progress;

515         /* Activate conditional macros for the target */
516         if (!target->added_pattern_conditionals) {
517                 add_pattern_conditionals(target);
518                 target->added_pattern_conditionals = true;
519         }
520         if (target->conditional_cnt > 0) {
521                 old_locals = (Property) alloca(target->conditional_cnt *
522                                                sizeof (Property_rec));
523                 set_locals(target, old_locals);
```

```
524         }

526 /*
527  * after making the call to dynamic_dependecies unconditional we can handle
528  * target names that are same as file name. In this case $$@ in the
529  * dependencies did not mean anything. WIth this change it expands it
530  * as expected.
531  */
532         if (!target->has_depe_list_expanded)
533         {
534 #ifdef NSE
535                 save_readdep = reading_dependencies;
536                 reading_dependencies= true;
537 #endif
538                 dynamic_dependencies(target);
539 #ifdef NSE
540                 reading_dependencies= save_readdep;
541 #endif
542         }

544 /*
545  *      FIRST SECTION -- GO THROUGH DEPENDENCIES AND COLLECT EXPLICIT
546  *      COMMANDS TO RUN
547  */
548         if ((line = get_prop(target->prop, line_prop)) != NULL) {
549                 if (check_dependencies(&result,
550                                        line,
551                                        do_get,
552                                        target,
553                                        true_target,
554                                        doing_subtree,
555                                        &out_of_date_list,
556                                        old_locals,
557                                        implicit,
558                                        &command,
559                                        less,
560                                        rechecking_target,
561                                        recheck_conditionals)) {
562                         return build_running;
563                 }
564                 if (line->body.line.query != NULL) {
565                         delete_query_chain(line->body.line.query);
566                 }
567                 line->body.line.query = out_of_date_list;
568         }

570 #ifdef PARALLEL
571         if (doing_subtree) {
572                 parallel = false;
573         }
574 #endif

576 /*
577  * If the target is a :: type, do not try to find the rule for the target,
578  * all actions will be taken by separate branches.
579  * Else, we try to find an implicit rule using various methods,
580  * we quit as soon as one is found.
581  *
582  * [tolik, 12 Sep 2002] Do not try to find implicit rule for the target
583  * being rechecked - the target is being rechecked means that it already
584  * has explicit dependencies derived from an implicit rule found
585  * in previous step.
586  */
587         if (target->colon_splits == 0 && !rechecking_target) {
588                 /* Look for percent matched rule */
589                 if ((result == build_dont_know) &&
```



```
590                     (command == NULL)) {
591                         switch (find_percent_rule(
592                                         target,
593                                         &command,
594                                         recheck_conditionals)) {
595                         case build_failed:
596                                 result = build_failed;
597                                 break;
598 #ifdef TEAMWARE_MAKE_CMN
599                         case build_running:
600                                 target->state = build_running;
601                                 add_pending(target,
602                                             --recursion_level,
603                                             do_get,
604                                             implicit,
605                                             false);
606                                 if (target->conditional_cnt > 0) {
607                                         reset_locals(target,
608                                                      old_locals,
609                                                      get_prop(target->prop,
610                                                           conditional_prop),
611                                                      0);
612                                 }
613                                 return build_running;
614 #endif
615                         case build_ok:
616                                 result = build_ok;
617                                 break;
618                         }
619                 }
620                 /* Look for double suffix rule */
621                 if (result == build_dont_know) {
622                         Property member;

624                         if (target->is_member &&
625                             ((member = get_prop(target->prop, member_prop)) !=
626                              NULL)) {
627                                 switch (find_ar_suffix_rule(target,
628                                                 member->body.
629                                                 member.member,
630                                                 &command,
631                                                 recheck_conditionals)) {
632                                 case build_failed:
633                                         result = build_failed;
634                                         break;
635 #ifdef TEAMWARE_MAKE_CMN
636                                 case build_running:
637                                         target->state = build_running;
638                                         add_pending(target,
639                                                     --recursion_level,
640                                                     do_get,
641                                                     implicit,
642                                                     false);
643                                     if (target->conditional_cnt > 0) {
644                                         reset_locals(target,
645                                                      old_locals,
646                                                      get_prop(target->prop,
647                                                          conditional_prop),
648                                                      0);
649                                     }
650                                         return build_running;
651 #endif
652                                 default:
653                                         /* ALWAYS bind $% for old style */
654                                         /* ar rules */
655                                         if (line == NULL) {
```

```
656                                             line =
657                                               maybe_append_prop(target,
658                                                                 line_prop);
659                                     }
660                                     line->body.line.percent =
661                                       member->body.member.member;
662                                     break;
663                             }
664                     } else {
665                             switch (find_double_suffix_rule(target,
666                                     &command,
667                                     recheck_conditionals)) {
668                             case build_failed:
669                                     result = build_failed;
670                                     break;
671 #ifdef TEAMWARE_MAKE_CMN
672                             case build_running:
673                                     target->state = build_running;
674                                     add_pending(target,
675                                                 --recursion_level,
676                                                 do_get,
677                                                 implicit,
678                                                 false);
679                                     if (target->conditional_cnt > 0) {
680                                             reset_locals(target,
681                                                          old_locals,
682                                                          get_prop(target->
683                                                                   prop,
684                                                                   conditiona
685                                                          0);
686                                     }
687                                     return build_running;
688 #endif
689                             }
690                     }
691             }
692             /* Look for single suffix rule */

694 /* /tolik/
695  * I commented !implicit to fix bug 1247448: Suffix Rules failed when combine wi
696  * This caused problem with SVR4 tilde rules (infinite recursion). So I made som
697  */
698 /* /tolik, 06.21.96/
699  * Regression! See BugId 1255360
700  * If more than one percent rules are defined for the same target then
701  * the behaviour of 'make' with my previous fix may be different from one
702  * of the 'old make'.
703  * The global variable second_pass (maybe it should be an argument to doname())
704  * is intended to avoid this regression. It is set in doname_check().
705  * First, 'make' will work as it worked before. Only when it is
706  * going to say "don't know how to make target" it sets second_pass to true and
707  * run 'doname' again but now trying to use Single Suffix Rules.
708  */
709             if ((result == build_dont_know) && !automatic && (!implicit || s
710                 ((line == NULL) ||
711                  ((line->body.line.target != NULL) &&
712                   !line->body.line.target->has_regular_dependency))) {
713                     switch (find_suffix_rule(target,
714                                              target,
715                                              empty_name,
716                                              &command,
717                                              recheck_conditionals)) {
718                     case build_failed:
719                             result = build_failed;
720                             break;
721 #ifdef TEAMWARE_MAKE_CMN
```



```
722                     case build_running:
723                             target->state = build_running;
724                             add_pending(target,
725                                         --recursion_level,
726                                         do_get,
727                                         implicit,
728                                         false);
729                             if (target->conditional_cnt > 0) {
730                                     reset_locals(target,
731                                                  old_locals,
732                                                  get_prop(target->prop,
733                                                           conditional_prop),
734                                                  0);
735                             }
736                             return build_running;
737 #endif
738                     }
739             }
740             /* Try to sccs get */
741             if ((command == NULL) &&
742                 (result == build_dont_know) &&
743                 do_get) {
744                     result = sccs_get(target, &command);
745             }

747             /* Use .DEFAULT rule if it is defined. */
748             if ((command == NULL) &&
749                 (result == build_dont_know) &&
750                 (true_target->colons == no_colon) &&
751                 default_rule &&
752                 !implicit) {
753                     /* Make sure we have a line prop */
754                     line = maybe_append_prop(target, line_prop);
755                     command = line;
756                     Boolean out_of_date;
757                     if (true_target->is_member) {
758                             out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
759                                                                     line->bo
760                     } else {
761                             out_of_date = (Boolean) OUT_OF_DATE(true_target-
762                                                                 line->body.l
763                     }
764                     if (build_unconditional || out_of_date) {
765                             line->body.line.is_out_of_date = true;
766                             if (debug_level > 0) {
767                                     (void) printf(catgets(catd, 1, 17, "%*sB
768                                                   recursion_level,
769                                                   "",
770                                                   true_target->string_mb);
771                             }
772                     }
773                     line->body.line.sccs_command = false;
774                     line->body.line.command_template = default_rule;
775                     line->body.line.target = true_target;
776                     line->body.line.star = NULL;
777                     line->body.line.less = true_target;
778                     line->body.line.percent = NULL;
779             }
780     }

782     /* We say "target up to date" if no cmd were executed for the target */
783     if (!target->is_double_colon_parent) {
784             commands_done = false;
785     }

787     silent = silent_all;
```

```
788         ignore_errors = ignore_errors_all;
789         if  (posix)
790         {
791           if  (!silent)
792           {
793             silent = (Boolean) target->silent_mode;
794           }
795           if  (!ignore_errors)
796           {
797             ignore_errors = (Boolean) target->ignore_error_mode;
798           }
799         }

801         int doname_dyntarget = 0;
802 r_command:
803         /* Run commands if any. */
804         if ((command != NULL) &&
805             (command->body.line.command_template != NULL)) {
806                 if (result != build_failed) {
807                         result = run_command(command,
808                                              (Boolean) ((parallel || save_parall
809 #ifdef NSE
810                         nse_check_no_deps_no_rule(target,
811                                 get_prop(target->prop, line_prop), command);
812 #endif
813                 }
814                 switch (result) {
815 #ifdef TEAMWARE_MAKE_CMN
816                 case build_running:
817                         add_running(target,
818                                     true_target,
819                                     command,
820                                     --recursion_level,
821                                     auto_count,
822                                     automatics,
823                                     do_get,
824                                     implicit);
825                         target->state = build_running;
826                         if ((line = get_prop(target->prop,
827                                              line_prop)) != NULL) {
828                                 if (line->body.line.query != NULL) {
829                                      delete_query_chain(line->body.line.query
830                                 }
831                                 line->body.line.query = NULL;
832                         }
833                         if (target->conditional_cnt > 0) {
834                                 reset_locals(target,
835                                              old_locals,
836                                              get_prop(target->prop,
837                                                       conditional_prop),
838                                              0);
839                         }
840                         return build_running;
841                 case build_serial:
842                         add_serial(target,
843                                    --recursion_level,
844                                    do_get,
845                                    implicit);
846                         target->state = build_running;
847                         line = get_prop(target->prop, line_prop);
848                         if (line != NULL) {
849                                 if (line->body.line.query != NULL) {
850                                      delete_query_chain(line->body.line.query
851                                 }
852                                 line->body.line.query = NULL;
853                         }
```



```
854                         if (target->conditional_cnt > 0) {
855                                 reset_locals(target,
856                                              old_locals,
857                                              get_prop(target->prop,
858                                                       conditional_prop),
859                                              0);
860                         }
861                         return build_running;
862 #endif
863                 case build_ok:
864                         /* If all went OK set a nice timestamp */
865                         if (true_target->stat.time == file_doesnt_exist) {
866                                 true_target->stat.time = file_max_time;
867                         }
868                         break;
869                 }
870         } else {
871                 /*
872                  * If no command was found for the target, and it doesn't
873                  * exist, and it is mentioned as a target in the makefile,
874                  * we say it is extremely new and that it is OK.
875                  */
876                 if (target->colons != no_colon) {
877                         if (true_target->stat.time == file_doesnt_exist){
878                                 true_target->stat.time = file_max_time;
879                         }
880                         result = build_ok;
881                 }
882                 /*
883                  * Trying dynamic targets.
884                  */
885                 if(!doname_dyntarget) {
886                         doname_dyntarget = 1;
887                         Name dtarg = find_dyntarget(target);
888                         if(dtarg!=NULL) {
889                                 if (!target->has_depe_list_expanded) {
890                                         dynamic_dependencies(target);
891                                 }
892                                 if ((line = get_prop(target->prop, line_prop)) !
893                                         if (check_dependencies(&result,
894                                                                line,
895                                                                do_get,
896                                                                target,
897                                                                true_target,
898                                                                doing_subtree,
899                                                                &out_of_date_list
900                                                                old_locals,
901                                                                implicit,
902                                                                &command,
903                                                                less,
904                                                                rechecking_target
905                                                                recheck_condition
906                                                 {
907                                                         return build_running;
908                                                 }
909                                                 if (line->body.line.query != NULL) {
910                                                         delete_query_chain(line->body.li
911                                                 }
912                                                 line->body.line.query = out_of_date_list
913                                 }
914                                 goto r_command;
915                         }
916                 }
917                 /*
918                  * If the file exists, it is OK that we couldnt figure
919                  * out how to build it.
```

```
920                  */
921                 (void) exists(target);
922                 if ((target->stat.time != file_doesnt_exist) &&
923                     (result == build_dont_know)) {
924                         result = build_ok;
925                 }
926         }
927
928         /*
929          * Some of the following is duplicated in the function finish_doname.
930          * If anything is changed here, check to see if it needs to be
931          * changed there.
932          */
933         if ((line = get_prop(target->prop, line_prop)) != NULL) {
934                 if (line->body.line.query != NULL) {
935                         delete_query_chain(line->body.line.query);
936                 }
937                 line->body.line.query = NULL;
938         }
939         target->state = result;
940         parallel = save_parallel;
941         if (target->conditional_cnt > 0) {
942                 reset_locals(target,
943                              old_locals,
944                              get_prop(target->prop, conditional_prop),
945                              0);
946         }
947         recursion_level--;
948         if (target->is_member) {
949                 Property member;
950
951                 /* Propagate the timestamp from the member file to the member*/
952                 if ((target->stat.time != file_max_time) &&
953                     ((member = get_prop(target->prop, member_prop)) != NULL) &&
954                     (exists(member->body.member.member) > file_doesnt_exist)) {
955                         target->stat.time =
956                           member->body.member.member->stat.time;
957                 }
958         }
959         /*
960          * Check if we found any new auto dependencies when we
961          * built the target.
962          */
963         if ((result == build_ok) && check_auto_dependencies(target,
964                                                             auto_count,
965                                                             automatics)) {
966                 if (debug_level > 0) {
967                         (void) printf(catgets(catd, 1, 18, "%*sTarget `%s' acqui
968                                       recursion_level,
969                                       "",
970                                       true_target->string_mb);
971                 }
972                 rechecking_target = true;
973                 saved_commands_done = commands_done;
974                 goto recheck_target;
975         }
976
977         if (rechecking_target && !commands_done) {
978                 commands_done = saved_commands_done;
979         }
980
981         return result;
982 }
983
984 /*
985  * DONE.
```



```
986  *
987  *      check_dependencies(result, line, do_get,
988  *                         target, true_target, doing_subtree, out_of_date_tail,
989  *                         old_locals, implicit, command, less, rechecking_target)
990  *
991  *      Return value:
992  *                              True returned if some dependencies left running
993  *
994  *      Parameters:
995  *              result          Pointer to cell we update if build failed
996  *              line            We get the dependencies from here
997  *              do_get          Allow use of sccs get in recursive doname()
998  *              target          The target to chase dependencies for
999  *              true_target     The real one for :: and lib(member)
1000 *              doing_subtree   True if building a conditional macro subtree
1001 *              out_of_date_tail Used to set the $? list
1002 *              old_locals      Used for resetting the local macros
1003 *              implicit        Called when scanning for implicit rules?
1004 *              command         Place to stuff command
1005 *              less            Set to $< value
1006 *
1007 *      Global variables used:
1008 *              command_changed Set if we suspect .make.state needs rewrite
1009 *              debug_level     Should we trace actions?
1010 *              force           The Name " FORCE", compared against
1011 *              recursion_level Used for tracing
1012 *              rewrite_statefile Set if .make.state needs rewriting
1013 *              wait_name       The Name ".WAIT", compared against
1014 */
1015 static Boolean
1016 #ifdef TEAMWARE_MAKE_CMN
1017 check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
1018 #else
1019 check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
1020 #endif
1021 {
1022         Boolean                 dependencies_running;
1023         register Dependency     dependency;
1024         Doname                  dep_result;
1025         Boolean                 dependency_changed = false;
1026
1027         line->body.line.dependency_time = file_doesnt_exist;
1028         if (line->body.line.query != NULL) {
1029                 delete_query_chain(line->body.line.query);
1030         }
1031         line->body.line.query = NULL;
1032         line->body.line.is_out_of_date = false;
1033         dependencies_running = false;
1034         /*
1035          * Run thru all the dependencies and call doname() recursively
1036          * on each of them.
1037          */
1038         for (dependency = line->body.line.dependencies;
1039              dependency != NULL;
1040              dependency = dependency->next) {
1041                 Boolean this_dependency_changed = false;
1042
1043                 if (!dependency->automatic &&
1044                     (rechecking_target || target->rechecking_target)) {
1045                         /*
1046                          * We only bother with the autos when rechecking
1047                          */
1048                         continue;
1049                 }
1050
1051                 if (dependency->name == wait_name) {
```

```
1052                        /*
1053                         * The special target .WAIT means finish all of
1054                         * the prior dependencies before continuing.
1055                         */
1056                        if (dependencies_running) {
1057                                break;
1058                        }
1059 #ifdef DISTRIBUTED
1060                } else if ((!parallel_ok(dependency->name, false)) &&
1061                           (dependencies_running)) {
1062                        /*
1063                         * If we can't execute the current dependency in
1064                         * parallel, hold off the dependency processing
1065                         * to preserve the order of the dependencies.
1066                         */
1067                        break;
1068 #endif
1069                } else {
1070                        timestruc_t     depe_time = file_doesnt_exist;


1073                        if (true_target->is_member) {
1074                                depe_time = exists(dependency->name);
1075                        }
1076                        if (dependency->built ||
1077                            (dependency->name->state == build_failed)) {
1078                                dep_result = (Doname) dependency->name->state;
1079                        } else {
1080 #ifdef NSE
1081                                nse_check_sccs(target->string,
1082                                               dependency->name->string);
1083                                nse_check_derived_src(target,
1084                                               dependency->name->string,
1085                                               line->body.line.command_template)
1086 #endif
1087                                dep_result = doname_check(dependency->name,
1088                                                          do_get,
1089                                                          false,
1090                                                          (Boolean) dependency->
1091                        }
1092                        if (true_target->is_member || dependency->name->is_membe
1093                                /* should compare only secs, cause lib members d
1094                                if (depe_time.tv_sec != dependency->name->stat.t
1095                                        this_dependency_changed =
1096                                          dependency_changed =
1097                                            true;
1098                                }
1099                        } else {
1100                                if (depe_time != dependency->name->stat.time) {
1101                                        this_dependency_changed =
1102                                          dependency_changed =
1103                                            true;
1104                                }
1105                        }
1106                        dependency->built = true;
1107                        switch (dep_result) {
1108                        case build_running:
1109                                dependencies_running = true;
1110                                continue;
1111                        case build_failed:
1112                                *result = build_failed;
1113                                break;
1114                        case build_dont_know:
1115 /*
1116  * If make can't figure out how to make a dependency, maybe the dependency
1117  * is out of date. In this case, we just declare the target out of date
```



```
1118  * and go on. If we really need the dependency, the make'ing of the target
1119  * will fail. This will only happen for automatic (hidden) dependencies.
1120  */
1121                                if(!recheck_conditionals) {
1122                                        line->body.line.is_out_of_date = true;
1123                                }
1124                                /*
1125                                 * Make sure the dependency is not saved
1126                                 * in the state file.
1127                                 */
1128                                dependency->stale = true;
1129                                rewrite_statefile =
1130                                  command_changed =
1131                                    true;
1132                                if (debug_level > 0) {
1133                                        (void) printf(catgets(catd, 1, 19, "Targ
1134                                                      true_target->string_mb,
1135                                                      dependency->name->string_mb
1136                                }
1137                                break;
1138                        }
1139                        if (dependency->name->depends_on_conditional) {
1140                                target->depends_on_conditional = true;
1141                        }
1142                        if (dependency->name == force) {
1143                                target->stat.time =
1144                                  dependency->name->stat.time;
1145                        }
1146                        /*
1147                         * Propagate new timestamp from "member" to
1148                         * "lib.a(member)".
1149                         */
1150                        (void) exists(dependency->name);

1152                        /* Collect the timestamp of the youngest dependency */
1153                        line->body.line.dependency_time =
1154                          MAX(dependency->name->stat.time,
1155                              line->body.line.dependency_time);

1157                        /* Correction: do not consider nanosecs for members */
1158                        if(true_target->is_member || dependency->name->is_member
1159                                line->body.line.dependency_time.tv_nsec = 0;
1160                        }

1162                        if (debug_level > 1) {
1163                                (void) printf(catgets(catd, 1, 20, "%*sDate(%s)=
1164                                              recursion_level,
1165                                              "",
1166                                              dependency->name->string_mb,
1167                                              time_to_string(dependency->name->
1168                                                             stat.time));
1169                                if (dependency->name->stat.time > line->body.lin
1170                                        (void) printf(catgets(catd, 1, 21, "%*sD
1171                                                      recursion_level,
1172                                                      "",
1173                                                      true_target->string_mb,
1174                                                      time_to_string(line->body.
1175                                                                     dependency_
1176                                }
1177                        }

1179                        /* Build the $? list */
1180                        if (true_target->is_member) {
1181                                if (this_dependency_changed == true) {
1182                                        true_target->stat.time = dependency->nam
1183                                        true_target->stat.time.tv_sec--;
```

```
1184                                } else {
1185                                        /* Dina:
1186                                         * The next statement is commented
1187                                         * out as a fix for bug #1051032.
1188                                         * if dependency hasn't changed
1189                                         * then there's no need to invalidate
1190                                         * true_target. This statemnt causes
1191                                         * make to take much longer to process
1192                                         * an already-built archive. Soren
1193                                         * said it was a quick fix for some
1194                                         * problem he doesn't remember.
1195                                        true_target->stat.time = file_no_time;
1196                                         */
1197                                        (void) exists(true_target);
1198                                }
1199                        } else {
1200                                (void) exists(true_target);
1201                        }
1202                        Boolean out_of_date;
1203                        if (true_target->is_member || dependency->name->is_membe
1204                                out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
1205                                                                        dependen
1206                        } else {
1207                                out_of_date = (Boolean) OUT_OF_DATE(true_target-
1208                                                                    dependency->
1209                        }
1210                        if ((build_unconditional || out_of_date) &&
1211                            (dependency->name != force) &&
1212                            (dependency->stale == false)) {
1213                                *out_of_date_tail = ALLOC(Chain);
1214                                if (dependency->name->is_member &&
1215                                    (get_prop(dependency->name->prop,
1216                                              member_prop) != NULL)) {
1217                                        (*out_of_date_tail)->name =
1218                                          get_prop(dependency->name->prop,
1219                                                   member_prop)->
1220                                                     body.member.member;
1221                                } else {
1222                                        (*out_of_date_tail)->name =
1223                                          dependency->name;
1224                                }
1225                                (*out_of_date_tail)->next = NULL;
1226                                out_of_date_tail = &(*out_of_date_tail)->next;
1227                                if (debug_level > 0) {
1228                                        if (dependency->name->stat.time == file_
1229                                                (void) printf(catgets(catd, 1, 2
1230                                                                      recursion_level,
1231                                                                      "",
1232                                                                      true_target->strin
1233                                                                      dependency->name->
1234                                        } else {
1235                                                (void) printf(catgets(catd, 1, 2
1236                                                                      recursion_level,
1237                                                                      "",
1238                                                                      true_target->strin
1239                                                                      dependency->name->
1240                                        }
1241                                }
1242                        }
1243                        if (dependency->name == force) {
1244                                force->stat.time =
1245                                  file_max_time;
1246                                force->state = build_dont_know;
1247                        }
1248                }
1249        }
```



```
1250 #ifdef TEAMWARE_MAKE_CMN
1251         if (dependencies_running) {
1252                 if (doing_subtree) {
1253                         if (target->conditional_cnt > 0) {
1254                                 reset_locals(target,
1255                                              old_locals,
1256                                              get_prop(target->prop,
1257                                                       conditional_prop),
1258                                              0);
1259                         }
1260                         return true;
1261                 } else {
1262                         target->state = build_running;
1263                         add_pending(target,
1264                                     --recursion_level,
1265                                     do_get,
1266                                     implicit,
1267                                     false);
1268                         if (target->conditional_cnt > 0) {
1269                                 reset_locals(target,
1270                                              old_locals,
1271                                              get_prop(target->prop,
1272                                                       conditional_prop),
1273                                              0);
1274                         }
1275                         return true;
1276                 }
1277         }
1278 #endif
1279         /*
1280          * Collect the timestamp of the youngest double colon target
1281          * dependency.
1282          */
1283         if (target->is_double_colon_parent) {
1284                 for (dependency = line->body.line.dependencies;
1285                      dependency != NULL;
1286                      dependency = dependency->next) {
1287                         Property        tmp_line;

1289                         if ((tmp_line = get_prop(dependency->name->prop, line_pr
1290                                 if(tmp_line->body.line.dependency_time != file_m
1291                                         target->stat.time =
1292                                           MAX(tmp_line->body.line.dependency_tim
1293                                               target->stat.time);
1294                                 }
1295                         }
1296                 }
1297         }
1298         if ((true_target->is_member) && (dependency_changed == true)) {
1299                 true_target->stat.time = file_no_time;
1300         }
1301         /*
1302          * After scanning all the dependencies, we check the rule
1303          * if we found one.
1304          */
1305         if (line->body.line.command_template != NULL) {
1306                 if (line->body.line.command_template_redefined) {
1307                         warning(catgets(catd, 1, 24, "Too many rules defined for
1308                                 target->string_mb);
1309                 }
1310                 *command = line;
1311                 /* Check if the target is out of date */
1312                 Boolean out_of_date;
1313                 if (true_target->is_member) {
1314                         out_of_date = (Boolean) OUT_OF_DATE_SEC(true_target->sta
1315                                                                 line->body.line.
```

```
1316                } else {
1317                        out_of_date = (Boolean) OUT_OF_DATE(true_target->stat.ti
1318                                                            line->body.line.depe
1319                }
1320                if (build_unconditional || out_of_date){
1321                        if(!recheck_conditionals) {
1322                                line->body.line.is_out_of_date = true;
1323                        }
1324                }
1325                line->body.line.sccs_command = false;
1326                line->body.line.target = true_target;
1327                if(gnu_style) {

1329                        // set $< for explicit rule
1330                        if(line->body.line.dependencies != NULL) {
1331                                less = line->body.line.dependencies->name;
1332                        }

1334                        // set $* for explicit rule
1335                        Name                    target_body;
1336                        Name                    tt = true_target;
1337                        Property                member;
1338                        register wchar_t        *target_end;
1339                        register Dependency     suffix;
1340                        register int            suffix_length;
1341                        Wstring                 targ_string;
1342                        Wstring                 suf_string;

1344                        if (true_target->is_member &&
1345                            ((member = get_prop(target->prop, member_prop)) !=
1346                             NULL)) {
1347                                tt = member->body.member.member;
1348                        }
1349                        targ_string.init(tt);
1350                        target_end = targ_string.get_string() + tt->hash.length;
1351                        for (suffix = suffixes; suffix != NULL; suffix = suffix-
1352                                suffix_length = suffix->name->hash.length;
1353                                suf_string.init(suffix->name);
1354                                if (tt->hash.length < suffix_length) {
1355                                        continue;
1356                                } else if (!IS_WEQUALN(suf_string.get_string(),
1357                                                (target_end - suffix_length),
1358                                                suffix_length)) {
1359                                        continue;
1360                                }
1361                                target_body = GETNAME(
1362                                        targ_string.get_string(),
1363                                        (int)(tt->hash.length - suffix_length)
1364                                );
1365                                line->body.line.star = target_body;
1366                        }

1368                        // set result = build_ok so that implicit rules are not
1369                        if(*result == build_dont_know) {
1370                                *result = build_ok;
1371                        }
1372                }
1373                if (less != NULL) {
1374                        line->body.line.less = less;
1375                }
1376        }

1378        return false;
1379 }

1381 /*
```



```
1382  *      dynamic_dependencies(target)
1383  *
1384  *      Checks if any dependency contains a macro ref
1385  *      If so, it replaces the dependency with the expanded version.
1386  *      Here, "$@" gets translated to target->string. That is
1387  *      the current name on the left of the colon in the
1388  *      makefile.  Thus,
1389  *              xyz:    s.$@.c
1390  *      translates into
1391  *              xyz:    s.xyz.c
1392  *
1393  *      Also, "$(@F)" translates to the same thing without a preceeding
1394  *      directory path (if one exists).
1395  *      Note, to enter "$@" on a dependency line in a makefile
1396  *      "$$@" must be typed. This is because make expands
1397  *      macros in dependency lists upon reading them.
1398  *      dynamic_dependencies() also expands file wildcards.
1399  *      If there are any Shell meta characters in the name,
1400  *      search the directory, and replace the dependency
1401  *      with the set of files the pattern matches
1402  *
1403  *      Parameters:
1404  *              target          Target to sanitize dependencies for
1405  *
1406  *      Global variables used:
1407  *              c_at            The Name "@", used to set macro value
1408  *              debug_level     Should we trace actions?
1409  *              dot             The Name ".", used to read directory
1410  *              recursion_level Used for tracing
1411  */
1412 void
1413 dynamic_dependencies(Name target)
1414 {
1415         wchar_t                 pattern[MAXPATHLEN];
1416         register wchar_t        *p;
1417         Property                line;
1418         register Dependency     dependency;
1419         register Dependency     *remove;
1420         String_rec              string;
1421         wchar_t                 buffer[MAXPATHLEN];
1422         register Boolean        set_at = false;
1423         register wchar_t        *start;
1424         Dependency              new_depe;
1425         register Boolean        reuse_cell;
1426         Dependency              first_member;
1427         Name                    directory;
1428         Name                    lib;
1429         Name                    member;
1430         Property                prop;
1431         Name                    true_target = target;
1432         wchar_t                 *library;

1434         if ((line = get_prop(target->prop, line_prop)) == NULL) {
1435                 return;
1436         }
1437         /* If the target is constructed from a "::" target we consider that */
1438         if (target->has_target_prop) {
1439                 true_target = get_prop(target->prop,
1440                                        target_prop)->body.target.target;
1441         }
1442         /* Scan all dependencies and process the ones that contain "$" chars */
1443         for (dependency = line->body.line.dependencies;
1444              dependency != NULL;
1445              dependency = dependency->next) {
1446                 if (!dependency->name->dollar) {
1447                         continue;
```

```
1448                }
1449                target->has_depe_list_expanded = true;

1451                /* The make macro $@ is bound to the target name once per */
1452                /* invocation of dynamic_dependencies() */
1453                if (!set_at) {
1454                        (void) SETVAR(c_at, true_target, false);
1455                        set_at = true;
1456                }
1457                /* Expand this dependency string */
1458                INIT_STRING_FROM_STACK(string, buffer);
1459                expand_value(dependency->name, &string, false);
1460                /* Scan the expanded string. It could contain whitespace */
1461                /* which mean it expands to several dependencies */
1462                start = string.buffer.start;
1463                while (iswspace(*start)) {
1464                        start++;
1465                }
1466                /* Remove the cell (later) if the macro was empty */
1467                if (start[0] == (int) nul_char) {
1468                        dependency->name = NULL;
1469                }

1471 /* azv 10/26/95 to fix bug BID_1170218 */
1472                if ((start[0] == (int) period_char) &&
1473                    (start[1] == (int) slash_char)) {
1474                        start += 2;
1475                }
1476 /* azv */

1478                first_member = NULL;
1479                /* We use the original dependency cell for the first */
1480                /* dependency from the expansion */
1481                reuse_cell = true;
1482                /* We also have to deal with dependencies that expand to */
1483                /* lib.a(members) notation */
1484                for (p = start; *p != (int) nul_char; p++) {
1485                        if ((*p == (int) parenleft_char)) {
1486                                lib = GETNAME(start, p - start);
1487                                lib->is_member = true;
1488                                first_member = dependency;
1489                                start = p + 1;
1490                                while (iswspace(*start)) {
1491                                        start++;
1492                                }
1493                                break;
1494                        }
1495                }
1496                do {
1497                    /* First skip whitespace */
1498                        for (p = start; *p != (int) nul_char; p++) {
1499                                if ((*p == (int) nul_char) ||
1500                                    iswspace(*p) ||
1501                                    (*p == (int) parenright_char)) {
1502                                        break;
1503                                }
1504                        }
1505                        /* Enter dependency from expansion */
1506                        if (p != start) {
1507                                /* Create new dependency cell if */
1508                                /* this is not the first dependency */
1509                                /* picked from the expansion */
1510                                if (!reuse_cell) {
1511                                        new_depe = ALLOC(Dependency);
1512                                        new_depe->next = dependency->next;
1513                                        new_depe->automatic = false;
```



```
1514                                        new_depe->stale = false;
1515                                        new_depe->built = false;
1516                                        dependency->next = new_depe;
1517                                        dependency = new_depe;
1518                                }
1519                                reuse_cell = false;
1520                                /* Internalize the dependency name */
1521                                // tolik. Fix for bug 4110429: inconsistent expa
1522                                // include "//" and "/./"
1523                                //dependency->name = GETNAME(start, p - start);
1524                                dependency->name = normalize_name(start, p - sta
1525                                if ((debug_level > 0) &&
1526                                    (first_member == NULL)) {
1527                                        (void) printf(catgets(catd, 1, 25, "%*sD
1528                                                      recursion_level,
1529                                                      "",
1530                                                      dependency->name->string_m
1531                                                      true_target->string_mb);
1532                                }
1533                                for (start = p; iswspace(*start); start++);
1534                                p = start;
1535                        }
1536                } while ((*p != (int) nul_char) &&
1537                         (*p != (int) parenright_char));
1538                /* If the expansion was of lib.a(members) format we now */
1539                /* enter the proper member cells */
1540                if (first_member != NULL) {
1541                        /* Scan the new dependencies and transform them from */
1542                        /* "foo" to "lib.a(foo)" */
1543                        for (; 1; first_member = first_member->next) {
1544                                /* Build "lib.a(foo)" name */
1545                                INIT_STRING_FROM_STACK(string, buffer);
1546                                APPEND_NAME(lib,
1547                                            &string,
1548                                            (int) lib->hash.length);
1549                                append_char((int) parenleft_char, &string);
1550                                APPEND_NAME(first_member->name,
1551                                            &string,
1552                                            FIND_LENGTH);
1553                                append_char((int) parenright_char, &string);
1554                                member = first_member->name;
1555                                /* Replace "foo" with "lib.a(foo)" */
1556                                first_member->name =
1557                                  GETNAME(string.buffer.start, FIND_LENGTH);
1558                                if (string.free_after_use) {
1559                                        retmem(string.buffer.start);
1560                                }
1561                                if (debug_level > 0) {
1562                                        (void) printf(catgets(catd, 1, 26, "%*sD
1563                                                      recursion_level,
1564                                                      "",
1565                                                      first_member->name->
1566                                                      string_mb,
1567                                                      true_target->string_mb);
1568                                }
1569                                first_member->name->is_member = lib->is_member;
1570                                /* Add member property to member */
1571                                prop = maybe_append_prop(first_member->name,
1572                                                         member_prop);
1573                                prop->body.member.library = lib;
1574                                prop->body.member.entry = NULL;
1575                                prop->body.member.member = member;
1576                                if (first_member == dependency) {
1577                                        break;
1578                                }
1579                        }
```

```
1580                }
1581        }
1582        Wstring wcb;
1583        /* Then scan all the dependencies again. This time we want to expand */
1584        /* shell file wildcards */
1585        for (remove = &line->body.line.dependencies, dependency = *remove;
1586             dependency != NULL;
1587             dependency = *remove) {
1588                if (dependency->name == NULL) {
1589                        dependency = *remove = (*remove)->next;
1590                        continue;
1591                }
1592                /* If dependency name string contains shell wildcards */
1593                /* replace the name with the expansion */
1594                if (dependency->name->wildcard) {
1595 #ifdef NSE
1596                        nse_wildcard(target->string, dependency->name->string);
1597 #endif
1598                        wcb.init(dependency->name);
1599                        if ((start = (wchar_t *) wschr(wcb.get_string(),
1600                                        (int) parenleft_char)) != NULL) {
1601                                /* lib(*) type pattern */
1602                                library = buffer;
1603                                (void) wsncpy(buffer,
1604                                              wcb.get_string(),
1605                                              start - wcb.get_string());
1606                                buffer[start-wcb.get_string()] =
1607                                  (int) nul_char;
1608                                (void) wsncpy(pattern,
1609                                              start + 1,
1610 (int) (dependency->name->hash.length-(start-wcb.get_string())-2));
1611                                pattern[dependency->name->hash.length -
1612                                        (start-wcb.get_string()) - 2] =
1613                                          (int) nul_char;
1614                        } else {
1615                                library = NULL;
1616                                (void) wsncpy(pattern,
1617                                              wcb.get_string(),
1618                                              (int) dependency->name->hash.lengt
1619                                pattern[dependency->name->hash.length] =
1620                                  (int) nul_char;
1621                        }
1622                        start = (wchar_t *) wsrchr(pattern, (int) slash_char);
1623                        if (start == NULL) {
1624                                directory = dot;
1625                                p = pattern;
1626                        } else {
1627                                directory = GETNAME(pattern, start-pattern);
1628                                p = start+1;
1629                        }
1630                        /* The expansion is handled by the read_dir() routine*/
1631                        if (read_dir(directory, p, line, library)) {
1632                                *remove = (*remove)->next;
1633                        } else {
1634                                remove = &dependency->next;
1635                        }
1636                } else {
1637                        remove = &dependency->next;
1638                }
1639        }

1641        /* Then unbind $@ */
1642        (void) SETVAR(c_at, (Name) NULL, false);
1643 }

1645 /*
```



```
1646  * DONE.
1647  *
1648  *      run_command(line)
1649  *
1650  *      Takes one Cmd_line and runs the commands from it.
1651  *
1652  *      Return value:
1653  *                              Indicates if the command failed or not
1654  *
1655  *      Parameters:
1656  *              line            The command line to run
1657  *
1658  *      Global variables used:
1659  *              commands_done   Set if we do run command
1660  *              current_line    Set to the line we run a command from
1661  *              current_target  Set to the target we run a command for
1662  *              file_number     Used to form temp file name
1663  *              keep_state      Indicates that .KEEP_STATE is on
1664  *              make_state      The Name ".make.state", used to check timestamp
1665  *              parallel        True if currently building in parallel
1666  *              parallel_process_cnt Count of parallel processes running
1667  *              quest           Indicates that make -q is on
1668  *              rewrite_statefile Set if we do run a command
1669  *              sunpro_dependencies The Name "SUNPRO_DEPENDENCIES", set value
1670  *              temp_file_directory Used to form temp fie name
1671  *              temp_file_name  Set to the name of the temp file
1672  *              touch           Indicates that make -t is on
1673  */
1674 static Doname
1675 run_command(register Property line, Boolean)
1676 {
1677         register Doname         result = build_ok;
1678         register Boolean        remember_only = false;
1679         register Name           target = line->body.line.target;
1680         wchar_t                 *string;
1681         char                    tmp_file_path[MAXPATHLEN];

1683         if (!line->body.line.is_out_of_date && target->rechecking_target) {
1684                 target->rechecking_target = false;
1685                 return build_ok;
1686         }

1688         /*
1689          * Build the command if we know the target is out of date,
1690          * or if we want to check cmd consistency.
1691          */
1692         if (line->body.line.is_out_of_date || keep_state) {
1693                 /* Hack for handling conditional macros in DMake. */
1694                 if (!line->body.line.dont_rebuild_command_used) {
1695                         build_command_strings(target, line);
1696                 }
1697         }
1698         /* Never mind */
1699         if (!line->body.line.is_out_of_date) {
1700                 return build_ok;
1701         }
1702         /* If quest, then exit(1) because the target is out of date */
1703         if (quest) {
1704                 if (posix) {
1705 #ifdef TEAMWARE_MAKE_CMN
1706                         result = execute_parallel(line, true);
1707 #else
1708                         result = execute_serial(line);
1709 #endif
1710                 }
1711 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
```

```
1712                exit_status = 1;
1713 #endif
1714                exit(1);
1715        }
1716        /* We actually had to do something this time */
1717        rewrite_statefile = commands_done = true;
1718        /*
1719         * If this is an sccs command, we have to do some extra checking
1720         * and possibly complain. If the file can't be gotten because it's
1721         * checked out, we complain and behave as if the command was
1722         * executed eventhough we ignored the command.
1723         */
1724        if (!touch &&
1725            line->body.line.sccs_command &&
1726            (target->stat.time != file_doesnt_exist) &&
1727            ((target->stat.mode & 0222) != 0)) {
1728                fatal(catgets(catd, 1, 27, "%s is writable so it cannot be sccs
1729                      target->string_mb);
1730                target->has_complained = remember_only = true;
1731        }
1732        /*
1733         * If KEEP_STATE is on, we make sure we have the timestamp for
1734         * .make.state. If .make.state changes during the command run,
1735         * we reread .make.state after the command. We also setup the
1736         * environment variable that asks utilities to report dependencies.
1737         */
1738        if (!touch &&
1739            keep_state &&
1740            !remember_only) {
1741                (void) exists(make_state);
1742                if((strlen(temp_file_directory) == 1) &&
1743                       (temp_file_directory[0] == '/')) {
1744                   tmp_file_path[0] = '\0';
1745                } else {
1746                   strcpy(tmp_file_path, temp_file_directory);
1747                }
1748                sprintf(mbs_buffer,
1749                                NOCATGETS("%s/.make.dependency.%08x.%d.%d"),
1750                                tmp_file_path,
1751                                hostid,
1752                                getpid(),
1753                                file_number++);
1754                MBSTOWCS(wcs_buffer, mbs_buffer);
1755                Boolean fnd;
1756                temp_file_name = getname_fn(wcs_buffer, FIND_LENGTH, false, &fnd
1757                temp_file_name->stat.is_file = true;
1758                int len = 2*MAXPATHLEN + strlen(target->string_mb) + 2;
1759                wchar_t *to = string = ALLOC_WC(len);
1760                for (wchar_t *from = wcs_buffer; *from != (int) nul_char; ) {
1761                       if (*from == (int) space_char) {
1762                               *to++ = (int) backslash_char;
1763                       }
1764                       *to++ = *from++;
1765                }
1766                *to++ = (int) space_char;
1767                MBSTOWCS(to, target->string_mb);
1768                Name sprodep_name = getname_fn(string, FIND_LENGTH, false, &fnd)
1769                (void) SETVAR(sunpro_dependencies,
1770                              sprodep_name,
1771                              false);
1772                retmem(string);
1773        } else {
1774                temp_file_name = NULL;
1775        }

1777        /*
```



```
1778         * In case we are interrupted, we need to know what was going on.
1779         */
1780        current_target = target;
1781        /*
1782         * We also need to be able to save an empty command instead of the
1783         * interrupted one in .make.state.
1784         */
1785        current_line = line;
1786        if (remember_only) {
1787                /* Empty block!!! */
1788        } else if (touch) {
1789                result = touch_command(line, target, result);
1790                if (posix) {
1791 #ifdef TEAMWARE_MAKE_CMN
1792                        result = execute_parallel(line, true);
1793 #else
1794                        result = execute_serial(line);
1795 #endif
1796                }
1797        } else {
1798                /*
1799                 * If this is not a touch run, we need to execute the
1800                 * proper command(s) for the target.
1801                 */
1802 #ifdef TEAMWARE_MAKE_CMN
1803                if (parallel) {
1804                        if (!parallel_ok(target, true)) {
1805                                /*
1806                                 * We are building in parallel, but
1807                                 * this target must be built in serial.
1808                                 */
1809                                /*
1810                                 * If nothing else is building,
1811                                 * do this one, else wait.
1812                                 */
1813                                if (parallel_process_cnt == 0) {
1814 #ifdef TEAMWARE_MAKE_CMN
1815                                        result = execute_parallel(line, true, ta
1816 #else
1817                                        result = execute_serial(line);
1818 #endif
1819                                } else {
1820                                        current_target = NULL;
1821                                        current_line = NULL;
1822 /*
1823                                        line->body.line.command_used = NULL;
1824  */
1825                                        line->body.line.dont_rebuild_command_use
1826                                        return build_serial;
1827                                }
1828                        } else {
1829                                result = execute_parallel(line, false);
1830                                switch (result) {
1831                                case build_running:
1832                                        return build_running;
1833                                case build_serial:
1834                                        if (parallel_process_cnt == 0) {
1835 #ifdef TEAMWARE_MAKE_CMN
1836                                                result = execute_parallel(line,
1837 #else
1838                                                result = execute_serial(line);
1839 #endif
1840                                        } else {
1841                                                current_target = NULL;
1842                                                current_line = NULL;
1843                                                target->parallel = false;
```

```
1844                                                line->body.line.command_used =
1845                                                                        NULL;
1846                                                return build_serial;
1847                                        }
1848                                }
1849                        }
1850                } else {
1851 #endif
1852 #ifdef TEAMWARE_MAKE_CMN
1853                        result = execute_parallel(line, true, target->localhost)
1854 #else
1855                        result = execute_serial(line);
1856 #endif
1857 #ifdef TEAMWARE_MAKE_CMN
1858                }
1859 #endif
1860        }
1861        temp_file_name = NULL;
1862        if (report_dependencies_level == 0){
1863                update_target(line, result);
1864        }
1865        current_target = NULL;
1866        current_line = NULL;
1867        return result;
1868 }

1870 /*
1871  *      execute_serial(line)
1872  *
1873  *      Runs thru the command line for the target and
1874  *      executes the rules one by one.
1875  *
1876  *      Return value:
1877  *                              The result of the command build
1878  *
1879  *      Parameters:
1880  *              line            The command to execute
1881  *
1882  *      Static variables used:
1883  *
1884  *      Global variables used:
1885  *              continue_after_error -k flag
1886  *              do_not_exec_rule -n flag
1887  *              report_dependencies -P flag
1888  *              silent          Don't echo commands before executing
1889  *              temp_file_name  Temp file for auto dependencies
1890  *              vpath_defined   If true, translate path for command
1891  */
1892 Doname
1893 execute_serial(Property line)
1894 {
1895        int                     child_pid = 0;
1896 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
1897        Avo_MToolJobResultMsg   *job_result_msg;
1898        RWCollectable           *xdr_msg;
1899 #endif
1900        Boolean                 printed_serial;
1901        Doname                  result = build_ok;
1902        Cmd_line                rule, cmd_tail, command = NULL;
1903        char                    mbstring[MAXPATHLEN];
1904        int                     filed;
1905        Name                    target = line->body.line.target;

1907        SEND_MTOOL_MSG(
1908                if (!sent_rsrc_info_msg) {
1909                        if (userName[0] == '\0') {
```



```
1910                                avo_get_user(userName, NULL);
1911                        }
1912                        if (hostName[0] == '\0') {
1913                                strcpy(hostName, avo_hostname());
1914                        }
1915                        send_rsrc_info_msg(1, hostName, userName);
1916                        sent_rsrc_info_msg = 1;
1917                }
1918                send_job_start_msg(line);
1919                job_result_msg = new Avo_MToolJobResultMsg();
1920        );

1922        target->has_recursive_dependency = false;
1923        // We have to create a copy of the rules chain for processing because
1924        // the original one can be destroyed during .make.state file rereading.
1925        for (rule = line->body.line.command_used;
1926             rule != NULL;
1927             rule = rule->next) {
1928                if (command == NULL) {
1929                        command = cmd_tail = ALLOC(Cmd_line);
1930                } else {
1931                        cmd_tail->next = ALLOC(Cmd_line);
1932                        cmd_tail = cmd_tail->next;
1933                }
1934                *cmd_tail = *rule;
1935        }
1936        if (command) {
1937                cmd_tail->next = NULL;
1938        }
1939        for (rule = command; rule != NULL; rule = rule->next) {
1940                if (posix && (touch || quest) && !rule->always_exec) {
1941                        continue;
1942                }
1943                if (vpath_defined) {
1944                        rule->command_line =
1945                          vpath_translation(rule->command_line);
1946                }
1947                /* Echo command line, maybe. */
1948                if ((rule->command_line->hash.length > 0) &&
1949                    !silent &&
1950                    (!rule->silent || do_not_exec_rule) &&
1951                    (report_dependencies_level == 0)) {
1952                        (void) printf("%s\n", rule->command_line->string_mb);
1953                        SEND_MTOOL_MSG(
1954                                job_result_msg->appendOutput(AVO_STRDUP(rule->co
1955                        );
1956                }
1957                if (rule->command_line->hash.length > 0) {
1958                        SEND_MTOOL_MSG(
1959                                (void) sprintf(mbstring,
1960                                        NOCATGETS("%s/make.stdout.%d.%d.
1961                                        tmpdir,
1962                                        getpid(),
1963                                        file_number++);

1965                                int tmp_fd = mkstemp(mbstring);
1966                                if(tmp_fd) {
1967                                        (void) close(tmp_fd);
1968                                }

1970                                stdout_file = strdup(mbstring);
1971                                stderr_file = NULL;
1972                                child_pid = pollResults(stdout_file,
1973                                                        (char *)NULL,
1974                                                        (char *)NULL);
1975                        );
```

```
1976                        /* Do assignment if command line prefixed with "=" */
1977                        if (rule->assign) {
1978                                result = build_ok;
1979                                do_assign(rule->command_line, target);
1980                        } else if (report_dependencies_level == 0) {
1981                                /* Execute command line. */
1982 #ifdef DISTRIBUTED
1983                                setvar_envvar((Avo_DoJobMsg *)NULL);
1984 #else
1985                                setvar_envvar();
1986 #endif
1987                                result = dosys(rule->command_line,
1988                                               (Boolean) rule->ignore_error,
1989                                               (Boolean) rule->make_refd,
1990                                               /* ds 98.04.23 bug #4085164. make
1991                                               false,
1992                                               /* BOOLEAN(rule->silent &&
1993                                                       rule->ignore_error), */
1994                                               (Boolean) rule->always_exec,
1995                                               target,
1996                                               send_mtool_msgs);
1997 #ifdef NSE
1998                                nse_did_recursion= false;
1999 #endif
2000                                check_state(temp_file_name);
2001 #ifdef NSE
2002                                nse_check_cd(line);
2003 #endif
2004                        }
2005                        SEND_MTOOL_MSG(
2006                                append_job_result_msg(job_result_msg);
2007                                if (child_pid > 0) {
2008                                        kill(child_pid, SIGUSR1);
2009                                        while (!((waitpid(child_pid, 0, 0) == -1
2010                                                 && (errno == ECHILD)));
2011                                }
2012                                child_pid = 0;
2013                                (void) unlink(stdout_file);
2014                                retmem_mb(stdout_file);
2015                                stdout_file = NULL;
2016                        );
2017                } else {
2018                        result = build_ok;
2019                }
2020                if (result == build_failed) {
2021                        if (silent || rule->silent) {
2022                                (void) printf(catgets(catd, 1, 242, "The followi
2023                                              rule->command_line->string_mb);
2024                                SEND_MTOOL_MSG(
2025                                        job_result_msg->appendOutput(AVO_STRDUP(
2026                                        job_result_msg->appendOutput(AVO_STRDUP(
2027                                );
2028                        }
2029                        if (!rule->ignore_error && !ignore_errors) {
2030                                if (!continue_after_error) {
2031                                        SEND_MTOOL_MSG(
2032                                                job_result_msg->setResult(job_ms
2033                                                xdr_msg = (RWCollectable*)
2034                                                        job_result_msg;
2035                                                xdr(&xdrs, xdr_msg);
2036                                                (void) fflush(mtool_msgs_fp);
2037                                                delete job_result_msg;
2038                                        );
2039                                        fatal(catgets(catd, 1, 244, "Command fai
2040                                              target->string_mb);
2041                                }
```



```
2042                                /*
2043                                 * Make sure a failing command is not
2044                                 * saved in .make.state.
2045                                 */
2046                                line->body.line.command_used = NULL;
2047                                break;
2048                        } else {
2049                                result = build_ok;
2050                        }
2051                }
2052        }
2053        for (rule = command; rule != NULL; rule = cmd_tail) {
2054                cmd_tail = rule->next;
2055                free(rule);
2056        }
2057        command = NULL;
2058        SEND_MTOOL_MSG(
2059                job_result_msg->setResult(job_msg_id, (result == build_ok) ? 0 :
2060                xdr_msg = (RWCollectable*) job_result_msg;
2061                xdr(&xdrs, xdr_msg);
2062                (void) fflush(mtool_msgs_fp);

2064                delete job_result_msg;
2065        );
2066        if (temp_file_name != NULL) {
2067                free_name(temp_file_name);
2068        }
2069        temp_file_name = NULL;

2071        Property spro = get_prop(sunpro_dependencies->prop, macro_prop);
2072        if(spro != NULL) {
2073                Name val = spro->body.macro.value;
2074                if(val != NULL) {
2075                        free_name(val);
2076                        spro->body.macro.value = NULL;
2077                }
2078        }
2079        spro = get_prop(sunpro_dependencies->prop, env_mem_prop);
2080        if(spro) {
2081                char *val = spro->body.env_mem.value;
2082                if(val != NULL) {
2083                        /*
2084                         * Do not return memory allocated for SUNPRO_DEPENDENCIE
2085                         * It will be returned in setvar_daemon() in macro.cc
2086                         */
2087                        //      retmem_mb(val);
2088                        spro->body.env_mem.value = NULL;
2089                }
2090        }
2091
2092        return result;
2093 }


2096 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */

2098 /*
2099  * Create and send an Avo_MToolRsrcInfoMsg.
2100  */
2101 void
2102 send_rsrc_info_msg(int max_jobs, char *hostname, char *username)
2103 {
2104        static int              first = 1;
2105        Avo_MToolRsrcInfoMsg    *msg;
2106        RWSlistCollectables     server_list;
2107        Avo_ServerState         *server_state;
```

```
2108         RWCollectable           *xdr_msg;

2110         if (!first) {
2111                 return;
2112         }
2113         first = 0;

2115         create_xdrs_ptr();

2117         server_state = new Avo_ServerState(max_jobs, hostname, username);
2118         server_list.append(server_state);
2119         msg = new Avo_MToolRsrcInfoMsg(&server_list);

2121         xdr_msg = (RWCollectable *)msg;
2122         xdr(get_xdrs_ptr(), xdr_msg);
2123         (void) fflush(get_mtool_msgs_fp());

2125         delete server_state;
2126         delete msg;
2127 }

2129 /*
2130  * Create and send an Avo_MToolJobStartMsg.
2131  */
2132 void
2133 send_job_start_msg(Property line)
2134 {
2135         int                     cmd_options = 0;
2136         Avo_MToolJobStartMsg    *msg;
2137         Cmd_line                rule;
2138         Name                    target = line->body.line.target;
2139         RWCollectable           *xdr_msg;

2141         if (userName[0] == '\0') {
2142                 avo_get_user(userName, NULL);
2143         }
2144         if (hostName[0] == '\0') {
2145                 strcpy(hostName, avo_hostname());
2146         }

2148         msg = new Avo_MToolJobStartMsg();
2149         msg->setJobId(++job_msg_id);
2150         msg->setTarget(AVO_STRDUP(target->string_mb));
2151         msg->setHost(AVO_STRDUP(hostName));
2152         msg->setUser(AVO_STRDUP(userName));

2154         for (rule = line->body.line.command_used;
2155              rule != NULL;
2156              rule = rule->next) {
2157                 if (posix && (touch || quest) && !rule->always_exec) {
2158                         continue;
2159                 }
2160                 if (vpath_defined) {
2161                         rule->command_line =
2162                           vpath_translation(rule->command_line);
2163                 }
2164                 cmd_options = 0;
2165                 if (rule->ignore_error || ignore_errors) {
2166                         cmd_options |= ignore_mask;
2167                 }
2168                 if (rule->silent || silent) {
2169                         cmd_options |= silent_mask;
2170                 }
2171                 if (rule->command_line->meta) {
2172                         cmd_options |= meta_mask;
2173                 }
```



```
2174                 if (!touch && (rule->command_line->hash.length > 0)) {
2175                         msg->appendCmd(new Avo_DmakeCommand(rule->command_line->
2176                 }
2177         }

2179         xdr_msg = (RWCollectable*) msg;
2180         xdr(&xdrs, xdr_msg);
2181         (void) fflush(mtool_msgs_fp);

2183 /* tolik, 08/39/2002.
2184    I commented out this code because it causes using unallocated memory.
2185         delete msg;
2186 */
2187 }

2189 /*
2190  * Append the stdout/err to Avo_MToolJobResultMsg.
2191  */
2192 static void
2193 append_job_result_msg(Avo_MToolJobResultMsg *job_result_msg)
2194 {
2195         FILE            *fp;
2196         char            line[MAXPATHLEN];
2197         char            stdout_file2[MAXPATHLEN];

2199         if (stdout_file != NULL) {
2200                 fp = fopen(stdout_file, "r");
2201                 if (fp == NULL) {
2202                         /* Hmmm... what should we do here? */
2203                         warning(catgets(catd, 1, 326, "fopen() of stdout_file fa
2204                         return;
2205                 }
2206                 while (fgets(line, MAXPATHLEN, fp) != NULL) {
2207                         if (line[strlen(line) - 1] == '\n') {
2208                                 line[strlen(line) - 1] = '\0';
2209                         }
2210                         job_result_msg->appendOutput(AVO_STRDUP(line));
2211                 }
2212                 (void) fclose(fp);
2213                 us_sleep(STAT_RETRY_SLEEP_TIME);
2214         } else {
2215                 /* Hmmm... stdout_file shouldn't be NULL */
2216                 warning(catgets(catd, 1, 327, "Internal stdout_file variable sho
2217         }
2218 }
2219 #endif /* TEAMWARE_MAKE_CMN */

2221 /*
2222  *      vpath_translation(cmd)
2223  *
2224  *      Translates one command line by
2225  *      checking each word. If the word has an alias it is translated.
2226  *
2227  *      Return value:
2228  *                              The translated command
2229  *
2230  *      Parameters:
2231  *              cmd             Command to translate
2232  *
2233  *      Global variables used:
2234  */
2235 Name
2236 vpath_translation(register Name cmd)
2237 {
2238         wchar_t                 buffer[STRING_BUFFER_LENGTH];
2239         String_rec              new_cmd;
```

```
2240         wchar_t                *p;
2241         wchar_t                *start;

2243         if (!vpath_defined || (cmd == NULL) || (cmd->hash.length == 0)) {
2244                 return cmd;
2245         }
2246         INIT_STRING_FROM_STACK(new_cmd, buffer);

2248         Wstring wcb(cmd);
2249         p = wcb.get_string();

2251         while (*p != (int) nul_char) {
2252                 while (iswspace(*p) && (*p != (int) nul_char)) {
2253                         append_char(*p++, &new_cmd);
2254                 }
2255                 start = p;
2256                 while (!iswspace(*p) && (*p != (int) nul_char)) {
2257                         p++;
2258                 }
2259                 cmd = GETNAME(start, p - start);
2260                 if (cmd->has_vpath_alias_prop) {
2261                         cmd = get_prop(cmd->prop, vpath_alias_prop)->
2262                                                 body.vpath_alias.alias;
2263                         APPEND_NAME(cmd,
2264                                     &new_cmd,
2265                                     (int) cmd->hash.length);
2266                 } else {
2267                         append_string(start, &new_cmd, p - start);
2268                 }
2269         }
2270         cmd = GETNAME(new_cmd.buffer.start, FIND_LENGTH);
2271         if (new_cmd.free_after_use) {
2272                 retmem(new_cmd.buffer.start);
2273         }
2274         return cmd;
2275 }

2277 /*
2278  *      check_state(temp_file_name)
2279  *
2280  *      Reads and checks the state changed by the previously executed command.
2281  *
2282  *      Parameters:
2283  *              temp_file_name  The auto dependency temp file
2284  *
2285  *      Global variables used:
2286  */
2287 void
2288 check_state(Name temp_file_name)
2289 {
2290         if (!keep_state) {
2291                 return;
2292         }

2294         /*
2295          * Then read the temp file that now might
2296          * contain dependency reports from utilities
2297          */
2298         read_dependency_file(temp_file_name);

2300         /*
2301          * And reread .make.state if it
2302          * changed (the command ran recursive makes)
2303          */
2304         check_read_state_file();
2305         if (temp_file_name != NULL) {
```



```
2306                 (void) unlink(temp_file_name->string_mb);
2307         }
2308 }

2310 /*
2311  *      read_dependency_file(filename)
2312  *
2313  *      Read the temp file used for reporting dependencies to make
2314  *
2315  *      Parameters:
2316  *              filename        The name of the file with the state info
2317  *
2318  *      Global variables used:
2319  *              makefile_type   The type of makefile being read
2320  *              read_trace_level Debug flag
2321  *              temp_file_number The always increasing number for unique files
2322  *              trace_reader    Debug flag
2323  */
2324 static void
2325 read_dependency_file(register Name filename)
2326 {
2327         register Makefile_type  save_makefile_type;

2329         if (filename == NULL) {
2330                 return;
2331         }
2332         filename->stat.time = file_no_time;
2333         if (exists(filename) > file_doesnt_exist) {
2334                 save_makefile_type = makefile_type;
2335                 makefile_type = reading_cpp_file;
2336                 if (read_trace_level > 1) {
2337                         trace_reader = true;
2338                 }
2339                 temp_file_number++;
2340                 (void) read_simple_file(filename,
2341                                         false,
2342                                         false,
2343                                         false,
2344                                         false,
2345                                         false,
2346                                         false);
2347                 trace_reader = false;
2348                 makefile_type = save_makefile_type;
2349         }
2350 }

2352 /*
2353  *      check_read_state_file()
2354  *
2355  *      Check if .make.state has changed
2356  *      If it has we reread it
2357  *
2358  *      Parameters:
2359  *
2360  *      Global variables used:
2361  *              make_state      Make state file name
2362  *              makefile_type   Type of makefile being read
2363  *              read_trace_level Debug flag
2364  *              trace_reader    Debug flag
2365  */
2366 static void
2367 check_read_state_file(void)
2368 {
2369         timestruc_t             previous = make_state->stat.time;
2370         register Makefile_type  save_makefile_type;
2371         register Property       makefile;
```

```
2373         make_state->stat.time = file_no_time;
2374         if ((exists(make_state) == file_doesnt_exist) ||
2375             (make_state->stat.time == previous)) {
2376                 return;
2377         }
2378         save_makefile_type = makefile_type;
2379         makefile_type = rereading_statefile;
2380         /* Make sure we clear the old cached contents of .make.state */
2381         makefile = maybe_append_prop(make_state, makefile_prop);
2382         if (makefile->body.makefile.contents != NULL) {
2383                 retmem(makefile->body.makefile.contents);
2384                 makefile->body.makefile.contents = NULL;
2385         }
2386         if (read_trace_level > 1) {
2387                 trace_reader = true;
2388         }
2389         temp_file_number++;
2390         (void) read_simple_file(make_state,
2391                                 false,
2392                                 false,
2393                                 false,
2394                                 false,
2395                                 false,
2396                                 true);
2397         trace_reader = false;
2398         makefile_type = save_makefile_type;
2399 }

2401 /*
2402  *      do_assign(line, target)
2403  *
2404  *      Handles runtime assignments for command lines prefixed with "=".
2405  *
2406  *      Parameters:
2407  *              line            The command that contains an assignment
2408  *              target          The Name of the target, used for error reports
2409  *
2410  *      Global variables used:
2411  *              assign_done     Set to indicate doname needs to reprocess
2412  */
2413 static void
2414 do_assign(register Name line, register Name target)
2415 {
2416         Wstring wcb(line);
2417         register wchar_t        *string = wcb.get_string();
2418         register wchar_t        *equal;
2419         register Name           name;
2420         register Boolean        append = false;

2422         /*
2423          * If any runtime assignments are done, doname() must reprocess all
2424          * targets in the future since the macro values used to build the
2425          * command lines for the targets might have changed.
2426          */
2427         assign_done = true;
2428         /* Skip white space. */
2429         while (iswspace(*string)) {
2430                 string++;
2431         }
2432         equal = string;
2433         /* Find "+=" or "=". */
2434         while (!iswspace(*equal) &&
2435                (*equal != (int) plus_char) &&
2436                (*equal != (int) equal_char)) {
2437                 equal++;
```



```
2438         }
2439         /* Internalize macro name. */
2440         name = GETNAME(string, equal - string);
2441         /* Skip over "+=" "=". */
2442         while (!((*equal == (int) nul_char) ||
2443                  (*equal == (int) equal_char) ||
2444                  (*equal == (int) plus_char))) {
2445                 equal++;
2446         }
2447         switch (*equal) {
2448         case nul_char:
2449                 fatal(catgets(catd, 1, 31, "= expected in rule ‘%s’ for target ‘
2450                       line->string_mb,
2451                       target->string_mb);
2452         case plus_char:
2453                 append = true;
2454                 equal++;
2455                 break;
2456         }
2457         equal++;
2458         /* Skip over whitespace in front of value. */
2459         while (iswspace(*equal)) {
2460                 equal++;
2461         }
2462         /* Enter new macro value. */
2463         enter_equal(name,
2464                     GETNAME(equal, wcb.get_string() + line->hash.length - equal)
2465                     append);
2466 }

2468 /*
2469  *      build_command_strings(target, line)
2470  *
2471  *      Builds the command string to used when
2472  *      building a target. If the string is different from the previous one
2473  *      is_out_of_date is set.
2474  *
2475  *      Parameters:
2476  *              target          Target to build commands for
2477  *              line            Where to stuff result
2478  *
2479  *      Global variables used:
2480  *              c_at            The Name "@", used to set macro value
2481  *              command_changed Set if command is different from old
2482  *              debug_level     Should we trace activities?
2483  *              do_not_exec_rule Always echo when running -n
2484  *              empty_name      The Name "", used for empty rule
2485  *              funny           Semantics of characters
2486  *              ignore_errors   Used to init field for line
2487  *              is_conditional  Set to false befor evaling macro, checked
2488  *                              after expanding macros
2489  *              keep_state      Indicates that .KEEP_STATE is on
2490  *              make_word_mentioned Set by macro eval, inits field for cmd
2491  *              query           The Name "?", used to set macro value
2492  *              query_mentioned Set by macro eval, inits field for cmd
2493  *              recursion_level Used for tracing
2494  *              silent          Used to init field for line
2495  */
2496 static void
2497 build_command_strings(Name target, register Property line)
2498 {
2499         String_rec              command_line;
2500         register Cmd_line       command_template = line->body.line.command_templ
2501         register Cmd_line       *insert = &line->body.line.command_used;
2502         register Cmd_line       used = *insert;
2503         wchar_t                 buffer[STRING_BUFFER_LENGTH];
```

```
2504        wchar_t                 *start;
2505        Name                    new_command_line;
2506        register Boolean        new_command_longer = false;
2507        register Boolean        ignore_all_command_dependency = true;
2508        Property                member;
2509        static Name             less_name;
2510        static Name             percent_name;
2511        static Name             star;
2512        Name                    tmp_name;

2514        if (less_name == NULL) {
2515                MBSTOWCS(wcs_buffer, "<");
2516                less_name = GETNAME(wcs_buffer, FIND_LENGTH);
2517                MBSTOWCS(wcs_buffer, "%");
2518                percent_name = GETNAME(wcs_buffer, FIND_LENGTH);
2519                MBSTOWCS(wcs_buffer, "*");
2520                star = GETNAME(wcs_buffer, FIND_LENGTH);
2521        }

2523        /* We have to check if a target depends on conditional macros */
2524        /* Targets that do must be reprocessed by doname() each time around */
2525        /* since the macro values used when building the target might have */
2526        /* changed */
2527        conditional_macro_used = false;
2528        /* If we are building a lib.a(member) target $@ should be bound */
2529        /* to lib.a */
2530        if (target->is_member &&
2531            ((member = get_prop(target->prop, member_prop)) != NULL)) {
2532                target = member->body.member.library;
2533        }
2534        /* If we are building a "::" help target $@ should be bound to */
2535        /* the real target name */
2536        /* A lib.a(member) target is never :: */
2537        if (target->has_target_prop) {
2538                target = get_prop(target->prop, target_prop)->
2539                  body.target.target;
2540        }
2541        /* Bind the magic macros that make supplies */
2542        tmp_name = target;
2543        if(tmp_name != NULL) {
2544                if (tmp_name->has_vpath_alias_prop) {
2545                        tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
2546                                        body.vpath_alias.alias;
2547                }
2548        }
2549        (void) SETVAR(c_at, tmp_name, false);

2551        tmp_name = line->body.line.star;
2552        if(tmp_name != NULL) {
2553                if (tmp_name->has_vpath_alias_prop) {
2554                        tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
2555                                        body.vpath_alias.alias;
2556                }
2557        }
2558        (void) SETVAR(star, tmp_name, false);

2560        tmp_name = line->body.line.less;
2561        if(tmp_name != NULL) {
2562                if (tmp_name->has_vpath_alias_prop) {
2563                        tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
2564                                        body.vpath_alias.alias;
2565                }
2566        }
2567        (void) SETVAR(less_name, tmp_name, false);

2569        tmp_name = line->body.line.percent;
```



```
2570        if(tmp_name != NULL) {
2571                if (tmp_name->has_vpath_alias_prop) {
2572                        tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
2573                                        body.vpath_alias.alias;
2574                }
2575        }
2576        (void) SETVAR(percent_name, tmp_name, false);

2578        /* $? is seldom used and it is expensive to build */
2579        /* so we store the list form and build the string on demand */
2580        Chain query_list = NULL;
2581        Chain *query_list_tail = &query_list;

2583        for (Chain ch = line->body.line.query; ch != NULL; ch = ch->next) {
2584                *query_list_tail = ALLOC(Chain);
2585                (*query_list_tail)->name = ch->name;
2586                if ((*query_list_tail)->name->has_vpath_alias_prop) {
2587                        (*query_list_tail)->name =
2588                                get_prop((*query_list_tail)->name->prop,
2589                                        vpath_alias_prop)->body.vpath_alias.alia
2590                }
2591                (*query_list_tail)->next = NULL;
2592                query_list_tail = &(*query_list_tail)->next;
2593        }
2594        (void) setvar_daemon(query,
2595                             (Name) query_list,
2596                             false,
2597                             chain_daemon,
2598                             false,
2599                             debug_level);

2601        /* build $^ */
2602        Chain hat_list = NULL;
2603        Chain *hat_list_tail = &hat_list;

2605        for (Dependency dependency = line->body.line.dependencies;
2606                dependency != NULL;
2607                dependency = dependency->next) {
2608                /* skip automatic dependencies */
2609                if (!dependency->automatic) {
2610                        if ((dependency->name != force) &&
2611                                (dependency->stale == false)) {
2612                                *hat_list_tail = ALLOC(Chain);
2613
2614                                if (dependency->name->is_member &&
2615                                        (get_prop(dependency->name->prop, member
2616                                        (*hat_list_tail)->name =
2617                                                        get_prop(dependency->nam
2618                                                                member_prop)->bo
2619                                } else {
2620                                        (*hat_list_tail)->name = dependency->nam
2621                                }

2623                                if((*hat_list_tail)->name != NULL) {
2624                                        if ((*hat_list_tail)->name->has_vpath_al
2625                                                (*hat_list_tail)->name =
2626                                                        get_prop((*hat_list_tail
2627                                                                vpath_alias_prop
2628                                        }
2629                                }

2631                                (*hat_list_tail)->next = NULL;
2632                                hat_list_tail = &(*hat_list_tail)->next;
2633                        }
2634                }
2635        }
```

```
2636         (void) setvar_daemon(hat,
2637                              (Name) hat_list,
2638                              false,
2639                              chain_daemon,
2640                              false,
2641                              debug_level);

2643 /* We have two command sequences we need to handle */
2644 /* The old one that we probably read from .make.state */
2645 /* and the new one we are building that will replace the old one */
2646 /* Even when KEEP_STATE is not on we build a new command sequence and store */
2647 /* it in the line prop. This command sequence is then executed by */
2648 /* run_command(). If KEEP_STATE is on it is also later written to */
2649 /* .make.state. The routine replaces the old command line by line with the */
2650 /* new one trying to reuse Cmd_lines */

2652         /* If there is no old command_used we have to start creating */
2653         /* Cmd_lines to keep the new cmd in */
2654         if (used == NULL) {
2655                 new_command_longer = true;
2656                 *insert = used = ALLOC(Cmd_line);
2657                 used->next = NULL;
2658                 used->command_line = NULL;
2659                 insert = &used->next;
2660         }
2661         /* Run thru the template for the new command and build the expanded */
2662         /* new command lines */
2663         for (;
2664              command_template != NULL;
2665              command_template = command_template->next, insert = &used->next, us
2666                 /* If there is no old command_used Cmd_line we need to */
2667                 /* create one and say that cmd consistency failed */
2668                 if (used == NULL) {
2669                         new_command_longer = true;
2670                         *insert = used = ALLOC(Cmd_line);
2671                         used->next = NULL;
2672                         used->command_line = empty_name;
2673                 }
2674                 /* Prepare the Cmd_line for the processing */
2675                 /* The command line prefixes "@-=?" are stripped and that */
2676                 /* information is saved in the Cmd_line */
2677                 used->assign = false;
2678                 used->ignore_error = ignore_errors;
2679                 used->silent = silent;
2680                 used->always_exec = false;
2681                 /* Expand the macros in the command line */
2682                 INIT_STRING_FROM_STACK(command_line, buffer);
2683                 make_word_mentioned =
2684                   query_mentioned =
2685                     false;
2686                 expand_value(command_template->command_line, &command_line, true
2687                 /* If the macro $(MAKE) is mentioned in the command */
2688                 /* "make -n" runs actually execute the command */
2689                 used->make_refd = make_word_mentioned;
2690                 used->ignore_command_dependency = query_mentioned;
2691                 /* Strip the prefixes */
2692                 start = command_line.buffer.start;
2693                 for (;
2694                      iswspace(*start) ||
2695                      (get_char_semantics_value(*start) & (int) command_prefix_se
2696                      start++) {
2697                         switch (*start) {
2698                         case question_char:
2699                                 used->ignore_command_dependency = true;
2700                                 break;
2701                         case exclam_char:
```



```
2702                                 used->ignore_command_dependency = false;
2703                                 break;
2704                         case equal_char:
2705                                 used->assign = true;
2706                                 break;
2707                         case hyphen_char:
2708                                 used->ignore_error = true;
2709                                 break;
2710                         case at_char:
2711                                 if (!do_not_exec_rule) {
2712                                         used->silent = true;
2713                                 }
2714                                 break;
2715                         case plus_char:
2716                                 if(posix) {
2717                                   used->always_exec  = true;
2718                                 }
2719                                 break;
2720                         }
2721                 }
2722                 /* If all command lines of the template are prefixed with "?"*/
2723                 /* the VIRTUAL_ROOT is not used for cmd consistency checks */
2724                 if (!used->ignore_command_dependency) {
2725                         ignore_all_command_dependency = false;
2726                 }
2727                 /* Internalize the expanded and stripped command line */
2728                 new_command_line = GETNAME(start, FIND_LENGTH);
2729                 if ((used->command_line == NULL) &&
2730                     (line->body.line.sccs_command)) {
2731                         used->command_line = new_command_line;
2732                         new_command_longer = false;
2733                 }
2734                 /* Compare it with the old one for command consistency */
2735                 if (used->command_line != new_command_line) {
2736                         Name vpath_translated = vpath_translation(new_command_li
2737                         if (keep_state &&
2738                             !used->ignore_command_dependency && (vpath_translate
2739                                 if (debug_level > 0) {
2740                                         if (used->command_line != NULL
2741                                             && *used->command_line->string_mb !=
2742                                             '\0') {
2743                                                 (void) printf(catgets(catd, 1, 3
2744                                                               recursion_level,
2745                                                               "",
2746                                                               target->string_mb,
2747                                                               vpath_translated->
2748                                                               recursion_level,
2749                                                               "",
2750                                                               used->
2751                                                               command_line->
2752                                                               string_mb);
2753                                         } else {
2754                                                 (void) printf(catgets(catd, 1, 3
2755                                                               recursion_level,
2756                                                               "",
2757                                                               target->string_mb,
2758                                                               vpath_translated->
2759                                                               recursion_level,
2760                                                               "");
2761                                         }
2762                                 }
2763                                 command_changed = true;
2764                                 line->body.line.is_out_of_date = true;
2765                         }
2766                         used->command_line = new_command_line;
2767                 }
```

```
2768                 if (command_line.free_after_use) {
2769                         retmem(command_line.buffer.start);
2770                 }
2771         }
2772         /* Check if the old command is longer than the new for */
2773         /* command consistency */
2774         if (used != NULL) {
2775                 *insert = NULL;
2776                 if (keep_state &&
2777                     !ignore_all_command_dependency) {
2778                         if (debug_level > 0) {
2779                                 (void) printf(catgets(catd, 1, 34, "%*sBuilding
2780                                               recursion_level,
2781                                               "",
2782                                               target->string_mb);
2783                         }
2784                         command_changed = true;
2785                         line->body.line.is_out_of_date = true;
2786                 }
2787         }
2788         /* Check if the new command is longer than the old command for */
2789         /* command consistency */
2790         if (new_command_longer &&
2791             !ignore_all_command_dependency &&
2792             keep_state) {
2793                 if (debug_level > 0) {
2794                         (void) printf(catgets(catd, 1, 35, "%*sBuilding %s becau
2795                                       recursion_level,
2796                                       "",
2797                                       target->string_mb);
2798                 }
2799                 command_changed = true;
2800                 line->body.line.is_out_of_date = true;
2801         }
2802         /* Unbind the magic macros */
2803         (void) SETVAR(c_at, (Name) NULL, false);
2804         (void) SETVAR(star, (Name) NULL, false);
2805         (void) SETVAR(less_name, (Name) NULL, false);
2806         (void) SETVAR(percent_name, (Name) NULL, false);
2807         (void) SETVAR(query, (Name) NULL, false);
2808         if (query_list != NULL) {
2809                 delete_query_chain(query_list);
2810         }
2811         (void) SETVAR(hat, (Name) NULL, false);
2812         if (hat_list != NULL) {
2813                 delete_query_chain(hat_list);
2814         }

2816         if (conditional_macro_used) {
2817                 target->conditional_macro_list = cond_macro_list;
2818                 cond_macro_list = NULL;
2819                 target->depends_on_conditional = true;
2820         }
2821 }

2823 /*
2824  *      touch_command(line, target, result)
2825  *
2826  *      If this is an "make -t" run we do this.
2827  *      We touch all targets in the target group ("foo + fie:") if any.
2828  *
2829  *      Return value:
2830  *                              Indicates if the command failed or not
2831  *
2832  *      Parameters:
2833  *              line            The command line to update
```



```
2834  *              target          The target we are touching
2835  *              result          Initial value for the result we return
2836  *
2837  *      Global variables used:
2838  *              do_not_exec_rule Indicates that -n is on
2839  *              silent          Do not echo commands
2840  */
2841 static Doname
2842 touch_command(register Property line, register Name target, Doname result)
2843 {
2844         Name                    name;
2845         register Chain          target_group;
2846         String_rec              touch_string;
2847         wchar_t                 buffer[MAXPATHLEN];
2848         Name                    touch_cmd;
2849         Cmd_line                rule;

2851 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
2852         Avo_MToolJobResultMsg   *job_result_msg;
2853         RWCollectable           *xdr_msg;
2854         int                     child_pid = 0;
2855         wchar_t                 string[MAXPATHLEN];
2856         char                    mbstring[MAXPATHLEN];
2857         int                     filed;
2858 #endif

2860         SEND_MTOOL_MSG(
2861                 if (!sent_rsrc_info_msg) {
2862                         if (userName[0] == '\0') {
2863                                 avo_get_user(userName, NULL);
2864                         }
2865                         if (hostName[0] == '\0') {
2866                                 strcpy(hostName, avo_hostname());
2867                         }
2868                         send_rsrc_info_msg(1, hostName, userName);
2869                         sent_rsrc_info_msg = 1;
2870                 }
2871                 send_job_start_msg(line);
2872                 job_result_msg = new Avo_MToolJobResultMsg();
2873         );
2874         for (name = target, target_group = NULL; name != NULL;) {
2875                 if (!name->is_member) {
2876                         /*
2877                          * Build a touch command that can be passed
2878                          * to dosys(). If KEEP_STATE is on, "make -t"
2879                          * will save the proper command, not the
2880                          * "touch" in .make.state.
2881                          */
2882                         INIT_STRING_FROM_STACK(touch_string, buffer);
2883                         MBSTOWCS(wcs_buffer, NOCATGETS("touch "));
2884                         append_string(wcs_buffer, &touch_string, FIND_LENGTH);
2885                         touch_cmd = name;
2886                         if (name->has_vpath_alias_prop) {
2887                                 touch_cmd = get_prop(name->prop,
2888                                                      vpath_alias_prop)->
2889                                               body.vpath_alias.alias;
2890                         }
2891                         APPEND_NAME(touch_cmd,
2892                                     &touch_string,
2893                                     FIND_LENGTH);
2894                         touch_cmd = GETNAME(touch_string.buffer.start,
2895                                             FIND_LENGTH);
2896                         if (touch_string.free_after_use) {
2897                                 retmem(touch_string.buffer.start);
2898                         }
2899                         if (!silent ||
```

```
2900                            do_not_exec_rule &&
2901                            (target_group == NULL)) {
2902                                (void) printf("%s\n", touch_cmd->string_mb);
2903                                SEND_MTOOL_MSG(
2904                                        job_result_msg->appendOutput(AVO_STRDUP(
2905                                );
2906                        }
2907                        /* Run the touch command, or simulate it */
2908                        if (!do_not_exec_rule) {

2910                                SEND_MTOOL_MSG(
2911                                        (void) sprintf(mbstring,
2912                                                        NOCATGETS("%s/make.stdou
2913                                                        tmpdir,
2914                                                        getpid(),
2915                                                        file_number++);
2916
2917                                        int tmp_fd = mkstemp(mbstring);
2918                                        if(tmp_fd) {
2919                                                (void) close(tmp_fd);
2920                                        }

2922                                        stdout_file = strdup(mbstring);
2923                                        stderr_file = NULL;
2924                                        child_pid = pollResults(stdout_file,
2925                                                                (char *)NULL,
2926                                                                (char *)NULL);
2927                                );

2929                                result = dosys(touch_cmd,
2930                                               false,
2931                                               false,
2932                                               false,
2933                                               false,
2934                                               name,
2935                                               send_mtool_msgs);

2937                                SEND_MTOOL_MSG(
2938                                        append_job_result_msg(job_result_msg);
2939                                        if (child_pid > 0) {
2940                                                kill(child_pid, SIGUSR1);
2941                                                while (!((waitpid(child_pid, 0,
2942                                                        && (errno == ECHILD)));
2943                                        }
2944                                        child_pid = 0;
2945                                        (void) unlink(stdout_file);
2946                                        retmem_mb(stdout_file);
2947                                        stdout_file = NULL;
2948                                );

2950                        } else {
2951                                result = build_ok;
2952                        }
2953                } else {
2954                        result = build_ok;
2955                }
2956                if (target_group == NULL) {
2957                        target_group = line->body.line.target_group;
2958                } else {
2959                        target_group = target_group->next;
2960                }
2961                if (target_group != NULL) {
2962                        name = target_group->name;
2963                } else {
2964                        name = NULL;
2965                }
```



```
2966        }
2967        SEND_MTOOL_MSG(
2968                job_result_msg->setResult(job_msg_id, (result == build_ok) ? 0 :
2969                xdr_msg = (RWCollectable*) job_result_msg;
2970                xdr(&xdrs, xdr_msg);
2971                (void) fflush(mtool_msgs_fp);
2972                delete job_result_msg;
2973        );
2974        return result;
2975 }

2977 /*
2978  *      update_target(line, result)
2979  *
2980  *      updates the status of a target after executing its commands.
2981  *
2982  *      Parameters:
2983  *              line            The command line block to update
2984  *              result          Indicates that build is OK so can update
2985  *
2986  *      Global variables used:
2987  *              do_not_exec_rule Indicates that -n is on
2988  *              touch           Fake the new timestamp if we are just touching
2989  */
2990 void
2991 update_target(Property line, Doname result)
2992 {
2993         Name                    target;
2994         Chain                   target_group;
2995         Property                line2;
2996         timestruc_t             old_stat_time;
2997         Property                member;

2999         /*
3000          * [tolik] Additional fix for bug 1063790. It was fixed
3001          * for serial make long ago, but DMake dumps core when
3002          * target is a symlink and sccs file is newer then target.
3003          * In this case, finish_children() calls update_target()
3004          * with line==NULL.
3005          */
3006         if(line == NULL) {
3007                 /* XXX. Should we do anything here? */
3008                 return;
3009         }

3011         target = line->body.line.target;

3013         if ((result == build_ok) && (line->body.line.command_used != NULL)) {
3014                 if (do_not_exec_rule ||
3015                     touch ||
3016                     (target->is_member &&
3017                      (line->body.line.command_template != NULL) &&
3018                      (line->body.line.command_template->command_line->string_mb[
3019                      (line->body.line.command_template->next == NULL))) {
3020                         /* If we are simulating execution we need to fake a */
3021                         /* new timestamp for the target we didnt build */
3022                         target->stat.time = file_max_time;
3023                 } else {
3024                         /*
3025                          * If we really built the target we read the new
3026                          * timestamp.
3027                          * Fix for bug #1110906: if .c file is newer than
3028                          * the corresponding .o file which is in an archive
3029                          * file, make will compile the .c file but it won't
3030                          * update the object in the .a file.
3031                          */
```

```
3032                        old_stat_time = target->stat.time;
3033                        target->stat.time = file_no_time;
3034                        (void) exists(target);
3035                        if ((target->is_member) &&
3036                            (target->stat.time == old_stat_time)) {
3037                                member = get_prop(target->prop, member_prop);
3038                                if (member != NULL) {
3039                                        target->stat.time = member->body.member.
3040                                        target->stat.time.tv_sec++;
3041                                }
3042                        }
3043                }
3044                /* If the target is part of a group we need to propagate the */
3045                /* result of the run to all members */
3046                for (target_group = line->body.line.target_group;
3047                     target_group != NULL;
3048                     target_group = target_group->next) {
3049                        target_group->name->stat.time = target->stat.time;
3050                        line2 = maybe_append_prop(target_group->name,
3051                                                  line_prop);
3052                        line2->body.line.command_used =
3053                          line->body.line.command_used;
3054                        line2->body.line.target = target_group->name;
3055                }
3056        }
3057        target->has_built = true;
3058 }

3060 /*
3061  *      sccs_get(target, command)
3062  *
3063  *      Figures out if it possible to sccs get a file
3064  *      and builds the command to do it if it is.
3065  *
3066  *      Return value:
3067  *                              Indicates if sccs get failed or not
3068  *
3069  *      Parameters:
3070  *              target          Target to get
3071  *              command         Where to deposit command to use
3072  *
3073  *      Global variables used:
3074  *              debug_level     Should we trace activities?
3075  *              recursion_level Used for tracing
3076  *              sccs_get_rule   The rule to used for sccs getting
3077  */
3078 static Doname
3079 sccs_get(register Name target, register Property *command)
3080 {
3081        register int            result;
3082        char                    link[MAXPATHLEN];
3083        String_rec              string;
3084        wchar_t                 name[MAXPATHLEN];
3085        register wchar_t        *p;
3086        timestruc_t             sccs_time;
3087        register Property       line;
3088        int                     sym_link_depth = 0;

3090        /* For sccs, we need to chase symlinks. */
3091        while (target->stat.is_sym_link) {
3092                if (sym_link_depth++ > 90) {
3093                        fatal(catgets(catd, 1, 95, "Can't read symbolic link '%s
3094                              target->string_mb);
3095                }
3096                /* Read the value of the link. */
3097                result = readlink_vroot(target->string_mb,
```



```
3098                                        link,
3099                                        sizeof(link),
3100                                        NULL,
3101                                        VROOT_DEFAULT);
3102                if (result == -1) {
3103                        fatal(catgets(catd, 1, 36, "Can't read symbolic link '%s
3104                              target->string_mb, errmsg(errno));
3105                }
3106                link[result] = 0;
3107                /* Use the value to build the proper filename. */
3108                INIT_STRING_FROM_STACK(string, name);

3110                Wstring wcb(target);
3111                if ((link[0] != slash_char) &&
3112                    ((p = (wchar_t *) wsrchr(wcb.get_string(), slash_char)) != N
3113                        append_string(wcb.get_string(), &string, p - wcb.get_str
3114                }
3115                append_string(link, &string, result);
3116                /* Replace the old name with the translated name. */
3117                target = normalize_name(string.buffer.start, string.text.p - str
3118                (void) exists(target);
3119                if (string.free_after_use) {
3120                        retmem(string.buffer.start);
3121                }
3122        }

3124        /*
3125         * read_dir() also reads the ?/SCCS dir and saves information
3126         * about which files have SCSC/s. files.
3127         */
3128        if (target->stat.has_sccs == DONT_KNOW_SCCS) {
3129                read_directory_of_file(target);
3130        }
3131        switch (target->stat.has_sccs) {
3132        case DONT_KNOW_SCCS:
3133                /* We dont know by now there is no SCCS/s.* */
3134                target->stat.has_sccs = NO_SCCS;
3135        case NO_SCCS:
3136                /*
3137                 * If there is no SCCS/s.* but the plain file exists,
3138                 * we say things are OK.
3139                 */
3140                if (target->stat.time > file_doesnt_exist) {
3141                        return build_ok;
3142                }
3143                /* If we cant find the plain file, we give up. */
3144                return build_dont_know;
3145        case HAS_SCCS:
3146                /*
3147                 * Pay dirt. We now need to figure out if the plain file
3148                 * is out of date relative to the SCCS/s.* file.
3149                 */
3150                sccs_time = exists(get_prop(target->prop,
3151                                            sccs_prop)->body.sccs.file);
3152                break;
3153        }

3155        if ((!target->has_complained &&
3156            (sccs_time != file_doesnt_exist) &&
3157            (sccs_get_rule != NULL))) {
3158                /* only checking */
3159                if (command == NULL) {
3160                        return build_ok;
3161                }
3162                /*
3163                 * We provide a command line for the target. The line is a
```

```
3164                 * "sccs get" command from default.mk.
3165                 */
3166                line = maybe_append_prop(target, line_prop);
3167                *command = line;
3168                if (sccs_time > target->stat.time) {
3169                        /*
3170                         * And only if the plain file is out of date do we
3171                         * request execution of the command.
3172                         */
3173                        line->body.line.is_out_of_date = true;
3174                        if (debug_level > 0) {
3175                                (void) printf(catgets(catd, 1, 37, "%*sSccs gett
3176                                              recursion_level,
3177                                              "",
3178                                              target->string_mb);
3179                        }
3180                }
3181                line->body.line.sccs_command = true;
3182                line->body.line.command_template = sccs_get_rule;
3183                if(!svr4 && (!allrules_read || posix)) {
3184                   if((target->prop) &&
3185                      (target->prop->body.sccs.file) &&
3186                      (target->prop->body.sccs.file->string_mb)) {
3187                      if((strlen(target->prop->body.sccs.file->string_mb) ==
3188                        strlen(target->string_mb) + 2) &&
3189                        (target->prop->body.sccs.file->string_mb[0] == 's') &&
3190                        (target->prop->body.sccs.file->string_mb[1] == '.')) {

3192                         line->body.line.command_template = get_posix_rule;
3193                      }
3194                   }
3195                }
3196                line->body.line.target = target;
3197                /*
3198                 * Also make sure the rule is build with $* and $<
3199                 * bound properly.
3200                 */
3201                line->body.line.star = NULL;
3202                line->body.line.less = NULL;
3203                line->body.line.percent = NULL;
3204                return build_ok;
3205        }
3206        return build_dont_know;
3207 }

3209 /*
3210  *      read_directory_of_file(file)
3211  *
3212  *      Reads the directory the specified file lives in.
3213  *
3214  *      Parameters:
3215  *              file            The file we need to read dir for
3216  *
3217  *      Global variables used:
3218  *              dot             The Name ".", used as the default dir
3219  */
3220 void
3221 read_directory_of_file(register Name file)
3222 {

3224        Wstring file_string(file);
3225        wchar_t * wcb = file_string.get_string();
3226        wchar_t usr_include_buf[MAXPATHLEN];
3227        wchar_t usr_include_sys_buf[MAXPATHLEN];

3229        register Name           directory = dot;
```



```
3230        register wchar_t        *p = (wchar_t *) wsrchr(wcb,
3231                                                        (int) slash_char);
3232        register int            length = p - wcb;
3233        static Name             usr_include;
3234        static Name             usr_include_sys;

3236        if (usr_include == NULL) {
3237                MBSTOWCS(usr_include_buf, NOCATGETS("/usr/include"));
3238                usr_include = GETNAME(usr_include_buf, FIND_LENGTH);
3239                MBSTOWCS(usr_include_sys_buf, NOCATGETS("/usr/include/sys"));
3240                usr_include_sys = GETNAME(usr_include_sys_buf, FIND_LENGTH);
3241        }

3243        /*
3244         * If the filename contains a "/" we have to extract the path
3245         * Else the path defaults to ".".
3246         */
3247        if (p != NULL) {
3248                /*
3249                 * Check some popular directories first to possibly
3250                 * save time. Compare string length first to gain speed.
3251                 */
3252                if ((usr_include->hash.length == length) &&
3253                    IS_WEQUALN(usr_include_buf,
3254                               wcb,
3255                               length)) {
3256                        directory = usr_include;
3257                } else if ((usr_include_sys->hash.length == length) &&
3258                           IS_WEQUALN(usr_include_sys_buf,
3259                                      wcb,
3260                                      length)) {
3261                        directory = usr_include_sys;
3262                } else {
3263                        directory = GETNAME(wcb, length);
3264                }
3265        }
3266        (void) read_dir(directory,
3267                        (wchar_t *) NULL,
3268                        (Property) NULL,
3269                        (wchar_t *) NULL);
3270 }

3272 /*
3273  *      add_pattern_conditionals(target)
3274  *
3275  *      Scan the list of conditionals defined for pattern targets and add any
3276  *      that match this target to its list of conditionals.
3277  *
3278  *      Parameters:
3279  *              target          The target we should add conditionals for
3280  *
3281  *      Global variables used:
3282  *              conditionals    The list of pattern conditionals
3283  */
3284 static void
3285 add_pattern_conditionals(register Name target)
3286 {
3287        register Property       conditional;
3288        Property                new_prop;
3289        Property                *previous;
3290        Name_rec                dummy;
3291        wchar_t                 *pattern;
3292        wchar_t                 *percent;
3293        int                     length;

3295        Wstring wcb(target);
```

```
3296         Wstring wcb1;

3298         for (conditional = get_prop(conditionals->prop, conditional_prop);
3299              conditional != NULL;
3300              conditional = get_prop(conditional->next, conditional_prop)) {
3301                 wcb1.init(conditional->body.conditional.target);
3302                 pattern = wcb1.get_string();
3303                 if (pattern[1] != 0) {
3304                         percent = (wchar_t *) wschr(pattern, (int) percent_char)
3305                         if (!wcb.equaln(pattern, percent-pattern) ||
3306                             !IS_WEQUAL(wcb.get_string(wcb.length()-wslen(percent
3307                                 continue;
3308                         }
3309                 }
3310                 for (previous = &target->prop;
3311                      *previous != NULL;
3312                      previous = &(*previous)->next) {
3313                         if (((*previous)->type == conditional_prop) &&
3314                             ((*previous)->body.conditional.sequence >
3315                              conditional->body.conditional.sequence)) {
3316                                 break;
3317                         }
3318                 }
3319                 if (*previous == NULL) {
3320                         new_prop = append_prop(target, conditional_prop);
3321                 } else {
3322                         dummy.prop = NULL;
3323                         new_prop = append_prop(&dummy, conditional_prop);
3324                         new_prop->next = *previous;
3325                         *previous = new_prop;
3326                 }
3327                 target->conditional_cnt++;
3328                 new_prop->body.conditional = conditional->body.conditional;
3329         }
3330 }

3332 /*
3333  *      set_locals(target, old_locals)
3334  *
3335  *      Sets any conditional macros for the target.
3336  *      Each target carries a possibly empty set of conditional properties.
3337  *
3338  *      Parameters:
3339  *              target          The target to set conditional macros for
3340  *              old_locals      Space to store old values in
3341  *
3342  *      Global variables used:
3343  *              debug_level     Should we trace activity?
3344  *              is_conditional  We need to preserve this value
3345  *              recursion_level Used for tracing
3346  */
3347 void
3348 set_locals(register Name target, register Property old_locals)
3349 {
3350         register Property       conditional;
3351         register int            i;
3352         register Boolean        saved_conditional_macro_used;
3353         Chain                   cond_name;
3354         Chain                   cond_chain;

3356 #ifdef DISTRIBUTED
3357         if (target->dont_activate_cond_values) {
3358                 return;
3359         }
3360 #endif
```



```
3362         saved_conditional_macro_used = conditional_macro_used;

3364         /* Scan the list of conditional properties and apply each one */
3365         for (conditional = get_prop(target->prop, conditional_prop), i = 0;
3366              conditional != NULL;
3367              conditional = get_prop(conditional->next, conditional_prop),
3368              i++) {
3369                 /* Save the old value */
3370                 old_locals[i].body.macro =
3371                   maybe_append_prop(conditional->body.conditional.name,
3372                                     macro_prop)->body.macro;
3373                 if (debug_level > 1) {
3374                         (void) printf(catgets(catd, 1, 38, "%*sActivating condit
3375                                       recursion_level,
3376                                       "");
3377                 }
3378                 /* Set the conditional value. Macros are expanded when the */
3379                 /* macro is refd as usual */
3380                 if ((conditional->body.conditional.name != virtual_root) ||
3381                     (conditional->body.conditional.value != virtual_root)) {
3382                         (void) SETVAR(conditional->body.conditional.name,
3383                                       conditional->body.conditional.value,
3384                                       (Boolean) conditional->body.conditional.ap
3385                 }
3386                 cond_name = ALLOC(Chain);
3387                 cond_name->name = conditional->body.conditional.name;
3388         }
3389         /* Put this target on the front of the chain of conditional targets */
3390         cond_chain = ALLOC(Chain);
3391         cond_chain->name = target;
3392         cond_chain->next = conditional_targets;
3393         conditional_targets = cond_chain;
3394         conditional_macro_used = saved_conditional_macro_used;
3395 }

3397 /*
3398  *      reset_locals(target, old_locals, conditional, index)
3399  *
3400  *      Removes any conditional macros for the target.
3401  *
3402  *      Parameters:
3403  *              target          The target we are retoring values for
3404  *              old_locals      The values to restore
3405  *              conditional     The first conditional block for the target
3406  *              index           into the old_locals vector
3407  *      Global variables used:
3408  *              debug_level     Should we trace activities?
3409  *              recursion_level Used for tracing
3410  */
3411 void
3412 reset_locals(register Name target, register Property old_locals, register Proper
3413 {
3414         register Property       this_conditional;
3415         Chain                   cond_chain;

3417 #ifdef DISTRIBUTED
3418         if (target->dont_activate_cond_values) {
3419                 return;
3420         }
3421 #endif

3423         /* Scan the list of conditional properties and restore the old value */
3424         /* to each one Reverse the order relative to when we assigned macros */
3425         this_conditional = get_prop(conditional->next, conditional_prop);
3426         if (this_conditional != NULL) {
3427                 reset_locals(target, old_locals, this_conditional, index+1);
```

```
3428         } else {
3429                 /* Remove conditional target from chain */
3430                 if (conditional_targets == NULL ||
3431                     conditional_targets->name != target) {
3432                         warning(catgets(catd, 1, 39, "Internal error: reset targ
3433                 } else {
3434                         cond_chain = conditional_targets->next;
3435                         retmem_mb((caddr_t) conditional_targets);
3436                         conditional_targets = cond_chain;
3437                 }
3438         }
3439         get_prop(conditional->body.conditional.name->prop,
3440                  macro_prop)->body.macro = old_locals[index].body.macro;
3441         if (conditional->body.conditional.name == virtual_root) {
3442                 (void) SETVAR(virtual_root, getvar(virtual_root), false);
3443         }
3444         if (debug_level > 1) {
3445                 if (old_locals[index].body.macro.value != NULL) {
3446                         (void) printf(catgets(catd, 1, 40, "%*sdeactivating cond
3447                                       recursion_level,
3448                                       "",
3449                                       conditional->body.conditional.name->
3450                                       string_mb,
3451                                       old_locals[index].body.macro.value->
3452                                       string_mb);
3453                 } else {
3454                         (void) printf(catgets(catd, 1, 41, "%*sdeactivating cond
3455                                       recursion_level,
3456                                       "",
3457                                       conditional->body.conditional.name->
3458                                       string_mb);
3459                 }
3460         }
3461 }

3463 /*
3464  *      check_auto_dependencies(target, auto_count, automatics)
3465  *
3466  *      Returns true if the target now has a dependency
3467  *      it didn't previously have (saved on automatics).
3468  *
3469  *      Return value:
3470  *                              true if new dependency found
3471  *
3472  *      Parameters:
3473  *              target          Target we check
3474  *              auto_count      Number of old automatic vars
3475  *              automatics      Saved old automatics
3476  *
3477  *      Global variables used:
3478  *              keep_state      Indicates that .KEEP_STATE is on
3479  */
3480 Boolean
3481 check_auto_dependencies(Name target, int auto_count, Name *automatics)
3482 {
3483         Name            *p;
3484         int             n;
3485         Property        line;
3486         Dependency      dependency;

3488         if (keep_state) {
3489                 if ((line = get_prop(target->prop, line_prop)) == NULL) {
3490                         return false;
3491                 }
3492                 /* Go thru new list of automatic depes */
3493                 for (dependency = line->body.line.dependencies;
```



```
3494                      dependency != NULL;
3495                      dependency = dependency->next) {
3496                         /* And make sure that each one existed before we */
3497                         /* built the target */
3498                         if (dependency->automatic && !dependency->stale) {
3499                                 for (n = auto_count, p = automatics;
3500                                      n > 0;
3501                                      n--) {
3502                                         if (*p++ == dependency->name) {
3503                                                 /* If we can find it on the */
3504                                                 /* saved list of autos we */
3505                                                 /* are OK  */
3506                                                 goto not_new;
3507                                         }
3508                                 }
3509                                 /* But if we scan over the old list */
3510                                 /* of auto. without finding it it is */
3511                                 /* new and we must check it */
3512                                 return true;
3513                         }
3514                 not_new:;
3515                 }
3516                 return false;
3517         } else {
3518                 return false;
3519         }
3520 }

3522 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
3523 void
3524 create_xdrs_ptr(void)
3525 {
3526         static int      xdrs_init = 0;

3528         if (!xdrs_init) {
3529                 xdrs_init = 1;
3530                 mtool_msgs_fp = fdopen(mtool_msgs_fd, "a");
3531                 xdrstdio_create(&xdrs,
3532                                 mtool_msgs_fp,
3533                                 XDR_ENCODE);
3534         }
3535 }

3537 XDR *
3538 get_xdrs_ptr(void)
3539 {
3540         return &xdrs;
3541 }

3543 FILE *
3544 get_mtool_msgs_fp(void)
3545 {
3546         return mtool_msgs_fp;
3547 }

3549 int
3550 get_job_msg_id(void)
3551 {
3552         return job_msg_id;
3553 }

3555 // Continuously poll and show the results of remotely executing a job,
3556 // i.e., output the stdout and stderr files.

3558 static int
3559 pollResults(char *outFn, char *errFn, char *hostNm)
```

```
3560 {
3561         int             child;

3563         child = fork();
3564         switch (child) {
3565         case -1:
3566                 break;
3567         case 0:
3568                 enable_interrupt((void (*) (int))SIG_DFL);
3569 #ifdef linux
3570                 (void) signal(SIGUSR1, Avo_PollResultsAction_Sigusr1Handler);
3571 #else
3572                 (void) sigset(SIGUSR1, Avo_PollResultsAction_Sigusr1Handler);
3573 #endif
3574                 pollResultsAction(outFn, errFn);

3576                 exit(0);
3577                 break;
3578         default:
3579                 break;
3580         }
3581         return child;
3582 }

3584 // This is the PollResultsAction SIGUSR1 handler.

3586 static bool_t pollResultsActionTimeToFinish = FALSE;

3588 extern "C" void
3589 Avo_PollResultsAction_Sigusr1Handler(int foo)
3590 {
3591         pollResultsActionTimeToFinish = TRUE;
3592 }

3594 static void
3595 pollResultsAction(char *outFn, char *errFn)
3596 {
3597         int                     fd;
3598         time_t                  file_time = 0;
3599         long                    file_time_nsec = 0;
3600         struct stat             statbuf;
3601         int                     stat_rc;

3603         // Keep stat'ing until file exists.
3604         while (((stat_rc = stat(outFn, &statbuf)) != 0) &&
3605                (errno == ENOENT) &&
3606                !pollResultsActionTimeToFinish) {
3607                 us_sleep(STAT_RETRY_SLEEP_TIME);
3608         }
3609         // The previous stat() could be failed due to EINTR
3610         // So one more try is needed
3611         if (stat_rc != 0 && stat(outFn, &statbuf) != 0) {
3612                 // stat() failed
3613                 warning(NOCATGETS("Internal error: stat(\"%s\", ...) failed: %s\
3614                         outFn, strerror(errno));
3615                 exit(1);
3616         }

3618         if ((fd = open(outFn, O_RDONLY)) < 0
3619                 && (errno != EINTR || (fd = open(outFn, O_RDONLY)) < 0)) {
3620                 // open() failed
3621                 warning(NOCATGETS("Internal error: open(\"%s\", O_RDONLY) failed
3622                         outFn, strerror(errno));
3623                 exit(1);
3624         }
```



```
3626         while (!pollResultsActionTimeToFinish && stat(outFn, &statbuf) == 0) {
3627 #ifdef linux
3628                 if ((statbuf.st_mtime > file_time)
3629                    ) {
3630                         file_time = statbuf.st_mtime;
3631                         rxmGetNextResultsBlock(fd);
3632                 }
3633 #else
3634                 if ((statbuf.st_mtim.tv_sec > file_time) ||
3635                     ((statbuf.st_mtim.tv_sec == file_time) &&
3636                      (statbuf.st_mtim.tv_nsec > file_time_nsec))
3637                    ) {
3638                         file_time = statbuf.st_mtim.tv_sec;
3639                         file_time_nsec = statbuf.st_mtim.tv_nsec;
3640                         rxmGetNextResultsBlock(fd);
3641                 }
3642 #endif
3643                 us_sleep(STAT_RETRY_SLEEP_TIME);
3644         }
3645         // Check for the rest of output
3646         rxmGetNextResultsBlock(fd);

3648         (void) close(fd);
3649 }

3651 static void
3652 rxmGetNextResultsBlock(int fd)
3653 {
3654         size_t                  to_read = 8 * 1024;
3655         ssize_t                 bytes_read;
3656         ssize_t                 bytes_written;
3657         char                    results_buf[8 * 1024];
3658         sigset_t                newset;
3659         sigset_t                oldset;

3661         // Read some more from the output/results file.
3662         // Hopefully the kernel managed to prefetch the stuff.
3663         bytes_read = read(fd, results_buf, to_read);
3664         while (bytes_read > 0) {
3665                 AVO_BLOCK_INTERUPTS;
3666                 bytes_written = write(1, results_buf, bytes_read);
3667                 AVO_UNBLOCK_INTERUPTS;
3668                 if (bytes_written != bytes_read) {
3669                         // write() failed
3670                         warning(NOCATGETS("Internal error: write(1, ...) failed:
3671                                 strerror(errno));
3672                         exit(1);
3673                 }
3674                 bytes_read = read(fd, results_buf, to_read);
3675         }
3676 }

3678 // Generic, interruptable microsecond resolution sleep member function.

3680 static int
3681 us_sleep(unsigned int nusecs)
3682 {
3683         struct pollfd dummy;
3684         int timeout;

3686         if ((timeout = nusecs/1000) <= 0) {
3687                 timeout = 1;
3688         }
3689         return poll(&dummy, 0, timeout);
3690 }
3691 #endif /* TEAMWARE_MAKE_CMN */
```

```
3693 // Recursively delete each of the Chain struct on the chain.

3695 static void
3696 delete_query_chain(Chain ch)
3697 {
3698         if (ch == NULL) {
3699                 return;
3700         } else {
3701                 delete_query_chain(ch->next);
3702                 retmem_mb((char *) ch);
3703         }
3704 }

3706 Doname
3707 target_can_be_built(register Name target) {
3708         Doname          result = build_dont_know;
3709         Name            true_target = target;
3710         Property        line;

3712         if (target == wait_name) {
3713                 return(build_ok);
3714         }
3715         /*
3716          * If the target is a constructed one for a "::" target,
3717          * we need to consider that.
3718          */
3719         if (target->has_target_prop) {
3720                 true_target = get_prop(target->prop,
3721                                        target_prop)->body.target.target;
3722         }

3724         (void) exists(true_target);

3726         if (true_target->state == build_running) {
3727                 return(build_running);
3728         }
3729         if (true_target->stat.time != file_doesnt_exist) {
3730                 result = build_ok;
3731         }

3733         /* get line property for the target */
3734         line = get_prop(true_target->prop, line_prop);

3736         /* first check for explicit rule */
3737         if (line != NULL && line->body.line.command_template != NULL) {
3738                 result = build_ok;
3739         }
3740         /* try to find pattern rule */
3741         if (result == build_dont_know) {
3742                 result = find_percent_rule(target, NULL, false);
3743         }
3744
3745         /* try to find double suffix rule */
3746         if (result == build_dont_know) {
3747                 if (target->is_member) {
3748                         Property member = get_prop(target->prop, member_prop);
3749                         if (member != NULL && member->body.member.member != NULL
3750                                 result = find_ar_suffix_rule(target, member->bod
3751                         } else {
3752                                 result = find_double_suffix_rule(target, NULL, f
3753                         }
3754                 } else {
3755                         result = find_double_suffix_rule(target, NULL, false);
3756                 }
3757         }
```



```
3758
3759         /* try to find suffix rule */
3760         if ((result == build_dont_know) && second_pass) {
3761                 result = find_suffix_rule(target, target, empty_name, NULL, fals
3762         }
3763
3764         /* check for sccs */
3765         if (result == build_dont_know) {
3766                 result = sccs_get(target, NULL);
3767         }
3768
3769         /* try to find dyn target */
3770         if (result == build_dont_know) {
3771                 Name dtarg = find_dyntarget(target);
3772                 if (dtarg != NULL) {
3773                         result = target_can_be_built(dtarg);
3774                 }
3775         }
3776
3777         /* check whether target was mentioned in makefile */
3778         if (result == build_dont_know) {
3779                 if (target->colons != no_colon) {
3780                         result = build_ok;
3781                 }
3782         }

3784         /* result */
3785         return result;
3786 }
3787 #endif /* ! codereview */
```

```
**********************************************************
    4403 Wed May 20 11:04:08 2015
new/usr/src/cmd/make/bin/make/common/dosys.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)dosys.cc 1.45 06/12/12
 */

#pragma ident   "@(#)dosys.cc   1.45    06/12/12"

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

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

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

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

/*
```



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

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

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

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

```
128                   silent_error,
129                   target,
130                   wcb.get_string(),
131                   command->meta ?
132                     doshell(q, ignore_error, redirect_out_err,
133                             stdout_file, stderr_file, 0) :
134                     doexec(q, ignore_error, redirect_out_err,
135                            stdout_file, stderr_file,
136                            vroot_path, 0),
137                   send_mtool_msgs,
138 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
139                   get_xdrs_ptr(),
140                   get_job_msg_id()
141 #else
142                   NULL,
143                   -1
144 #endif
145                   )) {
146                 result = build_ok;
147         } else {
148                 result = build_failed;
149         }
150         retmem(q);

152         if ((report_dependencies_level == 0) &&
153             call_make) {
154                 make_state->stat.time = file_no_time;
155                 (void)exists(make_state);
156                 if (before == make_state->stat.time) {
157                         return result;
158                 }
159                 makefile_type = reading_statefile;
160                 if (read_trace_level > 1) {
161                         trace_reader = true;
162                 }
163                 temp_file_number++;
164                 (void) read_simple_file(make_state,
165                                         false,
166                                         false,
167                                         false,
168                                         false,
169                                         false,
170                                         true);
171                 trace_reader = false;
172         }
173         return result;
174 }
175 #endif /* ! codereview */
```

```
**********************************************************
   18919 Wed May 20 11:04:08 2015
new/usr/src/cmd/make/bin/make/common/files.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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.
 */
/*
 * @(#)files.cc 1.37 06/12/12
 */

#pragma ident   "@(#)files.cc   1.37    06/12/12"

/*
 *      files.c
 *
 *      Various file related routines:
 *              Figure out if file exists
 *              Wildcard resolution for directory reader
 *              Directory reader
 */



/*
 * Included files
 */
#if defined(SUN5_0) || defined(HP_UX)
#include <dirent.h>             /* opendir() */
#else
#include <sys/dir.h>            /* opendir() */
#endif
#include <errno.h>              /* errno */
#include <mk/defs.h>
#include <mksh/macro.h>         /* getvar() */
#include <mksh/misc.h>          /* get_prop(), append_prop() */
#include <sys/stat.h>           /* lstat() */

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */
```



```
/*
 * Static variables
 */

/*
 * File table of contents
 */
extern  timestruc_t&    exists(register Name target);
extern  void            set_target_stat(register Name target, struct stat buf);
static  timestruc_t&    vpath_exists(register Name target);
static  Name            enter_file_name(wchar_t *name_string, wchar_t *library);
static  Boolean         star_match(register char *string, register char *pattern
static  Boolean         amatch(register wchar_t *string, register wchar_t *patte

/*
 *      exists(target)
 *
 *      Figure out the timestamp for one target.
 *
 *      Return value:
 *                              The time the target was created
 *
 *      Parameters:
 *              target          The target to check
 *
 *      Global variables used:
 *              debug_level     Should we trace the stat call?
 *              recursion_level Used for tracing
 *              vpath_defined   Was the variable VPATH defined in environment?
 */
timestruc_t&
exists(register Name target)
{
        struct stat             buf;
        register int            result;

        /* We cache stat information. */
        if (target->stat.time != file_no_time) {
                return target->stat.time;
        }

        /*
         * If the target is a member, we have to extract the time
         * from the archive.
         */
        if (target->is_member &&
            (get_prop(target->prop, member_prop) != NULL)) {
                return read_archive(target);
        }

        if (debug_level > 1) {
                (void) printf(NOCATGETS("%*sstat(%s)\n"),
                                recursion_level,
                                "",
                                target->string_mb);
        }

        result = lstat_vroot(target->string_mb, &buf, NULL, VROOT_DEFAULT);
        if ((result != -1) && ((buf.st_mode & S_IFMT) == S_IFLNK)) {
                /*
                 * If the file is a symbolic link, we remember that
                 * and then we get the status for the refd file.
                 */
                target->stat.is_sym_link = true;
                result = stat_vroot(target->string_mb, &buf, NULL, VROOT_DEFAULT
```

```
128         } else {
129                 target->stat.is_sym_link = false;
130         }

132         if (result < 0) {
133                 target->stat.time = file_doesnt_exist;
134                 target->stat.stat_errno = errno;
135                 if ((errno == ENOENT) &&
136                     vpath_defined &&
137 /* azv, fixing bug 1262942, VPATH works with a leaf name
138  * but not a directory name.
139  */
140                     (target->string_mb[0] != (int) slash_char) ) {
141 /* BID_1214655 */
142 /* azv */
143                         vpath_exists(target);
144                         // return vpath_exists(target);
145                 }
146         } else {
147                 /* Save all the information we need about the file */
148                 target->stat.stat_errno = 0;
149                 target->stat.is_file = true;
150                 target->stat.mode = buf.st_mode & 0777;
151                 target->stat.size = buf.st_size;
152                 target->stat.is_dir =
153                   BOOLEAN((buf.st_mode & S_IFMT) == S_IFDIR);
154                 if (target->stat.is_dir) {
155                         target->stat.time = file_is_dir;
156                 } else {
157                         /* target->stat.time = buf.st_mtime; */
158 /* BID_1129806 */
159 /* vis@nbsp.nsk.su */
160 #if defined(linux)
161                         timestruc_t ttime = { buf.st_mtime, 0 };
162                         target->stat.time = MAX(ttime, file_min_time);
163 #else
164                         target->stat.time = MAX(buf.st_mtim, file_min_time);
165 #endif
166                 }
167         }
168         if ((target->colon_splits > 0) &&
169             (get_prop(target->prop, time_prop) == NULL)) {
170                 append_prop(target, time_prop)->body.time.time =
171                   target->stat.time;
172         }
173         return target->stat.time;
174 }

176 /*
177  *      set_target_stat( target, buf)
178  *
179  *      Called by exists() to set some stat fields in the Name structure
180  *      to those read by the stat_vroot() call (from disk).
181  *
182  *      Parameters:
183  *              target          The target whose stat field is set
184  *              buf             stat values (on disk) of the file
185  *                              represented by target.
186  */
187 void
188 set_target_stat(register Name target, struct stat buf)
189 {
190         target->stat.stat_errno = 0;
191         target->stat.is_file = true;
192         target->stat.mode = buf.st_mode & 0777;
193         target->stat.size = buf.st_size;
```



```
194         target->stat.is_dir =
195           BOOLEAN((buf.st_mode & S_IFMT) == S_IFDIR);
196         if (target->stat.is_dir) {
197                 target->stat.time = file_is_dir;
198         } else {
199                 /* target->stat.time = buf.st_mtime; */
200 /* BID_1129806 */
201 /* vis@nbsp.nsk.su */
202 #if defined(linux)
203                 timestruc_t ttime = { buf.st_mtime, 0 };
204                 target->stat.time = ttime;
205 #else
206                 target->stat.time = MAX(buf.st_mtim, file_min_time);
207 #endif
208         }
209 }


212 /*
213  *      vpath_exists(target)
214  *
215  *      Called if exists() discovers that there is a VPATH defined.
216  *      This function stats the VPATH translation of the target.
217  *
218  *      Return value:
219  *                              The time the target was created
220  *
221  *      Parameters:
222  *              target          The target to check
223  *
224  *      Global variables used:
225  *              vpath_name      The Name "VPATH", used to get macro value
226  */
227 static timestruc_t&
228 vpath_exists(register Name target)
229 {
230         wchar_t                 *vpath;
231         wchar_t                 file_name[MAXPATHLEN];
232         wchar_t                 *name_p;
233         Name                    alias;

235         /*
236          * To avoid recursive search through VPATH when exists(alias) is called
237          */
238         vpath_defined = false;

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

243         vpath = wcb.get_string();

245         while (*vpath != (int) nul_char) {
246                 name_p = file_name;
247                 while ((*vpath != (int) colon_char) &&
248                        (*vpath != (int) nul_char)) {
249                         *name_p++ = *vpath++;
250                 }
251                 *name_p++ = (int) slash_char;
252                 (void) wscpy(name_p, wcb1.get_string());
253                 alias = GETNAME(file_name, FIND_LENGTH);
254                 if (exists(alias) != file_doesnt_exist) {
255                         target->stat.is_file = true;
256                         target->stat.mode = alias->stat.mode;
257                         target->stat.size = alias->stat.size;
258                         target->stat.is_dir = alias->stat.is_dir;
259                         target->stat.time = alias->stat.time;
```

```
260                         maybe_append_prop(target, vpath_alias_prop)->
261                                                 body.vpath_alias.alias = alias;
262                         target->has_vpath_alias_prop = true;
263                         vpath_defined = true;
264                         return alias->stat.time;
265                 }
266                 while ((*vpath != (int) nul_char) &&
267                        ((*vpath == (int) colon_char) || iswspace(*vpath))) {
268                         vpath++;
269                 }
270         }
271         /*
272          * Restore vpath_defined
273          */
274         vpath_defined = true;
275         return target->stat.time;
276 }

278 /*
279  *      read_dir(dir, pattern, line, library)
280  *
281  *      Used to enter the contents of directories into makes namespace.
282  *      Presence of a file is important when scanning for implicit rules.
283  *      read_dir() is also used to expand wildcards in dependency lists.
284  *
285  *      Return value:
286  *                              Non-0 if we found files to match the pattern
287  *
288  *      Parameters:
289  *              dir             Path to the directory to read
290  *              pattern         Pattern for that files should match or NULL
291  *              line            When we scan using a pattern we enter files
292  *                              we find as dependencies for this line
293  *              library         If we scan for "lib.a(<wildcard-member>)"
294  *
295  *      Global variables used:
296  *              debug_level     Should we trace the dir reading?
297  *              dot             The Name ".", compared against
298  *              sccs_dir_path   The path to the SCCS dir (from PROJECTDIR)
299  *              vpath_defined   Was the variable VPATH defined in environment?
300  *              vpath_name      The Name "VPATH", use to get macro value
301  */
302 int
303 read_dir(Name dir, wchar_t *pattern, Property line, wchar_t *library)
304 {
305         wchar_t                 file_name[MAXPATHLEN];
306         wchar_t                 *file_name_p = file_name;
307         Name                    file;
308         wchar_t                 plain_file_name[MAXPATHLEN];
309         wchar_t                 *plain_file_name_p;
310         Name                    plain_file;
311         wchar_t                 tmp_wcs_buffer[MAXPATHLEN];
312         DIR                     *dir_fd;
313         int                     m_local_dependency=0;
314 #if defined(SUN5_0) || defined(HP_UX)
315 #define d_fileno d_ino
316         register struct dirent  *dp;
317 #else
318         register struct direct  *dp;
319 #endif
320         wchar_t                 *vpath = NULL;
321         wchar_t                 *p;
322         int                     result = 0;

324         if(dir->hash.length >= MAXPATHLEN) {
325                 return 0;
```



```
326         }

328         Wstring wcb(dir);
329         Wstring vps;

331         /* A directory is only read once unless we need to expand wildcards. */
332         if (pattern == NULL) {
333                 if (dir->has_read_dir) {
334                         return 0;
335                 }
336                 dir->has_read_dir = true;
337         }
338         /* Check if VPATH is active and setup list if it is. */
339         if (vpath_defined && (dir == dot)) {
340                 vps.init(getvar(vpath_name));
341                 vpath = vps.get_string();
342         }

344         /*
345          * Prepare the string where we build the full name of the
346          * files in the directory.
347          */
348         if ((dir->hash.length > 1) || (wcb.get_string()[0] != (int) period_char)
349                 (void) wscpy(file_name, wcb.get_string());
350                 MBSTOWCS(wcs_buffer, "/");
351                 (void) wscat(file_name, wcs_buffer);
352                 file_name_p = file_name + wslen(file_name);
353         }

355         /* Open the directory. */
356 vpath_loop:
357         dir_fd = opendir(dir->string_mb);
358         if (dir_fd == NULL) {
359                 return 0;
360         }

362         /* Read all the directory entries. */
363         while ((dp = readdir(dir_fd)) != NULL) {
364                 /* We ignore "." and ".." */
365                 if ((dp->d_fileno == 0) ||
366                     ((dp->d_name[0] == (int) period_char) &&
367                      ((dp->d_name[1] == 0) ||
368                       ((dp->d_name[1] == (int) period_char) &&
369                        (dp->d_name[2] == 0))))) {
370                         continue;
371                 }
372                 /*
373                  * Build the full name of the file using whatever
374                  * path supplied to the function.
375                  */
376                 MBSTOWCS(tmp_wcs_buffer, dp->d_name);
377                 (void) wscpy(file_name_p, tmp_wcs_buffer);
378                 file = enter_file_name(file_name, library);
379                 if ((pattern != NULL) && amatch(tmp_wcs_buffer, pattern)) {
380                         /*
381                          * If we are expanding a wildcard pattern, we
382                          * enter the file as a dependency for the target.
383                          */
384                         if (debug_level > 0){
385                                 WCSTOMBS(mbs_buffer, pattern);
386                                 (void) printf(catgets(catd, 1, 231, "'%s: %s' du
387                                                 line->body.line.target->string_mb,
388                                                 file->string_mb,
389                                                 mbs_buffer);
390                         }
391                         enter_dependency(line, file, false);
```

```
392                         result++;
393                 } else {
394                         /*
395                          * If the file has an SCCS/s. file,
396                          * we will detect that later on.
397                          */
398                         file->stat.has_sccs = NO_SCCS;
399                 /*
400                  * If this is an s. file, we also enter it as if it
401                  * existed in the plain directory.
402                  */
403                 if ((dp->d_name[0] == 's') &&
404                     (dp->d_name[1] == (int) period_char)) {
405
406                         MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
407                         plain_file_name_p = plain_file_name;
408                         (void) wscpy(plain_file_name_p, tmp_wcs_buffer);
409                         plain_file = GETNAME(plain_file_name, FIND_LENGTH);
410                         plain_file->stat.is_file = true;
411                         plain_file->stat.has_sccs = HAS_SCCS;
412                         /*
413                          * Enter the s. file as a dependency for the
414                          * plain file.
415                          */
416                         maybe_append_prop(plain_file, sccs_prop)->
417                           body.sccs.file = file;
418                         MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
419                         if ((pattern != NULL) &&
420                             amatch(tmp_wcs_buffer, pattern)) {
421                                 if (debug_level > 0) {
422                                         WCSTOMBS(mbs_buffer, pattern);
423                                         (void) printf(catgets(catd, 1, 232, "'%s
424                                                       line->body.line.target->
425                                                       string_mb,
426                                                       plain_file->string_mb,
427                                                       mbs_buffer);
428                                 }
429                                 enter_dependency(line, plain_file, false);
430                                 result++;
431                         }
432                 }
433               }
434         }
435         (void) closedir(dir_fd);
436         if ((vpath != NULL) && (*vpath != (int) nul_char)) {
437                 while ((*vpath != (int) nul_char) &&
438                        (iswspace(*vpath) || (*vpath == (int) colon_char))) {
439                         vpath++;
440                 }
441                 p = vpath;
442                 while ((*vpath != (int) colon_char) &&
443                        (*vpath != (int) nul_char)) {
444                         vpath++;
445                 }
446                 if (vpath > p) {
447                         dir = GETNAME(p, vpath - p);
448                         goto vpath_loop;
449                 }
450         }
451 /*
452  * look into SCCS directory only if it's not svr4. For svr4 dont do that.
453  */

455 /*
456  * Now read the SCCS directory.
457  * Files in the SCSC directory are considered to be part of the set of
```



```
458  * files in the plain directory. They are also entered in their own right.
459  * Prepare the string where we build the true name of the SCCS files.
460  */
461         (void) wsncpy(plain_file_name,
462                       file_name,
463                       file_name_p - file_name);
464         plain_file_name[file_name_p - file_name] = 0;
465         plain_file_name_p = plain_file_name + wslen(plain_file_name);

467         if(!svr4) {

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

474                 if (file_name_p - file_name > 0) {
475                         tmp_wchar = *file_name_p;
476                         *file_name_p = 0;
477                         WCSTOMBS(mbs_buffer, file_name);
478                         (void) sprintf(mb_path, NOCATGETS("%s/%s/SCCS"),
479                                        sccs_dir_path,
480                                        mbs_buffer);
481                         *file_name_p = tmp_wchar;
482                 } else {
483                         (void) sprintf(mb_path, NOCATGETS("%s/SCCS"), sccs_dir_p
484                 }
485                 MBSTOWCS(path, mb_path);
486                 (void) wscpy(file_name, path);
487           } else {
488                 MBSTOWCS(wcs_buffer, NOCATGETS("SCCS"));
489                 (void) wscpy(file_name_p, wcs_buffer);
490           }
491         } else {
492                 MBSTOWCS(wcs_buffer, NOCATGETS("."));
493                 (void) wscpy(file_name_p, wcs_buffer);
494         }
495         /* Internalize the constructed SCCS dir name. */
496         (void) exists(dir = GETNAME(file_name, FIND_LENGTH));
497         /* Just give up if the directory file doesnt exist. */
498         if (!dir->stat.is_file) {
499                 return result;
500         }
501         /* Open the directory. */
502         dir_fd = opendir(dir->string_mb);
503         if (dir_fd == NULL) {
504                 return result;
505         }
506         MBSTOWCS(wcs_buffer, "/");
507         (void) wscat(file_name, wcs_buffer);
508         file_name_p = file_name + wslen(file_name);

510         while ((dp = readdir(dir_fd)) != NULL) {
511                 if ((dp->d_fileno == 0) ||
512                     ((dp->d_name[0] == (int) period_char) &&
513                      ((dp->d_name[1] == 0) ||
514                       ((dp->d_name[1] == (int) period_char) &&
515                        (dp->d_name[2] == 0))))) {
516                         continue;
517                 }
518                 /* Construct and internalize the true name of the SCCS file. */
519                 MBSTOWCS(wcs_buffer, dp->d_name);
520                 (void) wscpy(file_name_p, wcs_buffer);
521                 file = GETNAME(file_name, FIND_LENGTH);
522                 file->stat.is_file = true;
523                 file->stat.has_sccs = NO_SCCS;
```

```
524                 /*
525                  * If this is an s. file, we also enter it as if it
526                  * existed in the plain directory.
527                  */
528                 if ((dp->d_name[0] == 's') &&
529                     (dp->d_name[1] == (int) period_char)) {
530
531                         MBSTOWCS(wcs_buffer, dp->d_name + 2);
532                         (void) wscpy(plain_file_name_p, wcs_buffer);
533                         plain_file = GETNAME(plain_file_name, FIND_LENGTH);
534                         plain_file->stat.is_file = true;
535                         plain_file->stat.has_sccs = HAS_SCCS;
536                                 /* if sccs dependency is already set,skip */
537                         if(plain_file->prop) {
538                                 Property sprop = get_prop(plain_file->prop,sccs_
539                                 if(sprop != NULL) {
540                                         if (sprop->body.sccs.file) {
541                                                 goto try_pattern;
542                                         }
543                                 }
544                         }
545
546                         /*
547                          * Enter the s. file as a dependency for the
548                          * plain file.
549                          */
550                         maybe_append_prop(plain_file, sccs_prop)->
551                           body.sccs.file = file;
552 try_pattern:
553                         MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
554                         if ((pattern != NULL) &&
555                             amatch(tmp_wcs_buffer, pattern)) {
556                                 if (debug_level > 0) {
557                                         WCSTOMBS(mbs_buffer, pattern);
558                                         (void) printf(catgets(catd, 1, 233, "'%s
559                                                       line->body.line.target->
560                                                       string_mb,
561                                                       plain_file->string_mb,
562                                                       mbs_buffer);
563                                 }
564                                 enter_dependency(line, plain_file, false);
565                                 result++;
566                         }
567                 }
568         }
569         (void) closedir(dir_fd);
570
571         return result;
572 }
573
574 /*
575  *      enter_file_name(name_string, library)
576  *
577  *      Helper function for read_dir().
578  *
579  *      Return value:
580  *                              The Name that was entered
581  *
582  *      Parameters:
583  *              name_string     Name of the file we want to enter
584  *              library         The library it is a member of, if any
585  *
586  *      Global variables used:
587  */
588 static Name
589 enter_file_name(wchar_t *name_string, wchar_t *library)
```



```
590 {
591         wchar_t         buffer[STRING_BUFFER_LENGTH];
592         String_rec      lib_name;
593         Name            name;
594         Property        prop;
595
596         if (library == NULL) {
597                 name = GETNAME(name_string, FIND_LENGTH);
598                 name->stat.is_file = true;
599                 return name;
600         }
601
602         INIT_STRING_FROM_STACK(lib_name, buffer);
603         append_string(library, &lib_name, FIND_LENGTH);
604         append_char((int) parenleft_char, &lib_name);
605         append_string(name_string, &lib_name, FIND_LENGTH);
606         append_char((int) parenright_char, &lib_name);
607
608         name = GETNAME(lib_name.buffer.start, FIND_LENGTH);
609         name->stat.is_file = true;
610         name->is_member = true;
611         prop = maybe_append_prop(name, member_prop);
612         prop->body.member.library = GETNAME(library, FIND_LENGTH);
613         prop->body.member.library->stat.is_file = true;
614         prop->body.member.entry = NULL;
615         prop->body.member.member = GETNAME(name_string, FIND_LENGTH);
616         prop->body.member.member->stat.is_file = true;
617         return name;
618 }
619
620 /*
621  *      star_match(string, pattern)
622  *
623  *      This is a regular shell type wildcard pattern matcher
624  *      It is used when xpanding wildcards in dependency lists
625  *
626  *      Return value:
627  *                              Indication if the string matched the pattern
628  *
629  *      Parameters:
630  *              string          String to match
631  *              pattern         Pattern to match it against
632  *
633  *      Global variables used:
634  */
635 static Boolean
636 star_match(register wchar_t *string, register wchar_t *pattern)
637 {
638         register int            pattern_ch;
639
640         switch (*pattern) {
641         case 0:
642                 return succeeded;
643         case bracketleft_char:
644         case question_char:
645         case asterisk_char:
646                 while (*string) {
647                         if (amatch(string++, pattern)) {
648                                 return succeeded;
649                         }
650                 }
651                 break;
652         default:
653                 pattern_ch = (int) *pattern++;
654                 while (*string) {
655                         if ((*string++ == pattern_ch) &&
```

```
656                             amatch(string, pattern)) {
657                                 return succeeded;
658                         }
659                 }
660                 break;
661         }
662         return failed;
663 }

665 /*
666  *      amatch(string, pattern)
667  *
668  *      Helper function for shell pattern matching
669  *
670  *      Return value:
671  *                              Indication if the string matched the pattern
672  *
673  *      Parameters:
674  *              string          String to match
675  *              pattern         Pattern to match it against
676  *
677  *      Global variables used:
678  */
679 static Boolean
680 amatch(register wchar_t *string, register wchar_t *pattern)
681 {
682         register long           lower_bound;
683         register long           string_ch;
684         register long           pattern_ch;
685         register int            k;

687 top:
688         for (; 1; pattern++, string++) {
689                 lower_bound = 017777777777;
690                 string_ch = *string;
691                 switch (pattern_ch = *pattern) {
692                 case bracketleft_char:
693                         k = 0;
694                         while ((pattern_ch = *++pattern) != 0) {
695                                 switch (pattern_ch) {
696                                 case bracketright_char:
697                                         if (!k) {
698                                                 return failed;
699                                         }
700                                         string++;
701                                         pattern++;
702                                         goto top;
703                                 case hyphen_char:
704                                         k |= (lower_bound <= string_ch) &&
705                                              (string_ch <=
706                                               (pattern_ch = pattern[1]));
707                                 default:
708                                         if (string_ch ==
709                                             (lower_bound = pattern_ch)) {
710                                                 k++;
711                                         }
712                                 }
713                         }
714                         return failed;
715                 case asterisk_char:
716                         return star_match(string, ++pattern);
717                 case 0:
718                         return BOOLEAN(!string_ch);
719                 case question_char:
720                         if (string_ch == 0) {
721                                 return failed;
```



```
722                         }
723                         break;
724                 default:
725                         if (pattern_ch != string_ch) {
726                                 return failed;
727                         }
728                         break;
729                 }
730         }
731         /* NOTREACHED */
732 }

734 #endif /* ! codereview */
```

```
**********************************************************
    5465 Wed May 20 11:04:08 2015
new/usr/src/cmd/make/bin/make/common/globals.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)globals.cc 1.42 06/12/12
 */

#pragma ident   "@(#)globals.cc 1.42    06/12/12"

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

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

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

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

/*
 * Global variables used by make
 */
        Boolean         allrules_read=false;
        Name            posix_name;
```



```
        Name            svr4_name;
        Boolean         sdot_target;    /* used to identify s.m(/M)akefile */
        Boolean         all_parallel;                   /* TEAMWARE_MAKE_CMN */
        Boolean         assign_done;
        int foo;
        Boolean         build_failed_seen;
#ifdef DISTRIBUTED
        Boolean         building_serial;
#endif
        Name            built_last_make_run;
        Name            c_at;
#ifdef DISTRIBUTED
        Boolean         called_make = false;
#endif
        Boolean         cleanup;
        Boolean         close_report;
        Boolean         command_changed;
        Boolean         commands_done;
        Chain           conditional_targets;
        Name            conditionals;
        Boolean         continue_after_error;           /* '-k' */
        Property        current_line;
        Name            current_make_version;
        Name            current_target;
        short           debug_level;
        Cmd_line        default_rule;
        Name            default_rule_name;
        Name            default_target_to_build;
        Name            dmake_group;
        Name            dmake_max_jobs;
        Name            dmake_mode;
        DMake_mode      dmake_mode_type;
        Name            dmake_output_mode;
        DMake_output_mode       output_mode = txt1_mode;
        Name            dmake_odir;
        Name            dmake_rcfile;
        Name            done;
        Name            dot;
        Name            dot_keep_state;
        Name            dot_keep_state_file;
        Name            empty_name;
#if defined(HP_UX) ||  defined(linux)
        int             exit_status;
#endif
        Boolean         fatal_in_progress;
        int             file_number;
#if 0
        Boolean         filter_stderr;                  /* '-X' */
#endif
        Name            force;
        Name            ignore_name;
        Boolean         ignore_errors;                  /* '-i' */
        Boolean         ignore_errors_all;              /* '-i' */
        Name            init;
        int             job_msg_id;
        Boolean         keep_state;
        Name            make_state;
#ifdef TEAMWARE_MAKE_CMN
        timestruc_t     make_state_before;
#endif
        Dependency      makefiles_used;
        Name            makeflags;
//      Boolean         make_state_locked; // Moved to lib/mksh
        Name            make_version;
        char            mbs_buffer2[(MAXPATHLEN * MB_LEN_MAX)];
        char            *mbs_ptr;
```

```
128         char            *mbs_ptr2;
129         int             mtool_msgs_fd;
130         Boolean         depinfo_already_read = false;
131 #ifdef NSE
132         Name            derived_src;
133         Boolean         nse;                            /* NSE on */
134         Name            nse_backquote_seen;
135         char            nse_depinfo_lockfile[MAXPATHLEN];
136         Boolean         nse_depinfo_locked;
137         Boolean         nse_did_recursion;
138         Name            nse_shell_var_used;
139         Boolean         nse_watch_vars = false;
140         wchar_t         current_makefile[MAXPATHLEN];
141 #endif
142         Boolean         no_action_was_taken = true;     /* true if we've not **
143                                                         ** run any command   */

145         Boolean         no_parallel = false;            /* TEAMWARE_MAKE_CMN */
146 #ifdef SGE_SUPPORT
147         Boolean         grid = false;                   /* TEAMWARE_MAKE_CMN */
148 #endif
149         Name            no_parallel_name;
150         Name            not_auto;
151         Boolean         only_parallel;                  /* TEAMWARE_MAKE_CMN */
152         Boolean         parallel;                       /* TEAMWARE_MAKE_CMN */
153         Name            parallel_name;
154         Name            localhost_name;
155         int             parallel_process_cnt;
156         Percent         percent_list;
157         Dyntarget       dyntarget_list;
158         Name            plus;
159         Name            pmake_machinesfile;
160         Name            precious;
161         Name            primary_makefile;
162         Boolean         quest;                          /* '-q' */
163         short           read_trace_level;
164         Boolean         reading_dependencies = false;
165         Name            recursive_name;
166         int             recursion_level;
167         short           report_dependencies_level = 0;  /* -P */
168         Boolean         report_pwd;
169         Boolean         rewrite_statefile;
170         Running         running_list;
171         char            *sccs_dir_path;
172         Name            sccs_get_name;
173         Name            sccs_get_posix_name;
174         Cmd_line        sccs_get_rule;
175         Cmd_line        sccs_get_org_rule;
176         Cmd_line        sccs_get_posix_rule;
177         Name            get_name;
178         Cmd_line        get_rule;
179         Name            get_posix_name;
180         Cmd_line        get_posix_rule;
181         Boolean         send_mtool_msgs;                /* '-K' */
182         Boolean         all_precious;
183         Boolean         silent_all;                     /* '-s' */
184         Boolean         report_cwd;                     /* '-w' */
185         Boolean         silent;                         /* '-s' */
186         Name            silent_name;
187         char            *stderr_file = NULL;
188         char            *stdout_file = NULL;
189 #ifdef SGE_SUPPORT
190         char            script_file[MAXPATHLEN] = "";
191 #endif
192         Boolean         stdout_stderr_same;
193         Dependency      suffixes;
```



```
194         Name            suffixes_name;
195         Name            sunpro_dependencies;
196         Boolean         target_variants;
197         char            *tmpdir = NOCATGETS("/tmp");
198         char            *temp_file_directory = NOCATGETS(".");
199         Name            temp_file_name;
200         short           temp_file_number;
201         time_t          timing_start;
202         wchar_t         *top_level_target;
203         Boolean         touch;                          /* '-t' */
204         Boolean         trace_reader;                   /* '-D' */
205         Boolean         build_unconditional;            /* '-u' */
206         pathpt          vroot_path = VROOT_DEFAULT;
207         Name            wait_name;
208         wchar_t         wcs_buffer2[MAXPATHLEN];
209         wchar_t         *wcs_ptr;
210         wchar_t         *wcs_ptr2;
211         nl_catd         catd;
212         long int        hostid;

214 /*
215  * File table of contents
216  */

218 #endif /* ! codereview */
```

```
**********************************************************
   43664 Wed May 20 11:04:08 2015
new/usr/src/cmd/make/bin/make/common/implicit.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)implicit.cc 1.64 06/12/12
 */

#pragma ident   "@(#)implicit.cc        1.64    06/12/12"

/*
 *      implicit.c
 *
 *      Handle suffix and percent rules
 */

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

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

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

/*
 * File table of contents
 */
extern  Doname          find_suffix_rule(Name target, Name target_body, Name tar
extern  Doname          find_ar_suffix_rule(register Name target, Name true_targ
```



```
extern  Doname          find_double_suffix_rule(register Name target, Property *
extern  void            build_suffix_list(register Name target_suffix);
extern  Doname          find_percent_rule(register Name target, Property *comman
static  void            create_target_group_and_dependencies_list(Name target, P
static  Boolean         match_found_with_pattern(Name target, Percent pat_rule,
static  void            construct_string_from_pattern(Percent pat_rule, String p
static  Boolean         dependency_exists(Name target, Property line);
extern  Property        maybe_append_prop(Name, Property_id);
extern  void            add_target_to_chain(Name target, Chain * query);

/*
 *      find_suffix_rule(target, target_body, target_suffix, command, rechecking
 *
 *      Does the lookup for single and double suffix rules.
 *      It calls build_suffix_list() to build the list of possible suffixes
 *      for the given target.
 *      It then scans the list to find the first possible source file that
 *      exists. This is done by concatenating the body of the target name
 *      (target name less target suffix) and the source suffix and checking
 *      if the resulting file exists.
 *
 *      Return value:
 *                              Indicates if search failed or not
 *
 *      Parameters:
 *              target          The target we need a rule for
 *              target_body     The target name without the suffix
 *              target_suffix   The suffix of the target
 *              command         Pointer to slot to deposit cmd in if found
 *              rechecking      true if we are rechecking target which depends
 *                              on conditional macro and keep_state is set
 *
 *      Global variables used:
 *              debug_level     Indicates how much tracing to do
 *              recursion_level Used for tracing
 */

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

static Boolean actual_doname = false;

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

static Boolean we_are_in_tilde = false;

Doname
find_suffix_rule(Name target, Name target_body, Name target_suffix, Property *co
{
        static wchar_t          static_string_buf_3M [ 3 * MAXPATHLEN ];
        Name                    true_target = target;
        wchar_t                 *sourcename = (wchar_t*)static_string_buf_3M;
        register wchar_t        *put_suffix;
        register Property       source_suffix;
        register Name           source;
        Doname                  result;
        register Property       line;
        extern Boolean          tilde_rule;
        Boolean                 name_found = true;
        Boolean                 posix_tilde_attempt = true;
```

```
128         int                     src_len = MAXPATHLEN + strlen(target_body->strin
129
130         /*
131          * To avoid infinite recursion
132          */
133         if(we_are_in_tilde) {
134                 we_are_in_tilde = false;
135                 return(build_dont_know);
136         }
137
138         /*
139          * If the target is a constructed one for a "::" target,
140          * we need to consider that.
141          */
142         if (target->has_target_prop) {
143                 true_target = get_prop(target->prop,
144                                        target_prop)->body.target.target;
145         }
146         if (debug_level > 1) {
147                 (void) printf(NOCATGETS("%*sfind_suffix_rule(%s,%s,%s)\n"),
148                               recursion_level,
149                               "",
150                               true_target->string_mb,
151                               target_body->string_mb,
152                               target_suffix->string_mb);
153         }
154         if (command != NULL) {
155                 if ((true_target->suffix_scan_done == true) && (*command == NULL
156                         return build_ok;
157                 }
158         }
159         true_target->suffix_scan_done = true;
160         /*
161          * Enter all names from the directory where the target lives as
162          * files that makes sense.
163          * This will make finding the synthesized source possible.
164          */
165         read_directory_of_file(target_body);
166         /* Cache the suffixes for this target suffix if not done. */
167         if (!target_suffix->has_read_suffixes) {
168                 build_suffix_list(target_suffix);
169         }
170         /* Preload the sourcename vector with the head of the target name. */
171         if (src_len >= sizeof(static_string_buf_3M)) {
172                 sourcename = ALLOC_WC(src_len);
173         }
174         (void) mbstowcs(sourcename,
175                         target_body->string_mb,
176                         (int) target_body->hash.length);
177         put_suffix = sourcename + target_body->hash.length;
178         /* Scan the suffix list for the target if one exists. */
179         if (target_suffix->has_suffixes) {
180 posix_attempts:
181                 for (source_suffix = get_prop(target_suffix->prop,
182                                               suffix_prop);
183                      source_suffix != NULL;
184                      source_suffix = get_prop(source_suffix->next,
185                                               suffix_prop)) {
186                         /* Build the synthesized source name. */
187                         (void) mbstowcs(put_suffix,
188                                         source_suffix->body.
189                                         suffix.suffix->string_mb,
190                                         (int) source_suffix->body.
191                                         suffix.suffix->hash.length);
192                         put_suffix[source_suffix->body.
193                                    suffix.suffix->hash.length] =
```



```
194                                   (int) nul_char;
195                         if (debug_level > 1) {
196                                 WCSTOMBS(mbs_buffer, sourcename);
197                                 (void) printf(catgets(catd, 1, 218, "%*sTrying %
198                                               recursion_level,
199                                               "",
200                                               mbs_buffer);
201                         }
202                         source = getname_fn(sourcename, FIND_LENGTH, false, &nam
203                         /*
204                          * If the source file is not registered as
205                          * a file, this source suffix did not match.
206                          */
207                         if(vpath_defined && !posix && !svr4) {
208                                 (void) exists(source);
209                         }
210                         if (!source->stat.is_file) {
211                            if(!(posix|svr4))
212                            {
213                                 if(!name_found) {
214                                         free_name(source);
215                                 }
216                                 continue;
217                            }
218
219                            /* following code will ensure that the corresponding
220                            ** tilde rules are executed when corresponding s. fil
221                            ** exists in the current directory. Though the curren
222                            ** target ends with a ~ character, there wont be any
223                            ** any file in the current directory with that suffix
224                            ** as it's fictitious. Even if it exists, it'll
225                            ** execute all the rules for the ~ target.
226                            */
227
228                            if(source->string_mb[source->hash.length - 1] == '~'
229                               ( svr4 || posix_tilde_attempt ) )
230                            {
231                                 char *p, *np;
232                                 char *tmpbuf;
233
234                                 tmpbuf = getmem(source->hash.length + 8);
235                                 /* + 8 to add "s." or "SCCS/s." */
236                                 memset(tmpbuf,0,source->hash.length + 8);
237                                 source->string_mb[source->hash.length - 1] = '\0
238                                 if(p = (char *) memchr((char *)source->string_mb
239                                 {
240                                   while(1) {
241                                     if(np = (char *) memchr((char *)p+1,'/',sour
242                                       p = np;
243                                     } else {break;}
244                                   }
245                                   /* copy everything including '/' */
246                                   strncpy(tmpbuf, source->string_mb, p - source-
247                                   strcat(tmpbuf, NOCATGETS("s."));
248                                   strcat(tmpbuf, p+1);
249                                   retmem((wchar_t *) source->string_mb);
250                                   source->string_mb = tmpbuf;
251
252                                 } else {
253                                   strcpy(tmpbuf, NOCATGETS("s."));
254                                   strcat(tmpbuf, source->string_mb);
255                                   retmem((wchar_t *) source->string_mb);
256                                   source->string_mb = tmpbuf;
257
258                                 }
259                                 source->hash.length = strlen(source->string_mb);
```

```
260                                 if(exists(source) == file_doesnt_exist)
261                                   continue;
262                                 tilde_rule = true;
263                                 we_are_in_tilde = true;
264                            } else {
265                                 if(!name_found) {
266                                         free_name(source);
267                                 }
268                                 continue;
269                            }
270                         } else {
271                            if(posix && posix_tilde_attempt) {
272                                 if(exists(source) == file_doesnt_exist) {
273                                         if(!name_found) {
274                                                 free_name(source);
275                                         }
276                                         continue;
277                                 }
278                            }
279                         }
280
281                         if (command != NULL) {
282                                 if(!name_found) {
283                                         store_name(source);
284                                 }
285                                 /*
286                                  * The source file is a file.
287                                  * Make sure it is up to date.
288                                  */
289                                 if (dependency_exists(source,
290                                                       get_prop(target->prop,
291                                                                line_prop))) {
292                                         result = (Doname) source->state;
293                                 } else {
294 #ifdef NSE
295                                         nse_check_derived_src(target, source->st
296                                          source_suffix->body.suffix.command_temp
297 #endif
298 #if 0  /* with_squiggle sends false, which is buggy. : djay */
299                                         result = doname(source,
300                                                         (Boolean) source_suffix-
301                                                         suffix.suffix->with_squi
302                                                         true);
303 #else
304                                         result = doname(source,
305                                                         true,
306                                                         true);
307 #endif
308                                 }
309                         } else {
310                                 result = target_can_be_built(source);
311
312                                 if (result == build_ok) {
313                                         return result;
314                                 } else {
315                                         if(!name_found) {
316                                                 free_name(source);
317                                         }
318                                         continue;
319                                 }
320                         }
321
322                         switch (result) {
323                         case build_dont_know:
324                                 /*
325                                  * If we still can't build the source,
```



```
326                                  * this rule is not a match,
327                                  * try the next one.
328                                  */
329                                 if (source->stat.time == file_doesnt_exist) {
330                                         if(!name_found) {
331                                                 free_name(source);
332                                         }
333                                         continue;
334                                 }
335                         case build_running:
336                                 if(!name_found) {
337                                         store_name(source);
338                                 }
339                                 true_target->suffix_scan_done = false;
340                                 line = maybe_append_prop(target, line_prop);
341                                 enter_dependency(line, source, false);
342                                 line->body.line.target = true_target;
343                                 return build_running;
344                         case build_ok:
345                                 if(!name_found) {
346                                         store_name(source);
347                                 }
348                                 break;
349                         case build_failed:
350                                 if(!name_found) {
351                                         store_name(source);
352                                 }
353                                 if (sourcename != static_string_buf_3M) {
354                                         retmem(sourcename);
355                                 }
356                                 return build_failed;
357                         }
358
359                         if (debug_level > 1) {
360                                 WCSTOMBS(mbs_buffer, sourcename);
361                                 (void) printf(catgets(catd, 1, 219, "%*sFound %s
362                                               recursion_level,
363                                               "",
364                                               mbs_buffer);
365                         }
366
367                         if (source->depends_on_conditional) {
368                                 target->depends_on_conditional = true;
369                         }
370 /*
371  * Since it is possible that the same target is built several times during
372  * the make run, we have to patch the target with all information we found
373  * here. Thus, the target will have an explicit rule the next time around.
374  */
375                         line = maybe_append_prop(target, line_prop);
376                         if (*command == NULL) {
377                                 *command = line;
378                         }
379                         if ((source->stat.time > (*command)->body.line.dependenc
380                             (debug_level > 1)) {
381                                 (void) printf(catgets(catd, 1, 220, "%*sDate(%s)
382                                               recursion_level,
383                                               "",
384                                               source->string_mb,
385                                               time_to_string(source->
386                                                              stat.time),
387                                               true_target->string_mb,
388                                               time_to_string((*command)->
389                                                              body.line.
390                                                              dependency_time));
391                         }
```

```
392                         /*
393                          * Determine if this new dependency made the
394                          * target out of date.
395                          */
396                         (*command)->body.line.dependency_time =
397                           MAX((*command)->body.line.dependency_time,
398                               source->stat.time);
399                         Boolean out_of_date;
400                         if (target->is_member) {
401                                 out_of_date = (Boolean) OUT_OF_DATE_SEC(target->
402                                                                         (*comman
403                         } else {
404                                 out_of_date = (Boolean) OUT_OF_DATE(target->stat
405                                                                     (*command)->
406                         }
407                         if (build_unconditional || out_of_date) {
408                                 if(!rechecking) {
409                                         line->body.line.is_out_of_date = true;
410                                 }
411                                 if (debug_level > 0) {
412                                         (void) printf(catgets(catd, 1, 221, "%*s
413                                                       recursion_level,
414                                                       "",
415                                                       true_target->string_mb,
416                                                       source_suffix->body.suffix
417                                                       target_suffix->string_mb,
418                                                       source->string_mb);
419                                 }
420                         }
421                         /*
422                          * Add the implicit rule as the target's explicit
423                          * rule if none actually given, and register
424                          * dependency.
425                          * The time checking above really should be
426                          * conditional on actual use of implicit rule
427                          * as well.
428                          */
429                         line->body.line.sccs_command = false;
430                         if (line->body.line.command_template == NULL) {
431                                 line->body.line.command_template =
432                                   source_suffix->body.suffix.command_template;
433                         }
434                         enter_dependency(line, source, false);
435                         line->body.line.target = true_target;
436                         /*
437                          * Also make sure the rule is built with
438                          * $* and $< bound properly.
439                          */
440                         line->body.line.star = target_body;
441                         if(svr4|posix) {
442                           char * p;
443                           char tstr[256];
444                           extern Boolean dollarless_flag;
445                           extern Name dollarless_value;

447                           if(tilde_rule) {
448                               MBSTOWCS(wcs_buffer, NOCATGETS(source->string_mb))
449                               dollarless_value = GETNAME(wcs_buffer,FIND_LENGTH)
450                           }
451                           else {
452                                    dollarless_flag = false;
453                           }
454                         }
455                         line->body.line.less = source;
456                         line->body.line.percent = NULL;
457                         add_target_to_chain(source, &(line->body.line.query));
```



```
458                                 if (sourcename != static_string_buf_3M) {
459                                         retmem(sourcename);
460                                 }
461                                 return build_ok;
462                         }
463                         if(posix && posix_tilde_attempt) {
464                                 posix_tilde_attempt = false;
465                                 goto posix_attempts;
466                         }
467                         if ((command != NULL) &&
468                             ((*command) != NULL) &&
469                             ((*command)->body.line.star == NULL)) {
470                                 (*command)->body.line.star = target_body;
471                         }
472                 }
473                 if (sourcename != static_string_buf_3M) {
474                         retmem(sourcename);
475                 }
476                 /* Return here in case no rule matched the target */
477                 return build_dont_know;
478         }

480         /*
481          *      find_ar_suffix_rule(target, true_target, command, rechecking)
482          *
483          *      Scans the .SUFFIXES list and tries
484          *      to find a suffix on it that matches the tail of the target member name.
485          *      If it finds a matching suffix it calls find_suffix_rule() to find
486          *      a rule for the target using the suffix ".a".
487          *
488          *      Return value:
489          *                              Indicates if search failed or not
490          *
491          *      Parameters:
492          *              target          The target we need a rule for
493          *              true_target     The proper name
494          *              command         Pointer to slot where we stuff cmd, if found
495          *              rechecking      true if we are rechecking target which depends
496          *                              on conditional macro and keep_state is set
497          *
498          *      Global variables used:
499          *              debug_level     Indicates how much tracing to do
500          *              dot_a           The Name ".a", compared against
501          *              recursion_level Used for tracing
502          *              suffixes        List of suffixes used for scan (from .SUFFIXES)
503          */
504         Doname
505         find_ar_suffix_rule(register Name target, Name true_target, Property *command, B
506         {
507                 wchar_t                 *target_end;
508                 register Dependency     suffix;
509                 register int            suffix_length;
510                 Property                line;
511                 Name                    body;
512                 static Name             dot_a;

514                 Wstring                 targ_string(true_target);
515                 Wstring                 suf_string;

517                 if (dot_a == NULL) {
518                         MBSTOWCS(wcs_buffer, NOCATGETS(".a"));
519                         dot_a = GETNAME(wcs_buffer, FIND_LENGTH);
520                 }
521                 target_end = targ_string.get_string() + true_target->hash.length;

523                 /*
```

```
524          * We compare the tail of the target name with the suffixes
525          * from .SUFFIXES.
526          */
527         if (debug_level > 1) {
528                 (void) printf(NOCATGETS("%*sfind_ar_suffix_rule(%s)\n"),
529                               recursion_level,
530                               "",
531                               true_target->string_mb);
532         }
533         /*
534          * Scan the .SUFFIXES list to see if the target matches any of
535          * those suffixes.
536          */
537         for (suffix = suffixes; suffix != NULL; suffix = suffix->next) {
538                 /* Compare one suffix. */
539                 suffix_length = suffix->name->hash.length;
540                 suf_string.init(suffix->name);
541                 if (!IS_WEQUALN(suf_string.get_string(),
542                                 target_end - suffix_length,
543                                 suffix_length)) {
544                         goto not_this_one;
545                 }
546                 /*
547                  * The target tail matched a suffix from the .SUFFIXES list.
548                  * Now check for a rule to match.
549                  */
550                 target->suffix_scan_done = false;
551                 body = GETNAME(targ_string.get_string(),
552                                (int)(true_target->hash.length -
553                                      suffix_length));
554                 we_are_in_tilde = false;
555                 switch (find_suffix_rule(target,
556                                          body,
557                                          dot_a,
558                                          command,
559                                          rechecking)) {
560                 case build_ok:
561                         line = get_prop(target->prop, line_prop);
562                         line->body.line.star = body;
563                         return build_ok;
564                 case build_running:
565                         return build_running;
566                 }
567                 /*
568                  * If no rule was found, we try the next suffix to see
569                  * if it matches the target tail, and so on.
570                  * Go here if the suffix did not match the target tail.
571                  */
572         not_this_one:;
573         }
574         return build_dont_know;
575 }

577 /*
578  *      find_double_suffix_rule(target, command, rechecking)
579  *
580  *      Scans the .SUFFIXES list and tries
581  *      to find a suffix on it that matches the tail of the target name.
582  *      If it finds a matching suffix it calls find_suffix_rule() to find
583  *      a rule for the target.
584  *
585  *      Return value:
586  *                              Indicates if scan failed or not
587  *
588  *      Parameters:
589  *              target          Target we need a rule for
```



```
590  *              command         Pointer to slot where we stuff cmd, if found
591  *              rechecking      true if we are rechecking target which depends
592  *                              on conditional macro and keep_state is set
593  *
594  *      Global variables used:
595  *              debug_level     Indicates how much tracing to do
596  *              recursion_level Used for tracing
597  *              suffixes        List of suffixes used for scan (from .SUFFIXES)
598  */
599 Doname
600 find_double_suffix_rule(register Name target, Property *command, Boolean recheck
601 {
602         Name                    true_target = target;
603         Name                    target_body;
604         register wchar_t        *target_end;
605         register Dependency     suffix;
606         register int            suffix_length;
607         Boolean                 scanned_once = false;
608         Boolean                 name_found = true;

610         Wstring                 targ_string;
611         Wstring                 suf_string;

613         /*
614          * If the target is a constructed one for a "::" target,
615          * we need to consider that.
616          */
617         if (target->has_target_prop) {
618                 true_target = get_prop(target->prop,
619                                        target_prop)->body.target.target;
620         }
621         targ_string.init(true_target);

623         /*
624          * We compare the tail of the target name with the
625          * suffixes from .SUFFIXES.
626          */
627         target_end = targ_string.get_string() + true_target->hash.length;
628         if (debug_level > 1) {
629                 (void) printf(NOCATGETS("%*sfind_double_suffix_rule(%s)\n"),
630                               recursion_level,
631                               "",
632                               true_target->string_mb);
633         }
634         /*
635          * Scan the .SUFFIXES list to see if the target matches
636          * any of those suffixes.
637          */
638         for (suffix = suffixes; suffix != NULL; suffix = suffix->next) {
639                 target->suffix_scan_done = false;
640                 true_target->suffix_scan_done = false;
641                 /* Compare one suffix. */
642                 suffix_length = suffix->name->hash.length;
643                 suf_string.init(suffix->name);
644                 /* Check the lengths, or else RTC will report rua. */
645                 if (true_target->hash.length < suffix_length) {
646                         goto not_this_one;
647                 } else if (!IS_WEQUALN(suf_string.get_string(),
648                                        (target_end - suffix_length),
649                                        suffix_length)) {
650                         goto not_this_one;
651                 }
652                 /*
653                  * The target tail matched a suffix from the .SUFFIXES list.
654                  * Now check for a rule to match.
655                  */
```

```
656                 we_are_in_tilde = false;
657                 target_body = GETNAME(
658                         targ_string.get_string(),
659                         (int)(true_target->hash.length - suffix_length)
660                 );
661                 switch (find_suffix_rule(target,
662                                          target_body,
663                                          suffix->name,
664                                          command,
665                                          rechecking)) {
666                 case build_ok:
667                         return build_ok;
668                 case build_running:
669                         return build_running;
670                 }
671                 if (true_target->suffix_scan_done == true) {
672                         scanned_once = true;
673                 }
674                 /*
675                  * If no rule was found, we try the next suffix to see
676                  * if it matches the target tail. And so on.
677                  * Go here if the suffix did not match the target tail.
678                  */
679         not_this_one:;
680         }
681         if (scanned_once)
682                 true_target->suffix_scan_done = true;
683         return build_dont_know;
684 }

686 /*
687  *      build_suffix_list(target_suffix)
688  *
689  *      Scans the .SUFFIXES list and figures out
690  *      which suffixes this target can be derived from.
691  *      The target itself is not know here, we just know the suffix of the
692  *      target. For each suffix on the list the target can be derived iff
693  *      a rule exists for the name "<suffix-on-list><target-suffix>".
694  *      A list of all possible building suffixes is built, with the rule for
695  *      each, and tacked to the target suffix nameblock.
696  *
697  *      Parameters:
698  *              target_suffix   The suffix we build a match list for
699  *
700  *      Global variables used:
701  *              debug_level     Indicates how much tracing to do
702  *              recursion_level Used for tracing
703  *              suffixes        List of suffixes used for scan (from .SUFFIXES)
704  *              working_on_targets Indicates that this is a real target
705  */
706 void
707 build_suffix_list(register Name target_suffix)
708 {
709         register Dependency     source_suffix;
710         wchar_t                 rule_name[MAXPATHLEN];
711         register Property       line;
712         register Property       suffix;
713         Name                    rule;

715         /* If this is before default.mk has been read we just return to try */
716         /* again later */
717         if ((suffixes == NULL) || !working_on_targets) {
718                 return;
719         }
720         if (debug_level > 1) {
721                 (void) printf(NOCATGETS("%*sbuild_suffix_list(%s) "),
```



```
722                               recursion_level,
723                               "",
724                               target_suffix->string_mb);
725         }
726         /* Mark the target suffix saying we cashed its list */
727         target_suffix->has_read_suffixes = true;
728         /* Scan the .SUFFIXES list */
729         for (source_suffix = suffixes;
730              source_suffix != NULL;
731              source_suffix = source_suffix->next) {
732                 /*
733                  * Build the name "<suffix-on-list><target-suffix>".
734                  * (a popular one would be ".c.o").
735                  */
736                 (void) mbstowcs(rule_name,
737                                 source_suffix->name->string_mb,
738                                 (int) source_suffix->name->hash.length);
739                 (void) mbstowcs(rule_name + source_suffix->name->hash.length,
740                                 target_suffix->string_mb,
741                                 (int) target_suffix->hash.length);
742                 /*
743                  * Check if that name has a rule. If not, it cannot match
744                  * any implicit rule scan and is ignored.
745                  * The GETNAME() call only checks for presence, it will not
746                  * enter the name if it is not defined.
747                  */
748                 if (((rule = getname_fn(rule_name,
749                                         (int) (source_suffix->name->
750                                                hash.length +
751                                                target_suffix->hash.length),
752                                         true)) != NULL) &&
753                     ((line = get_prop(rule->prop, line_prop)) != NULL)) {
754                         if (debug_level > 1) {
755                                 (void) printf("%s ", rule->string_mb);
756                         }
757                         /*
758                          * This makes it possible to quickly determine if
759                          * it will pay to look for a suffix property.
760                          */
761                         target_suffix->has_suffixes = true;
762                         /*
763                          * Add the suffix property to the target suffix
764                          * and save the rule with it.
765                          * All information the implicit rule scanner need
766                          * is saved in the suffix property.
767                          */
768                         suffix = append_prop(target_suffix, suffix_prop);
769                         suffix->body.suffix.suffix = source_suffix->name;
770                         suffix->body.suffix.command_template =
771                           line->body.line.command_template;
772                 }
773         }
774         if (debug_level > 1) {
775                 (void) printf("\n");
776         }
777 }

779 /*
780  *      find_percent_rule(target, command, rechecking)
781  *
782  *      Tries to find a rule from the list of wildcard matched rules.
783  *      It scans the list attempting to match the target.
784  *      For each target match it checks if the corresponding source exists.
785  *      If it does the match is returned.
786  *      The percent_list is built at makefile read time.
787  *      Each percent rule get one entry on the list.
```

```
 *
 *      Return value:
 *                              Indicates if the scan failed or not
 *
 *      Parameters:
 *              target          The target we need a rule for
 *              command         Pointer to slot where we stuff cmd, if found
 *              rechecking      true if we are rechecking target which depends
 *                              on conditional macro and keep_state is set
 *
 *      Global variables used:
 *              debug_level     Indicates how much tracing to do
 *              percent_list    List of all percent rules
 *              recursion_level Used for tracing
 *              empty_name
 */
Doname
find_percent_rule(register Name target, Property *command, Boolean rechecking)
{
        register Percent        pat_rule, pat_depe;
        register Name           depe_to_check;
        register Dependency     depe;
        register Property       line;
        String_rec              string;
        wchar_t                 string_buf[STRING_BUFFER_LENGTH];
        String_rec              percent;
        wchar_t                 percent_buf[STRING_BUFFER_LENGTH];
        Name                    true_target = target;
        Name                    less;
        Boolean                 nonpattern_less;
        Boolean                 dep_name_found = false;
        Doname                  result = build_dont_know;
        Percent                 rule_candidate = NULL;
        Boolean                 rule_maybe_ok;
        Boolean                 is_pattern;

        /* If the target is constructed for a "::" target we consider that */
        if (target->has_target_prop) {
                true_target = get_prop(target->prop,
                                       target_prop)->body.target.target;
        }
        if (target->has_long_member_name) {
                true_target = get_prop(target->prop,
                                       long_member_name_prop)->body.long_member_
        }
        if (debug_level > 1) {
                (void) printf(catgets(catd, 1, 222, "%*sLooking for %% rule for
                              recursion_level,
                              "",
                              true_target->string_mb);
        }
        for (pat_rule = percent_list;
             pat_rule != NULL;
             pat_rule = pat_rule->next) {
                /* Avoid infinite recursion when expanding patterns */
                if (pat_rule->being_expanded == true) {
                        continue;
                }

                /* Mark this pat_rule as "maybe ok". If no % rule is found
                   make will use this rule. The following algorithm is used:
                   1) make scans all pattern rules in order to find the rule
                      where ALL dependencies, including nonpattern ones, exist o
                      can be built (GNU behaviour). If such rule is found make
                      will apply it.
                   2) During this check make also remembers the first pattern ru
```



```
                      where all PATTERN dependencies can be build (no matter wha
                      happens with nonpattern dependencies).
                   3) If no rule satisfying 1) is found, make will apply the rul
                      remembered in 2) if there is one.
                */
                rule_maybe_ok = true;

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

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

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

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

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

```
921                                     pat_rule->being_expanded = true;

923                                     /* suppress message output */
924                                     int save_debug_level = debug_level;
925                                     debug_level = 0;

927                                     /* check whether dependency can be built
928                                     if (dependency_exists(depe_to_check,
929                                         get_prop(target->prop,
930                                                  line_prop)))
931                                     {
932                                             result = (Doname) depe_to_check-
933                                     } else {
934                                             if(actual_doname) {
935                                                     result = doname(depe_to_
936                                             } else {
937                                                     result = target_can_be_b
938                                             }
939                                             if(!dep_name_found) {
940                                                     if(result != build_ok &&
941                                                             free_name(depe_t
942                                                     } else {
943                                                             store_name(depe_
944                                                     }
945                                             }
946                                     }
947                                     if(result != build_ok && is_pattern) {
948                                             rule_maybe_ok = false;
949                                     }

951                                     /* restore debug_level */
952                                     debug_level = save_debug_level;
953                             }

955                             if (pat_depe->name->percent) {
956                                     if (string.free_after_use) {
957                                             retmem(string.buffer.start);
958                                     }
959                             }
960                             /* make can't figure out how to make this depend
961                             if (result != build_ok && result != build_runnin
962                                     pat_rule->being_expanded = false;
963                                     break;
964                             }
965                     }
966             } else {
967                     result = build_ok;
968             }

970             /* this pattern rule is the needed one since all dependencies co
971             if (result == build_ok || result == build_running) {
972                     break;
973             }

975             /* Make does not know how to build some of dependencies from thi
976                But if all "pattern" dependencies can be built, we remember t
977                as a candidate for the case if no other pattern rule found.
978             */
979             if(rule_maybe_ok && rule_candidate == NULL) {
980                     rule_candidate = pat_rule;
981             }
982     }

984     /* if no pattern matching rule was found, use the remembered candidate
985        or return build_dont_know if there is no candidate.
```



```
986     */
987     if (result != build_ok && result != build_running) {
988             if(rule_candidate) {
989                     pat_rule = rule_candidate;
990             } else {
991                     return build_dont_know;
992             }
993     }

995     /* if we are performing only check whether dependency could be built wit
996        return success */
997     if (command == NULL) {
998             if(pat_rule != NULL) {
999                     pat_rule->being_expanded = false;
1000            }
1001            return result;
1002    }

1004    if (debug_level > 1) {
1005            (void) printf(catgets(catd, 1, 224, "%*sMatched %s:"),
1006                                    recursion_level,
1007                                    "",
1008                                    target->string_mb);

1010            for (pat_depe = pat_rule->dependencies;
1011                 pat_depe != NULL;
1012                 pat_depe = pat_depe->next) {
1013                    if (pat_depe->name->percent) {
1014                            INIT_STRING_FROM_STACK(string, string_buf);
1015                            construct_string_from_pattern(pat_depe, &percent
1016                            depe_to_check = GETNAME(string.buffer.start, FIN
1017                    } else {
1018                            depe_to_check = pat_depe->name;
1019                            if(depe_to_check->dollar) {
1020                                    INIT_STRING_FROM_STACK(string, string_bu
1021                                    expand_value(depe_to_check, &string, fal
1022                                    depe_to_check = GETNAME(string.buffer.st
1023                            }
1024                    }

1026                    if (depe_to_check != empty_name) {
1027                            (void) printf(" %s", depe_to_check->string_mb);
1028                    }
1029            }

1031            (void) printf(catgets(catd, 1, 225, " from: %s:"),
1032                                  pat_rule->name->string_mb);

1034            for (pat_depe = pat_rule->dependencies;
1035                 pat_depe != NULL;
1036                 pat_depe = pat_depe->next) {
1037                    (void) printf(" %s", pat_depe->name->string_mb);
1038            }

1040            (void) printf("\n");
1041    }

1043    if (true_target->colons == no_colon) {
1044            true_target->colons = one_colon;
1045    }

1047    /* create deppendency list and target group from matched pattern rule */
1048    create_target_group_and_dependencies_list(target, pat_rule, &percent);

1050    /* save command */
1051    line = get_prop(target->prop, line_prop);
```

```
1052         *command = line;

1054         /* free query chain if one exist */
1055         while(line->body.line.query != NULL) {
1056                 Chain to_free = line->body.line.query;
1057                 line->body.line.query = line->body.line.query->next;
1058                 retmem_mb((char *) to_free);
1059         }

1061         if (line->body.line.dependencies != NULL) {
1062                 /* build all collected dependencies */
1063                 for (depe = line->body.line.dependencies;
1064                      depe != NULL;
1065                      depe = depe->next) {
1066                    actual_doname = true;
1067                    result = doname_check(depe->name, true, true, depe->auto

1069                    actual_doname = false;
1070                    if (result == build_failed) {
1071                           pat_rule->being_expanded = false;
1072                           return build_failed;
1073                    }
1074                    if (result == build_running) {
1075                           pat_rule->being_expanded = false;
1076                           return build_running;
1077                    }

1079                    if ((depe->name->stat.time > line->body.line.dependency_
1080                        (debug_level > 1)) {
1081                           (void) printf(catgets(catd, 1, 226, "%*sDate(%s)
1082                                         recursion_level,
1083                                         "",
1084                                         depe->name->string_mb,
1085                                         time_to_string(depe->name->stat.ti
1086                                         true_target->string_mb,
1087                                         time_to_string(line->body.line.dep
1088                    }

1090                    line->body.line.dependency_time =
1091                      MAX(line->body.line.dependency_time, depe->name->stat.

1093                    /* determine whether this dependency made target out of
1094                    Boolean out_of_date;
1095                    if (target->is_member || depe->name->is_member) {
1096                           out_of_date = (Boolean) OUT_OF_DATE_SEC(target->
1097                    } else {
1098                           out_of_date = (Boolean) OUT_OF_DATE(target->stat
1099                    }
1100                    if (build_unconditional || out_of_date) {
1101                           if(!rechecking) {
1102                                  line->body.line.is_out_of_date = true;
1103                           }
1104                           add_target_to_chain(depe->name, &(line->body.lin

1106                           if (debug_level > 0) {
1107                                  (void) printf(catgets(catd, 1, 227, "%*s
1108                                                recursion_level,
1109                                                "",
1110                                                true_target->string_mb,
1111                                                pat_rule->name->string_mb)

1113                                  for (pat_depe = pat_rule->dependencies;
1114                                       pat_depe != NULL;
1115                                       pat_depe = pat_depe->next) {
1116                                          (void) printf(" %s", pat_depe->n
1117                                  }
```



```
1119                                          (void) printf(catgets(catd, 1, 228, " be
1120                                                        depe->name->string_mb);
1121                                  }
1122                           }
1123                    }
1124             } else {
1125                    if ((true_target->stat.time <= file_doesnt_exist) ||
1126                        (true_target->stat.time < line->body.line.dependency_time))
1127                           if(!rechecking) {
1128                                  line->body.line.is_out_of_date = true;
1129                           }
1130                           if (debug_level > 0) {
1131                                  (void) printf(catgets(catd, 1, 229, "%*sBuilding
1132                                                recursion_level,
1133                                                "",
1134                                                true_target->string_mb,
1135                                                pat_rule->name->string_mb,
1136                                                (target->stat.time > file_doesnt_e
1137                                                catgets(catd, 1, 230, "because it
1138                                                catgets(catd, 1, 236, "because it
1139                           }
1140                    }
1141             }

1143             /* enter explicit rule from percent rule */
1144             Name lmn_target = true_target;
1145             if (true_target->has_long_member_name) {
1146                    lmn_target = get_prop(true_target->prop, long_member_name_prop)-
1147             }
1148             line->body.line.sccs_command = false;
1149             line->body.line.target = true_target;
1150             line->body.line.command_template = pat_rule->command_template;
1151             line->body.line.star = GETNAME(percent.buffer.start, FIND_LENGTH);
1152             line->body.line.less = less;

1154             if (lmn_target->parenleft) {
1155                    Wstring lmn_string(lmn_target);

1157                    wchar_t *left = (wchar_t *) wschr(lmn_string.get_string(), (int)
1158                    wchar_t *right = (wchar_t *) wschr(lmn_string.get_string(), (int

1160                    if ((left == NULL) || (right == NULL)) {
1161                           line->body.line.percent = NULL;
1162                    } else {
1163                           line->body.line.percent = GETNAME(left + 1, right - left
1164                    }
1165             } else {
1166                    line->body.line.percent = NULL;
1167             }
1168             pat_rule->being_expanded = false;

1170 #ifdef TEAMWARE_MAKE_CMN
1171             /*
1172              * This #ifdef fixes a dmake bug, but introduces bugid 1136156.
1173              */
1174             return result;
1175 #else
1176             return build_ok;
1177 #endif
1178 }

1180 /*
1181  *      match_found_with_pattern
1182  *           ( target, pat_rule, percent, percent_buf)
1183  *
```

```
 *      matches "target->string" with a % pattern.
 *      If pattern contains a MACRO definition, it's expanded first.
 *
 *      Return value:
 *                              true if a match was found
 *
 *      Parameters:
 *              target          The target we're trying to match
 *              pattern
 *              percent         record that contains "percent_buf" below
 *              percent_buf     This is where the patched % part of pattern is s
 *
 */

static Boolean
match_found_with_pattern(Name target, Percent pat_rule, String percent, wchar_t
        String_rec              string;
        wchar_t                 string_buf[STRING_BUFFER_LENGTH];

        /* construct prefix string and check whether prefix matches */
        Name prefix = pat_rule->patterns[0];
        int prefix_length;

        Wstring targ_string(target);
        Wstring pref_string(prefix);
        Wstring suf_string;

        if (prefix->dollar) {
                INIT_STRING_FROM_STACK(string, string_buf);
                expand_value(prefix, &string, false);
                prefix_length = string.text.p - string.buffer.start;
                if ((string.buffer.start[0] == (int) period_char) &&
                    (string.buffer.start[1] == (int) slash_char)) {
                        string.buffer.start += 2;
                        prefix_length -= 2;
                }
                if (!targ_string.equaln(string.buffer.start, prefix_length)) {
                        return false;
                }
        } else {
                prefix_length = prefix->hash.length;
                if (!targ_string.equaln(&pref_string, prefix_length)) {
                        return false;
                }
        }

        /* do the same with pattern suffix */
        Name suffix = pat_rule->patterns[pat_rule->patterns_total - 1];
        suf_string.init(suffix);

        int suffix_length;
        if (suffix->dollar) {
                INIT_STRING_FROM_STACK(string, string_buf);
                expand_value(suffix, &string, false);
                suffix_length = string.text.p - string.buffer.start;
                if(suffix_length > target->hash.length) {
                        return false;
                }
                if (!targ_string.equal(string.buffer.start, target->hash.length
                        return false;
                }
        } else {
                suffix_length = (int) suffix->hash.length;
                if(suffix_length > target->hash.length) {
                        return false;
                }
```



```
                if (!targ_string.equal(&suf_string, target->hash.length - suffix
                        return false;
                }
        }

        Boolean match_found = false;
        int percent_length = target->hash.length - prefix_length - suffix_length

        while (!match_found && (percent_length >= 0)) {
                /* init result string */
                INIT_STRING_FROM_STACK(string, string_buf);

                /* init percent string */
                percent->buffer.start = percent_buf;
                percent->text.p = percent_buf;
                percent->text.end = NULL;
                percent->free_after_use = false;
                percent->buffer.end = percent_buf + STRING_BUFFER_LENGTH;

                /* construct percent and result strings */
                targ_string.append_to_str(percent, prefix_length, percent_length
                construct_string_from_pattern(pat_rule, percent, &string);

                /* check for match */
                if (targ_string.equal(string.buffer.start, 0)) {
                        match_found = true;
                } else {
                        percent_length--;
                }
        }

        /* result */
        return match_found;
}


/*
 *      create_target_group_and_dependencies_list
 *           (target, pat_rule, percent)
 *
 *      constructs dependency list and a target group from pattern.
 *
 *      If we have the lines
 *              %/%.a + %/%.b + C%/CC%.c: yyy %.d bb%/BB%.e
 *                      commands
 *
 *      and we have matched the pattern xx/xx.a with %/%.a, then we
 *      construct a target group that looks like this:
 *              xx/xx.a + xx/xx.b + Cxx/CCxx.c: dependencies
 *
 *      and construct dependency list that looks like this:
 *      yyy xx.d bbxx/BBxx.e + already existed dependencies
 *
 *      Return value:
 *                              none
 *
 *      Parameters:
 *              target          The target we are building, in the previous
 *                              example, this is xx/xx.a
 *              pat_rule        the % pattern that matched "target", here %/%.a
 *              percent         string containing matched % part. In the example
 *
 *      Global variables used:
 *              empty_name
 */
```

```
1316 static void
1317 create_target_group_and_dependencies_list(Name target, Percent pat_rule, String
1318         String_rec      string;
1319         wchar_t         string_buf[STRING_BUFFER_LENGTH];
1320         Percent         pat_depe;
1321         Name            depe;
1322         Property        line = maybe_append_prop(target, line_prop);
1323         Chain           new_target_group = NULL;
1324         Chain           *new_target_group_tail = &new_target_group;
1325         Chain           group_member;
1326 
1327         /* create and append dependencies from rule */
1328         for (pat_depe = pat_rule->dependencies; pat_depe != NULL; pat_depe = pat
1329                 if (pat_depe->name->percent) {
1330                         INIT_STRING_FROM_STACK(string, string_buf);
1331                         construct_string_from_pattern(pat_depe, percent, &string
1332                         depe = GETNAME(string.buffer.start, FIND_LENGTH);
1333                         if (depe != empty_name) {
1334                                 enter_dependency(line, depe, false);
1335                         }
1336                 } else {
1337                         depe = pat_depe->name;
1338                         if(depe->dollar) {
1339                                 INIT_STRING_FROM_STACK(string, string_buf);
1340                                 expand_value(depe, &string, false);
1341                                 depe = GETNAME(string.buffer.start, FIND_LENGTH)
1342                         }
1343                         enter_dependency(line, depe, false);
1344                 }
1345         }
1346 
1347         /* if matched pattern is a group member, create new target group */
1348         for (group_member = pat_rule->target_group; group_member != NULL; group_
1349                 Name new_target = group_member->name;
1350                 if (group_member->name->percent) {
1351                         INIT_STRING_FROM_STACK(string, string_buf);
1352                         construct_string_from_pattern(group_member->percent_memb
1353                         new_target = GETNAME(string.buffer.start, FIND_LENGTH);
1354                         if (new_target == empty_name) {
1355                                 continue;
1356                         }
1357                 }
1359                 /* check for duplicates */
1360                 Chain   tgm;
1361                 for (tgm = new_target_group; tgm != NULL; tgm = tgm->next) {
1362                         if (new_target == tgm->name) {
1363                                 break;
1364                         }
1365                 }
1366                 if (tgm != NULL) {
1367                         continue;
1368                 }
1369 
1370                 /* insert it into the targets list */
1371                 (*new_target_group_tail) = ALLOC(Chain);
1372                 (*new_target_group_tail)->name = new_target;
1373                 (*new_target_group_tail)->next = NULL;
1374                 new_target_group_tail = &(*new_target_group_tail)->next;
1375         }
1376 
1377         /* now we gathered all dependencies and created target group */
1378         line->body.line.target_group = new_target_group;
1380         /* update properties for group members */
1381         for (group_member = new_target_group; group_member != NULL; group_member
```



```
1382                 if (group_member->name != target) {
1383                         group_member->name->prop = target->prop;
1384                         group_member->name->conditional_cnt = target->conditiona
1385                 }
1386         }
1387 }
1389 /*
1390  *      construct_string_from_pattern
1391  *              (pat_rule, percent, result)
1392  *
1393  *      after pattern matched a target this routine is called to construct targe
1394  *      strings from this matched pattern rule and a string (percent) with subst
1395  *
1396  *      Return value:
1397  *                              none
1398  *
1399  *      Parameters:
1400  *              pat_rule        matched pattern rule
1401  *              percent         string containing matched % sign part.
1402  *              result          holds the result of string construction.
1403  *
1404  */
1405 static void
1406 construct_string_from_pattern(Percent pat_rule, String percent, String result) {
1407         for (int i = 0; i < pat_rule->patterns_total; i++) {
1408                 if (pat_rule->patterns[i]->dollar) {
1409                         expand_value(pat_rule->patterns[i],
1410                                      result,
1411                                      false);
1412 
1413                 } else {
1414                         append_string(pat_rule->patterns[i]->string_mb,
1415                                       result,
1416                                       pat_rule->patterns[i]->hash.length);
1417                 }
1418 
1419                 if (i < pat_rule->patterns_total - 1) {
1420                         append_string(percent->buffer.start,
1421                                       result,
1422                                       percent->text.p - percent->buffer.start);
1423                 }
1424         }
1425 
1426         if ((result->buffer.start[0] == (int) period_char) &&
1427             (result->buffer.start[1] == (int) slash_char)) {
1428                 result->buffer.start += 2;
1429         }
1430 }
1432 /*
1433  *      dependency_exists(target, line)
1434  *
1435  *      Returns true if the target exists in the
1436  *      dependency list of the line.
1437  *
1438  *      Return value:
1439  *                              True if target is on dependency list
1440  *
1441  *      Parameters:
1442  *              target          Target we scan for
1443  *              line            We get the dependency list from here
1444  *
1445  *      Global variables used:
1446  */
1447 static Boolean
```

```
1448 dependency_exists(Name target, Property line)
1449 {
1450         Dependency      dp;

1452         if (line == NULL) {
1453                 return false;
1454         }
1455         for (dp = line->body.line.dependencies; dp != NULL; dp = dp->next) {
1456                 if (dp->name == target) {
1457                         return true;
1458                 }
1459         }
1460         return false;
1461 }

1463 void
1464 add_target_to_chain(Name target, Chain * query)
1465 {
1466         if (target->is_member && (get_prop(target->prop, member_prop) != NULL))
1467                 target = get_prop(target->prop, member_prop)->body.member.member
1468         }
1469         Chain   *query_tail;
1470         for (query_tail = query; *query_tail != NULL; query_tail = &(*query_tail
1471                 if ((*query_tail)->name == target) {
1472                         return;
1473                 }
1474         }
1475         *query_tail = ALLOC(Chain);
1476         (*query_tail)->name = target;
1477         (*query_tail)->next = NULL;
1478 }

1480 #endif /* ! codereview */
```

```
**********************************************************
    5112 Wed May 20 11:04:08 2015
new/usr/src/cmd/make/bin/make/common/macro.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)macro.cc 1.28 06/12/12
 */

#pragma ident   "@(#)macro.cc   1.28    06/12/12"

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

/*
 * Included files
 */
#ifdef DISTRIBUTED
#include <avo/strings.h>        /* AVO_STRDUP() */
#include <dm/Avo_DoJobMsg.h>
#endif
#include <mk/defs.h>
#include <mksh/macro.h>         /* getvar(), expand_value() */
#include <mksh/misc.h>          /* getmem() */

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */

/*
 * File table of contents
```



```
 */

void
setvar_append(register Name name, register Name value)
{
        register Property       macro_apx = get_prop(name->prop, macro_append_pr
        register Property       macro = get_prop(name->prop, macro_prop);
        int                     length;
        String_rec              destination;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        register Chain          chain;
        Name                    val = NULL;

        if(macro_apx == NULL) {
                macro_apx = append_prop(name, macro_append_prop);
                if(macro != NULL) {
                        macro_apx->body.macro_appendix.value = macro->body.macro
                }
        }

        val = macro_apx->body.macro_appendix.value_to_append;

        INIT_STRING_FROM_STACK(destination, buffer);
        buffer[0] = 0;
        if (val != NULL) {
                APPEND_NAME(val,
                                &destination,
                                (int) val->hash.length);
                if (value != NULL) {
                        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);
        if (destination.free_after_use) {
                retmem(destination.buffer.start);
        }
        macro_apx->body.macro_appendix.value_to_append = value;

        SETVAR(name, empty_name, true);
}

/*
 *      setvar_envvar()
 *
 *      This function scans the list of environment variables that have
 *      dynamic values and sets them.
 *
 *      Parameters:
 *
 *      Global variables used:
 *              envvar          A list of environment vars with $ in value
 */
void
#ifdef DISTRIBUTED
setvar_envvar(Avo_DoJobMsg *dmake_job_msg)
#else
setvar_envvar(void)
#endif
{
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
```

```
128         int                     length;
129 #ifdef DISTRIBUTED
130         Property                macro;
131 #endif
132         register        char    *mbs, *tmp_mbs_buffer = NULL;
133         register        char    *env, *tmp_mbs_buffer2 = NULL;
134         Envvar                  p;
135         String_rec              value;

137         for (p = envvar; p != NULL; p = p->next) {
138                 if (p->already_put
139 #ifdef DISTRIBUTED
140                     && !dmake_job_msg
141 #endif
142                     ) {
143                         continue;
144                 }
145                 INIT_STRING_FROM_STACK(value, buffer);
146                 expand_value(p->value, &value, false);
147                 if ((length = wslen(value.buffer.start)) >= MAXPATHLEN) {
148                         mbs = tmp_mbs_buffer = getmem((length + 1) * MB_LEN_MAX)
149                         (void) wcstombs(mbs,
150                                         value.buffer.start,
151                                         (length + 1) * MB_LEN_MAX);
152                 } else {
153                         mbs = mbs_buffer;
154                         WCSTOMBS(mbs, value.buffer.start);
155                 }
156                 length = 2 + strlen(p->name->string_mb) + strlen(mbs);
157                 if (!p->already_put || length > (MAXPATHLEN * MB_LEN_MAX)) {
158                         env = tmp_mbs_buffer2 = getmem(length);
159                 } else {
160                         env = mbs_buffer2;
161                 }
162                 (void) sprintf(env,
163                                 "%s=%s",
164                                 p->name->string_mb,
165                                 mbs);
166                 if (!p->already_put) {
167                         (void) putenv(env);
168                         p->already_put = true;
169                         if (p->env_string) {
170                                 retmem_mb(p->env_string);
171                         }
172                         p->env_string = env;
173                         tmp_mbs_buffer2 = NULL; // We should not return this mem
174                 }
175 #ifdef DISTRIBUTED
176                 if (dmake_job_msg) {
177                         dmake_job_msg->appendVar(env);
178                 }
179 #endif
180                 if (tmp_mbs_buffer2) {
181                         retmem_mb(tmp_mbs_buffer2);
182                         tmp_mbs_buffer2 = NULL;
183                 }
184                 if (tmp_mbs_buffer) {
185                         retmem_mb(tmp_mbs_buffer);
186                         tmp_mbs_buffer = NULL;
187                 }
188         }
189 #ifdef DISTRIBUTED
190         /* Append SUNPRO_DEPENDENCIES to the dmake_job_msg. */
191         if (keep_state && dmake_job_msg) {
192                 macro = get_prop(sunpro_dependencies->prop, macro_prop);
193                 length = 2 +
```



```
194                         strlen(sunpro_dependencies->string_mb) +
195                         strlen(macro->body.macro.value->string_mb);
196                 if (length > (MAXPATHLEN * MB_LEN_MAX)) {
197                         env = tmp_mbs_buffer2 = getmem(length);
198                 } else {
199                         env = mbs_buffer2;
200                 }
201                 (void) sprintf(env,
202                                 "%s=%s",
203                                 sunpro_dependencies->string_mb,
204                                 macro->body.macro.value->string_mb);
205                 dmake_job_msg->appendVar(env);
206                 if (tmp_mbs_buffer2) {
207                         retmem_mb(tmp_mbs_buffer2);
208                         tmp_mbs_buffer2 = NULL;
209                 }
210         }
211 #endif
212 }


215 #endif /* ! codereview */
```

```
**********************************************************
  103224 Wed May 20 11:04:09 2015
new/usr/src/cmd/make/bin/make/common/main.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)main.cc 1.158 06/12/12
 */

#pragma ident   "@(#)main.cc    1.158   06/12/12"

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

/*
 * Included files
 */
#if defined(TEAMWARE_MAKE_CMN)
#       include <avo/intl.h>
#       include <avo/libcli.h>          /* libcli_init() */
#       include <avo/cli_license.h>     /* avo_cli_get_license() */
#       include <avo/find_dir.h>        /* avo_find_run_dir() */
#       include <avo/version_string.h>
#       include <avo/util.h>            /* avo_init() */
#ifdef USE_DMS_CCR
#       include <avo/usage_tracking.h>
#else
#       include <avo/cleanup.h>
#endif
#endif

#if defined(TEAMWARE_MAKE_CMN)
/* This is for dmake only (not for Solaris make).
 * Include code to check updates (dmake patches)
 */
#ifdef _CHECK_UPDATE_H
#include <libAU.h>
#endif
#endif
```



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

#ifdef DISTRIBUTED
#       include <dm/Avo_AcknowledgeMsg.h>
#       include <rw/xdrstrea.h>
#       include <dmrc/dmrc.h> /* dmakerc file processing */
#endif

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

#if defined(TEAMWARE_MAKE_CMN)
#ifdef USE_DMS_CCR
#       include <pubdmsi18n/pubdmsi18n.h>       /* libpubdmsi18n_init() */
#endif
#endif

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

#ifdef SUN5_0
#       include <sys/systeminfo.h>      /* sysinfo() */
#endif

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

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

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

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

#if defined(linux)
#include <ctype.h>
#endif

/*
 * Defined macros
 */
#define LD_SUPPORT_ENV_VAR      NOCATGETS("SGS_SUPPORT")
#define LD_SUPPORT_MAKE_LIB     NOCATGETS("libmakestate.so.1")

/*
 * typedefs & structs
 */

/*
```

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

#ifdef DMAKE_STATISTICS
static  Boolean         getname_stat = false;
#endif

#if defined(TEAMWARE_MAKE_CMN)
        static  time_t          start_time;
        static  int             g_argc;
        static  char            **g_argv;
#ifdef USE_DMS_CCR
        static  Avo_usage_tracking *usageTracking = NULL;
#else
        static  Avo_cleanup     *cleanup = NULL;
#endif
#endif

/*
 * File table of contents
 */
#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
        extern "C" void         cleanup_after_exit(void);
#else
        extern  void            cleanup_after_exit(int, ...);
#endif

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

extern  Name            normalize_name(register wchar_t *name_string, register i

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

static  void            append_makeflags_string(Name, String);
static  void            doalarm(int);
static  void            enter_argv_values(int , char **, ASCII_Dyn_Array *);
static  void            make_targets(int, char **, Boolean);
static  int             parse_command_option(char);
static  void            read_command_options(int, char **);
```



```
static  void            read_environment(Boolean);
static  void            read_files_and_state(int, char **);
static  Boolean         read_makefile(Name, Boolean, Boolean, Boolean);
static  void            report_recursion(Name);
static  void            set_sgs_support(void);
static  void            setup_for_projectdir(void);
static  void            setup_makeflags_argv(void);
static  void            report_dir_enter_leave(Boolean entering);

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

#ifdef DISTRIBUTED
        extern  int             dmake_ofd;
        extern  FILE*           dmake_ofp;
        extern  int             rxmPid;
        extern  XDR             xdrs_out;
#endif
#ifdef TEAMWARE_MAKE_CMN
        extern  char            verstring[];
#endif

jmp_buf jmpbuffer;
#if !defined(linux)
nl_catd catd;
#endif

/*
 *      main(argc, argv)
 *
 *      Parameters:
 *              argc                    You know what this is
 *              argv                    You know what this is
 *
 *      Static variables used:
 *              list_all_targets        make -T seen
 *              trace_status            make -p seen
 *
 *      Global variables used:
 *              debug_level             Should we trace make actions?
 *              keep_state              Set if .KEEP_STATE seen
 *              makeflags               The Name "MAKEFLAGS", used to get macro
 *              remote_command_name     Name of remote invocation cmd ("on")
 *              running_list            List of parallel running processes
 *              stdout_stderr_same      true if stdout and stderr are the same
 *              auto_dependencies       The Name "SUNPRO_DEPENDENCIES"
 *              temp_file_directory     Set to the dir where we create tmp file
 *              trace_reader            Set to reflect tracing status
 *              working_on_targets      Set when building user targets
 */
int
main(int argc, char *argv[])
{
        /*
         * cp is a -> to the value of the MAKEFLAGS env var,
         * which has to be regular chars.
         */
        register char           *cp;
        char                    make_state_dir[MAXPATHLEN];
        Boolean                 parallel_flag = false;
        char                    *prognameptr;
        char                    *slash_ptr;
        mode_t                  um;
        int                     i;
#ifdef TEAMWARE_MAKE_CMN
        struct itimerval        value;
        char                    def_dmakerc_path[MAXPATHLEN];
```

```
260         Name                    dmake_name, dmake_name2;
261         Name                    dmake_value, dmake_value2;
262         Property                prop, prop2;
263         struct stat             statbuf;
264         int                     statval;
265 #endif

267 #ifndef PARALLEL
268         struct stat             out_stat, err_stat;
269 #endif
270         hostid = gethostid();
271 #ifdef TEAMWARE_MAKE_CMN
272         avo_get_user(NULL, NULL); // Initialize user name
273 #endif
274         bsd_signals();

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

278 #if defined (HP_UX) || defined(linux)
279        /* HP-UX users typically will not have NLSPATH set, and this binary
280         * requires that it be set.  On HP-UX 9.0x, /usr/lib/nls/%L/%N.cat is
281         * the path to set it to.
282         */

284        if (getenv(NOCATGETS("NLSPATH")) == NULL) {
285          putenv(NOCATGETS("NLSPATH=/usr/lib/nls/%L/%N.cat"));
286         }
287 #endif

289 #ifdef DMAKE_STATISTICS
290         if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
291                 getname_stat = true;
292         }
293 #endif



296         /*
297          * avo_init() sets the umask to 0.  Save it here and restore
298          * it after the avo_init() call.
299          */
300 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
301         um = umask(0);
302         avo_init(argv[0]);
303         umask(um);

305 #ifdef USE_DMS_CCR
306         usageTracking = new Avo_usage_tracking(NOCATGETS("dmake"), argc, argv);
307 #else
308         cleanup = new Avo_cleanup(NOCATGETS("dmake"), argc, argv);
309 #endif
310 #endif

312 #if defined(TEAMWARE_MAKE_CMN)
313         catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
314         libcli_init();

316 #ifdef _CHECK_UPDATE_H
317         /* This is for dmake only (not for Solaris make).
318          * Check (in background) if there is an update (dmake patch)
319          * and inform user
320          */
321         {
322                 Avo_err         *err;
323                 char            *dir;
324                 err = avo_find_run_dir(&dir);
325                 if (AVO_OK == err) {
```



```
326                         AU_check_update_service(NOCATGETS("Dmake"), dir);
327                 }
328         }
329 #endif /* _CHECK_UPDATE_H */
330 #endif

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


335 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
336 /*
337  * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
338  * from avo_util library.
339  */
340         libmksdmsi18n_init();
341 #ifdef USE_DMS_CCR
342         libpubdmsi18n_init();
343 #endif
344 #endif


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

351 #ifdef TEAMWARE_MAKE_CMN
352         g_argc = argc;
353         g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
354         for (i = 0; i < argc; i++) {
355                 g_argv[i] = argv[i];
356         }
357         g_argv[i] = NULL;
358 #endif /* TEAMWARE_MAKE_CMN */

360         /*
361          * Set argv_zero_string to some form of argv[0] for
362          * recursive MAKE builds.
363          */

365         if (*argv[0] == (int) slash_char) {
366                 /* argv[0] starts with a slash */
367                 argv_zero_string = strdup(argv[0]);
368         } else if (strchr(argv[0], (int) slash_char) == NULL) {
369                 /* argv[0] contains no slashes */
370                 argv_zero_string = strdup(argv[0]);
371         } else {
372                 /*
373                  * argv[0] contains at least one slash,
374                  * but doesn't start with a slash
375                  */
376                 char    *tmp_current_path;
377                 char    *tmp_string;

379                 tmp_current_path = get_current_path();
380                 tmp_string = getmem(strlen(tmp_current_path) + 1 +
381                                     strlen(argv[0]) + 1);
382                 (void) sprintf(tmp_string,
383                                "%s/%s",
384                                tmp_current_path,
385                                argv[0]);
386                 argv_zero_string = strdup(tmp_string);
387                 retmem_mb(tmp_string);
388         }

390         /*
391          * The following flags are reset if we don't have the
```

```
392          * (.nse_depinfo or .make.state) files locked and only set
393          * AFTER the file has been locked. This ensures that if the user
394          * interrupts the program while file_lock() is waiting to lock
395          * the file, the interrupt handler doesn't remove a lock
396          * that doesn't belong to us.
397          */
398         make_state_lockfile = NULL;
399         make_state_locked = false;

401 #ifdef NSE
402         nse_depinfo_lockfile[0] = '\0';
403         nse_depinfo_locked = false;
404 #endif

406         /*
407          * look for last slash char in the path to look at the binary
408          * name. This is to resolve the hard link and invoke make
409          * in svr4 mode.
410          */

412         /* Sun OS make standart */
413         svr4 = false;
414         posix = false;
415         if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
416                 svr4 = false;
417                 posix = true;
418         } else {
419                 prognameptr = strrchr(argv[0], '/');
420                 if(prognameptr) {
421                         prognameptr++;
422                 } else {
423                         prognameptr = argv[0];
424                 }
425                 if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
426                         svr4 = true;
427                         posix = false;
428                 }
429         }
430 #if !defined(HP_UX) && !defined(linux)
431         if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
432            svr4 = true;
433            posix = false;
434         }
435 #endif

437         /*
438          * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
439          */
440         char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
441         if (dmake_compat_mode_var != NULL) {
442                 if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
443                         gnu_style = true;
444                 }
445                 //svr4 = false;
446                 //posix = false;
447         }

449         /*
450          * Temporary directory set up.
451          */
452         char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
453         if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
454                 strcpy(mbs_buffer, tmpdir_var);
455                 for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
456                         *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
457                         *tmpdir_var = '\0');
```



```
458                 if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
459                         sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
460                                 mbs_buffer, getpid());
461                         int fd = mkstemp(mbs_buffer2);
462                         if (fd >= 0) {
463                                 close(fd);
464                                 unlink(mbs_buffer2);
465                                 tmpdir = strdup(mbs_buffer);
466                         }
467                 }
468         }

470 #ifndef PARALLEL
471         /* find out if stdout and stderr point to the same place */
472         if (fstat(1, &out_stat) < 0) {
473                 fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
474         }
475         if (fstat(2, &err_stat) < 0) {
476                 fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
477         }
478         if ((out_stat.st_dev == err_stat.st_dev) &&
479             (out_stat.st_ino == err_stat.st_ino)) {
480                 stdout_stderr_same = true;
481         } else {
482                 stdout_stderr_same = false;
483         }
484 #else
485         stdout_stderr_same = false;
486 #endif
487         /* Make the vroot package scan the path using shell semantics */
488         set_path_style(0);

490         setup_char_semantics();

492         setup_for_projectdir();

494         /*
495          * If running with .KEEP_STATE, curdir will be set with
496          * the connected directory.
497          */
498 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
499         (void) atexit(cleanup_after_exit);
500 #else
501         (void) on_exit(cleanup_after_exit, (char *) NULL);
502 #endif

504         load_cached_names();

506 /*
507  *      Set command line flags
508  */
509         setup_makeflags_argv();
510         read_command_options(mf_argc, mf_argv);
511         read_command_options(argc, argv);
512         if (debug_level > 0) {
513                 cp = getenv(makeflags->string_mb);
514                 (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
515         }

517         setup_interrupt(handle_interrupt);

519         read_files_and_state(argc, argv);

521 #ifdef TEAMWARE_MAKE_CMN
522         /*
523          * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
```

```
524          */
525         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
526         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
527         prop2 = get_prop(dmake_name2->prop, macro_prop);
528         if (prop2 == NULL) {
529                 /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
530                 output_mode = txt1_mode;
531         } else {
532                 dmake_value2 = prop2->body.macro.value;
533                 if ((dmake_value2 == NULL) ||
534                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
535                         output_mode = txt1_mode;
536                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
537                         output_mode = txt2_mode;
538                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))
539                         output_mode = html1_mode;
540                 } else {
541                         warning(catgets(catd, 1, 352, "Unsupported value ‘%s’ fo
542                               dmake_value2->string_mb);
543                 }
544         }
545         /*
546          * Find the dmake_mode: distributed, parallel, or serial.
547          */
548     if ((!pmake_cap_r_specified) &&
549         (!pmake_machinesfile_specified)) {
550         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
551         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
552         prop2 = get_prop(dmake_name2->prop, macro_prop);
553         if (prop2 == NULL) {
554                 /* DMAKE_MODE not defined, default to distributed mode */
555                 dmake_mode_type = distributed_mode;
556                 no_parallel = false;
557         } else {
558                 dmake_value2 = prop2->body.macro.value;
559                 if ((dmake_value2 == NULL) ||
560                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed"))
561                         dmake_mode_type = distributed_mode;
562                         no_parallel = false;
563                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel
564                         dmake_mode_type = parallel_mode;
565                         no_parallel = false;
566 #ifdef SGE_SUPPORT
567                         grid = false;
568                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("grid")))
569                         dmake_mode_type = parallel_mode;
570                         no_parallel = false;
571                         grid = true;
572 #endif
573                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
574                         dmake_mode_type = serial_mode;
575                         no_parallel = true;
576                 } else {
577                         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
578                 }
579         }

581         if ((!list_all_targets) &&
582             (report_dependencies_level == 0)) {
583                 /*
584                  * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
585                  * They could be defined in the env, in the makefile, or on the
586                  * command line.
587                  * If neither is defined, and $(HOME)/.dmakerc does not exists,
588                  * then print a message, and default to parallel mode.
589                  */
```



```
590 #ifdef DISTRIBUTED
591                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
592                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
593                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
594                 dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
595                 if ((((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
596                      ((dmake_value = prop->body.macro.value) == NULL)) &&
597                     (((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL)
598                      ((dmake_value2 = prop2->body.macro.value) == NULL))) {
599                         Boolean empty_dmakerc = true;
600                         char *homedir = getenv(NOCATGETS("HOME"));
601                         if ((homedir != NULL) && (strlen(homedir) < (sizeof(def_
602                                 sprintf(def_dmakerc_path, NOCATGETS("%s/.dmakerc
603                                 if ((((statval = stat(def_dmakerc_path, &statbuf
604                                         ((statval == 0) && (statbuf.st_size == 0
605                                 } else {
606                                         Avo_dmakerc     *rcfile = new Avo_dmaker
607                                         Avo_err         *err = rcfile->read(def_
608                                         if (err) {
609                                                 fatal(err->str);
610                                         }
611                                         empty_dmakerc = rcfile->was_empty();
612                                         delete rcfile;
613                                 }
614                         }
615                         if (empty_dmakerc) {
616                                 if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
617                                         putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE
618                                         (void) fprintf(stdout, catgets(catd, 1,
619                                         (void) fprintf(stdout, catgets(catd, 1,
620                                 }
621                                 dmake_mode_type = parallel_mode;
622                                 no_parallel = false;
623                         }
624                 }
625 #else
626                 if(dmake_mode_type == distributed_mode) {
627                         (void) fprintf(stdout, NOCATGETS("dmake: Distributed mod
628                         (void) fprintf(stdout, NOCATGETS("       Defaulting to p
629                         dmake_mode_type = parallel_mode;
630                         no_parallel = false;
631                 }
632 #endif  /* DISTRIBUTED */
633         }
634     }
635 #endif

637 #ifdef TEAMWARE_MAKE_CMN
638         parallel_flag = true;
639         /* XXX - This is a major hack for DMake/Licensing. */
640         if (getenv(NOCATGETS("DMAKE_CHILD")) == NULL) {
641                 if (!avo_cli_search_license(argv[0], dmake_exit_callback, TRUE,
642                         /*
643                          * If the user can not get a TeamWare license,
644                          * default to serial mode.
645                          */
646                         dmake_mode_type = serial_mode;
647                         no_parallel = true;
648                 } else {
649                         putenv(NOCATGETS("DMAKE_CHILD=TRUE"));
650                 }
651                 start_time = time(NULL);
652                 /*
653                  * XXX - Hack to disable SIGALRM’s from licensing library’s
654                  *       setitimer().
655                  */
```

```
656                 value.it_interval.tv_sec = 0;
657                 value.it_interval.tv_usec = 0;
658                 value.it_value.tv_sec = 0;
659                 value.it_value.tv_usec = 0;
660                 (void) setitimer(ITIMER_REAL, &value, NULL);
661         }

663 //
664 // If dmake is running with -t option, set dmake_mode_type to serial.
665 // This is done because doname() calls touch_command() that runs serially.
666 // If we do not do that, maketool will have problems.
667 //
668         if(touch) {
669                 dmake_mode_type = serial_mode;
670                 no_parallel = true;
671         }
672 #else
673         parallel_flag = false;
674 #endif

676 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
677         /*
678          * Check whether stdout and stderr are physically same.
679          * This is in order to decide whether we need to redirect
680          * stderr separately from stdout.
681          * This check is performed only if __DMAKE_SEPARATE_STDERR
682          * is not set. This variable may be used in order to preserve
683          * the 'old' behaviour.
684          */
685         out_err_same = true;
686         char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
687         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
688                 struct stat stdout_stat;
689                 struct stat stderr_stat;
690                 if( (fstat(1, &stdout_stat) == 0)
691                  && (fstat(2, &stderr_stat) == 0) )
692                 {
693                         if( (stdout_stat.st_dev != stderr_stat.st_dev)
694                          || (stdout_stat.st_ino != stderr_stat.st_ino) )
695                         {
696                                 out_err_same = false;
697                         }
698                 }
699         }
700 #endif

702 #ifdef DISTRIBUTED
703         /*
704          * At this point, DMake should startup an rxm with any and all
705          * DMake command line options. Rxm will, among other things,
706          * read the rc file.
707          */
708         if ((!list_all_targets) &&
709             (report_dependencies_level == 0) &&
710             (dmake_mode_type == distributed_mode)) {
711                 startup_rxm();
712         }
713 #endif
714
715 /*
716  *      Enable interrupt handler for alarms
717  */
718         (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

720 /*
721  *      Check if make should report
```



```
722  */
723         if (getenv(sunpro_dependencies->string_mb) != NULL) {
724                 FILE    *report_file;

726                 report_dependency("");
727                 report_file = get_report_file();
728                 if ((report_file != NULL) && (report_file != (FILE*)-1)) {
729                         (void) fprintf(report_file, "\n");
730                 }
731         }

733 /*
734  *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly
735  *      and NSE_DEP.
736  */
737         if (keep_state) {
738                 maybe_append_prop(sunpro_dependencies, macro_prop)->
739                   body.macro.exported = true;
740 #ifdef NSE
741                 (void) setenv(NOCATGETS("NSE_DEP"), get_current_path());
742 #endif
743         } else {
744                 maybe_append_prop(sunpro_dependencies, macro_prop)->
745                   body.macro.exported = false;
746         }

748         working_on_targets = true;
749         if (trace_status) {
750                 dump_make_state();
751 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
752                 fclose(stdout);
753                 fclose(stderr);
754                 exit_status = 0;
755 #endif
756                 exit(0);
757         }
758         if (list_all_targets) {
759                 dump_target_list();
760 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
761                 fclose(stdout);
762                 fclose(stderr);
763                 exit_status = 0;
764 #endif
765                 exit(0);
766         }
767         trace_reader = false;

769         /*
770          * Set temp_file_directory to the directory the .make.state
771          * file is written to.
772          */
773         if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
774                 temp_file_directory = strdup(get_current_path());
775         } else {
776                 *slash_ptr = (int) nul_char;
777                 (void) strcpy(make_state_dir, make_state->string_mb);
778                 *slash_ptr = (int) slash_char;
779                    /* when there is only one slash and it's the first
780                    ** character, make_state_dir should point to '/'.
781                    */
782                 if(make_state_dir[0] == '\0') {
783                    make_state_dir[0] = '/';
784                    make_state_dir[1] = '\0';
785                 }
786                 if (make_state_dir[0] == (int) slash_char) {
787                         temp_file_directory = strdup(make_state_dir);
```

```
788                 } else {
789                         char    tmp_current_path2[MAXPATHLEN];
790 
791                         (void) sprintf(tmp_current_path2,
792                                         "%s/%s",
793                                         get_current_path(),
794                                         make_state_dir);
795                         temp_file_directory = strdup(tmp_current_path2);
796                 }
797         }

799 #ifdef DISTRIBUTED
800         building_serial = false;
801 #endif

803         report_dir_enter_leave(true);

805         make_targets(argc, argv, parallel_flag);

807         report_dir_enter_leave(false);

809 #ifdef NSE
810         exit(nse_exit_status());
811 #else
812         if (build_failed_ever_seen) {
813 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
814                 if (posix) {
815                         exit_status = 1;
816                 }
817 #endif
818                 exit(1);
819         }
820 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
821         exit_status = 0;
822 #endif
823         exit(0);
824 #endif
825         /* NOTREACHED */
826 }

828 /*
829  *      cleanup_after_exit()
830  *
831  *      Called from exit(), performs cleanup actions.
832  *
833  *      Parameters:
834  *              status          The argument exit() was called with
835  *              arg             Address of an argument vector to
836  *                              cleanup_after_exit()
837  *
838  *      Global variables used:
839  *              command_changed Set if we think .make.state should be rewritten
840  *              current_line    Is set we set commands_changed
841  *              do_not_exec_rule
842  *                              True if -n flag on
843  *              done            The Name ".DONE", rule we run
844  *              keep_state      Set if .KEEP_STATE seen
845  *              parallel        True if building in parallel
846  *              quest           If -q is on we do not run .DONE
847  *              report_dependencies
848  *                              True if -P flag on
849  *              running_list    List of parallel running processes
850  *              temp_file_name  The temp file is removed, if any
851  *              catd    the message catalog file
852  *              usage_tracking  Should have been constructed in main()
853  *                              should destroyed just before exiting
```



```
854  */
855 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
856 extern "C" void
857 cleanup_after_exit(void)
858 #else
859 void cleanup_after_exit(int status, ...)
860 #endif
861 {
862         Running         rp;
863 #ifdef NSE
864         char            push_cmd[NSE_TFS_PUSH_LEN + 3 +
865                                 (MAXPATHLEN * MB_LEN_MAX) + 12];
866         char            *active;
867 #endif

869 extern long     getname_bytes_count;
870 extern long     getname_names_count;
871 extern long     getname_struct_count;
872 extern long     freename_bytes_count;
873 extern long     freename_names_count;
874 extern long     freename_struct_count;
875 extern long     other_alloc;

877 extern long     env_alloc_num;
878 extern long     env_alloc_bytes;


881 #ifdef DMAKE_STATISTICS
882 if(getname_stat) {
883         printf(NOCATGETS(">>> Getname statistics:\n"));
884         printf(NOCATGETS("  Allocated:\n"));
885         printf(NOCATGETS("        Names: %ld\n"), getname_names_count);
886         printf(NOCATGETS("      Strings: %ld Kb (%ld bytes)\n"), getname_bytes_c
887         printf(NOCATGETS("      Structs: %ld Kb (%ld bytes)\n"), getname_struct_
888         printf(NOCATGETS("  Total bytes: %ld Kb (%ld bytes)\n"), getname_struct_

890         printf(NOCATGETS("\n  Unallocated: %ld\n"), freename_names_count);
891         printf(NOCATGETS("        Names: %ld\n"), freename_names_count);
892         printf(NOCATGETS("      Strings: %ld Kb (%ld bytes)\n"), freename_bytes_
893         printf(NOCATGETS("      Structs: %ld Kb (%ld bytes)\n"), freename_struct
894         printf(NOCATGETS("  Total bytes: %ld Kb (%ld bytes)\n"), freename_struct

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

898         printf(NOCATGETS("\n>>> Other:\n"));
899         printf(
900                 NOCATGETS("       Env (%ld): %ld Kb (%ld bytes)\n"),
901                 env_alloc_num,
902                 env_alloc_bytes/1000,
903                 env_alloc_bytes
904         );

906 }
907 #endif

909 /*
910 #ifdef DISTRIBUTED
911     if (get_parent() == TRUE) {
912 #endif
913  */

915         parallel = false;
916 #ifdef SUN5_0
917         /* If we used the SVR4_MAKE, don't build .DONE or .FAILED */
918         if (!getenv(USE_SVR4_MAKE)){
919 #endif
```

```
920             /* Build the target .DONE or .FAILED if we caught an error */
921             if (!quest && !list_all_targets) {
922                 Name            failed_name;

924                 MBSTOWCS(wcs_buffer, NOCATGETS(".FAILED"));
925                 failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
926 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
927                 if ((exit_status != 0) && (failed_name->prop != NULL)) {
928 #else
929                 if ((status != 0) && (failed_name->prop != NULL)) {
930 #endif
931 #ifdef TEAMWARE_MAKE_CMN
932                         /*
933                          * [tolik] switch DMake to serial mode
934                          */
935                         dmake_mode_type = serial_mode;
936                         no_parallel = true;
937 #endif
938                         (void) doname(failed_name, false, true);
939                 } else {
940                     if (!trace_status) {
941 #ifdef TEAMWARE_MAKE_CMN
942                         /*
943                          * Switch DMake to serial mode
944                          */
945                         dmake_mode_type = serial_mode;
946                         no_parallel = true;
947 #endif
948                         (void) doname(done, false, true);
949                     }
950                 }
951             }
952 #ifdef SUN5_0
953         }
954 #endif
955         /*
956          * Remove the temp file utilities report dependencies thru if it
957          * is still around
958          */
959         if (temp_file_name != NULL) {
960                 (void) unlink(temp_file_name->string_mb);
961         }
962         /*
963          * Do not save the current command in .make.state if make
964          * was interrupted.
965          */
966         if (current_line != NULL) {
967                 command_changed = true;
968                 current_line->body.line.command_used = NULL;
969         }
970         /*
971          * For each parallel build process running, remove the temp files
972          * and zap the command line so it won't be put in .make.state
973          */
974         for (rp = running_list; rp != NULL; rp = rp->next) {
975                 if (rp->temp_file != NULL) {
976                         (void) unlink(rp->temp_file->string_mb);
977                 }
978                 if (rp->stdout_file != NULL) {
979                         (void) unlink(rp->stdout_file);
980                         retmem_mb(rp->stdout_file);
981                         rp->stdout_file = NULL;
982                 }
983                 if (rp->stderr_file != NULL) {
984                         (void) unlink(rp->stderr_file);
985                         retmem_mb(rp->stderr_file);
```



```
986                         rp->stderr_file = NULL;
987                 }
988                 command_changed = true;
989 /*
990                 line = get_prop(rp->target->prop, line_prop);
991                 if (line != NULL) {
992                         line->body.line.command_used = NULL;
993                 }
994  */
995         }
996         /* Remove the statefile lock file if the file has been locked */
997         if ((make_state_lockfile != NULL) && (make_state_locked)) {
998                 (void) unlink(make_state_lockfile);
999                 make_state_lockfile = NULL;
1000                make_state_locked = false;
1001        }
1002        /* Write .make.state */
1003        write_state_file(1, (Boolean) 1);

1005 #ifdef TEAMWARE_MAKE_CMN
1006        // Deleting the usage tracking object sends the usage mail
1007 #ifdef USE_DMS_CCR
1008        //usageTracking->setExitStatus(exit_status, NULL);
1009        //delete usageTracking;
1010 #else
1011        cleanup->set_exit_status(exit_status);
1012        delete cleanup;
1013 #endif
1014 #endif

1016 #ifdef NSE
1017        /* If running inside an activated environment, push the */
1018        /* .nse_depinfo file (if written) */
1019        active = getenv(NSE_VARIANT_ENV);
1020        if (keep_state &&
1021            (active != NULL) &&
1022            !IS_EQUAL(active, NSE_RT_SOURCE_NAME) &&
1023            !do_not_exec_rule &&
1024            (report_dependencies_level == 0)) {
1025                (void) sprintf(push_cmd,
1026                               "%s %s/%s",
1027                               NSE_TFS_PUSH,
1028                               get_current_path(),
1029                               NSE_DEPINFO);
1030                (void) system(push_cmd);
1031        }
1032 #endif

1034 /*
1035 #ifdef DISTRIBUTED
1036     }
1037 #endif
1038  */

1040 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
1041        job_adjust_fini();
1042 #endif

1044 #ifdef TEAMWARE_MAKE_CMN
1045        catclose(catd);
1046 #endif
1047 #ifdef DISTRIBUTED
1048        if (rxmPid > 0) {
1049                // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
1050                Avo_AcknowledgeMsg acknowledgeMsg;
1051                RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;
```

```
1053                 int xdrResult = xdr(&xdrs_out, msg);

1055                 if (xdrResult) {
1056                         fflush(dmake_ofp);
1057                 } else {
1058 /*
1059                         fatal(catgets(catd, 1, 266, "couldn't tell rxm to exit")
1060  */
1061                         kill(rxmPid, SIGTERM);
1062                 }

1064                 waitpid(rxmPid, NULL, 0);
1065                 rxmPid = 0;
1066         }
1067 #endif
1068 }

1070 /*
1071  *      handle_interrupt()
1072  *
1073  *      This is where C-C traps are caught.
1074  *
1075  *      Parameters:
1076  *
1077  *      Global variables used (except DMake 1.0):
1078  *              current_target          Sometimes the current target is removed
1079  *              do_not_exec_rule        But not if -n is on
1080  *              quest                   or -q
1081  *              running_list            List of parallel running processes
1082  *              touch                   Current target is not removed if -t on
1083  */
1084 void
1085 handle_interrupt(int)
1086 {
1087         Property                member;
1088         Running                 rp;

1090         (void) fflush(stdout);
1091 #ifdef DISTRIBUTED
1092         if (rxmPid > 0) {
1093                 // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
1094                 Avo_AcknowledgeMsg acknowledgeMsg;
1095                 RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;

1097                 int xdrResult = xdr(&xdrs_out, msg);

1099                 if (xdrResult) {
1100                         fflush(dmake_ofp);
1101                 } else {
1102                         kill(rxmPid, SIGTERM);
1103                         rxmPid = 0;
1104                 }
1105         }
1106 #endif
1107         if (childPid > 0) {
1108                 kill(childPid, SIGTERM);
1109                 childPid = -1;
1110         }
1111         for (rp = running_list; rp != NULL; rp = rp->next) {
1112                 if (rp->state != build_running) {
1113                         continue;
1114                 }
1115                 if (rp->pid > 0) {
1116                         kill(rp->pid, SIGTERM);
1117                         rp->pid = -1;
```



```
1118                 }
1119         }
1120         if (getpid() == getpgrp()) {
1121                 bsd_signal(SIGTERM, SIG_IGN);
1122                 kill (-getpid(), SIGTERM);
1123         }
1124 #ifdef TEAMWARE_MAKE_CMN
1125         /* Clean up all parallel/distributed children already finished */
1126         finish_children(false);
1127 #endif

1129         /* Make sure the processes running under us terminate first */

1131 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
1132         while (wait((int *) NULL) != -1);
1133 #else
1134         while (wait((union wait*) NULL) != -1);
1135 #endif
1136         /* Delete the current targets unless they are precious */
1137         if ((current_target != NULL) &&
1138             current_target->is_member &&
1139             ((member = get_prop(current_target->prop, member_prop)) != NULL)) {
1140                 current_target = member->body.member.library;
1141         }
1142         if (!do_not_exec_rule &&
1143             !touch &&
1144             !quest &&
1145             (current_target != NULL) &&
1146             !(current_target->stat.is_precious || all_precious)) {

1148 /* BID_1030811 */
1149 /* azv 16 Oct 95 */
1150                 current_target->stat.time = file_no_time;

1152                 if (exists(current_target) != file_doesnt_exist) {
1153                         (void) fprintf(stderr,
1154                                        "\n*** %s ",
1155                                        current_target->string_mb);
1156                         if (current_target->stat.is_dir) {
1157                                 (void) fprintf(stderr,
1158                                                catgets(catd, 1, 168, "not remove
1159                                                current_target->string_mb);
1160                         } else if (unlink(current_target->string_mb) == 0) {
1161                                 (void) fprintf(stderr,
1162                                                catgets(catd, 1, 169, "removed.\n
1163                                                current_target->string_mb);
1164                         } else {
1165                                 (void) fprintf(stderr,
1166                                                catgets(catd, 1, 170, "could not
1167                                                current_target->string_mb,
1168                                                errmsg(errno));
1169                         }
1170                 }
1171         }
1172         for (rp = running_list; rp != NULL; rp = rp->next) {
1173                 if (rp->state != build_running) {
1174                         continue;
1175                 }
1176                 if (rp->target->is_member &&
1177                     ((member = get_prop(rp->target->prop, member_prop)) !=
1178                      NULL)) {
1179                         rp->target = member->body.member.library;
1180                 }
1181                 if (!do_not_exec_rule &&
1182                     !touch &&
1183                     !quest &&
```

```
1184                    !(rp->target->stat.is_precious || all_precious)) {

1186                        rp->target->stat.time = file_no_time;
1187                        if (exists(rp->target) != file_doesnt_exist) {
1188                                (void) fprintf(stderr,
1189                                               "\n*** %s ",
1190                                               rp->target->string_mb);
1191                                if (rp->target->stat.is_dir) {
1192                                        (void) fprintf(stderr,
1193                                                       catgets(catd, 1, 171, "no
1194                                                       rp->target->string_mb);
1195                                } else if (unlink(rp->target->string_mb) == 0) {
1196                                        (void) fprintf(stderr,
1197                                                       catgets(catd, 1, 172, "re
1198                                                       rp->target->string_mb);
1199                                } else {
1200                                        (void) fprintf(stderr,
1201                                                       catgets(catd, 1, 173, "co
1202                                                       rp->target->string_mb,
1203                                                       errmsg(errno));
1204                                }
1205                        }
1206                }
1207        }

1209 #ifdef SGE_SUPPORT
1210        /* Remove SGE script file */
1211        if (grid) {
1212                unlink(script_file);
1213        }
1214 #endif

1216        /* Have we locked .make.state or .nse_depinfo? */
1217        if ((make_state_lockfile != NULL) && (make_state_locked)) {
1218                unlink(make_state_lockfile);
1219                make_state_lockfile = NULL;
1220                make_state_locked = false;
1221        }
1222 #ifdef NSE
1223        if ((nse_depinfo_lockfile[0] != '\0') && (nse_depinfo_locked)) {
1224                unlink(nse_depinfo_lockfile);
1225                nse_depinfo_lockfile[0] = '\0';
1226                nse_depinfo_locked = false;
1227        }
1228 #endif
1229        /*
1230         * Re-read .make.state file (it might be changed by recursive make)
1231         */
1232        check_state(NULL);

1234        report_dir_enter_leave(false);

1236 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
1237        exit_status = 2;
1238 #endif
1239        exit(2);
1240 }

1242 /*
1243  *      doalarm(sig, ...)
1244  *
1245  *      Handle the alarm interrupt but do nothing.  Side effect is to
1246  *      cause return from wait3.
1247  *
1248  *      Parameters:
1249  *              sig
```



```
1250  *
1251  *      Global variables used:
1252  */
1253 /*ARGSUSED*/
1254 static void
1255 doalarm(int)
1256 {
1257        return;
1258 }


1261 /*
1262  *      read_command_options(argc, argv)
1263  *
1264  *      Scan the cmd line options and process the ones that start with "-"
1265  *
1266  *      Return value:
1267  *                              -M argument, if any
1268  *
1269  *      Parameters:
1270  *              argc            You know what this is
1271  *              argv            You know what this is
1272  *
1273  *      Global variables used:
1274  */
1275 static void
1276 read_command_options(register int argc, register char **argv)
1277 {
1278        register int            ch;
1279        int                     current_optind = 1;
1280        int                     last_optind_with_double_hyphen = 0;
1281        int                     last_optind;
1282        int                     last_current_optind;
1283        register int            i;
1284        register int            j;
1285        register int            k;
1286        register int            makefile_next = 0; /*
1287                                                    * flag to note options:
1288                                                    * -c, f, g, j, m, o
1289                                                    */
1290        const char              *tptr;
1291        const char              *CMD_OPTS;

1293        extern char             *optarg;
1294        extern int              optind, opterr, optopt;

1296 #define SUNPRO_CMD_OPTS "-~Bbc:Ddef:g:ij:K:kM:m:NnO:o:PpqRrSsTtuVvwx:"

1298 #ifdef TEAMWARE_MAKE_CMN
1299 #       define SVR4_CMD_OPTS   "-c:ef:g:ij:km:nO:o:pqrsTtVv"
1300 #else
1301 #       define SVR4_CMD_OPTS   "-ef:iknpqrstV"
1302 #endif

1304        /*
1305         * Added V in SVR4_CMD_OPTS also, which is going to be a hidden
1306         * option, just to make sure that the getopt doesn't fail when some
1307         * users leave their USE_SVR4_MAKE set and try to use the makefiles
1308         * that are designed to issue commands like $(MAKE) -V. Anyway it
1309         * sets the same flag but ensures that getopt doesn't fail.
1310         */

1312        opterr = 0;
1313        optind = 1;
1314        while (1) {
1315                last_optind=optind;                     /* Save optind and curre
```

```
1316                last_current_optind=current_optind;     /* in case we have to re
1317                if (svr4) {
1318                        CMD_OPTS=SVR4_CMD_OPTS;
1319                        ch = getopt(argc, argv, SVR4_CMD_OPTS);
1320                } else {
1321                        CMD_OPTS=SUNPRO_CMD_OPTS;
1322                        ch = getopt(argc, argv, SUNPRO_CMD_OPTS);
1323                }
1324                if (ch == EOF) {
1325                        if(optind < argc) {
1326                                /*
1327                                 * Fixing bug 4102537:
1328                                 *    Strange behaviour of command make using --
1329                                 * Not all argv have been processed
1330                                 * Skip non-flag argv and continue processing.
1331                                 */
1332                                optind++;
1333                                current_optind++;
1334                                continue;
1335                        } else {
1336                                break;
1337                        }

1339                }
1340                if (ch == '?') {
1341                        if (optopt == '-') {
1342                                /* Bug 5060758: getopt() changed behavior (s10_6
1343                                 * and now we have to deal with cases when optio
1344                                 * with double hyphen appear here, from -$(MAKEF
1345                                 */
1346                                i = current_optind;
1347                                if (argv[i][0] == '-') {
1348                                  if (argv[i][1] == '-') {
1349                                    if (argv[i][2] != '\0') {
1350                                      /* Check if this option is allowed */
1351                                      tptr = strchr(CMD_OPTS, argv[i][2]);
1352                                      if (tptr) {
1353                                        if (last_optind_with_double_hyphen != cu
1354                                          /* This is first time we are trying to
1355                                           * problem with this option. If we com
1356                                           * time, we will go to fatal error.
1357                                           */
1358                                          last_optind_with_double_hyphen = curre
1359
1360                                          /* Eliminate first hyphen character */
1361                                          for (j=0; argv[i][j] != '\0'; j++) {
1362                                            argv[i][j] = argv[i][j+1];
1363                                          }
1364
1365                                          /* Repeat the processing of this argum
1366                                          optind=last_optind;
1367                                          current_optind=last_current_optind;
1368                                          continue;
1369                                        }
1370                                      }
1371                                    }
1372                                  }
1373                                }
1374                        }
1375                }

1377                if (ch == '?') {
1378                        if (svr4) {
1379 #ifdef TEAMWARE_MAKE_CMN
1380                                fprintf(stderr,
1381                                        catgets(catd, 1, 267, "Usage : dmake [ -
```



```
1382                                fprintf(stderr,
1383                                        catgets(catd, 1, 268, "              [ -
1384                                fprintf(stderr,
1385                                        catgets(catd, 1, 269, "              [ -
1386 #else
1387                                fprintf(stderr,
1388                                        catgets(catd, 1, 270, "Usage : make [ -f
1389                                fprintf(stderr,
1390                                        catgets(catd, 1, 271, "             [ -s
1391 #endif
1392                                tptr = strchr(SVR4_CMD_OPTS, optopt);
1393                        } else {
1394 #ifdef TEAMWARE_MAKE_CMN
1395                                fprintf(stderr,
1396                                        catgets(catd, 1, 272, "Usage : dmake [ -
1397                                fprintf(stderr,
1398                                        catgets(catd, 1, 273, "              [ -
1399                                fprintf(stderr,
1400                                        catgets(catd, 1, 274, "              [ -
1401                                fprintf(stderr,
1402                                        catgets(catd, 1, 275, "              [ -
1403 #else
1404                                fprintf(stderr,
1405                                        catgets(catd, 1, 276, "Usage : make [ -f
1406                                fprintf(stderr,
1407                                        catgets(catd, 1, 277, "             [ -e
1408                                fprintf(stderr,
1409                                        catgets(catd, 1, 278, "             [ -u
1410 #endif
1411                                tptr = strchr(SUNPRO_CMD_OPTS, optopt);
1412                        }
1413                        if (!tptr) {
1414                                fatal(catgets(catd, 1, 279, "Unknown option '-%c
1415                        } else {
1416                                fatal(catgets(catd, 1, 280, "Missing argument af
1417                        }
1418                }

1420 #if defined(linux)
1421                if (ch == 1) {
1422                        if(optind < argc) {
1423                                //optind++;
1424                                //current_optind++;
1425                        makefile_next = 0;
1426                        current_optind = optind;
1427                                continue;
1428                        } else {
1429                                break;
1430                        }
1431                }
1432 #endif


1435                makefile_next |= parse_command_option(ch);
1436                /*
1437                 * If we're done processing all of the options of
1438                 * ONE argument string...
1439                 */
1440                if (current_optind < optind) {
1441                        i = current_optind;
1442                        k = 0;
1443                        /* If there's an argument for an option... */
1444                        if ((optind - current_optind) > 1) {
1445                                k = i + 1;
1446                        }
1447                        switch (makefile_next) {
```

```
1448                        case 0:
1449                                argv[i] = NULL;
1450                                /* This shouldn't happen */
1451                                if (k) {
1452                                        argv[k] = NULL;
1453                                }
1454                                break;
1455                        case 1: /* -f seen */
1456                                argv[i] = NOCATGETS("-f");
1457                                break;
1458                        case 2: /* -c seen */
1459                                argv[i] = NOCATGETS("-c");
1460 #ifndef TEAMWARE_MAKE_CMN
1461                                warning(catgets(catd, 1, 281, "Ignoring Distribu
1462 #endif
1463                                break;
1464                        case 4: /* -g seen */
1465                                argv[i] = NOCATGETS("-g");
1466 #ifndef TEAMWARE_MAKE_CMN
1467                                warning(catgets(catd, 1, 282, "Ignoring Distribu
1468 #endif
1469                                break;
1470                        case 8: /* -j seen */
1471                                argv[i] = NOCATGETS("-j");
1472 #ifndef TEAMWARE_MAKE_CMN
1473                                warning(catgets(catd, 1, 283, "Ignoring Distribu
1474 #endif
1475                                break;
1476                        case 16: /* -M seen */
1477                                argv[i] = NOCATGETS("-M");
1478 #ifndef TEAMWARE_MAKE_CMN
1479                                warning(catgets(catd, 1, 284, "Ignoring Parallel
1480 #endif
1481                                break;
1482                        case 32: /* -m seen */
1483                                argv[i] = NOCATGETS("-m");
1484 #ifndef TEAMWARE_MAKE_CMN
1485                                warning(catgets(catd, 1, 285, "Ignoring Distribu
1486 #endif
1487                                break;
1488 #ifndef PARALLEL
1489                        case 128: /* -O seen */
1490                                argv[i] = NOCATGETS("-O");
1491                                break;
1492 #endif
1493                        case 256: /* -K seen */
1494                                argv[i] = NOCATGETS("-K");
1495                                break;
1496                        case 512:       /* -o seen */
1497                                argv[i] = NOCATGETS("-o");
1498 #ifndef TEAMWARE_MAKE_CMN
1499                                warning(catgets(catd, 1, 311, "Ignoring Distribu
1500 #endif
1501                                break;
1502                        case 1024: /* -x seen */
1503                                argv[i] = NOCATGETS("-x");
1504 #ifndef TEAMWARE_MAKE_CMN
1505                                warning(catgets(catd, 1, 353, "Ignoring Distribu
1506 #endif
1507                                break;
1508                        default: /* > 1 of -c, f, g, j, K, M, m, O, o, x seen */
1509                                fatal(catgets(catd, 1, 286, "Illegal command lin
1510                        }
1511
1512                        makefile_next = 0;
1513                        current_optind = optind;
```



```
1514                }
1515        }
1516 }
1517
1518 static void
1519 quote_str(char *str, char *qstr)
1520 {
1521        char            *to;
1522        char            *from;
1523
1524        to = qstr;
1525        for (from = str; *from; from++) {
1526                switch (*from) {
1527                case ';':       /* End of command */
1528                case '(':       /* Start group */
1529                case ')':       /* End group */
1530                case '{':       /* Start group */
1531                case '}':       /* End group */
1532                case '[':       /* Reg expr - any of a set of chars */
1533                case ']':       /* End of set of chars */
1534                case '|':       /* Pipe or logical-or */
1535                case '^':       /* Old-fashioned pipe */
1536                case '&':       /* Background or logical-and */
1537                case '<':       /* Redirect stdin */
1538                case '>':       /* Redirect stdout */
1539                case '*':       /* Reg expr - any sequence of chars */
1540                case '?':       /* Reg expr - any single char */
1541                case '$':       /* Variable substitution */
1542                case '\'':      /* Singe quote - turn off all magic */
1543                case '"':       /* Double quote - span whitespace */
1544                case '`':       /* Backquote - run a command */
1545                case '#':       /* Comment */
1546                case ' ':       /* Space (for MACRO=value1 value2  */
1547                case '\\':      /* Escape char - turn off magic of next char */
1548                        *to++ = '\\';
1549                        break;
1550
1551                default:
1552                        break;
1553                }
1554                *to++ = *from;
1555        }
1556        *to = '\0';
1557 }
1558
1559 static void
1560 unquote_str(char *str, char *qstr)
1561 {
1562        char            *to;
1563        char            *from;
1564
1565        to = qstr;
1566        for (from = str; *from; from++) {
1567                if (*from == '\\') {
1568                        from++;
1569                }
1570                *to++ = *from;
1571        }
1572        *to = '\0';
1573 }
1574
1575 /*
1576  * Convert the MAKEFLAGS string value into a vector of char *, similar
1577  * to argv.
1578  */
1579 static void
```

```
1580 setup_makeflags_argv()
1581 {
1582         char            *cp;
1583         char            *cp1;
1584         char            *cp2;
1585         char            *cp3;
1586         char            *cp_orig;
1587         Boolean         add_hyphen;
1588         int             i;
1589         char            tmp_char;

1591         mf_argc = 1;
1592         cp = getenv(makeflags->string_mb);
1593         cp_orig = cp;

1595         if (cp) {
1596                 /*
1597                  * If new MAKEFLAGS format, no need to add hyphen.
1598                  * If old MAKEFLAGS format, add hyphen before flags.
1599                  */

1601                 if ((strchr(cp, (int) hyphen_char) != NULL) ||
1602                     (strchr(cp, (int) equal_char) != NULL)) {

1604                         /* New MAKEFLAGS format */

1606                         add_hyphen = false;
1607 #ifdef ADDFIX5060758
1608                         /* Check if MAKEFLAGS value begins with multiple
1609                          * hyphen characters, and remove all duplicates.
1610                          * Usually it happens when the next command is
1611                          * used: $(MAKE) -$(MAKEFLAGS)
1612                          * This is a workaround for BugID 5060758.
1613                          */
1614                         while (*cp) {
1615                                 if (*cp != (int) hyphen_char) {
1616                                         break;
1617                                 }
1618                                 cp++;
1619                                 if (*cp == (int) hyphen_char) {
1620                                         /* There are two hyphens. Skip one */
1621                                         cp_orig = cp;
1622                                         cp++;
1623                                 }
1624                                 if (!(*cp)) {
1625                                         /* There are hyphens only. Skip all */
1626                                         cp_orig = cp;
1627                                         break;
1628                                 }
1629                         }
1630 #endif
1631                 } else {

1633                         /* Old MAKEFLAGS format */

1635                         add_hyphen = true;
1636                 }
1637         }

1639         /* Find the number of arguments in MAKEFLAGS */
1640         while (cp && *cp) {
1641                 /* Skip white spaces */
1642                 while (cp && *cp && isspace(*cp)) {
1643                         cp++;
1644                 }
1645                 if (cp && *cp) {
```



```
1646                         /* Increment arg count */
1647                         mf_argc++;
1648                         /* Go to next white space */
1649                         while (cp && *cp && !isspace(*cp)) {
1650                                 if(*cp == (int) backslash_char) {
1651                                         cp++;
1652                                 }
1653                                 cp++;
1654                         }
1655                 }
1656         }
1657         /* Allocate memory for the new MAKEFLAGS argv */
1658         mf_argv = (char **) malloc((mf_argc + 1) * sizeof(char *));
1659         mf_argv[0] = NOCATGETS("MAKEFLAGS");
1660         /*
1661          * Convert the MAKEFLAGS string value into a vector of char *,
1662          * similar to argv.
1663          */
1664         cp = cp_orig;
1665         for (i = 1; i < mf_argc; i++) {
1666                 /* Skip white spaces */
1667                 while (cp && *cp && isspace(*cp)) {
1668                         cp++;
1669                 }
1670                 if (cp && *cp) {
1671                         cp_orig = cp;
1672                         /* Go to next white space */
1673                         while (cp && *cp && !isspace(*cp)) {
1674                                 if(*cp == (int) backslash_char) {
1675                                         cp++;
1676                                 }
1677                                 cp++;
1678                         }
1679                         tmp_char = *cp;
1680                         *cp = (int) nul_char;
1681                         if (add_hyphen) {
1682                                 mf_argv[i] = getmem(2 + strlen(cp_orig));
1683                                 mf_argv[i][0] = '\0';
1684                                 (void) strcat(mf_argv[i], "-");
1685                                 // (void) strcat(mf_argv[i], cp_orig);
1686                                 unquote_str(cp_orig, mf_argv[i]+1);
1687                         } else {
1688                                 mf_argv[i] = getmem(2 + strlen(cp_orig));
1689                                 //mf_argv[i] = strdup(cp_orig);
1690                                 unquote_str(cp_orig, mf_argv[i]);
1691                         }
1692                         *cp = tmp_char;
1693                 }
1694         }
1695         mf_argv[i] = NULL;
1696 }

1698 /*
1699  *      parse_command_option(ch)
1700  *
1701  *      Parse make command line options.
1702  *
1703  *      Return value:
1704  *                              Indicates if any -f -c or -M were seen
1705  *
1706  *      Parameters:
1707  *              ch              The character to parse
1708  *
1709  *      Static variables used:
1710  *              dmake_group_specified   Set for make -g
1711  *              dmake_max_jobs_specified        Set for make -j
```

```
 *               dmake_mode_specified    Set for make -m
 *               dmake_add_mode_specified        Set for make -x
 *               dmake_compat_mode_specified     Set for make -x SUN_MAKE_COMPAT_
 *               dmake_output_mode_specified     Set for make -x DMAKE_OUTPUT_MOD
 *               dmake_odir_specified    Set for make -o
 *               dmake_rcfile_specified  Set for make -c
 *               env_wins                Set for make -e
 *               ignore_default_mk       Set for make -r
 *               trace_status            Set for make -p
 *
 *      Global variables used:
 *               .make.state path & name set for make -K
 *               continue_after_error    Set for make -k
 *               debug_level             Set for make -d
 *               do_not_exec_rule        Set for make -n
 *               filter_stderr           Set for make -X
 *               ignore_errors_all       Set for make -i
 *               no_parallel             Set for make -R
 *               quest                   Set for make -q
 *               read_trace_level        Set for make -D
 *               report_dependencies     Set for make -P
 *               send_mtool_msgs         Set for make -K
 *               silent_all              Set for make -s
 *               touch                   Set for make -t
 */
static int
parse_command_option(register char ch)
{
        static int              invert_next = 0;
        int                     invert_this = invert_next;

        invert_next = 0;
        switch (ch) {
        case '-':                       /* Ignore "--" */
                return 0;
        case '~':                       /* Invert next option */
                invert_next = 1;
                return 0;
        case 'B':                       /* Obsolete */
                return 0;
        case 'b':                       /* Obsolete */
                return 0;
        case 'c':                       /* Read alternative dmakerc file */
                if (invert_this) {
                        dmake_rcfile_specified = false;
                } else {
                        dmake_rcfile_specified = true;
                }
                return 2;
        case 'D':                       /* Show lines read */
                if (invert_this) {
                        read_trace_level--;
                } else {
                        read_trace_level++;
                }
                return 0;
        case 'd':                       /* Debug flag */
                if (invert_this) {
                        debug_level--;
                } else {
#if defined( HP_UX) || defined(linux)
                       if (debug_level < 2)  /* Fixes a bug on HP-UX */
#endif
                        debug_level++;
                }
                return 0;
```



```
#ifdef NSE
        case 'E':
                if (invert_this) {
                        nse = false;
                } else {
                        nse = true;
                }
                nse_init_source_suffixes();
                return 0;
#endif
        case 'e':                       /* Environment override flag */
                if (invert_this) {
                        env_wins = false;
                } else {
                        env_wins = true;
                }
                return 0;
        case 'f':                       /* Read alternative makefile(s) */
                return 1;
        case 'g':                       /* Use alternative DMake group */
                if (invert_this) {
                        dmake_group_specified = false;
                } else {
                        dmake_group_specified = true;
                }
                return 4;
        case 'i':                       /* Ignore errors */
                if (invert_this) {
                        ignore_errors_all = false;
                } else {
                        ignore_errors_all = true;
                }
                return 0;
        case 'j':                       /* Use alternative DMake max jobs */
                if (invert_this) {
                        dmake_max_jobs_specified = false;
                } else {
                        dmake_max_jobs_specified = true;
                }
                return 8;
        case 'K':                       /* Read alternative .make.state */
                return 256;
        case 'k':                       /* Keep making even after errors */
                if (invert_this) {
                        continue_after_error = false;
                } else {
                        continue_after_error = true;
                        continue_after_error_ever_seen = true;
                }
                return 0;
        case 'M':                       /* Read alternative make.machines file
                if (invert_this) {
                        pmake_machinesfile_specified = false;
                } else {
                        pmake_machinesfile_specified = true;
                        dmake_mode_type = parallel_mode;
                        no_parallel = false;
                }
                return 16;
        case 'm':                       /* Use alternative DMake build mode */
                if (invert_this) {
                        dmake_mode_specified = false;
                } else {
                        dmake_mode_specified = true;
                }
                return 32;
```

```
1844         case 'x':                          /* Use alternative DMake mode */
1845                 if (invert_this) {
1846                         dmake_add_mode_specified = false;
1847                 } else {
1848                         dmake_add_mode_specified = true;
1849                 }
1850                 return 1024;
1851         case 'N':                          /* Reverse -n */
1852                 if (invert_this) {
1853                         do_not_exec_rule = true;
1854                 } else {
1855                         do_not_exec_rule = false;
1856                 }
1857                 return 0;
1858         case 'n':                          /* Print, not exec commands */
1859                 if (invert_this) {
1860                         do_not_exec_rule = false;
1861                 } else {
1862                         do_not_exec_rule = true;
1863                 }
1864                 return 0;
1865 #ifndef PARALLEL
1866         case 'O':                          /* Send job start & result msgs */
1867                 if (invert_this) {
1868                         send_mtool_msgs = false;
1869                 } else {
1870 #ifdef DISTRIBUTED
1871                         send_mtool_msgs = true;
1872 #endif
1873                 }
1874                 return 128;
1875 #endif
1876         case 'o':                          /* Use alternative dmake output dir */
1877                 if (invert_this) {
1878                         dmake_odir_specified = false;
1879                 } else {
1880                         dmake_odir_specified = true;
1881                 }
1882                 return 512;
1883         case 'P':                          /* Print for selected targets */
1884                 if (invert_this) {
1885                         report_dependencies_level--;
1886                 } else {
1887                         report_dependencies_level++;
1888                 }
1889                 return 0;
1890         case 'p':                          /* Print description */
1891                 if (invert_this) {
1892                         trace_status = false;
1893                         do_not_exec_rule = false;
1894                 } else {
1895                         trace_status = true;
1896                         do_not_exec_rule = true;
1897                 }
1898                 return 0;
1899         case 'q':                          /* Question flag */
1900                 if (invert_this) {
1901                         quest = false;
1902                 } else {
1903                         quest = true;
1904                 }
1905                 return 0;
1906         case 'R':                          /* Don't run in parallel */
1907 #ifdef TEAMWARE_MAKE_CMN
1908                 if (invert_this) {
1909                         pmake_cap_r_specified = false;
```



```
1910                         no_parallel = false;
1911                 } else {
1912                         pmake_cap_r_specified = true;
1913                         dmake_mode_type = serial_mode;
1914                         no_parallel = true;
1915                 }
1916 #else
1917                 warning(catgets(catd, 1, 182, "Ignoring ParallelMake -R option")
1918 #endif
1919                 return 0;
1920         case 'r':                          /* Turn off internal rules */
1921                 if (invert_this) {
1922                         ignore_default_mk = false;
1923                 } else {
1924                         ignore_default_mk = true;
1925                 }
1926                 return 0;
1927         case 'S':                          /* Reverse -k */
1928                 if (invert_this) {
1929                         continue_after_error = true;
1930                 } else {
1931                         continue_after_error = false;
1932                         stop_after_error_ever_seen = true;
1933                 }
1934                 return 0;
1935         case 's':                          /* Silent flag */
1936                 if (invert_this) {
1937                         silent_all = false;
1938                 } else {
1939                         silent_all = true;
1940                 }
1941                 return 0;
1942         case 'T':                          /* Print target list */
1943                 if (invert_this) {
1944                         list_all_targets = false;
1945                         do_not_exec_rule = false;
1946                 } else {
1947                         list_all_targets = true;
1948                         do_not_exec_rule = true;
1949                 }
1950                 return 0;
1951         case 't':                          /* Touch flag */
1952                 if (invert_this) {
1953                         touch = false;
1954                 } else {
1955                         touch = true;
1956                 }
1957                 return 0;
1958         case 'u':                          /* Unconditional flag */
1959                 if (invert_this) {
1960                         build_unconditional = false;
1961                 } else {
1962                         build_unconditional = true;
1963                 }
1964                 return 0;
1965         case 'V':                         /* SVR4 mode */
1966                 svr4 = true;
1967                 return 0;
1968         case 'v':                          /* Version flag */
1969                 if (invert_this) {
1970                 } else {
1971 #ifdef TEAMWARE_MAKE_CMN
1972                         fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
1973 #ifdef SUN5_0
1974                         exit_status = 0;
1975 #endif
```

```
1976                        exit(0);
1977 #else
1978                        warning(catgets(catd, 1, 324, "Ignoring DistributedMake
1979 #endif
1980                }
1981                return 0;
1982        case 'w':                        /* Unconditional flag */
1983                if (invert_this) {
1984                        report_cwd = false;
1985                } else {
1986                        report_cwd = true;
1987                }
1988                return 0;
1989 #if 0
1990        case 'X':                       /* Filter stdout */
1991                if (invert_this) {
1992                        filter_stderr = false;
1993                } else {
1994                        filter_stderr = true;
1995                }
1996                return 0;
1997 #endif
1998        default:
1999                break;
2000        }
2001        return 0;
2002 }

2004 /*
2005  *      setup_for_projectdir()
2006  *
2007  *      Read the PROJECTDIR variable, if defined, and set the sccs path
2008  *
2009  *      Parameters:
2010  *
2011  *      Global variables used:
2012  *              sccs_dir_path   Set to point to SCCS dir to use
2013  */
2014 static void
2015 setup_for_projectdir(void)
2016 {
2017 static char     path[MAXPATHLEN];
2018 char            cwdpath[MAXPATHLEN];
2019 uid_t uid;
2020 int   done=0;

2022        /* Check if we should use PROJECTDIR when reading the SCCS dir. */
2023        sccs_dir_path = getenv(NOCATGETS("PROJECTDIR"));
2024        if ((sccs_dir_path != NULL) &&
2025            (sccs_dir_path[0] != (int) slash_char)) {
2026                struct passwd *pwent;

2028             {
2029                uid = getuid();
2030                pwent = getpwuid(uid);
2031                if (pwent == NULL) {
2032                   fatal(catgets(catd, 1, 188, "Bogus USERID "));
2033                }
2034                if ((pwent = getpwnam(sccs_dir_path)) == NULL) {
2035                        /*empty block : it'll go & check cwd  */
2036                }
2037                else {
2038                  (void) sprintf(path, NOCATGETS("%s/src"), pwent->pw_dir);
2039                  if (access(path, F_OK) == 0) {
2040                        sccs_dir_path = path;
2041                        done = 1;
```



```
2042                  } else {
2043                        (void) sprintf(path, NOCATGETS("%s/source"), pwent->pw_d
2044                        if (access(path, F_OK) == 0) {
2045                                sccs_dir_path = path;
2046                                done = 1;
2047                        }
2048                    }
2049                }
2050                if (!done) {
2051                    if (getcwd(cwdpath, MAXPATHLEN - 1 )) {

2053                        (void) sprintf(path, NOCATGETS("%s/%s"), cwdpath,sccs_dir
2054                        if (access(path, F_OK) == 0) {
2055                                sccs_dir_path = path;
2056                                done = 1;
2057                        } else {
2058                                fatal(catgets(catd, 1, 189, "Bogus PROJECTDIR '%
2059                        }
2060                    }
2061                }
2062           }
2063        }
2064 }

2066 /*
2067  *      set_sgs_support()
2068  *
2069  *      Add the libmakestate.so.1 lib to the env var SGS_SUPPORT
2070  *        if it's not already in there.
2071  *      The SGS_SUPPORT env var and libmakestate.so.1 is used by
2072  *        the linker ld to report .make.state info back to make.
2073  */
2074 static void
2075 set_sgs_support()
2076 {
2077        int             len;
2078        char            *newpath;
2079        char            *oldpath;
2080        static char     *prev_path;

2082        oldpath = getenv(LD_SUPPORT_ENV_VAR);
2083        if (oldpath == NULL) {
2084                len = strlen(LD_SUPPORT_ENV_VAR) + 1 +
2085                        strlen(LD_SUPPORT_MAKE_LIB) + 1;
2086                newpath = (char *) malloc(len);
2087                sprintf(newpath, "%s=", LD_SUPPORT_ENV_VAR);
2088        } else {
2089                len = strlen(LD_SUPPORT_ENV_VAR) + 1 + strlen(oldpath) + 1 +
2090                        strlen(LD_SUPPORT_MAKE_LIB) + 1;
2091                newpath = (char *) malloc(len);
2092                sprintf(newpath, "%s=%s", LD_SUPPORT_ENV_VAR, oldpath);
2093        }

2095 #if defined(TEAMWARE_MAKE_CMN)

2097        /* function maybe_append_str_to_env_var() is defined in avo_util library
2098         * Serial make should not use this library !!!
2099         */
2100        maybe_append_str_to_env_var(newpath, LD_SUPPORT_MAKE_LIB);
2101 #else
2102        if (oldpath == NULL) {
2103                sprintf(newpath, "%s%s", newpath, LD_SUPPORT_MAKE_LIB);
2104        } else {
2105                sprintf(newpath, "%s:%s", newpath, LD_SUPPORT_MAKE_LIB);
2106        }
2107 #endif
```

```
2108         putenv(newpath);
2109         if (prev_path) {
2110                 free(prev_path);
2111         }
2112         prev_path = newpath;
2113 }

2115 /*
2116  *      read_files_and_state(argc, argv)
2117  *
2118  *      Read the makefiles we care about and the environment
2119  *      Also read the = style command line options
2120  *
2121  *      Parameters:
2122  *              argc            You know what this is
2123  *              argv            You know what this is
2124  *
2125  *      Static variables used:
2126  *              env_wins        make -e, determines if env vars are RO
2127  *              ignore_default_mk make -r, determines if make.rules is read
2128  *              not_auto_depen  dwight
2129  *
2130  *      Global variables used:
2131  *              default_target_to_build Set to first proper target from file
2132  *              do_not_exec_rule Set to false when makfile is made
2133  *              dot             The Name ".", used to read current dir
2134  *              empty_name      The Name "", use as macro value
2135  *              keep_state      Set if KEEP_STATE is in environment
2136  *              make_state      The Name ".make.state", used to read file
2137  *              makefile_type   Set to type of file being read
2138  *              makeflags       The Name "MAKEFLAGS", used to set macro value
2139  *              not_auto        dwight
2140  *              nse             Set if NSE_ENV is in the environment
2141  *              read_trace_level Checked to se if the reader should trace
2142  *              report_dependencies If -P is on we do not read .make.state
2143  *              trace_reader    Set if reader should trace
2144  *              virtual_root    The Name "VIRTUAL_ROOT", used to check value
2145  */
2146 static void
2147 read_files_and_state(int argc, char **argv)
2148 {
2149         wchar_t                 buffer[1000];
2150         wchar_t                 buffer_posix[1000];
2151         register char           ch;
2152         register char           *cp;
2153         Property                def_make_macro = NULL;
2154         Name                    def_make_name;
2155         Name                    default_makefile;
2156         String_rec              dest;
2157         wchar_t                 destbuffer[STRING_BUFFER_LENGTH];
2158         register int            i;
2159         register int            j;
2160         Name                    keep_state_name;
2161         int                     length;
2162         Name                    Makefile;
2163         register Property       macro;
2164         struct stat             make_state_stat;
2165         Name                    makefile_name;
2166         register int            makefile_next = 0;
2167         register Boolean        makefile_read = false;
2168         String_rec              makeflags_string;
2169         String_rec              makeflags_string_posix;
2170         String_rec *            makeflags_string_current;
2171         Name                    makeflags_value_saved;
2172         register Name           name;
2173         Name                    new_make_value;
```



```
2174         Boolean                 save_do_not_exec_rule;
2175         Name                    sdotMakefile;
2176         Name                    sdotmakefile_name;
2177         static wchar_t          state_file_str;
2178         static char             state_file_str_mb[MAXPATHLEN];
2179         static struct _Name     state_filename;
2180         Boolean                 temp;
2181         char                    tmp_char;
2182         wchar_t                 *tmp_wcs_buffer;
2183         register Name           value;
2184         ASCII_Dyn_Array         makeflags_and_macro;
2185         Boolean                 is_xpg4;

2187 /*
2188  *      Remember current mode. It may be changed after reading makefile
2189  *      and we will have to correct MAKEFLAGS variable.
2190  */
2191         is_xpg4 = posix;

2193         MBSTOWCS(wcs_buffer, NOCATGETS("KEEP_STATE"));
2194         keep_state_name = GETNAME(wcs_buffer, FIND_LENGTH);
2195         MBSTOWCS(wcs_buffer, NOCATGETS("Makefile"));
2196         Makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2197         MBSTOWCS(wcs_buffer, NOCATGETS("makefile"));
2198         makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
2199         MBSTOWCS(wcs_buffer, NOCATGETS("s.makefile"));
2200         sdotmakefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
2201         MBSTOWCS(wcs_buffer, NOCATGETS("s.Makefile"));
2202         sdotMakefile = GETNAME(wcs_buffer, FIND_LENGTH);

2204 /*
2205  *      Set flag if NSE is active
2206  */
2207 #ifdef NSE
2208         if (getenv(NOCATGETS("NSE_ENV")) != NULL) {
2209                 nse = true;
2210         }
2211 #endif

2213 /*
2214  *      initialize global dependency entry for .NOT_AUTO
2215  */
2216         not_auto_depen->next = NULL;
2217         not_auto_depen->name = not_auto;
2218         not_auto_depen->automatic = not_auto_depen->stale = false;

2220 /*
2221  *      Read internal definitions and rules.
2222  */
2223         if (read_trace_level > 1) {
2224                 trace_reader = true;
2225         }
2226         if (!ignore_default_mk) {
2227 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
2228                 if (svr4) {
2229                         MBSTOWCS(wcs_buffer, NOCATGETS("svr4.make.rules"));
2230                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2231                 } else {
2232                         MBSTOWCS(wcs_buffer, NOCATGETS("make.rules"));
2233                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2234                 }
2235 #else
2236                 MBSTOWCS(wcs_buffer, NOCATGETS("default.mk"));
2237                 default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2238 #endif
2239                 default_makefile->stat.is_file = true;
```

```
2241                 (void) read_makefile(default_makefile,
2242                                      true,
2243                                      false,
2244                                      true);
2245         }

2247         /*
2248          * If the user did not redefine the MAKE macro in the
2249          * default makefile (make.rules), then we'd like to
2250          * change the macro value of MAKE to be some form
2251          * of argv[0] for recursive MAKE builds.
2252          */
2253         MBSTOWCS(wcs_buffer, NOCATGETS("MAKE"));
2254         def_make_name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2255         def_make_macro = get_prop(def_make_name->prop, macro_prop);
2256         if ((def_make_macro != NULL) &&
2257             (IS_EQUAL(def_make_macro->body.macro.value->string_mb,
2258                   NOCATGETS("make")))) {
2259                 MBSTOWCS(wcs_buffer, argv_zero_string);
2260                 new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2261                 (void) SETVAR(def_make_name,
2262                               new_make_value,
2263                               false);
2264         }

2266         default_target_to_build = NULL;
2267         trace_reader = false;

2269 /*
2270  *      Read environment args. Let file args which follow override unless
2271  *      -e option seen. If -e option is not mentioned.
2272  */
2273         read_environment(env_wins);
2274         if (getvar(virtual_root)->hash.length == 0) {
2275                 maybe_append_prop(virtual_root, macro_prop)
2276                   ->body.macro.exported = true;
2277                 MBSTOWCS(wcs_buffer, "/");
2278                 (void) SETVAR(virtual_root,
2279                               GETNAME(wcs_buffer, FIND_LENGTH),
2280                               false);
2281         }

2283 /*
2284  * We now scan mf_argv and argv to see if we need to set
2285  * any of the DMake-added options/variables in MAKEFLAGS.
2286  */

2288         makeflags_and_macro.start = 0;
2289         makeflags_and_macro.size = 0;
2290         enter_argv_values(mf_argc, mf_argv, &makeflags_and_macro);
2291         enter_argv_values(argc, argv, &makeflags_and_macro);

2293 /*
2294  *      Set MFLAGS and MAKEFLAGS
2295  *
2296  *      Before reading makefile we do not know exactly which mode
2297  *      (posix or not) is used. So prepare two MAKEFLAGS strings
2298  *      for both posix and solaris modes because they are different.
2299  */
2300         INIT_STRING_FROM_STACK(makeflags_string, buffer);
2301         INIT_STRING_FROM_STACK(makeflags_string_posix, buffer_posix);
2302         append_char((int) hyphen_char, &makeflags_string);
2303         append_char((int) hyphen_char, &makeflags_string_posix);

2305         switch (read_trace_level) {
```



```
2306         case 2:
2307                 append_char('D', &makeflags_string);
2308                 append_char('D', &makeflags_string_posix);
2309         case 1:
2310                 append_char('D', &makeflags_string);
2311                 append_char('D', &makeflags_string_posix);
2312         }
2313         switch (debug_level) {
2314         case 2:
2315                 append_char('d', &makeflags_string);
2316                 append_char('d', &makeflags_string_posix);
2317         case 1:
2318                 append_char('d', &makeflags_string);
2319                 append_char('d', &makeflags_string_posix);
2320         }
2321 #ifdef NSE
2322         if (nse) {
2323                 append_char('E', &makeflags_string);
2324         }
2325 #endif
2326         if (env_wins) {
2327                 append_char('e', &makeflags_string);
2328                 append_char('e', &makeflags_string_posix);
2329         }
2330         if (ignore_errors_all) {
2331                 append_char('i', &makeflags_string);
2332                 append_char('i', &makeflags_string_posix);
2333         }
2334         if (continue_after_error) {
2335                 if (stop_after_error_ever_seen) {
2336                         append_char('S', &makeflags_string_posix);
2337                         append_char((int) space_char, &makeflags_string_posix);
2338                         append_char((int) hyphen_char, &makeflags_string_posix);
2339                 }
2340                 append_char('k', &makeflags_string);
2341                 append_char('k', &makeflags_string_posix);
2342         } else {
2343                 if (stop_after_error_ever_seen
2344                     && continue_after_error_ever_seen) {
2345                         append_char('k', &makeflags_string_posix);
2346                         append_char((int) space_char, &makeflags_string_posix);
2347                         append_char((int) hyphen_char, &makeflags_string_posix);
2348                         append_char('S', &makeflags_string_posix);
2349                 }
2350         }
2351         if (do_not_exec_rule) {
2352                 append_char('n', &makeflags_string);
2353                 append_char('n', &makeflags_string_posix);
2354         }
2355         switch (report_dependencies_level) {
2356         case 4:
2357                 append_char('P', &makeflags_string);
2358                 append_char('P', &makeflags_string_posix);
2359         case 3:
2360                 append_char('P', &makeflags_string);
2361                 append_char('P', &makeflags_string_posix);
2362         case 2:
2363                 append_char('P', &makeflags_string);
2364                 append_char('P', &makeflags_string_posix);
2365         case 1:
2366                 append_char('P', &makeflags_string);
2367                 append_char('P', &makeflags_string_posix);
2368         }
2369         if (trace_status) {
2370                 append_char('p', &makeflags_string);
2371                 append_char('p', &makeflags_string_posix);
```

```
2372         }
2373         if (quest) {
2374                 append_char('q', &makeflags_string);
2375                 append_char('q', &makeflags_string_posix);
2376         }
2377         if (silent_all) {
2378                 append_char('s', &makeflags_string);
2379                 append_char('s', &makeflags_string_posix);
2380         }
2381         if (touch) {
2382                 append_char('t', &makeflags_string);
2383                 append_char('t', &makeflags_string_posix);
2384         }
2385         if (build_unconditional) {
2386                 append_char('u', &makeflags_string);
2387                 append_char('u', &makeflags_string_posix);
2388         }
2389         if (report_cwd) {
2390                 append_char('w', &makeflags_string);
2391                 append_char('w', &makeflags_string_posix);
2392         }
2393 #ifndef PARALLEL
2394         /* -c dmake_rcfile */
2395         if (dmake_rcfile_specified) {
2396                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2397                 dmake_rcfile = GETNAME(wcs_buffer, FIND_LENGTH);
2398                 append_makeflags_string(dmake_rcfile, &makeflags_string);
2399                 append_makeflags_string(dmake_rcfile, &makeflags_string_posix);
2400         }
2401         /* -g dmake_group */
2402         if (dmake_group_specified) {
2403                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
2404                 dmake_group = GETNAME(wcs_buffer, FIND_LENGTH);
2405                 append_makeflags_string(dmake_group, &makeflags_string);
2406                 append_makeflags_string(dmake_group, &makeflags_string_posix);
2407         }
2408         /* -j dmake_max_jobs */
2409         if (dmake_max_jobs_specified) {
2410                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
2411                 dmake_max_jobs = GETNAME(wcs_buffer, FIND_LENGTH);
2412                 append_makeflags_string(dmake_max_jobs, &makeflags_string);
2413                 append_makeflags_string(dmake_max_jobs, &makeflags_string_posix)
2414         }
2415         /* -m dmake_mode */
2416         if (dmake_mode_specified) {
2417                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2418                 dmake_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2419                 append_makeflags_string(dmake_mode, &makeflags_string);
2420                 append_makeflags_string(dmake_mode, &makeflags_string_posix);
2421         }
2422         /* -x dmake_compat_mode */
2423 //      if (dmake_compat_mode_specified) {
2424 //              MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE_COMPAT_MODE"));
2425 //              dmake_compat_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2426 //              append_makeflags_string(dmake_compat_mode, &makeflags_string);
2427 //              append_makeflags_string(dmake_compat_mode, &makeflags_string_pos
2428 //      }
2429         /* -x dmake_output_mode */
2430         if (dmake_output_mode_specified) {
2431                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
2432                 dmake_output_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2433                 append_makeflags_string(dmake_output_mode, &makeflags_string);
2434                 append_makeflags_string(dmake_output_mode, &makeflags_string_pos
2435         }
2436         /* -o dmake_odir */
2437         if (dmake_odir_specified) {
```



```
2438                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2439                 dmake_odir = GETNAME(wcs_buffer, FIND_LENGTH);
2440                 append_makeflags_string(dmake_odir, &makeflags_string);
2441                 append_makeflags_string(dmake_odir, &makeflags_string_posix);
2442         }
2443         /* -M pmake_machinesfile */
2444         if (pmake_machinesfile_specified) {
2445                 MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
2446                 pmake_machinesfile = GETNAME(wcs_buffer, FIND_LENGTH);
2447                 append_makeflags_string(pmake_machinesfile, &makeflags_string);
2448                 append_makeflags_string(pmake_machinesfile, &makeflags_string_po
2449         }
2450         /* -R */
2451         if (pmake_cap_r_specified) {
2452                 append_char((int) space_char, &makeflags_string);
2453                 append_char((int) hyphen_char, &makeflags_string);
2454                 append_char('R', &makeflags_string);
2455                 append_char((int) space_char, &makeflags_string_posix);
2456                 append_char((int) hyphen_char, &makeflags_string_posix);
2457                 append_char('R', &makeflags_string_posix);
2458         }
2459 #endif

2461 /*
2462  *      Make sure MAKEFLAGS is exported
2463  */
2464         maybe_append_prop(makeflags, macro_prop)->
2465           body.macro.exported = true;

2467         if (makeflags_string.buffer.start[1] != (int) nul_char) {
2468                 if (makeflags_string.buffer.start[1] != (int) space_char) {
2469                         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2470                         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2471                                       GETNAME(makeflags_string.buffer.start,
2472                                               FIND_LENGTH),
2473                                       false);
2474                 } else {
2475                         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2476                         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2477                                       GETNAME(makeflags_string.buffer.start + 2,
2478                                               FIND_LENGTH),
2479                                       false);
2480                 }
2481         }

2483 /*
2484  *      Add command line macro to POSIX makeflags_string
2485  */
2486         if (makeflags_and_macro.start) {
2487                 tmp_char = (char) space_char;
2488                 cp = makeflags_and_macro.start;
2489                 do {
2490                         append_char(tmp_char, &makeflags_string_posix);
2491                 } while ( tmp_char = *cp++ );
2492                 retmem_mb(makeflags_and_macro.start);
2493         }

2495 /*
2496  *      Now set the value of MAKEFLAGS macro in accordance
2497  *      with current mode.
2498  */
2499         macro = maybe_append_prop(makeflags, macro_prop);
2500         temp = (Boolean) macro->body.macro.read_only;
2501         macro->body.macro.read_only = false;
2502         if(posix || gnu_style) {
2503                 makeflags_string_current = &makeflags_string_posix;
```

```
2504         } else {
2505                 makeflags_string_current = &makeflags_string;
2506         }
2507         if (makeflags_string_current->buffer.start[1] == (int) nul_char) {
2508                 makeflags_value_saved =
2509                         GETNAME( makeflags_string_current->buffer.start + 1
2510                                , FIND_LENGTH
2511                                );
2512         } else {
2513                 if (makeflags_string_current->buffer.start[1] != (int) space_cha
2514                         makeflags_value_saved =
2515                                 GETNAME( makeflags_string_current->buffer.start
2516                                        , FIND_LENGTH
2517                                        );
2518                 } else {
2519                         makeflags_value_saved =
2520                                 GETNAME( makeflags_string_current->buffer.start
2521                                        , FIND_LENGTH
2522                                        );
2523                 }
2524         }
2525         (void) SETVAR( makeflags
2526                      , makeflags_value_saved
2527                      , false
2528                      );
2529         macro->body.macro.read_only = temp;

2531 /*
2532  *      Read command line "-f" arguments and ignore -c, g, j, K, M, m, O and o a
2533  */
2534         save_do_not_exec_rule = do_not_exec_rule;
2535         do_not_exec_rule = false;
2536         if (read_trace_level > 0) {
2537                 trace_reader = true;
2538         }

2540         for (i = 1; i < argc; i++) {
2541                 if (argv[i] &&
2542                     (argv[i][0] == (int) hyphen_char) &&
2543                     (argv[i][1] == 'f') &&
2544                     (argv[i][2] == (int) nul_char)) {
2545                         argv[i] = NULL;         /* Remove -f */
2546                         if (i >= argc - 1) {
2547                                 fatal(catgets(catd, 1, 190, "No filename argumen
2548                         }
2549                         MBSTOWCS(wcs_buffer, argv[++i]);
2550                         primary_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2551                         (void) read_makefile(primary_makefile, true, true, true)
2552                         argv[i] = NULL;         /* Remove filename */
2553                         makefile_read = true;
2554                 } else if (argv[i] &&
2555                            (argv[i][0] == (int) hyphen_char) &&
2556                            (argv[i][1] == 'c' ||
2557                             argv[i][1] == 'g' ||
2558                             argv[i][1] == 'j' ||
2559                             argv[i][1] == 'K' ||
2560                             argv[i][1] == 'M' ||
2561                             argv[i][1] == 'm' ||
2562                             argv[i][1] == 'O' ||
2563                             argv[i][1] == 'o') &&
2564                           (argv[i][2] == (int) nul_char)) {
2565                         argv[i] = NULL;
2566                         argv[++i] = NULL;
2567                 }
2568         }
```



```
2570 /*
2571  *      If no command line "-f" args then look for "makefile", and then for
2572  *      "Makefile" if "makefile" isn't found.
2573  */
2574         if (!makefile_read) {
2575                 (void) read_dir(dot,
2576                                 (wchar_t *) NULL,
2577                                 (Property) NULL,
2578                                 (wchar_t *) NULL);
2579             if (!posix) {
2580                 if (makefile_name->stat.is_file) {
2581                         if (Makefile->stat.is_file) {
2582                                 warning(catgets(catd, 1, 310, "Both 'makefile' a
2583                         }
2584                         primary_makefile = makefile_name;
2585                         makefile_read = read_makefile(makefile_name,
2586                                                       false,
2587                                                       false,
2588                                                       true);
2589                 }
2590                 if (!makefile_read &&
2591                     Makefile->stat.is_file) {
2592                         primary_makefile = Makefile;
2593                         makefile_read = read_makefile(Makefile,
2594                                                       false,
2595                                                       false,
2596                                                       true);
2597                 }
2598             } else {

2600                 enum sccs_stat save_m_has_sccs = NO_SCCS;
2601                 enum sccs_stat save_M_has_sccs = NO_SCCS;

2603                 if (makefile_name->stat.is_file) {
2604                         if (Makefile->stat.is_file) {
2605                                 warning(catgets(catd, 1, 191, "Both 'makefile' a
2606                         }
2607                 }
2608                 if (makefile_name->stat.is_file) {
2609                         if (makefile_name->stat.has_sccs == NO_SCCS) {
2610                            primary_makefile = makefile_name;
2611                            makefile_read = read_makefile(makefile_name,
2612                                                          false,
2613                                                          false,
2614                                                          true);
2615                         } else {
2616                           save_m_has_sccs = makefile_name->stat.has_sccs;
2617                           makefile_name->stat.has_sccs = NO_SCCS;
2618                           primary_makefile = makefile_name;
2619                           makefile_read = read_makefile(makefile_name,
2620                                                         false,
2621                                                         false,
2622                                                         true);
2623                         }
2624                 }
2625                 if (!makefile_read &&
2626                     Makefile->stat.is_file) {
2627                         if (Makefile->stat.has_sccs == NO_SCCS) {
2628                            primary_makefile = Makefile;
2629                            makefile_read = read_makefile(Makefile,
2630                                                          false,
2631                                                          false,
2632                                                          true);
2633                         } else {
2634                           save_M_has_sccs = Makefile->stat.has_sccs;
2635                           Makefile->stat.has_sccs = NO_SCCS;
```

```
2636                          primary_makefile = Makefile;
2637                          makefile_read = read_makefile(Makefile,
2638                                                        false,
2639                                                        false,
2640                                                        true);
2641                  }
2642          }
2643          if (!makefile_read &&
2644                  makefile_name->stat.is_file) {
2645                     makefile_name->stat.has_sccs = save_m_has_sccs;
2646                     primary_makefile = makefile_name;
2647                     makefile_read = read_makefile(makefile_name,
2648                                                   false,
2649                                                   false,
2650                                                   true);
2651          }
2652          if (!makefile_read &&
2653              Makefile->stat.is_file) {
2654                     Makefile->stat.has_sccs = save_M_has_sccs;
2655                     primary_makefile = Makefile;
2656                     makefile_read = read_makefile(Makefile,
2657                                                   false,
2658                                                   false,
2659                                                   true);
2660          }
2661      }
2662  }
2663  do_not_exec_rule = save_do_not_exec_rule;
2664  allrules_read = makefile_read;
2665  trace_reader = false;

2667 /*
2668  *      Now get current value of MAKEFLAGS and compare it with
2669  *      the saved value we set before reading makefile.
2670  *      If they are different then MAKEFLAGS is subsequently set by
2671  *      makefile, just leave it there. Otherwise, if make mode
2672  *      is changed by using .POSIX target in makefile we need
2673  *      to correct MAKEFLAGS value.
2674  */
2675  Name mf_val = getvar(makeflags);
2676  if( (posix != is_xpg4)
2677   && (!strcmp(mf_val->string_mb, makeflags_value_saved->string_mb)))
2678  {
2679          if (makeflags_string_posix.buffer.start[1] == (int) nul_char) {
2680                  (void) SETVAR(makeflags,
2681                                GETNAME(makeflags_string_posix.buffer.star
2682                                        FIND_LENGTH),
2683                                false);
2684          } else {
2685                  if (makeflags_string_posix.buffer.start[1] != (int) spac
2686                          (void) SETVAR(makeflags,
2687                                        GETNAME(makeflags_string_posix.buf
2688                                                FIND_LENGTH),
2689                                        false);
2690                  } else {
2691                          (void) SETVAR(makeflags,
2692                                        GETNAME(makeflags_string_posix.buf
2693                                                FIND_LENGTH),
2694                                        false);
2695                  }
2696          }
2697  }

2699  if (makeflags_string.free_after_use) {
2700          retmem(makeflags_string.buffer.start);
2701  }
```



```
2702  if (makeflags_string_posix.free_after_use) {
2703          retmem(makeflags_string_posix.buffer.start);
2704  }
2705  makeflags_string.buffer.start = NULL;
2706  makeflags_string_posix.buffer.start = NULL;

2708  if (posix) {
2709          /*
2710           * If the user did not redefine the ARFLAGS macro in the
2711           * default makefile (make.rules), then we'd like to
2712           * change the macro value of ARFLAGS to be in accordance
2713           * with "POSIX" requirements.
2714           */
2715          MBSTOWCS(wcs_buffer, NOCATGETS("ARFLAGS"));
2716          name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2717          macro = get_prop(name->prop, macro_prop);
2718          if ((macro != NULL) && /* Maybe (macro == NULL) || ? */
2719              (IS_EQUAL(macro->body.macro.value->string_mb,
2720                        NOCATGETS("rv")))) {
2721                  MBSTOWCS(wcs_buffer, NOCATGETS("-rv"));
2722                  value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2723                  (void) SETVAR(name,
2724                                value,
2725                                false);
2726          }
2727  }

2729  if (!posix && !svr4) {
2730          set_sgs_support();
2731  }


2734 /*
2735  *      Make sure KEEP_STATE is in the environment if KEEP_STATE is on.
2736  */
2737  macro = get_prop(keep_state_name->prop, macro_prop);
2738  if ((macro != NULL) &&
2739      macro->body.macro.exported) {
2740          keep_state = true;
2741  }
2742  if (keep_state) {
2743          if (macro == NULL) {
2744                  macro = maybe_append_prop(keep_state_name,
2745                                            macro_prop);
2746          }
2747          macro->body.macro.exported = true;
2748          (void) SETVAR(keep_state_name,
2749                        empty_name,
2750                        false);

2752          /*
2753           *      Read state file
2754           */

2756          /* Before we read state, let's make sure we have
2757          ** right state file.
2758          */
2759          /* just in case macro references are used in make_state file
2760          ** name, we better expand them at this stage using expand_value.
2761          */
2762          INIT_STRING_FROM_STACK(dest, destbuffer);
2763          expand_value(make_state, &dest, false);

2765          make_state = GETNAME(dest.buffer.start, FIND_LENGTH);

2767          if(!stat(make_state->string_mb, &make_state_stat)) {
```

```
2768                    if(!(make_state_stat.st_mode & S_IFREG) ) {
2769                         /* copy the make_state structure to the other
2770                         ** and then let make_state point to the new
2771                         ** one.
2772                         */
2773                       memcpy(&state_filename, make_state,sizeof(state_filename))
2774                       state_filename.string_mb = state_file_str_mb;
2775                /* Just a kludge to avoid two slashes back to back */
2776                       if((make_state->hash.length == 1)&&
2777                                (make_state->string_mb[0] == '/')) {
2778                          make_state->hash.length = 0;
2779                          make_state->string_mb[0] = '\0';
2780                       }
2781                       sprintf(state_file_str_mb,NOCATGETS("%s%s"),
2782                        make_state->string_mb,NOCATGETS("/.make.state"));
2783                       make_state = &state_filename;
2784                         /* adjust the length to reflect the appended string */
2785                       make_state->hash.length += 12;
2786                    }
2787                } else { /* the file doesn't exist or no permission */
2788                    char tmp_path[MAXPATHLEN];
2789                    char *slashp;

2791                    if (slashp = strrchr(make_state->string_mb, '/')) {
2792                       strncpy(tmp_path, make_state->string_mb,
2793                                (slashp - make_state->string_mb));
2794                         tmp_path[slashp - make_state->string_mb]=0;
2795                       if(strlen(tmp_path)) {
2796                         if(stat(tmp_path, &make_state_stat)) {
2797                           warning(catgets(catd, 1, 192, "directory %s for .KEEP_
2798                         }
2799                         if (access(tmp_path, F_OK) != 0) {
2800                           warning(catgets(catd, 1, 193, "can't access dir %s"),t
2801                         }
2802                       }
2803                    }
2804                }
2805                if (report_dependencies_level != 1) {
2806                        Makefile_type   makefile_type_temp = makefile_type;
2807                        makefile_type = reading_statefile;
2808                        if (read_trace_level > 1) {
2809                                trace_reader = true;
2810                        }
2811                        (void) read_simple_file(make_state,
2812                                                false,
2813                                                false,
2814                                                false,
2815                                                false,
2816                                                false,
2817                                                true);
2818                        trace_reader = false;
2819                        makefile_type = makefile_type_temp;
2820                }
2821        }
2822 }

2824 /*
2825  * Scan the argv for options and "=" type args and make them readonly.
2826  */
2827 static void
2828 enter_argv_values(int argc, char *argv[], ASCII_Dyn_Array *makeflags_and_macro)
2829 {
2830         register char           *cp;
2831         register int            i;
2832         int                     length;
2833         register Name           name;
```



```
2834         int                     opt_separator = argc;
2835         char                    tmp_char;
2836         wchar_t                 *tmp_wcs_buffer;
2837         register Name           value;
2838         Boolean                 append = false;
2839         Property                macro;
2840         struct stat             statbuf;


2843         /* Read argv options and "=" type args and make them readonly. */
2844         makefile_type = reading_nothing;
2845         for (i = 1; i < argc; ++i) {
2846                 append = false;
2847                 if (argv[i] == NULL) {
2848                         continue;
2849                 } else if (((argv[i][0] == '-') && (argv[i][1] == '-')) ||
2850                            ((argv[i][0] == (int) ' ') &&
2851                             (argv[i][1] == (int) '-') &&
2852                             (argv[i][2] == (int) ' ') &&
2853                             (argv[i][3] == (int) '-'))) {
2854                         argv[i] = NULL;
2855                         opt_separator = i;
2856                         continue;
2857                 } else if ((i < opt_separator) && (argv[i][0] == (int) hyphen_ch
2858                         switch (parse_command_option(argv[i][1])) {
2859                         case 1: /* -f seen */
2860                                 ++i;
2861                                 continue;
2862                         case 2: /* -c seen */
2863                                 if (argv[i+1] == NULL) {
2864                                         fatal(catgets(catd, 1, 194, "No dmake rc
2865                                 }
2866                                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2867                                 name = GETNAME(wcs_buffer, FIND_LENGTH);
2868                                 break;
2869                         case 4: /* -g seen */
2870                                 if (argv[i+1] == NULL) {
2871                                         fatal(catgets(catd, 1, 195, "No dmake gr
2872                                 }
2873                                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
2874                                 name = GETNAME(wcs_buffer, FIND_LENGTH);
2875                                 break;
2876                         case 8: /* -j seen */
2877                                 if (argv[i+1] == NULL) {
2878                                         fatal(catgets(catd, 1, 196, "No dmake ma
2879                                 }
2880                                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS")
2881                                 name = GETNAME(wcs_buffer, FIND_LENGTH);
2882                                 break;
2883                         case 16: /* -M seen */
2884                                 if (argv[i+1] == NULL) {
2885                                         fatal(catgets(catd, 1, 323, "No pmake ma
2886                                 }
2887                                 MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFI
2888                                 name = GETNAME(wcs_buffer, FIND_LENGTH);
2889                                 break;
2890                         case 32: /* -m seen */
2891                                 if (argv[i+1] == NULL) {
2892                                         fatal(catgets(catd, 1, 197, "No dmake mo
2893                                 }
2894                                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2895                                 name = GETNAME(wcs_buffer, FIND_LENGTH);
2896                                 break;
2897                         case 128: /* -O seen */
2898                                 if (argv[i+1] == NULL) {
2899                                         fatal(catgets(catd, 1, 287, "No file des
```

```
2900                                    }
2901                                    mtool_msgs_fd = atoi(argv[i+1]);
2902                                    /* find out if mtool_msgs_fd is a valid file des
2903                                    if (fstat(mtool_msgs_fd, &statbuf) < 0) {
2904                                            fatal(catgets(catd, 1, 355, "Invalid fil
2905                                    }
2906                                    argv[i] = NULL;
2907                                    argv[i+1] = NULL;
2908                                    continue;
2909                            case 256: /* -K seen */
2910                                    if (argv[i+1] == NULL) {
2911                                            fatal(catgets(catd, 1, 288, "No makestat
2912                                    }
2913                                    MBSTOWCS(wcs_buffer, argv[i+1]);
2914                                    make_state = GETNAME(wcs_buffer, FIND_LENGTH);
2915                                    keep_state = true;
2916                                    argv[i] = NULL;
2917                                    argv[i+1] = NULL;
2918                                    continue;
2919                            case 512:       /* -o seen */
2920                                    if (argv[i+1] == NULL) {
2921                                            fatal(catgets(catd, 1, 312, "No dmake ou
2922                                    }
2923                                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2924                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2925                                    break;
2926                            case 1024: /* -x seen */
2927                                    if (argv[i+1] == NULL) {
2928                                            fatal(catgets(catd, 1, 351, "No argument
2929                                    }
2930                                    length = strlen( NOCATGETS("SUN_MAKE_COMPAT_MODE
2931                                    if (strncmp(argv[i+1], NOCATGETS("SUN_MAKE_COMPA
2932                                            argv[i+1] = &argv[i+1][length];
2933                                            MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE
2934                                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2935                                            dmake_compat_mode_specified = dmake_add_
2936                                            break;
2937                                    }
2938                                    length = strlen( NOCATGETS("DMAKE_OUTPUT_MODE=")
2939                                    if (strncmp(argv[i+1], NOCATGETS("DMAKE_OUTPUT_M
2940                                            argv[i+1] = &argv[i+1][length];
2941                                            MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OU
2942                                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2943                                            dmake_output_mode_specified = dmake_add_
2944                                    } else {
2945                                            warning(catgets(catd, 1, 354, "Unknown a
2946                                                  argv[i+1]);
2947                                            argv[i] = argv[i + 1] = NULL;
2948                                            continue;
2949                                    }
2950                                    break;
2951                            default: /* Shouldn't reach here */
2952                                    argv[i] = NULL;
2953                                    continue;
2954                            }
2955                            argv[i] = NULL;
2956                            if (i == (argc - 1)) {
2957                                    break;
2958                            }
2959                            if ((length = strlen(argv[i+1])) >= MAXPATHLEN) {
2960                                    tmp_wcs_buffer = ALLOC_WC(length + 1);
2961                                    (void) mbstowcs(tmp_wcs_buffer, argv[i+1], lengt
2962                                    value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2963                                    retmem(tmp_wcs_buffer);
2964                            } else {
2965                                    MBSTOWCS(wcs_buffer, argv[i+1]);
```



```
2966                                    value = GETNAME(wcs_buffer, FIND_LENGTH);
2967                            }
2968                            argv[i+1] = NULL;
2969                    } else if ((cp = strchr(argv[i], (int) equal_char)) != NULL) {
2970 /*
2971  * Combine all macro in dynamic array
2972  */
2973                            if(*(cp-1) == (int) plus_char)
2974                            {
2975                                    if(isspace(*(cp-2))) {
2976                                            append = true;
2977                                            cp--;
2978                                    }
2979                            }
2980                            if(!append)
2981                                    append_or_replace_macro_in_dyn_array(makeflags_a
2983                            while (isspace(*(cp-1))) {
2984                                    cp--;
2985                            }
2986                            tmp_char = *cp;
2987                            *cp = (int) nul_char;
2988                            MBSTOWCS(wcs_buffer, argv[i]);
2989                            *cp = tmp_char;
2990                            name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2991                            while (*cp != (int) equal_char) {
2992                                    cp++;
2993                            }
2994                            cp++;
2995                            while (isspace(*cp) && (*cp != (int) nul_char)) {
2996                                    cp++;
2997                            }
2998                            if ((length = strlen(cp)) >= MAXPATHLEN) {
2999                                    tmp_wcs_buffer = ALLOC_WC(length + 1);
3000                                    (void) mbstowcs(tmp_wcs_buffer, cp, length + 1);
3001                                    value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
3002                                    retmem(tmp_wcs_buffer);
3003                            } else {
3004                                    MBSTOWCS(wcs_buffer, cp);
3005                                    value = GETNAME(wcs_buffer, FIND_LENGTH);
3006                            }
3007                            argv[i] = NULL;
3008                    } else {
3009                            /* Illegal MAKEFLAGS argument */
3010                            continue;
3011                    }
3012                    if(append) {
3013                            setvar_append(name, value);
3014                            append = false;
3015                    } else {
3016                            macro = maybe_append_prop(name, macro_prop);
3017                            macro->body.macro.exported = true;
3018                            SETVAR(name, value, false)->body.macro.read_only = true;
3019                    }
3020            }
3021 }
3023 /*
3024  * Append the DMake option and value to the MAKEFLAGS string.
3025  */
3026 static void
3027 append_makeflags_string(Name name, register String makeflags_string)
3028 {
3029            char            *option;
3031            if (strcmp(name->string_mb, NOCATGETS("DMAKE_GROUP")) == 0) {
```

```
3032                 option = NOCATGETS(" -g ");
3033         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_MAX_JOBS")) == 0) {
3034                 option = NOCATGETS(" -j ");
3035         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_MODE")) == 0) {
3036                 option = NOCATGETS(" -m ");
3037         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_ODIR")) == 0) {
3038                 option = NOCATGETS(" -o ");
3039         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_RCFILE")) == 0) {
3040                 option = NOCATGETS(" -c ");
3041         } else if (strcmp(name->string_mb, NOCATGETS("PMAKE_MACHINESFILE")) == 0
3042                 option = NOCATGETS(" -M ");
3043         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_OUTPUT_MODE")) == 0)
3044                 option = NOCATGETS(" -x DMAKE_OUTPUT_MODE=");
3045         } else if (strcmp(name->string_mb, NOCATGETS("SUN_MAKE_COMPAT_MODE")) ==
3046                 option = NOCATGETS(" -x SUN_MAKE_COMPAT_MODE=");
3047         } else {
3048                 fatal(catgets(catd, 1, 289, "Internal error: name not recognized
3049         }
3050         Property prop = maybe_append_prop(name, macro_prop);
3051         if( prop == 0 || prop->body.macro.value == 0 ||
3052             prop->body.macro.value->string_mb == 0 ) {
3053                 return;
3054         }
3055         char mbs_value[MAXPATHLEN + 100];
3056         strcpy(mbs_value, option);
3057         strcat(mbs_value, prop->body.macro.value->string_mb);
3058         MBSTOWCS(wcs_buffer, mbs_value);
3059         append_string(wcs_buffer, makeflags_string, FIND_LENGTH);
3060 }

3062 /*
3063  *      read_environment(read_only)
3064  *
3065  *      This routine reads the process environment when make starts and enters
3066  *      it as make macros. The environment variable SHELL is ignored.
3067  *
3068  *      Parameters:
3069  *              read_only       Should we make env vars read only?
3070  *
3071  *      Global variables used:
3072  *              report_pwd      Set if this make was started by other make
3073  */
3074 static void
3075 read_environment(Boolean read_only)
3076 {
3077         register char           **environment;
3078         int                     length;
3079         wchar_t                 *tmp_wcs_buffer;
3080         Boolean                 alloced_tmp_wcs_buffer = false;
3081         register wchar_t        *name;
3082         register wchar_t        *value;
3083         register Name           macro;
3084         Property                val;
3085         Boolean                 read_only_saved;

3087         reading_environment = true;
3088         environment = environ;
3089         for (; *environment; environment++) {
3090                 read_only_saved = read_only;
3091                 if ((length = strlen(*environment)) >= MAXPATHLEN) {
3092                         tmp_wcs_buffer = ALLOC_WC(length + 1);
3093                         alloced_tmp_wcs_buffer = true;
3094                         (void) mbstowcs(tmp_wcs_buffer, *environment, length + 1
3095                         name = tmp_wcs_buffer;
3096                 } else {
3097                         MBSTOWCS(wcs_buffer, *environment);
```



```
3098                         name = wcs_buffer;
3099                 }
3100                 value = (wchar_t *) wschr(name, (int) equal_char);

3102                 /*
3103                  * Looks like there's a bug in the system, but sometimes
3104                  * you can get blank lines in *environment.
3105                  */
3106                 if (!value) {
3107                         continue;
3108                 }
3109                 MBSTOWCS(wcs_buffer2, NOCATGETS("SHELL="));
3110                 if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
3111                         continue;
3112                 }
3113                 MBSTOWCS(wcs_buffer2, NOCATGETS("MAKEFLAGS="));
3114                 if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
3115                         report_pwd = true;
3116                         /*
3117                          * In POSIX mode we do not want MAKEFLAGS to be readonly
3118                          * If the MAKEFLAGS macro is subsequently set by the mak
3119                          * it replaces the MAKEFLAGS variable currently found in
3120                          * environment.
3121                          * See Assertion 50 in section 6.2.5.3 of standard P1003
3122                          */
3123                         if(posix) {
3124                                 read_only_saved = false;
3125                         }
3126                 }

3128                 /*
3129                  * We ignore SUNPRO_DEPENDENCIES and NSE_DEP. Those
3130                  * environment variables are set by make and read by
3131                  * cpp which then writes info to .make.dependency.xxx and
3132                  * .nse_depinfo. When make is invoked by another make
3133                  * (recursive make), we don't want to read this because
3134                  * then the child make will end up writing to the parent
3135                  * directory's .make.state and .nse_depinfo and clobbering
3136                  * them.
3137                  */
3138                 MBSTOWCS(wcs_buffer2, NOCATGETS("SUNPRO_DEPENDENCIES"));
3139                 if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
3140                         continue;
3141                 }
3142 #ifdef NSE
3143                 MBSTOWCS(wcs_buffer2, NOCATGETS("NSE_DEP"));
3144                 if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
3145                         continue;
3146                 }
3147 #endif

3149                 macro = GETNAME(name, value - name);
3150                 maybe_append_prop(macro, macro_prop)->body.macro.exported =
3151                   true;
3152                 if ((value == NULL) || ((value + 1)[0] == (int) nul_char)) {
3153                         val = setvar_daemon(macro,
3154                                             (Name) NULL,
3155                                             false, no_daemon, false, debug_level
3156                 } else {
3157                         val = setvar_daemon(macro,
3158                                             GETNAME(value + 1, FIND_LENGTH),
3159                                             false, no_daemon, false, debug_level
3160                 }
3161 #ifdef NSE
3162                 /*
3163                  * Must be after the call to setvar() as it sets
```

```
3164                  * imported to false.
3165                  */
3166                 maybe_append_prop(macro, macro_prop)->body.macro.imported = true
3167 #endif
3168                 val->body.macro.read_only = read_only_saved;
3169                 if (alloced_tmp_wcs_buffer) {
3170                         retmem(tmp_wcs_buffer);
3171                         alloced_tmp_wcs_buffer = false;
3172                 }
3173         }
3174         reading_environment = false;
3175 }

3177 /*
3178  *      read_makefile(makefile, complain, must_exist, report_file)
3179  *
3180  *      Read one makefile and check the result
3181  *
3182  *      Return value:
3183  *                              false is the read failed
3184  *
3185  *      Parameters:
3186  *              makefile        The file to read
3187  *              complain        Passed thru to read_simple_file()
3188  *              must_exist      Passed thru to read_simple_file()
3189  *              report_file     Passed thru to read_simple_file()
3190  *
3191  *      Global variables used:
3192  *              makefile_type   Set to indicate we are reading main file
3193  *              recursion_level Initialized
3194  */
3195 static Boolean
3196 read_makefile(register Name makefile, Boolean complain, Boolean must_exist, Bool
3197 {
3198         Boolean                 b;
3199
3200         makefile_type = reading_makefile;
3201         recursion_level = 0;
3202 #ifdef NSE
3203         wscpy(current_makefile, makefile->string);
3204 #endif
3205         reading_dependencies = true;
3206         b = read_simple_file(makefile, true, true, complain,
3207                              must_exist, report_file, false);
3208         reading_dependencies = false;
3209         return b;
3210 }

3212 /*
3213  *      make_targets(argc, argv, parallel_flag)
3214  *
3215  *      Call doname on the specified targets
3216  *
3217  *      Parameters:
3218  *              argc            You know what this is
3219  *              argv            You know what this is
3220  *              parallel_flag   True if building in parallel
3221  *
3222  *      Global variables used:
3223  *              build_failed_seen Used to generated message after failed -k
3224  *              commands_done   Used to generate message "Up to date"
3225  *              default_target_to_build First proper target in makefile
3226  *              init            The Name ".INIT", use to run command
3227  *              parallel        Global parallel building flag
3228  *              quest           make -q, suppresses messages
3229  *              recursion_level Initialized, used for tracing
```



```
3230  *              report_dependencies make -P, regroves whole process
3231  */
3232 static void
3233 make_targets(int argc, char **argv, Boolean parallel_flag)
3234 {
3235         int                     i;
3236         char                    *cp;
3237         Doname                  result;
3238         register Boolean        target_to_make_found = false;

3240         (void) doname(init, true, true);
3241         recursion_level = 1;
3242         parallel = parallel_flag;
3243 /*
3244  *      make remaining args
3245  */
3246 #ifdef TEAMWARE_MAKE_CMN
3247 /*
3248         if ((report_dependencies_level == 0) && parallel) {
3249  */
3250         if (parallel) {
3251                 /*
3252                  * If building targets in parallel, start all of the
3253                  * remaining args to build in parallel.
3254                  */
3255                 for (i = 1; i < argc; i++) {
3256                         if ((cp = argv[i]) != NULL) {
3257                                 commands_done = false;
3258                                 if ((cp[0] == (int) period_char) &&
3259                                     (cp[1] == (int) slash_char)) {
3260                                         cp += 2;
3261                                 }
3262                                  if((cp[0] == (int) ' ') &&
3263                                     (cp[1] == (int) '-') &&
3264                                     (cp[2] == (int) ' ') &&
3265                                     (cp[3] == (int) '-')) {
3266                                     argv[i] = NULL;
3267                                         continue;
3268                                 }
3269                                 MBSTOWCS(wcs_buffer, cp);
3270                                 //default_target_to_build = GETNAME(wcs_buffer,
3271                                 //                                  FIND_LENGTH);
3272                                 default_target_to_build = normalize_name(wcs_buf
3273                                                                   wslen(wcs_buff
3274                                 if (default_target_to_build == wait_name) {
3275                                         if (parallel_process_cnt > 0) {
3276                                                 finish_running();
3277                                         }
3278                                         continue;
3279                                 }
3280                                 top_level_target = get_wstring(default_target_to
3281                                 /*
3282                                  * If we can't execute the current target in
3283                                  * parallel, hold off the target processing
3284                                  * to preserve the order of the targets as they
3285                                  * in command line.
3286                                  */
3287                                 if (!parallel_ok(default_target_to_build, false)
3288                                                 && parallel_process_cnt > 0) {
3289                                         finish_running();
3290                                 }
3291                                 result = doname_check(default_target_to_build,
3292                                                       true,
3293                                                       false,
3294                                                       false);
3295                                 gather_recursive_deps();
```

```
3296                                if (/* !commands_done && */
3297                                    (result == build_ok) &&
3298                                    !quest &&
3299                                    (report_dependencies_level == 0) /*  &&
3300                                    (exists(default_target_to_build) > file_does
3301                                        if (posix) {
3302                                                if (!commands_done) {
3303                                                        (void) printf(catgets(ca
3304                                                                              default_ta
3305                                                } else {
3306                                                        if (no_action_was_taken)
3307                                                                (void) printf(ca
3308                                                                              de
3309                                                        }
3310                                                }
3311                                        } else {
3312                                                default_target_to_build->stat.ti
3313                                                if (!commands_done &&
3314                                                    (exists(default_target_to_bu
3315                                                        (void) printf(catgets(ca
3316                                                                              default_ta
3317                                                }
3318                                        }
3319                                }
3320                        }
3321                }
3322                /* Now wait for all of the targets to finish running */
3323                finish_running();
3324                //              setjmp(jmpbuffer);
3325
3326        }
3327 #endif
3328        for (i = 1; i < argc; i++) {
3329                if ((cp = argv[i]) != NULL) {
3330                        target_to_make_found = true;
3331                        if ((cp[0] == (int) period_char) &&
3332                            (cp[1] == (int) slash_char)) {
3333                                cp += 2;
3334                        }
3335                                 if((cp[0] == (int) ' ') &&
3336                                    (cp[1] == (int) '-') &&
3337                                    (cp[2] == (int) ' ') &&
3338                                    (cp[3] == (int) '-')) {
3339                                    argv[i] = NULL;
3340                                        continue;
3341                                }
3342                        MBSTOWCS(wcs_buffer, cp);
3343                        default_target_to_build = normalize_name(wcs_buffer, wsl
3344                        top_level_target = get_wstring(default_target_to_build->
3345                        report_recursion(default_target_to_build);
3346                        commands_done = false;
3347                        if (parallel) {
3348                                result = (Doname) default_target_to_build->state
3349                        } else {
3350                                result = doname_check(default_target_to_build,
3351                                                      true,
3352                                                      false,
3353                                                      false);
3354                        }
3355                        gather_recursive_deps();
3356                        if (build_failed_seen) {
3357                                build_failed_ever_seen = true;
3358                                warning(catgets(catd, 1, 200, "Target '%s' not r
3359                                        default_target_to_build->string_mb);
3360                        }
3361                        build_failed_seen = false;
```



```
3362                        if (report_dependencies_level > 0) {
3363                                print_dependencies(default_target_to_build,
3364                                                   get_prop(default_target_to_bu
3365                                                            line_prop));
3366                        }
3367                        default_target_to_build->stat.time =
3368                          file_no_time;
3369                        if (default_target_to_build->colon_splits > 0) {
3370                                default_target_to_build->state =
3371                                  build_dont_know;
3372                        }
3373                        if (!parallel &&
3374                            /* !commands_done && */
3375                            (result == build_ok) &&
3376                            !quest &&
3377                            (report_dependencies_level == 0) /*  &&
3378                            (exists(default_target_to_build) > file_doesnt_exist
3379                                if (posix) {
3380                                        if (!commands_done) {
3381                                                (void) printf(catgets(catd, 1, 2
3382                                                                      default_target_to_
3383                                        } else {
3384                                                if (no_action_was_taken) {
3385                                                        (void) printf(catgets(ca
3386                                                                              default_ta
3387                                                }
3388                                        }
3389                                } else {
3390                                        if (!commands_done &&
3391                                            (exists(default_target_to_build) > f
3392                                                (void) printf(catgets(catd, 1, 2
3393                                                                      default_target_to_
3394                                        }
3395                                }
3396                        }
3397                }
3398        }

3400 /*
3401  *      If no file arguments have been encountered,
3402  *      make the first name encountered that doesnt start with a dot
3403  */
3404        if (!target_to_make_found) {
3405                if (default_target_to_build == NULL) {
3406                        fatal(catgets(catd, 1, 202, "No arguments to build"));
3407                }
3408                commands_done = false;
3409                top_level_target = get_wstring(default_target_to_build->string_m
3410                report_recursion(default_target_to_build);


3413                if (getenv(NOCATGETS("SPRO_EXPAND_ERRORS"))){
3414                        (void) printf(NOCATGETS("::(%s)\n"),
3415                                      default_target_to_build->string_mb);
3416                }


3419 #ifdef TEAMWARE_MAKE_CMN
3420                result = doname_parallel(default_target_to_build, true, false);
3421 #else
3422                result = doname_check(default_target_to_build, true,
3423                                      false, false);
3424 #endif
3425                gather_recursive_deps();
3426                if (build_failed_seen) {
3427                        build_failed_ever_seen = true;
```

```
3428                         warning(catgets(catd, 1, 203, "Target ‘%s’ not remade be
3429                                 default_target_to_build->string_mb);
3430                 }
3431                 build_failed_seen = false;
3432                 if (report_dependencies_level > 0) {
3433                         print_dependencies(default_target_to_build,
3434                                            get_prop(default_target_to_build->
3435                                                     prop,
3436                                                     line_prop));
3437                 }
3438                 default_target_to_build->stat.time = file_no_time;
3439                 if (default_target_to_build->colon_splits > 0) {
3440                         default_target_to_build->state = build_dont_know;
3441                 }
3442                 if (/* !commands_done && */
3443                     (result == build_ok) &&
3444                     !quest &&
3445                     (report_dependencies_level == 0) /*  &&
3446                     (exists(default_target_to_build) > file_doesnt_exist)  */) {
3447                         if (posix) {
3448                                 if (!commands_done) {
3449                                         (void) printf(catgets(catd, 1, 299, "‘%s
3450                                                       default_target_to_build->s
3451                                 } else {
3452                                         if (no_action_was_taken) {
3453                                                 (void) printf(catgets(catd, 1, 3
3454                                                               default_target_to_
3455                                         }
3456                                 }
3457                         } else {
3458                                 if (!commands_done &&
3459                                     (exists(default_target_to_build) > file_does
3460                                         (void) printf(catgets(catd, 1, 301, "‘%s
3461                                                       default_target_to_build->s
3462                                 }
3463                         }
3464                 }
3465         }
3466 }

3468 /*
3469  *      report_recursion(target)
3470  *
3471  *      If this is a recursive make and the parent make has KEEP_STATE on
3472  *      this routine reports the dependency to the parent make
3473  *
3474  *      Parameters:
3475  *              target          Target to report
3476  *
3477  *      Global variables used:
3478  *              makefiles_used          List of makefiles read
3479  *              recursive_name          The Name ".RECURSIVE", printed
3480  *              report_dependency       dwight
3481  */
3482 static void
3483 report_recursion(register Name target)
3484 {
3485         register FILE           *report_file = get_report_file();

3487         if ((report_file == NULL) || (report_file == (FILE*)-1)) {
3488                 return;
3489         }
3490         if (primary_makefile == NULL) {
3491                 /*
3492                  * This can happen when there is no makefile and
3493                  * only implicit rules are being used.
```



```
3494                  */
3495 #ifdef NSE
3496                 nse_no_makefile(target);
3497 #endif
3498                 return;
3499         }
3500         (void) fprintf(report_file,
3501                        "%s: %s ",
3502                        get_target_being_reported_for(),
3503                        recursive_name->string_mb);
3504         report_dependency(get_current_path());
3505         report_dependency(target->string_mb);
3506         report_dependency(primary_makefile->string_mb);
3507         (void) fprintf(report_file, "\n");
3508 }

3510 /* Next function "append_or_replace_macro_in_dyn_array" must be in "misc.cc". */
3511 /* NIKMOL */
3512 extern void
3513 append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar, char *macro)
3514 {
3515         register char   *cp0;   /* work pointer in macro */
3516         register char   *cp1;   /* work pointer in array */
3517         register char   *cp2;   /* work pointer in array */
3518         register char   *cp3;   /* work pointer in array */
3519         register char   *name;  /* macro name */
3520         register char   *value; /* macro value */
3521         register int    len_array;
3522         register int    len_macro;

3524         char * esc_value = NULL;
3525         int esc_len;

3527         if (!(len_macro = strlen(macro))) return;
3528         name = macro;
3529         while (isspace(*(name))) {
3530                 name++;
3531         }
3532         if (!(value = strchr(name, (int) equal_char))) {
3533                 /* no ’=’ in macro */
3534                 goto ERROR_MACRO;
3535         }
3536         cp0 = value;
3537         value++;
3538         while (isspace(*(value))) {
3539                 value++;
3540         }
3541         while (isspace(*(cp0-1))) {
3542                 cp0--;
3543         }
3544         if (cp0 <= name) goto ERROR_MACRO; /* no name */
3545         if (!(Ar->size)) goto ALLOC_ARRAY;
3546         cp1 = Ar->start;

3548 LOOK_FOR_NAME:
3549         if (!(cp1 = strchr(cp1, name[0]))) goto APPEND_MACRO;
3550         if (!(cp2 = strchr(cp1, (int) equal_char))) goto APPEND_MACRO;
3551         if (strncmp(cp1, name, (size_t)(cp0-name))) {
3552                 /* another name */
3553                 cp1++;
3554                 goto LOOK_FOR_NAME;
3555         }
3556         if (cp1 != Ar->start) {
3557                 if (!isspace(*(cp1-1))) {
3558                         /* another name */
3559                         cp1++;
```

```
3560                        goto LOOK_FOR_NAME;
3561                }
3562        }
3563        for (cp3 = cp1 + (cp0-name); cp3 < cp2; cp3++) {
3564                if (isspace(*cp3)) continue;
3565                /* else: another name */
3566                cp1++;
3567                goto LOOK_FOR_NAME;
3568        }
3569        /* Look for the next macro name in array */
3570        cp3 = cp2+1;
3571        if (*cp3 != (int) doublequote_char) {
3572                /* internal error */
3573                goto ERROR_MACRO;
3574        }
3575        if (!(cp3 = strchr(cp3+1, (int) doublequote_char))) {
3576                /* internal error */
3577                goto ERROR_MACRO;
3578        }
3579        cp3++;
3580        while (isspace(*cp3)) {
3581                cp3++;
3582        }
3583
3584        cp2 = cp1;  /* remove old macro */
3585        if ((*cp3) && (cp3 < Ar->start + Ar->size)) {
3586                for (; cp3 < Ar->start + Ar->size; cp3++) {
3587                        *cp2++ = *cp3;
3588                }
3589        }
3590        for (; cp2 < Ar->start + Ar->size; cp2++) {
3591                *cp2 = 0;
3592        }
3593        if (*cp1) {
3594                /* check next name */
3595                goto LOOK_FOR_NAME;
3596        }
3597        goto APPEND_MACRO;

3599 ALLOC_ARRAY:
3600        if (Ar->size) {
3601                cp1 = Ar->start;
3602        } else {
3603                cp1 = 0;
3604        }
3605        Ar->size += 128;
3606        Ar->start = getmem(Ar->size);
3607        for (len_array=0; len_array < Ar->size; len_array++) {
3608                Ar->start[len_array] = 0;
3609        }
3610        if (cp1) {
3611                strcpy(Ar->start, cp1);
3612                retmem((wchar_t *) cp1);
3613        }

3615 APPEND_MACRO:
3616        len_array = strlen(Ar->start);
3617        esc_value = (char*)malloc(strlen(value)*2 + 1);
3618        quote_str(value, esc_value);
3619        esc_len = strlen(esc_value) - strlen(value);
3620        if (len_array + len_macro + esc_len + 5 >= Ar->size) goto  ALLOC_ARRAY;
3621        strcat(Ar->start, " ");
3622        strncat(Ar->start, name, cp0-name);
3623        strcat(Ar->start, "=");
3624        strncat(Ar->start, esc_value, strlen(esc_value));
3625        free(esc_value);
```



```
3626        return;
3627 ERROR_MACRO:
3628        /* Macro without '=' or with invalid left/right part */
3629        return;
3630 }

3632 #ifdef TEAMWARE_MAKE_CMN
3633 /*
3634  * This function, if registered w/ avo_cli_get_license(), will be called
3635  * if the application is about to exit because:
3636  *   1) there has been certain unrecoverable error(s) that cause the
3637  *      application to exit immediately.
3638  *   2) the user has lost a license while the application is running.
3639  */
3640 extern "C" void
3641 dmake_exit_callback(void)
3642 {
3643        fatal(catgets(catd, 1, 306, "can not get a license, exiting..."));
3644        exit(1);
3645 }

3647 /*
3648  * This function, if registered w/ avo_cli_get_license(), will be called
3649  * if the application can not get a license.
3650  */
3651 extern "C" void
3652 dmake_message_callback(char *err_msg)
3653 {
3654        static Boolean  first = true;

3656        if (!first) {
3657                return;
3658        }
3659        first = false;
3660        if ((!list_all_targets) &&
3661            (report_dependencies_level == 0) &&
3662            (dmake_mode_type != serial_mode)) {
3663                warning(catgets(catd, 1, 313, "can not get a TeamWare license, d
3664        }
3665 }
3666 #endif

3668 #ifdef DISTRIBUTED
3669 /*
3670  * Returns whether -c is set or not.
3671  */
3672 Boolean
3673 get_dmake_rcfile_specified(void)
3674 {
3675        return(dmake_rcfile_specified);
3676 }

3678 /*
3679  * Returns whether -g is set or not.
3680  */
3681 Boolean
3682 get_dmake_group_specified(void)
3683 {
3684        return(dmake_group_specified);
3685 }

3687 /*
3688  * Returns whether -j is set or not.
3689  */
3690 Boolean
3691 get_dmake_max_jobs_specified(void)
```

```
{
        return(dmake_max_jobs_specified);
}

/*
 * Returns whether -m is set or not.
 */
Boolean
get_dmake_mode_specified(void)
{
        return(dmake_mode_specified);
}

/*
 * Returns whether -o is set or not.
 */
Boolean
get_dmake_odir_specified(void)
{
        return(dmake_odir_specified);
}

#endif

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

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

        if(report_cwd) {
                if(make_level_val <= 0) {
                        if(entering) {
#ifdef TEAMWARE_MAKE_CMN
                                sprintf( rcwd
                                       , catgets(catd, 1, 329, "dmake: Entering
                                       , get_current_path());
#else
                                sprintf( rcwd
                                       , catgets(catd, 1, 330, "make: Entering d
                                       , get_current_path());
#endif
                        } else {
#ifdef TEAMWARE_MAKE_CMN
                                sprintf( rcwd
                                       , catgets(catd, 1, 331, "dmake: Leaving d
                                       , get_current_path());
#else
                                sprintf( rcwd
```



```
                                       , catgets(catd, 1, 332, "make: Leaving di
                                       , get_current_path());
#endif
                        }
                } else {
                        if(entering) {
#ifdef TEAMWARE_MAKE_CMN
                                sprintf( rcwd
                                       , catgets(catd, 1, 333, "dmake[%d]: Enter
                                       , make_level_val, get_current_path());
#else
                                sprintf( rcwd
                                       , catgets(catd, 1, 334, "make[%d]: Enteri
                                       , make_level_val, get_current_path());
#endif
                        } else {
#ifdef TEAMWARE_MAKE_CMN
                                sprintf( rcwd
                                       , catgets(catd, 1, 335, "dmake[%d]: Leavi
                                       , make_level_val, get_current_path());
#else
                                sprintf( rcwd
                                       , catgets(catd, 1, 336, "make[%d]: Leavin
                                       , make_level_val, get_current_path());
#endif
                        }
                }
                printf(NOCATGETS("%s"), rcwd);
        }
}
#endif /* ! codereview */
```

```
**********************************************************
    1002 Wed May 20 11:04:09 2015
new/usr/src/cmd/make/bin/make/common/make.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)make.cc 1.2 06/12/12
 */

#pragma ident   "@(#)make.cc    1.2     06/12/12"
#endif /* ! codereview */
```

```
**********************************************************
   10257 Wed May 20 11:04:09 2015
new/usr/src/cmd/make/bin/make/common/make.rules.file
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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.
#
# @(#)make.rules.file 1.22 06/12/12
#

SUFFIXES = .o .c .c~ .cc .cc~ .y .y~ .l .l~ .s .s~ .sh .sh~ .S .S~ .i .ln \
        .h .h~ .f .f~ .for .for~ .F .F~ .f90 .f90~ .ftn .ftn~ .mod .mod~ \
        .sym .def .def~ .p .p~ .r .r~ .cps .cps~ .C .C~ .Y .Y~ .L .L~ \
        .java .java~ .class

.SUFFIXES: $(SUFFIXES)

# OUTPUT_OPTION should be defined to "-o $@" when
# the default rules are used for non-local files.
OUTPUT_OPTION=

#       C language section.
CC=cc
CFLAGS=
CPPFLAGS=
LINT=lint
LINTFLAGS=
COMPILE.c=$(CC) $(CFLAGS) $(CPPFLAGS) -c
LINK.c=$(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS)
LINT.c=$(LINT) $(LINTFLAGS) $(CPPFLAGS)
.c:
        $(LINK.c) -o $@ $< $(LDLIBS)
.c~:
        $(GET) $(GFLAGS) -p $< > $*.c
        $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c
.c.o:
        $(COMPILE.c) $(OUTPUT_OPTION) $<
.c~.o:
        $(GET) $(GFLAGS) -p $< > $*.c
        $(CC) $(CFLAGS) -c  $*.c
.c.i:
        $(CC) $(CFLAGS) $(CPPFLAGS) -P $<
.c~.i:
        $(GET) $(GFLAGS) -p $< > $*.c
        $(CC) $(CFLAGS) $(CPPFLAGS) -P $*.c
```



```
.c.ln:
        $(LINT.c) $(OUTPUT_OPTION) -c $<
.c~.ln:
        $(GET) $(GFLAGS) -p $< > $*.c
        $(LINT.c) $(OUTPUT_OPTION) -c $*.c
.c.a:
        $(COMPILE.c) -o $% $<
        $(AR) $(ARFLAGS) $@ $%
        $(RM) $%
.c~.a:
        $(GET) $(GFLAGS) -p $< > $*.c
        $(COMPILE.c) -o $% $*.c
        $(AR) $(ARFLAGS) $@ $%
        $(RM) $%

#       C language section. yacc.
YACC=yacc
YFLAGS=
YACC.y=$(YACC) $(YFLAGS)
.y:
        $(YACC.y) $<
        $(LINK.c) -o $@ y.tab.c $(LDLIBS)
        $(RM) y.tab.c
.y~:
        $(GET) $(GFLAGS) -p $< > $*.y
        $(YACC) $(YFLAGS) $*.y
        $(COMPILE.c) -o $@ y.tab.c
        $(RM) y.tab.c

.y.c:
        $(YACC.y) $<
        mv y.tab.c $@
.y~.c:
        $(GET) $(GFLAGS) -p $< > $*.y
        $(YACC) $(YFLAGS) $*.y
        mv y.tab.c $@
.y.ln:
        $(YACC.y) $<
        $(LINT.c) -o $@ -i y.tab.c
        $(RM) y.tab.c
.y~.ln:
        $(GET) $(GFLAGS) -p $< > $*.y
        $(YACC.y) $*.y
        $(LINT.c) -o $@ -i y.tab.c
        $(RM) y.tab.c
.y.o:
        $(YACC.y) $<
        $(COMPILE.c) -o $@ y.tab.c
        $(RM) y.tab.c
.y~.o:
        $(GET) $(GFLAGS) -p $< > $*.y
        $(YACC) $(YFLAGS) $*.y
        $(CC) $(CFLAGS) -c y.tab.c
        rm -f y.tab.c
        mv y.tab.o $@

#       C language section. lex.
LEX=lex
LFLAGS=
LEX.l=$(LEX) $(LFLAGS) -t
.l:
        $(RM) $*.c
        $(LEX.l) $< > $*.c
        $(LINK.c) -o $@ $*.c -ll $(LDLIBS)
        $(RM) $*.c
.l~:
```

```
128         $(GET) $(GFLAGS) -p $< > $*.l
129         $(LEX) $(LFLAGS) $*.l
130         $(CC) $(CFLAGS) -c lex.yy.c
131         rm -f lex.yy.c
132         mv lex.yy.c $@

133 .l.c :
134         $(RM) $@
135         $(LEX.l) $< > $@
136 .l~.c:
137         $(GET) $(GFLAGS) -p $< > $*.l
138         $(LEX) $(LFLAGS) $*.l
139         mv lex.yy.c $@
140 .l.ln:
141         $(RM) $*.c
142         $(LEX.l) $< > $*.c
143         $(LINT.c) -o $@ -i $*.c
144         $(RM) $*.c
145 .l~.ln:
146         $(GET) $(GFLAGS) -p $< > $*.l
147         $(RM) $*.c
148         $(LEX.l) $*.l > $*.c
149         $(LINT.c) -o $@ -i $*.c
150         $(RM) $*.c
151 .l.o:
152         $(RM) $*.c
153         $(LEX.l) $< > $*.c
154         $(COMPILE.c) -o $@ $*.c
155         $(RM) $*.c
156 .l~.o:
157         $(GET) $(GFLAGS) -p $< > $*.l
158         $(LEX) $(LFLAGS) $*.l
159         $(CC) $(CFLAGS) -c lex.yy.c
160         rm -f lex.yy.c
161         mv lex.yy.c $@

162 #       C++ language section.
163 CCC=CC
164 CCFLAGS=
165 COMPILE.cc=$(CCC) $(CCFLAGS) $(CPPFLAGS)  -c
166 LINK.cc=$(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS)
167 COMPILE.C=$(CCC) $(CCFLAGS) $(CPPFLAGS)  -c
168 LINK.C=$(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS)
169 .cc:
170         $(LINK.cc) -o $@ $< $(LDLIBS)
171 .cc~:
172         $(GET) $(GFLAGS) -p $< > $*.cc
173         $(LINK.cc) -o $@ $*.cc $(LDLIBS)
174 .cc.o:
175         $(COMPILE.cc) $(OUTPUT_OPTION) $<
176 .cc~.o:
177         $(GET) $(GFLAGS) -p $< > $*.cc
178         $(COMPILE.cc) $(OUTPUT_OPTION) $*.cc
179 .cc.i:
180         $(CCC) $(CCFLAGS) $(CPPFLAGS) -P $<
181 .cc~.i:
182         $(GET) $(GFLAGS) -p $< > $*.cc
183         $(CCC) $(CCFLAGS) $(CPPFLAGS) -P $*.cc
184 .cc.a:
185         $(COMPILE.cc) -o $% $<
186         $(AR) $(ARFLAGS) $@ $%
187         $(RM) $%
188 .cc~.a:
189         $(GET) $(GFLAGS) -p $< > $*.cc
190         $(COMPILE.cc) -o $% $*.cc
191         $(AR) $(ARFLAGS) $@ $%
192         $(RM) $%

193 .C:
194         $(LINK.C) -o $@ $< $(LDLIBS)
195 .C~:
196         $(GET) $(GFLAGS) -p $< > $*.C
197         $(LINK.C) -o $@ $*.C $(LDLIBS)
198 .C.o:
199         $(COMPILE.C) $(OUTPUT_OPTION) $<
200 .C~.o:
201         $(GET) $(GFLAGS) -p $< > $*.C
202         $(COMPILE.C) $(OUTPUT_OPTION) $*.C
203 .C.i:
204         $(CCC) $(CCFLAGS) $(CPPFLAGS) -P $<
205 .C~.i:
206         $(GET) $(GFLAGS) -p $< > $*.C
207         $(CCC) $(CCFLAGS) $(CPPFLAGS) -P  $*.C
208 .C.a:
209         $(COMPILE.C) -o $% $<
210         $(AR) $(ARFLAGS) $@ $%
211         $(RM) $%
212 .C~.a:
213         $(GET) $(GFLAGS) -p $< > $*.C
214         $(COMPILE.C) -o $% $*.C
215         $(AR) $(ARFLAGS) $@ $%
216         $(RM) $%

217 #       FORTRAN section.
218 FC=f77
219 FFLAGS=
220 COMPILE.f=$(FC) $(FFLAGS)  -c
221 LINK.f=$(FC) $(FFLAGS) $(LDFLAGS)
222 COMPILE.F=$(FC) $(FFLAGS) $(CPPFLAGS)  -c
223 LINK.F=$(FC) $(FFLAGS) $(CPPFLAGS) $(LDFLAGS)
224 .f:
225         $(LINK.f) -o $@ $< $(LDLIBS)
226 .f~:
227         $(GET) $(GFLAGS) -p $< > $*.f
228         $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.f
229 .f.o:
230         $(COMPILE.f) $(OUTPUT_OPTION) $<
231 .f~.o:
232         $(GET) $(GFLAGS) -p $< > $*.f
233         $(FC) $(FFLAGS) -c  $*.f
234 .f.a:
235         $(COMPILE.f) -o $% $<
236         $(AR) $(ARFLAGS) $@ $%
237         $(RM) $%
238 .f~.a:
239         $(GET) $(GFLAGS) -p $< > $*.f
240         $(COMPILE.f) -o $% $*.f
241         $(AR) $(ARFLAGS) $@ $%
242         $(RM) $%
243 .for:
244         $(LINK.f) -o $@ $< $(LDLIBS)
245 .for~:
246         $(GET) $(GFLAGS) -p $< > $*.for
247         $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.for
248 .for.o:
249         $(COMPILE.f) $(OUTPUT_OPTION) $<
250 .for~.o:
251         $(GET) $(GFLAGS) -p $< > $*.for
252         $(FC) $(FFLAGS) -c  $*.for
253 .for.a:
254         $(COMPILE.f) -o $% $<
255         $(AR) $(ARFLAGS) $@ $%
```

```
260         $(RM) $%
261 .for~.a:
262         $(GET) $(GFLAGS) -p $< > $*.for
263         $(COMPILE.f) -o $% $*.for
264         $(AR) $(ARFLAGS) $@ $%
265         $(RM) $%
266 .F:
267         $(LINK.F) -o $@ $< $(LDLIBS)
268 .F~:
269         $(GET) $(GFLAGS) -p $< > $*.F
270         $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.F
271 .F.o:
272         $(COMPILE.F) $(OUTPUT_OPTION) $<
273 .F~.o:
274         $(GET) $(GFLAGS) -p $< > $*.F
275         $(FC) $(FFLAGS) -c  $*.F
276 .F.a:
277         $(COMPILE.F) -o $% $<
278         $(AR) $(ARFLAGS) $@ $%
279         $(RM) $%
280 .F~.a:
281         $(GET) $(GFLAGS) -p $< > $*.F
282         $(COMPILE.F) -o $% $*.F
283         $(AR) $(ARFLAGS) $@ $%
284         $(RM) $%

286 #       FORTRAN section. ratfor.
287 RFLAGS=
288 COMPILE.r=$(FC) $(FFLAGS) $(RFLAGS)  -c
289 LINK.r=$(FC) $(FFLAGS) $(RFLAGS) $(LDFLAGS)
290 .r:
291         $(LINK.r) -o $@ $< $(LDLIBS)
292 .r~:
293         $(GET) $(GFLAGS) -p $< > $*.r
294         $(LINK.r) -o $@ $*.r $(LDLIBS)
295 .r.o:
296         $(COMPILE.r) $(OUTPUT_OPTION) $<
297 .r~.o:
298         $(GET) $(GFLAGS) -p $< > $*.r
299         $(COMPILE.r) $(OUTPUT_OPTION) $*.r
300 .r.a:
301         $(COMPILE.r) -o $% $<
302         $(AR) $(ARFLAGS) $@ $%
303         $(RM) $%
304 .r~.a:
305         $(GET) $(GFLAGS) -p $< > $*.r
306         $(COMPILE.r) -o $% $*.r
307         $(AR) $(ARFLAGS) $@ $%
308         $(RM) $%

310 #       FORTRAN 90 section.
311 F90C=f90
312 F90FLAGS=
313 COMPILE.f90=$(F90C) $(F90FLAGS) -c
314 LINK.f90=$(F90C) $(F90FLAGS) $(LDFLAGS)
315 COMPILE.ftn=$(F90C) $(F90FLAGS) -c
316 LINK.ftn=$(F90C) $(F90FLAGS) $(LDFLAGS)
317 .f90:
318         $(LINK.f90) -o $@ $< $(LDLIBS)
319 .f90~:
320         $(GET) $(GFLAGS) -p $< > $*.f90
321         $(LINK.f90) -o $@ $*.f90 $(LDLIBS)
322 .f90.o:
323         $(COMPILE.f90) $(OUTPUT_OPTION) $<
324 .f90~.o:
325         $(GET) $(GFLAGS) -p $< > $*.f90
```



```
326         $(COMPILE.f90) $(OUTPUT_OPTION) $*.f90
327 .f90.a:
328         $(COMPILE.f90) -o $% $<
329         $(AR) $(ARFLAGS) $@ $%
330         $(RM) $%
331 .f90~.a:
332         $(GET) $(GFLAGS) -p $< > $*.f90
333         $(COMPILE.f90) -o $% $*.f90
334         $(AR) $(ARFLAGS) $@ $%
335         $(RM) $%
336 .ftn:
337         $(LINK.ftn) -o $@ $< $(LDLIBS)
338 .ftn~:
339         $(GET) $(GFLAGS) -p $< > $*.ftn
340         $(LINK.ftn) -o $@ $*.ftn $(LDLIBS)
341 .ftn.o:
342         $(COMPILE.ftn) $(OUTPUT_OPTION) $<
343 .ftn~.o:
344         $(GET) $(GFLAGS) -p $< > $*.ftn
345         $(COMPILE.ftn) $(OUTPUT_OPTION) $*.ftn
346 .ftn.a:
347         $(COMPILE.ftn) -o $% $<
348         $(AR) $(ARFLAGS) $@ $%
349         $(RM) $%
350 .ftn~.a:
351         $(GET) $(GFLAGS) -p $< > $*.ftn
352         $(COMPILE.ftn) -o $% $*.ftn
353         $(AR) $(ARFLAGS) $@ $%
354         $(RM) $%

356 #       Modula-2 section.
357 M2C=m2c
358 M2FLAGS=
359 MODFLAGS=
360 DEFFLAGS=
361 COMPILE.def=$(M2C) $(M2FLAGS) $(DEFFLAGS)
362 COMPILE.mod=$(M2C) $(M2FLAGS) $(MODFLAGS)
363 .def.sym:
364         $(COMPILE.def) -o $@ $<
365 .def~.sym:
366         $(GET) $(GFLAGS) -p $< > $*.def
367         $(COMPILE.def) -o $@ $*.def
368 .mod:
369         $(COMPILE.mod) -o $@ -e $@ $<
370 .mod~:
371         $(GET) $(GFLAGS) -p $< > $*.mod
372         $(COMPILE.mod) -o $@ -e $@ $*.mod
373 .mod.o:
374         $(COMPILE.mod) -o $@ $<
375 .mod~.o:
376         $(GET) $(GFLAGS) -p $< > $*.mod
377         $(COMPILE.mod) -o $@ $*.mod
378 .mod.a:
379         $(COMPILE.mod) -o $% $<
380         $(AR) $(ARFLAGS) $@ $%
381         $(RM) $%
382 .mod~.a:
383         $(GET) $(GFLAGS) -p $< > $*.mod
384         $(COMPILE.mod) -o $% $*.mod
385         $(AR) $(ARFLAGS) $@ $%
386         $(RM) $%

388 #       Pascal section.
389 PC=pc
390 PFLAGS=
391 COMPILE.p=$(PC) $(PFLAGS) $(CPPFLAGS)  -c
```

```
392 LINK.p=$(PC) $(PFLAGS) $(CPPFLAGS) $(LDFLAGS)
393 .p:
394         $(LINK.p) -o $@ $< $(LDLIBS)
395 .p~:
396         $(GET) $(GFLAGS) -p $< > $*.p
397         $(LINK.p) -o $@ $*.p $(LDLIBS)
398 .p.o:
399         $(COMPILE.p) $(OUTPUT_OPTION) $<
400 .p~.o:
401         $(GET) $(GFLAGS) -p $< > $*.p
402         $(COMPILE.p) $(OUTPUT_OPTION) $*.p
403 .p.a:
404         $(COMPILE.p) -o $% $<
405         $(AR) $(ARFLAGS) $@ $%
406         $(RM) $%
407 .p~.a:
408         $(GET) $(GFLAGS) -p $< > $*.p
409         $(COMPILE.p) -o $% $*.p
410         $(AR) $(ARFLAGS) $@ $%
411         $(RM) $%

413 #       Assembly section.
414 AS=as
415 ASFLAGS=
416 COMPILE.s=$(AS) $(ASFLAGS)
417 COMPILE.S=$(CC) $(ASFLAGS) $(CPPFLAGS)  -c
418 .s.o:
419         $(COMPILE.s) -o $@ $<
420 .s~.o:
421         $(GET) $(GFLAGS) -p $< > $*.s
422         $(COMPILE.s) -o $@ $*.s
423 .s.a:
424         $(COMPILE.s) -o $% $<
425         $(AR) $(ARFLAGS) $@ $%
426         $(RM) $%
427 .s~.a:
428         $(GET) $(GFLAGS) -p $< > $*.s
429         $(COMPILE.s) -o $% $*.s
430         $(AR) $(ARFLAGS) $@ $%
431         $(RM) $%
432 .S.o:
433         $(COMPILE.S) -o $@ $<
434 .S~.o:
435         $(GET) $(GFLAGS) -p $< > $*.S
436         $(COMPILE.S) -o $@ $*.S
437 .S.a:
438         $(COMPILE.S) -o $% $<
439         $(AR) $(ARFLAGS) $@ $%
440         $(RM) $%
441 .S~.a:
442         $(GET) $(GFLAGS) -p $< > $*.S
443         $(COMPILE.S) -o $% $*.S
444         $(AR) $(ARFLAGS) $@ $%
445         $(RM) $%

447 #       Shell section.
448 .sh:
449         $(RM) $@
450         cat $< > $@
451         chmod +x $@
452 .sh~:
453         $(GET) $(GFLAGS) -p $< > $*.sh
454         cp $*.sh $@
455         chmod a+x $@

457 #       NeWS section
```



```
458 CPS=cps
459 CPSFLAGS=
460 .cps.h:
461         $(CPS) $(CPSFLAGS) $*.cps
462 .cps~.h:
463         $(GET) $(GFLAGS) -p $< > $*.cps
464         $(CPS) $(CPSFLAGS) $*.cps

466 #       JAVA section
467 JAVAC=javac
468 JAVACFLAGS=
469 .java.class:
470         $(JAVAC) $(JAVACFLAGS) $<
471 .java~.class:
472         $(GET) $(GFLAGS) -p $< > $*.java
473         $(JAVAC) $(JAVACFLAGS) $<

475 #       Miscellaneous section.
476 LD=ld
477 LDFLAGS=
478 LDLIBS=
479 MAKE=make
480 RM=rm -f
481 AR=ar
482 ARFLAGS=rv
483 GET=get
484 GFLAGS=

486 markfile.o:     markfile
487         echo "static char _sccsid[] = \"`grep @(#) markfile`\";" > markfile.c
488         cc -c markfile.c
489         $(RM) markfile.c

491 SCCSFLAGS=
492 SCCSGETFLAGS=-s
493 .SCCS_GET:
494         sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $@ -G$@

496 .SCCS_GET_POSIX:
497         sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $@

499 .GET_POSIX:
500         $(GET) $(GFLAGS) s.$@
501 #endif /* ! codereview */
```

```
**********************************************************
   26968 Wed May 20 11:04:09 2015
new/usr/src/cmd/make/bin/make/common/misc.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)misc.cc 1.50 06/12/12
 */

#pragma ident   "@(#)misc.cc    1.34    95/10/04"

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

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

#if defined(HP_UX) || defined(linux)
#include <unistd.h>
#endif

#ifdef TEAMWARE_MAKE_CMN
#define MAXJOBS_ADJUST_RFE4694000

#ifdef MAXJOBS_ADJUST_RFE4694000
```



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

#if defined(linux)
#include <time.h>               /* localtime() */
#endif

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */

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

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

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

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

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

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

```
 */

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

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

        va_start(args, message);
        (void) fflush(stdout);
#ifdef DISTRIBUTED
        (void) fprintf(stderr, catgets(catd, 1, 262, "dmake: Fatal error: "));
#else
        (void) fprintf(stderr, catgets(catd, 1, 263, "make: Fatal error: "));
#endif
        (void) vfprintf(stderr, message, args);
        (void) fprintf(stderr, "\n");
        va_end(args);
        if (report_pwd) {
                (void) fprintf(stderr,
                                catgets(catd, 1, 156, "Current working directory
                                get_current_path());
        }
        (void) fflush(stderr);
        if (fatal_in_progress) {
#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
                exit_status = 1;
#endif
                exit(1);
        }
        fatal_in_progress = true;
#ifdef TEAMWARE_MAKE_CMN
        /* Let all parallel children finish */
        if ((dmake_mode_type == parallel_mode) &&
            (parallel_process_cnt > 0)) {
                (void) fprintf(stderr,
                                catgets(catd, 1, 157, "Waiting for %d %s to finis
                                parallel_process_cnt,
                                parallel_process_cnt == 1 ?
                                catgets(catd, 1, 158, "job") : catgets(catd, 1, 1
                (void) fflush(stderr);
        }

        while (parallel_process_cnt > 0) {
#ifdef DISTRIBUTED
                if (dmake_mode_type == distributed_mode) {
                        (void) await_dist(false);
```



```
                } else {
                        await_parallel(true);
                }
#else
                await_parallel(true);
#endif
                finish_children(false);
        }
#endif

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

#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
        exit_status = 1;
#endif
        exit(1);
}

/*
 *      warning(format, args...)
 *
 *      Print a message and continue.
 *
 *      Parameters:
 *              format          printf type format string
 *              args            Arguments to match the format
 *
 *      Global variables used:
 *              report_pwd      Should we report the current path?
 */
/*VARARGS*/
void
warning(char * message, ...)
{
        va_list args;

        va_start(args, message);
        (void) fflush(stdout);
#ifdef DISTRIBUTED
        (void) fprintf(stderr, catgets(catd, 1, 264, "dmake: Warning: "));
#else
        (void) fprintf(stderr, catgets(catd, 1, 265, "make: Warning: "));
#endif
        (void) vfprintf(stderr, message, args);
        (void) fprintf(stderr, "\n");
        va_end(args);
        if (report_pwd) {
                (void) fprintf(stderr,
                                catgets(catd, 1, 161, "Current working directory
                                get_current_path());
        }
        (void) fflush(stderr);
}

/*
 *      time_to_string(time)
 *
 *      Take a numeric time value and produce
 *      a proper string representation.
 *
 *      Return value:
 *                              The string representation of the time
 *
 *      Parameters:
```

```
 *              time            The time we need to translate
 *
 *      Global variables used:
 */
char *
time_to_string(const timestruc_t &time)
{
        struct tm               *tm;
        char                    buf[128];

        if (time == file_doesnt_exist) {
                return catgets(catd, 1, 163, "File does not exist");
        }
        if (time == file_max_time) {
                return catgets(catd, 1, 164, "Younger than any file");
        }
        tm = localtime(&time.tv_sec);
        strftime(buf, sizeof (buf), NOCATGETS("%c %Z"), tm);
        buf[127] = (int) nul_char;
        return strdup(buf);
}

/*
 *      get_current_path()
 *
 *      Stuff current_path with the current path if it isnt there already.
 *
 *      Parameters:
 *
 *      Global variables used:
 */
char *
get_current_path(void)
{
        char                    pwd[(MAXPATHLEN * MB_LEN_MAX)];
        static char             *current_path;

        if (current_path == NULL) {
#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
                getcwd(pwd, sizeof(pwd));
#else
                (void) getwd(pwd);
#endif
                if (pwd[0] == (int) nul_char) {
                        pwd[0] = (int) slash_char;
                        pwd[1] = (int) nul_char;
#ifdef DISTRIBUTED
                        current_path = strdup(pwd);
                } else if (IS_EQUALN(pwd, NOCATGETS("/tmp_mnt"), 8)) {
                        current_path = strdup(pwd + 8);
                } else {
                        current_path = strdup(pwd);
                }
#else
                }
                current_path = strdup(pwd);
#endif
        }
        return current_path;
}

/*************************************
 *
 *      Make internal state dumping
 *
 *      This is a set  of routines for dumping the internal make state
```



```
 *      Used for the -p option
 */

/*
 *      dump_make_state()
 *
 *      Dump make's internal state to stdout
 *
 *      Parameters:
 *
 *      Global variables used:
 *              svr4                    Was ".SVR4" seen in makefile?
 *              svr4_name               The Name ".SVR4", printed
 *              posix                   Was ".POSIX" seen in makefile?
 *              posix_name              The Name ".POSIX", printed
 *              default_rule            Points to the .DEFAULT rule
 *              default_rule_name       The Name ".DEFAULT", printed
 *              default_target_to_build The first target to print
 *              dot_keep_state          The Name ".KEEP_STATE", printed
 *              dot_keep_state_file     The Name ".KEEP_STATE_FILE", printed
 *              hashtab                 The make hash table for Name blocks
 *              ignore_errors           Was ".IGNORE" seen in makefile?
 *              ignore_name             The Name ".IGNORE", printed
 *              keep_state              Was ".KEEP_STATE" seen in makefile?
 *              percent_list            The list of % rules
 *              precious                The Name ".PRECIOUS", printed
 *              sccs_get_name           The Name ".SCCS_GET", printed
 *              sccs_get_posix_name     The Name ".SCCS_GET_POSIX", printed
 *              get_name                The Name ".GET", printed
 *              get_posix_name          The Name ".GET_POSIX", printed
 *              sccs_get_rule           Points to the ".SCCS_GET" rule
 *              silent                  Was ".SILENT" seen in makefile?
 *              silent_name             The Name ".SILENT", printed
 *              suffixes                The suffix list from ".SUFFIXES"
 *              suffixes_name           The Name ".SUFFIX", printed
 */
void
dump_make_state(void)
{
        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);
        }
        (void) printf("\n");

        /* .POSIX */
        if (posix) {
                (void) printf("%s:\n", posix_name->string_mb);
        }

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

        /* .IGNORE */
        if (ignore_errors) {
```

```
392                 (void) printf("%s:\n", ignore_name->string_mb);
393         }
395         /* .KEEP_STATE: */
396         if (keep_state) {
397                 (void) printf("%s:\n\n", dot_keep_state->string_mb);
398         }
400         /* .PRECIOUS */
401         (void) printf("%s:", precious->string_mb);
402         for (p = hashtab.begin(), e = hashtab.end(); p != e; p++) {
403                         if ((p->stat.is_precious) || (all_precious)) {
404                                 (void) printf(" %s", p->string_mb);
405                         }
406         }
407         (void) printf("\n");
409         /* .SCCS_GET */
410         if (sccs_get_rule != NULL) {
411                 (void) printf("%s:\n", sccs_get_name->string_mb);
412                 for (rule = sccs_get_rule; rule != NULL; rule = rule->next) {
413                         (void) printf("\t%s\n", rule->command_line->string_mb);
414                 }
415         }
417         /* .SILENT */
418         if (silent) {
419                 (void) printf("%s:\n", silent_name->string_mb);
420         }
422         /* .SUFFIXES: */
423         (void) printf("%s:", suffixes_name->string_mb);
424         for (dep = suffixes; dep != NULL; dep = dep->next) {
425                 (void) printf(" %s", dep->name->string_mb);
426                 build_suffix_list(dep->name);
427         }
428         (void) printf("\n\n");
430         /* % rules */
431         for (percent = percent_list;
432              percent != NULL;
433              percent = percent->next) {
434                 (void) printf("%s:",
435                               percent->name->string_mb);
436
437                 for (percent_depe = percent->dependencies;
438                      percent_depe != NULL;
439                      percent_depe = percent_depe->next) {
440                         (void) printf(" %s", percent_depe->name->string_mb);
441                 }
442
443                 (void) printf("\n");
445                 for (rule = percent->command_template;
446                      rule != NULL;
447                      rule = rule->next) {
448                         (void) printf("\t%s\n", rule->command_line->string_mb);
449                 }
450         }
452         /* Suffix rules */
453         for (p = hashtab.begin(), e = hashtab.end(); p != e; p++) {
454                         Wstring wcb(p);
455                         if (wcb.get_string()[0] == (int) period_char) {
456                                 print_rule(p);
457                         }
```



```
458         }
460         /* Macro assignments */
461         for (p = hashtab.begin(), e = hashtab.end(); p != e; p++) {
462                         if (((prop = get_prop(p->prop, macro_prop)) != NULL) &&
463                             (prop->body.macro.value != NULL)) {
464                                 (void) printf("%s", p->string_mb);
465                                 print_value(prop->body.macro.value,
466                                             (Daemon) prop->body.macro.daemon);
467                         }
468         }
469         (void) printf("\n");
471         /* Conditional macro assignments */
472         for (p = hashtab.begin(), e = hashtab.end(); p != e; p++) {
473                         for (prop = get_prop(p->prop, conditional_prop);
474                              prop != NULL;
475                              prop = get_prop(prop->next, conditional_prop)) {
476                                 (void) printf("%s := %s",
477                                               p->string_mb,
478                                               prop->body.conditional.name->
479                                               string_mb);
480                                 if (prop->body.conditional.append) {
481                                         printf(" +");
482                                 }
483                                 else {
484                                         printf(" ");
485                                 }
486                                 print_value(prop->body.conditional.value,
487                                             no_daemon);
488                         }
489         }
490         (void) printf("\n");
492         /* All other dependencies */
493         for (p = hashtab.begin(), e = hashtab.end(); p != e; p++) {
494                         if (p->colons != no_colon) {
495                                 print_rule(p);
496                         }
497         }
498         (void) printf("\n");
499 }
501 /*
502  *      print_rule(target)
503  *
504  *      Print the rule for one target
505  *
506  *      Parameters:
507  *              target          Target we print rule for
508  *
509  *      Global variables used:
510  */
511 static void
512 print_rule(register Name target)
513 {
514         register Cmd_line       rule;
515         register Property       line;
516         register Dependency     dependency;
518         if (target->dependency_printed ||
519             ((line = get_prop(target->prop, line_prop)) == NULL) ||
520             ((line->body.line.command_template == NULL) &&
521              (line->body.line.dependencies == NULL))) {
522                 return;
523         }
```

```
524         target->dependency_printed = true;

526         (void) printf("%s:", target->string_mb);

528         for (dependency = line->body.line.dependencies;
529              dependency != NULL;
530              dependency = dependency->next) {
531                 (void) printf(" %s", dependency->name->string_mb);
532         }

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

536         for (rule = line->body.line.command_template;
537              rule != NULL;
538              rule = rule->next) {
539                 (void) printf("\t%s\n", rule->command_line->string_mb);
540         }
541 }

543 void
544 dump_target_list(void)
545 {
546         Name_set::iterator      p, e;
547         Wstring str;

549         for (p = hashtab.begin(), e = hashtab.end(); p != e; p++) {
550                         str.init(p);
551                         wchar_t * wcb = str.get_string();
552                         if ((p->colons != no_colon) &&
553                             ((wcb[0] != (int) period_char) ||
554                              ((wcb[0] == (int) period_char) &&
555                               (wschr(wcb, (int) slash_char)))) {
556                                 print_target_n_deps(p);
557                         }
558         }
559 }

561 static void
562 print_target_n_deps(register Name target)
563 {
564         register Cmd_line       rule;
565         register Property       line;
566         register Dependency     dependency;

568         if (target->dependency_printed) {
569                 return;
570         }
571         target->dependency_printed = true;

573         (void) printf("%s\n", target->string_mb);

575         if ((line = get_prop(target->prop, line_prop)) == NULL) {
576                 return;
577         }
578         for (dependency = line->body.line.dependencies;
579              dependency != NULL;
580              dependency = dependency->next) {
581                 if (!dependency->automatic) {
582                         print_target_n_deps(dependency->name);
583                 }
584         }
585 }

587 /*****************************************
588  *
589  *      main() support
```



```
590  */

592 /*
593  *      load_cached_names()
594  *
595  *      Load the vector of cached names
596  *
597  *      Parameters:
598  *
599  *      Global variables used:
600  *              Many many pointers to Name blocks.
601  */
602 void
603 load_cached_names(void)
604 {
605         char            *cp;
606         Name            dollar;

608         /* Load the cached_names struct */
609         MBSTOWCS(wcs_buffer, NOCATGETS(".BUILT_LAST_MAKE_RUN"));
610         built_last_make_run = GETNAME(wcs_buffer, FIND_LENGTH);
611         MBSTOWCS(wcs_buffer, NOCATGETS("@"));
612         c_at = GETNAME(wcs_buffer, FIND_LENGTH);
613         MBSTOWCS(wcs_buffer, NOCATGETS(" *conditionals* "));
614         conditionals = GETNAME(wcs_buffer, FIND_LENGTH);
615         /*
616          * A version of make was released with NSE 1.0 that used
617          * VERSION-1.1 but this version is identical to VERSION-1.0.
618          * The version mismatch code makes a special case for this
619          * situation.  If the version number is changed from 1.0
620          * it should go to 1.2.
621          */
622         MBSTOWCS(wcs_buffer, NOCATGETS("VERSION-1.0"));
623         current_make_version = GETNAME(wcs_buffer, FIND_LENGTH);
624         MBSTOWCS(wcs_buffer, NOCATGETS(".SVR4"));
625         svr4_name = GETNAME(wcs_buffer, FIND_LENGTH);
626         MBSTOWCS(wcs_buffer, NOCATGETS(".POSIX"));
627         posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
628         MBSTOWCS(wcs_buffer, NOCATGETS(".DEFAULT"));
629         default_rule_name = GETNAME(wcs_buffer, FIND_LENGTH);
630 #ifdef NSE
631         MBSTOWCS(wcs_buffer, NOCATGETS(".DERIVED_SRC"));
632         derived_src= GETNAME(wcs_buffer, FIND_LENGTH);
633 #endif
634         MBSTOWCS(wcs_buffer, NOCATGETS("$"));
635         dollar = GETNAME(wcs_buffer, FIND_LENGTH);
636         MBSTOWCS(wcs_buffer, NOCATGETS(".DONE"));
637         done = GETNAME(wcs_buffer, FIND_LENGTH);
638         MBSTOWCS(wcs_buffer, NOCATGETS("."));
639         dot = GETNAME(wcs_buffer, FIND_LENGTH);
640         MBSTOWCS(wcs_buffer, NOCATGETS(".KEEP_STATE"));
641         dot_keep_state = GETNAME(wcs_buffer, FIND_LENGTH);
642         MBSTOWCS(wcs_buffer, NOCATGETS(".KEEP_STATE_FILE"));
643         dot_keep_state_file = GETNAME(wcs_buffer, FIND_LENGTH);
644         MBSTOWCS(wcs_buffer, NOCATGETS(""));
645         empty_name = GETNAME(wcs_buffer, FIND_LENGTH);
646         MBSTOWCS(wcs_buffer, NOCATGETS(" FORCE"));
647         force = GETNAME(wcs_buffer, FIND_LENGTH);
648         MBSTOWCS(wcs_buffer, NOCATGETS("HOST_ARCH"));
649         host_arch = GETNAME(wcs_buffer, FIND_LENGTH);
650         MBSTOWCS(wcs_buffer, NOCATGETS("HOST_MACH"));
651         host_mach = GETNAME(wcs_buffer, FIND_LENGTH);
652         MBSTOWCS(wcs_buffer, NOCATGETS(".IGNORE"));
653         ignore_name = GETNAME(wcs_buffer, FIND_LENGTH);
654         MBSTOWCS(wcs_buffer, NOCATGETS(".INIT"));
655         init = GETNAME(wcs_buffer, FIND_LENGTH);
```

```
656         MBSTOWCS(wcs_buffer, NOCATGETS(".LOCAL"));
657         localhost_name = GETNAME(wcs_buffer, FIND_LENGTH);
658         MBSTOWCS(wcs_buffer, NOCATGETS(".make.state"));
659         make_state = GETNAME(wcs_buffer, FIND_LENGTH);
660         MBSTOWCS(wcs_buffer, NOCATGETS("MAKEFLAGS"));
661         makeflags = GETNAME(wcs_buffer, FIND_LENGTH);
662         MBSTOWCS(wcs_buffer, NOCATGETS(".MAKE_VERSION"));
663         make_version = GETNAME(wcs_buffer, FIND_LENGTH);
664         MBSTOWCS(wcs_buffer, NOCATGETS(".NO_PARALLEL"));
665         no_parallel_name = GETNAME(wcs_buffer, FIND_LENGTH);
666         MBSTOWCS(wcs_buffer, NOCATGETS(".NOT_AUTO"));
667         not_auto = GETNAME(wcs_buffer, FIND_LENGTH);
668         MBSTOWCS(wcs_buffer, NOCATGETS(".PARALLEL"));
669         parallel_name = GETNAME(wcs_buffer, FIND_LENGTH);
670         MBSTOWCS(wcs_buffer, NOCATGETS("PATH"));
671         path_name = GETNAME(wcs_buffer, FIND_LENGTH);
672         MBSTOWCS(wcs_buffer, NOCATGETS("+"));
673         plus = GETNAME(wcs_buffer, FIND_LENGTH);
674         MBSTOWCS(wcs_buffer, NOCATGETS(".PRECIOUS"));
675         precious = GETNAME(wcs_buffer, FIND_LENGTH);
676         MBSTOWCS(wcs_buffer, NOCATGETS("?"));
677         query = GETNAME(wcs_buffer, FIND_LENGTH);
678         MBSTOWCS(wcs_buffer, NOCATGETS("^"));
679         hat = GETNAME(wcs_buffer, FIND_LENGTH);
680         MBSTOWCS(wcs_buffer, NOCATGETS(".RECURSIVE"));
681         recursive_name = GETNAME(wcs_buffer, FIND_LENGTH);
682         MBSTOWCS(wcs_buffer, NOCATGETS(".SCCS_GET"));
683         sccs_get_name = GETNAME(wcs_buffer, FIND_LENGTH);
684         MBSTOWCS(wcs_buffer, NOCATGETS(".SCCS_GET_POSIX"));
685         sccs_get_posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
686         MBSTOWCS(wcs_buffer, NOCATGETS(".GET"));
687         get_name = GETNAME(wcs_buffer, FIND_LENGTH);
688         MBSTOWCS(wcs_buffer, NOCATGETS(".GET_POSIX"));
689         get_posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
690         MBSTOWCS(wcs_buffer, NOCATGETS("SHELL"));
691         shell_name = GETNAME(wcs_buffer, FIND_LENGTH);
692         MBSTOWCS(wcs_buffer, NOCATGETS(".SILENT"));
693         silent_name = GETNAME(wcs_buffer, FIND_LENGTH);
694         MBSTOWCS(wcs_buffer, NOCATGETS(".SUFFIXES"));
695         suffixes_name = GETNAME(wcs_buffer, FIND_LENGTH);
696         MBSTOWCS(wcs_buffer, SUNPRO_DEPENDENCIES);
697         sunpro_dependencies = GETNAME(wcs_buffer, FIND_LENGTH);
698         MBSTOWCS(wcs_buffer, NOCATGETS("TARGET_ARCH"));
699         target_arch = GETNAME(wcs_buffer, FIND_LENGTH);
700         MBSTOWCS(wcs_buffer, NOCATGETS("TARGET_MACH"));
701         target_mach = GETNAME(wcs_buffer, FIND_LENGTH);
702         MBSTOWCS(wcs_buffer, NOCATGETS("VIRTUAL_ROOT"));
703         virtual_root = GETNAME(wcs_buffer, FIND_LENGTH);
704         MBSTOWCS(wcs_buffer, NOCATGETS("VPATH"));
705         vpath_name = GETNAME(wcs_buffer, FIND_LENGTH);
706         MBSTOWCS(wcs_buffer, NOCATGETS(".WAIT"));
707         wait_name = GETNAME(wcs_buffer, FIND_LENGTH);

709         wait_name->state = build_ok;

711         /* Mark special targets so that the reader treats them properly */
712         svr4_name->special_reader = svr4_special;
713         posix_name->special_reader = posix_special;
714         built_last_make_run->special_reader = built_last_make_run_special;
715         default_rule_name->special_reader = default_special;
716 #ifdef NSE
717         derived_src->special_reader= derived_src_special;
718 #endif
719         dot_keep_state->special_reader = keep_state_special;
720         dot_keep_state_file->special_reader = keep_state_file_special;
721         ignore_name->special_reader = ignore_special;
```



```
722         make_version->special_reader = make_version_special;
723         no_parallel_name->special_reader = no_parallel_special;
724         parallel_name->special_reader = parallel_special;
725         localhost_name->special_reader = localhost_special;
726         precious->special_reader = precious_special;
727         sccs_get_name->special_reader = sccs_get_special;
728         sccs_get_posix_name->special_reader = sccs_get_posix_special;
729         get_name->special_reader = get_special;
730         get_posix_name->special_reader = get_posix_special;
731         silent_name->special_reader = silent_special;
732         suffixes_name->special_reader = suffixes_special;

734         /* The value of $$ is $ */
735         (void) SETVAR(dollar, dollar, false);
736         dollar->dollar = false;

738         /* Set the value of $(SHELL) */
739         #ifdef HP_UX
740         MBSTOWCS(wcs_buffer, NOCATGETS("/bin/posix/sh"));
741         #else
742         #if defined(SUN5_0)
743         if (posix) {
744           MBSTOWCS(wcs_buffer, NOCATGETS("/usr/xpg4/bin/sh"));
745         } else {
746           MBSTOWCS(wcs_buffer, NOCATGETS("/bin/sh"));
747         }
748         #else  /* ^SUN5_0 */
749         MBSTOWCS(wcs_buffer, NOCATGETS("/bin/sh"));
750         #endif /* ^SUN5_0 */
751         #endif
752         (void) SETVAR(shell_name, GETNAME(wcs_buffer, FIND_LENGTH), false);

754         /*
755          * Use " FORCE" to simulate a FRC dependency for :: type
756          * targets with no dependencies.
757          */
758         (void) append_prop(force, line_prop);
759         force->stat.time = file_max_time;

761         /* Make sure VPATH is defined before current dir is read */
762         if ((cp = getenv(vpath_name->string_mb)) != NULL) {
763                 MBSTOWCS(wcs_buffer, cp);
764                 (void) SETVAR(vpath_name,
765                               GETNAME(wcs_buffer, FIND_LENGTH),
766                               false);
767         }

769         /* Check if there is NO PATH variable. If not we construct one. */
770         if (getenv(path_name->string_mb) == NULL) {
771                 vroot_path = NULL;
772                 add_dir_to_path(NOCATGETS("."), &vroot_path, -1);
773                 add_dir_to_path(NOCATGETS("/bin"), &vroot_path, -1);
774                 add_dir_to_path(NOCATGETS("/usr/bin"), &vroot_path, -1);
775         }
776 }

778 /*
779  * iterate on list of conditional macros in np, and place them in
780  * a String_rec starting with, and separated by the '$' character.
781  */
782 void
783 cond_macros_into_string(Name np, String_rec *buffer)
784 {
785         Macro_list      macro_list;

787         /*
```

```
788          * Put the version number at the start of the string
789          */
790         MBSTOWCS(wcs_buffer, DEPINFO_FMT_VERSION);
791         append_string(wcs_buffer, buffer, FIND_LENGTH);
792         /*
793          * Add the rest of the conditional macros to the buffer
794          */
795         if (np->depends_on_conditional){
796                 for (macro_list = np->conditional_macro_list;
797                      macro_list != NULL; macro_list = macro_list->next){
798                         append_string(macro_list->macro_name, buffer,
799                                       FIND_LENGTH);
800                         append_char((int) equal_char, buffer);
801                         append_string(macro_list->value, buffer, FIND_LENGTH);
802                         append_char((int) dollar_char, buffer);
803                 }
804         }
805 }
806 /*
807  *      Copyright (c) 1987-1992 Sun Microsystems, Inc.  All Rights Reserved.
808  *      Sun considers its source code as an unpublished, proprietary
809  *      trade secret, and it is available only under strict license
810  *      provisions.  This copyright notice is placed here only to protect
811  *      Sun in the event the source is deemed a published work.  Dissassembly,
812  *      decompilation, or other means of reducing the object code to human
813  *      readable form is prohibited by the license agreement under which
814  *      this code is provided to the user or company in possession of this
815  *      copy.
816  *      RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the
817  *      Government is subject to restrictions as set forth in subparagraph
818  *      (c)(1)(ii) of the Rights in Technical Data and Computer Software
819  *      clause at DFARS 52.227-7013 and in similar clauses in the FAR and
820  *      NASA FAR Supplement.
821  *
822  * 1.3 91/09/30
823  */


826 /* Some includes are commented because of the includes at the beginning */
827 /* #include <signal.h> */
828 #include <sys/types.h>
829 #include <sys/stat.h>
830 #include <sys/param.h>
831 /* #include <string.h> */
832 #include <unistd.h>
833 #include <stdlib.h>
834 /* #include <stdio.h> */
835 /* #include <avo/find_dir.h> */
836 /* #ifndef TEAMWARE_MAKE_CMN
837 #include <avo/find_dir.h>
838 #endif */

840 /* Routines to find the base directory name from which the various components
841  * -executables, *crt* libraries etc will be accessed
842  */

844 /* This routine checks to see if a given filename is an executable or not.
845    Logically similar to the csh statement : if  ( -x $i && ! -d $i )
846  */

848 static int
849 check_if_exec(char *file)
850 {
851         struct stat stb;
852         if (stat(file, &stb) < 0) {
853                 return ( -1);
```



```
854         }
855         if (S_ISDIR(stb.st_mode)) {
856                 return (-1);
857         }
858         if (!(stb.st_mode & S_IEXEC)) {
859                 return ( -1);
860         }
861         return (0);
862 }

864 /* resolve - check for specified file in specified directory
865  *      sets up dir, following symlinks.
866  *      returns zero for success, or
867  *      -1 for error (with errno set properly)
868  */
869 static int
870 resolve (char   *indir, /* search directory */
871          char   *cmd,   /* search for name */
872          char   *dir,   /* directory buffer */
873          char   **run)  /* resultion name ptr ptr */
874 {
875     char               *p;
876     int                 rv = -1;
877     int                 sll;
878     char                symlink[MAXPATHLEN + 1];

880     do {
881         errno = ENAMETOOLONG;
882         if ((strlen (indir) + strlen (cmd) + 2) > (size_t) MAXPATHLEN)
883             break;

885         sprintf(dir, "%s/%s", indir, cmd);
886         if (check_if_exec(dir) != 0)  /* check if dir is an executable */
887         {
888                 break;          /* Not an executable program */
889         }

891         /* follow symbolic links */
892         while ((sll = readlink (dir, symlink, MAXPATHLEN)) >= 0) {
893             symlink[sll] = 0;
894             if (*symlink == '/')
895                 strcpy (dir, symlink);
896             else
897                 sprintf (strrchr (dir, '/'), "/%s", symlink);
898         }
899         if (errno != EINVAL)
900             break;

902         p = strrchr (dir, '/');
903         *p++ = 0;
904         if (run)                /* user wants resolution name */
905             *run = p;
906         rv = 0;                 /* complete, with success! */

908     } while (0);

910     return rv;
911 }

913 /*
914  *find_run_directory - find executable file in PATH
915  *
916  * PARAMETERS:
917  *      cmd     filename as typed by user (argv[0])
918  *      cwd     buffer from which is read the working directory
919  *               if first character is '/' or into which is
```

```
 *                copied working directory name otherwise
 *      dir     buffer into which is copied program's directory
 *      pgm     where to return pointer to tail of cmd (may be NULL
 *               if not wanted)
 *      run     where to return pointer to tail of final resolved
 *               name ( dir/run is the program) (may be NULL
 *               if not wanted)
 *      path    user's path from environment
 *
 * Note: run and pgm will agree except when symbolic links have
 *      renamed files
 *
 * RETURNS:
 *      returns zero for success,
 *      -1 for error (with errno set properly).
 *
 * EXAMPLE:
 *      find_run_directory (argv[0], ".", &charray1, (char **) 0, (char **) 0,
 *                          getenv(NOGETTEXT("PATH")));
 */
extern int
find_run_directory (char        *cmd,
                    char        *cwd,
                    char        *dir,
                    char        **pgm,
                    char        **run,
                    char        *path)
{
    int                 rv = 0;
    char                *f, *s;
    int                 i;
    char                tmp_path[MAXPATHLEN];

    if (!cmd || !*cmd || !cwd || !dir) {
        errno = EINVAL;         /* stupid arguments! */
        return -1;
    }

    if (*cwd != '/')
        if (!(getcwd (cwd, MAXPATHLEN)))
            return -1;          /* can not get working directory */

    f = strrchr (cmd, '/');
    if (pgm)                    /* user wants program name */
        *pgm = f ? f + 1 : cmd;

    /* get program directory */
    rv = -1;
    if (*cmd == '/')    /* absname given */
        rv = resolve ("", cmd + 1, dir, run);
    else if (f)         /* relname given */
        rv = resolve (cwd, cmd, dir, run);
    else {  /* from searchpath */
            if (!path || !*path) {      /* if missing or null path */
                    tmp_path[0] = '.';  /* assume sanity */
                    tmp_path[1] = '\0';
            } else {
                    strcpy(tmp_path, path);
            }
        f = tmp_path;
        rv = -1;
        errno = ENOENT; /* errno gets this if path empty */
        while (*f && (rv < 0)) {
            s = f;
            while (*f && (*f != ':'))
                ++f;
```



```
            if (*f)
                *f++ = 0;
            if (*s == '/')
                rv = resolve (s, cmd, dir, run);
            else {
                char                abuf[MAXPATHLEN];

                sprintf (abuf, "%s/%s", cwd, s);
                rv = resolve (abuf, cmd, dir, run);
            }
        }
    }

    /* Remove any trailing /. */
    i = strlen(dir);
    if ( dir[i-2] == '/' && dir[i-1] == '.') {
            dir[i-2] = '\0';
    }

    return rv;
}

#endif /* ! codereview */
```

```
**********************************************************
   14622 Wed May 20 11:04:09 2015
new/usr/src/cmd/make/bin/make/common/nse.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 1994 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)nse.cc 1.15 06/12/12
 */

#pragma ident   "@(#)nse.cc     1.15    06/12/12"

#ifdef NSE

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

/*
 * This file does some extra checking on behalf of the NSE.
 * It does some stuff that is analogous to lint in that it
 * looks for things which may be legal but that give the NSE
 * trouble.  Currently it looks for:
 *      1) recursion by cd'ing to a directory specified by a
 *         make variable that is defined from the shell environment.
 *      2) recursion by cd'ing to a directory specified by a
 *         make variable that has backquotes in it.
 *      3) recursion to another makefile in the same directory.
 *      4) a dependency upon an SCCS file (SCCS/s.*)
 *      5) backquotes in a file name
 *      6) a make variable being defined on the command-line that
 *         ends up affecting dependencies
 *      7) wildcards (*[) in dependencies
 *      8) recursion to the same directory
 *      9) potential source files on the left-hand-side so
 *         that they appear as derived files
 *
 * Things it should look for:
 *      1) makefiles that are symlinks (why are these a problem?)
 */
```



```
#define TARG_SUFX       "/usr/nse/lib/nse_targ.sufx"

typedef struct _Nse_suffix      *Nse_suffix, Nse_suffix_rec;
struct _Nse_suffix {
        wchar_t                 *suffix;        /* The suffix */
        struct _Nse_suffix      *next;          /* Linked list */
};
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;
        }
```

```
128         line = &prop->body.line;
129 #ifdef LTEST
130         printf("string = %s\n", line->command_template->command_line->string_mb)
131 #endif

133         wscpy(tmpbuf, line->command_template->command_line->string);
134         our_template = tmpbuf;
135         cd = false;
136         while (nse_gettoken(&our_template, tok)) {
137 #ifdef LTEST
138                 printf("in gettoken loop\n");
139 #endif
140 #ifdef SUNOS4_AND_AFTER
141                 if (IS_WEQUAL(tok, (wchar_t *) "cd")) {
142 #else
143                 if (is_equal(tok, "cd")) {
144 #endif
145                         cd = true;
146                 } else if (cd && tok[0] == '$') {
147                         nse_backquote_seen = NULL;
148                         nse_shell_var_used = NULL;
149                         nse_watch_vars = true;
150 #ifdef SUNOS4_AND_AFTER
151                         INIT_STRING_FROM_STACK(string, strbuf);
152                         name = GETNAME(tok, FIND_LENGTH);
153 #else
154                         init_string_from_stack(string, strbuf);
155                         name = getname(tok, FIND_LENGTH);
156 #endif
157                         expand_value(name, &string, false);
158                         nse_watch_vars = false;

160 #ifdef LTEST
161                         printf("cd = %d, tok = $\n", cd);
162 #endif
163                         /*
164                          * Try to trim tok to just
165                          * the variable.
166                          */
167                         if (nse_shell_var_used != NULL) {
168                                 nse_warning();
169                                 fprintf(stderr, "\tMake invoked recursively by c
170                                     nse_shell_var_used->string_mb,
171                                     line->command_template->command_line->string
172                         }
173                         if (nse_backquote_seen != NULL) {
174                                 nse_warning();
175                                 fprintf(stderr, "\tMake invoked recursively by c
176                                     nse_backquote_seen->string_mb,
177                                     line->command_template->command_line->string
178                         }
179                         cd = false;
180                 } else if (cd && nse_backquotes(tok)) {
181                         nse_warning();
182                         fprintf(stderr, "\tMake invoked recursively by cd'ing to
183                             line->command_template->command_line->string_mb);
184                         cd = false;
185                 } else {
186                         cd = false;
187                 }
188         }

190         /*
191          * Now check for recursion to ".".
192          */
193         if (primary_makefile != NULL) {
```



```
194                 target = prop->body.line.target;
195                 recurse = get_prop(target->prop, recursive_prop);
196                 while (recurse != NULL) {
197                         r = &recurse->body.recursive;
198 #ifdef SUNOS4_AND_AFTER
199                         if (IS_WEQUAL(r->directory->string, (wchar_t *) ".") &&
200                             !IS_WEQUAL(r->makefiles->name->string,
201                             primary_makefile->string)) {
202 #else
203                         if (is_equal(r->directory->string, ".") &&
204                             !is_equal(r->makefiles->name->string,
205                             primary_makefile->string)) {
206 #endif
207                                 nse_warning();
208                                 fprintf(stderr, "\tRecursion to makefile '%s' in
209                                     r->makefiles->name->string_mb,
210                                     line->command_template->command_line->string
211                         }
212                         recurse = get_prop(recurse->next, recursive_prop);
213                 }
214         }
215 }

217 /*
218  * Print an NSE make warning line.
219  * If the -P flag was given then consider this a fatal
220  * error, otherwise, just a warning.
221  */
222 static void
223 nse_warning(void)
224 {
225 #ifdef SUNOS4_AND_AFTER
226         if (report_dependencies_level > 0) {
227 #else
228         if (is_true(flag.report_dependencies)) {
229 #endif
230                 our_exit_status = 1;
231         }
232         if (primary_makefile != NULL) {
233                 fprintf(stderr, "make: NSE warning from makefile %s/%s:\n",
234                         get_current_path(), primary_makefile->string_mb);
235         } else {
236                 fprintf(stderr, "make: NSE warning from directory %s:\n",
237                         get_current_path());
238         }
239 }

241 /*
242  * Get the next whitespace delimited token pointed to by *cp.
243  * Return it in tok.
244  */
245 static Boolean
246 nse_gettoken(wchar_t **cp, wchar_t *tok)
247 {
248         wchar_t         *to;
249         wchar_t         *p;

251         p = *cp;
252         while (*p && iswspace(*p)) {
253                 p++;
254         }
255         if (*p == '\0') {
256                 return false;
257         }
258         to = tok;
259         while (*p && !iswspace(*p)) {
```

```
260                 *to++ = *p++;
261         }
262         if (*p == '\0') {
263                 return false;
264         }
265         *to = '\0';
266         *cp = p;
267         return true;
268 }

270 /*
271  * Given a dependency and a target, see if the dependency
272  * is an SCCS file.  Check for the last component of its name
273  * beginning with "s." and the component before that being "SCCS".
274  * The NSE does not consider a source file to be derived from
275  * an SCCS file.
276  */
277 void
278 nse_check_sccs(wchar_t *targ, wchar_t *dep)
279 {
280         wchar_t         *slash;
281         wchar_t         *p;

283 #ifdef SUNOS4_AND_AFTER
284         if (!nse) {
285 #else
286         if (is_false(flag.nse)) {
287 #endif
288                 return;
289         }
290 #ifdef SUNOS4_AND_AFTER
291         slash = wsrchr(dep, (int) slash_char);
292 #else
293         slash = rindex(dep, '/');
294 #endif
295         if (slash == NULL) {
296                 return;
297         }
298         if (slash[1] != 's' || slash[2] != '.') {
299                 return;
300         }

302         /*
303          * Find the next to last filename component.
304          */
305         for (p = slash - 1; p >= dep; p--) {
306                 if (*p == '/') {
307                         break;
308                 }
309         }
310         p++;
311 #ifdef SUNOS4_AND_AFTER
312         MBSTOWCS(wcs_buffer, "SCCS/");
313         if (IS_WEQUALN(p, wcs_buffer, wslen(wcs_buffer))) {
314 #else
315         if (is_equaln(p, "SCCS/", 5)) {
316 #endif
317                 nse_warning();
318                 WCSTOMBS(mbs_buffer, targ);
319                 WCSTOMBS(mbs_buffer2, dep);
320                 fprintf(stderr, "\tFile '%s' depends upon SCCS file '%s'\n",
321                         mbs_buffer, mbs_buffer2);
322         }
323         return;
324 }
```



```
326 /*
327  * Given a filename check to see if it has 2 backquotes in it.
328  * Complain about this because the shell expands the backquotes
329  * but make does not so the files always appear to be out of date.
330  */
331 void
332 nse_check_file_backquotes(wchar_t *file)
333 {
334 #ifdef SUNOS4_AND_AFTER
335         if (!nse) {
336 #else
337         if (is_false(flag.nse)) {
338 #endif
339                 return;
340         }
341         if (nse_backquotes(file)) {
342                 nse_warning();
343                 WCSTOMBS(mbs_buffer, file);
344                 fprintf(stderr, "\tFilename \"%s\" has backquotes in it\n",
345                         mbs_buffer);
346         }
347 }

349 /*
350  * Return true if the string has two backquotes in it.
351  */
352 Boolean
353 nse_backquotes(wchar_t *str)
354 {
355         wchar_t         *bq;

357 #ifdef SUNOS4_AND_AFTER
358         bq = wschr(str, (int) backquote_char);
359         if (bq) {
360                 bq = wschr(&bq[1], (int) backquote_char);
361 #else
362         bq = index(str, '`');
363         if (bq) {
364                 bq = index(&bq[1], '`');
365 #endif
366                 if (bq) {
367                         return true;
368                 }
369         }
370         return false;
371 }

373 /*
374  * A macro that was defined on the command-line was found to affect the
375  * set of dependencies.  The NSE "target explode" will not know about
376  * this and will not get the same set of dependencies.
377  */
378 void
379 nse_dep_cmdmacro(wchar_t *macro)
380 {
381 #ifdef SUNOS4_AND_AFTER
382         if (!nse) {
383 #else
384         if (is_false(flag.nse)) {
385 #endif
386                 return;
387         }
388         nse_warning();
389         WCSTOMBS(mbs_buffer, macro);
390         fprintf(stderr, "\tVariable '%s' is defined on the command-line and\n\ta
391                 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;
                        }
                }
        }
}
```

```
525 /*
526  * See if a target is a potential source file and has no
527  * dependencies and no rule but shows up on the right-hand
528  * side.  This tends to occur from old "make depend" output.
529  */
530 void
531 nse_check_no_deps_no_rule(Name target, Property line, Property command)
532 {
533         Nse_suffix      sufx;
534         wchar_t         *suffix;

536 #ifdef SUNOS4_AND_AFTER
537         if (!nse) {
538 #else
539         if (is_false(flag.nse)) {
540 #endif
541                 return;
542         }
543 #ifdef SUNOS4_AND_AFTER
544         if (target->stat.is_derived_src) {
545 #else
546         if (is_true(target->stat.is_derived_src)) {
547 #endif
548                 return;
549         }
550         if (line != NULL && line->body.line.dependencies != NULL) {
551                 return;
552         }
553 #ifdef SUNOS4_AND_AFTER
554         if (command->body.line.sccs_command) {
555 #else
556         if (is_true(command->body.line.sccs_command)) {
557 #endif
558                 return;
559         }
560 #ifdef SUNOS4_AND_AFTER
561         suffix = wsrchr(target->string, (int) period_char);
562 #else
563         suffix = rindex(target->string, '.');
564 #endif
565         if (suffix != NULL) {
566                 for (sufx = sufx_hdr; sufx != NULL; sufx = sufx->next) {
567 #ifdef SUNOS4_AND_AFTER
568                         if (IS_WEQUAL(sufx->suffix, suffix)) {
569 #else
570                         if (is_equal(sufx->suffix, suffix)) {
571 #endif
572                                 if (command->body.line.command_template == NULL)
573                                         nse_warning();
574                                         fprintf(stderr, "\tProbable source file
575                                                 target->string_mb);
576                                 }
577                         }
578                 }
579         }
580 }

582 /*
583  * Detected a situation where a recursive make derived a file
584  * without using a makefile.
585  */
586 void
587 nse_no_makefile(Name target)
588 {
589 #ifdef SUNOS4_AND_AFTER
```



```
590         if (!nse) {
591 #else
592         if (is_false(flag.nse)) {
593 #endif
594                 return;
595         }
596         nse_warning();
597         fprintf(stderr, "Recursive make to derive %s did not use a makefile\n",
598                 target->string_mb);
599 }

601 /*
602  * Return the NSE exit status.
603  * If the -P flag was given then a warning is considered fatal
604  */
605 int
606 nse_exit_status(void)
607 {
608         return our_exit_status;
609 }
610 #endif
611 #endif /* ! codereview */
```

```
**********************************************************
   12975 Wed May 20 11:04:10 2015
new/usr/src/cmd/make/bin/make/common/nse_printdep.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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.
 */
/*
 * @(#)nse_printdep.cc 1.18 06/12/12
 */

#pragma ident   "@(#)nse_printdep.cc    1.18    06/12/12"

/*
 * 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);
        }
}

/*
 *      print_more_deps(target, name)
 *
 *      Print some special dependencies.
```

```
 *      These are the dependencies for the .INIT and .DONE targets.
 *
 *      Parameters:
 *              target          Target built during make run
 *              name            Special target to print dependencies for
 *
 *      Global variables used:
 */
static void
print_more_deps(Name target, Name name)
{
        Property        line;
        register Dependency     dependencies;

        line = get_prop(name->prop, line_prop);
        if (line != NULL && line->body.line.dependencies != NULL) {
                (void) printf("%s:\t", target->string_mb);
                print_deplist(line->body.line.dependencies);
                (void) printf("\n");
                for (dependencies= line->body.line.dependencies;
                     dependencies != NULL;
                     dependencies= dependencies->next) {
                        print_deps(dependencies->name,
                                get_prop(dependencies->name->prop, line_prop));
                }
        }
}

/*
 *      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));
                }
        }
}

static Boolean
is_out_of_date(Property line)
{
        Dependency      dep;
        Property        line2;

        if (line == NULL) {
                return false;
        }
        if (line->body.line.is_out_of_date) {
                return true;
        }
        for (dep = line->body.line.dependencies;
             dep != NULL;
             dep = dep->next) {
                line2 = get_prop(dep->name->prop, line_prop);
                if (is_out_of_date(line2)) {
                        line->body.line.is_out_of_date = true;
                        return true;
                }
        }
        return false;
}

/*
 * Given a dependency print it and all its siblings.
 */
static void
print_deplist(Dependency head)
{
        Dependency      dp;

        for (dp = head; dp != NULL; dp = dp->next) {
                if ((report_dependencies_level != 2) ||
                    ((!dp->automatic) ||
                     (dp->name->is_double_colon))) {
                        if (dp->name != force) {
                                putwchar(' ');
                                print_filename(dp->name);
                        }
                }
        }
}

/*
 * Print the name of a file for the -P option.
```

```
 * If the file is a directory put on a trailing slash.
 */
static void
print_filename(Name name)
{
        (void) printf("%s", name->string_mb);
/*
        if (name->stat.is_dir) {
                putwchar('/');
        }
 */
}

/*
 *      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) {
```

```
392                 (void)printf("%s ", dep->name->string_mb);
393                 build_suffix_list(dep->name);
394         };
395         (void)printf("\n\n");
396
397         /* % rules */
398         for (percent= percent_list; percent != NULL; percent= percent->next) {
399                 (void) printf("%s:", percent->name->string_mb);
400
401                 for (percent_depe= percent->dependencies; percent_depe != NULL;
402                         (void) printf(" %s", percent_depe->name->string_mb);
403
404                 (void) printf("\n");
405
406                 for (rule= percent->command_template; rule != NULL; rule= rule->
407                         (void)printf("\t%s\n", rule->command_line->string_mb);
408         };
409
410         /* Suffix rules */
411         for (p = hashtab.begin(), e = hashtab.end(); p != e; p++)
412                         if (is_false(p->dependency_printed) && (p->string[0] ==
413                                 print_rule(p);
414                                 p->dependency_printed= true;
415                         };
416
417         /* Macro assignments */
418         for (p = hashtab.begin(), e = hashtab.end(); p != e; p++)
419                         if (((prop= get_prop(p->prop, macro_prop)) != NULL) &&
420                            (prop->body.macro.value != NULL)) {
421                                 (void)printf("%s", p->string_mb);
422                                 print_value(prop->body.macro.value,
423                                             prop->body.macro.daemon);
424                         };
425         (void)printf("\n");
426
427         /* Delays */
428         for (p = hashtab.begin(), e = hashtab.end(); p != e; p++)
429                         for (prop= get_prop(p->prop, conditional_prop);
430                              prop != NULL;
431                              prop= get_prop(prop->next, conditional_prop)) {
432                                 (void)printf("%s := %s",
433                                              p->string_mb,
434                                              prop->body.conditional.name->string
435                                 print_value(prop->body.conditional.value, no_dae
436                         };
437         (void)printf("\n");
438
439         /* All other dependencies */
440         for (p = hashtab.begin(), e = hashtab.end(); p != e; p++)
441                         if (is_false(p->dependency_printed) && (p->colons != no_
442                                 print_rule(p);
443         (void)printf("\n");
444         exit(0);
445 }
446 #endif
447
448 /*
449  *      This is a set  of routines for dumping the internal make state
450  *      Used for the -p option
451  */
452 void
453 print_value(register Name value, Daemon daemon)
454
455 #ifdef SUNOS4_AND_AFTER
456
457 #else
```



```
458
459 #endif
460 {
461         Chain                   cp;
462
463         if (value == NULL)
464                 (void)printf("=\n");
465         else
466                 switch (daemon) {
467                     case no_daemon:
468                         (void)printf("= %s\n", value->string_mb);
469                         break;
470                     case chain_daemon:
471                         for (cp= (Chain) value; cp != NULL; cp= cp->next)
472                                 (void)printf(cp->next == NULL ? "%s" : "%s ",
473                                              cp->name->string_mb);
474                         (void)printf("\n");
475                         break;
476                 };
477 }
478
479 static void
480 print_rule(register Name target)
481 {
482         register Cmd_line       rule;
483         register Property       line;
484
485         if (((line= get_prop(target->prop, line_prop)) == NULL) ||
486             ((line->body.line.command_template == NULL) &&
487              (line->body.line.dependencies == NULL)))
488                 return;
489         print_dependencies(target, line);
490         for (rule= line->body.line.command_template; rule != NULL; rule= rule->n
491                 (void)printf("\t%s\n", rule->command_line->string_mb);
492 }
493
494
495 /*
496  *  If target is recursive,  print the following to standard out:
497  *      .RECURSIVE subdir targ Makefile
498  */
499 static void
500 print_rec_info(Name target)
501 {
502         Recursive_make  rp;
503         wchar_t         *colon;
504
505         report_recursive_init();
506
507         rp = find_recursive_target(target);
508
509         if (rp) {
510                 /*
511                  * if found,  print starting with the space after the ':'
512                  */
513                 colon = (wchar_t *) wschr(rp->oldline, (int) colon_char);
514                 (void) printf("%s", colon + 1);
515         }
516 }
517
518 #endif /* ! codereview */
```

```
**********************************************************
   61705 Wed May 20 11:04:10 2015
new/usr/src/cmd/make/bin/make/common/parallel.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)parallel.cc 1.75 06/12/12
 */

#pragma ident   "@(#)parallel.cc        1.75    06/12/12"

#ifdef TEAMWARE_MAKE_CMN

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

/*
 * Included files
 */
#ifdef DISTRIBUTED
#include <avo/strings.h>        /* AVO_STRDUP() */
#include <dm/Avo_DoJobMsg.h>
#include <dm/Avo_MToolJobResultMsg.h>
#endif
#include <errno.h>              /* errno */
#include <fcntl.h>
#include <avo/util.h>           /* avo_get_user(), avo_hostname() */
#include <mk/defs.h>
#include <mksh/dosys.h>         /* redirect_io() */
#include <mksh/macro.h>         /* expand_value() */
#include <mksh/misc.h>          /* getmem() */
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>
#include <sys/wait.h>
#include <unistd.h>

#ifdef SGE_SUPPORT
```



```
#include <dmthread/Avo_PathNames.h>
#endif


/*
 * Defined macros
 */
#define MAXRULES                100

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


/*
 * typedefs & structs
 */


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


/*
 * File table of contents
 */
#ifdef DISTRIBUTED
static  void            append_dmake_cmd(Avo_DoJobMsg *dmake_job_msg, char *orig
static  void            append_job_result_msg(Avo_MToolJobResultMsg *msg, char *
static  void            send_job_result_msg(Running rp);
#endif
static  void            delete_running_struct(Running rp);
static  Boolean         dependency_conflict(Name target);
static  Doname          distribute_process(char **commands, Property line);
static  void            doname_subtree(Name target, Boolean do_get, Boolean impl
static  void            dump_out_file(char *filename, Boolean err);
static  void            finish_doname(Running rp);
static  void            maybe_reread_make_state(void);
static  void            process_next(void);
static  void            reset_conditionals(int cnt, Name *targets, Property *loc
static  pid_t           run_rule_commands(char *host, char **commands);
static  Property        *set_conditionals(int cnt, Name *targets);
static  void            store_conditionals(Running rp);


/*
 *      execute_parallel(line, waitflg)
 *
 *      DMake 2.x:
 *      parallel mode: spawns a parallel process to execute the command group.
 *      distributed mode: sends the command group down the pipe to rxm.
 *
 *      Return value:
 *                              The result of the execution
```

```
 *
 *      Parameters:
 *              line            The command group to execute
 */
Doname
execute_parallel(Property line, Boolean waitflg, Boolean local)
{
        int                     argcnt;
        int                     cmd_options = 0;
        char                    *commands[MAXRULES + 5];
        char                    *cp;
#ifdef DISTRIBUTED
        Avo_DoJobMsg            *dmake_job_msg = NULL;
#endif
        Name                    dmake_name;
        Name                    dmake_value;
        int                     ignore;
        Name                    make_machines_name;
        char                    **p;
        Property                prop;
        Doname                  result = build_ok;
        Cmd_line                rule;
        Boolean                 silent_flag;
        Name                    target = line->body.line.target;
        Boolean                 wrote_state_file = false;

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



```
                }
#endif
        }

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

#ifdef DISTRIBUTED
        if (dmake_mode_type == distributed_mode) {
                if(local) {
//                      return (execute_serial(line));
                        waitflg = true;
                }
                dmake_job_msg = new Avo_DoJobMsg();
                dmake_job_msg->setJobId(++job_msg_id);
                dmake_job_msg->setTarget(target->string_mb);
                dmake_job_msg->setImmediateOutput(0);
                called_make = false;
        } else
#endif
        {
                p = commands;
        }

        argcnt = 0;
        for (rule = line->body.line.command_used;
             rule != NULL;
             rule = rule->next) {
                if (posix && (touch || quest) && !rule->always_exec) {
                        continue;
                }
                if (vpath_defined) {
                        rule->command_line =
                          vpath_translation(rule->command_line);
                }
                if (dmake_mode_type == distributed_mode) {
                        cmd_options = 0;
                        if(local) {
                                cmd_options |= local_host_mask;
                        }
                } else {
                        silent_flag = false;
                        ignore = 0;
                }
                if (rule->command_line->hash.length > 0) {
                        if (++argcnt == MAXRULES) {
                                if (dmake_mode_type == distributed_mode) {
                                        /* XXX - tell rxm to execute on local ho
                                        /* I WAS HERE!!! */
                                } else {
                                        /* Too many rules, run serially instead.
                                        return build_serial;
                                }
                        }
#ifdef DISTRIBUTED
                        if (dmake_mode_type == distributed_mode) {
                                /*
                                 * XXX - set assign_mask to tell rxm
                                 * to do the following.
                                 */
/* From execute_serial():
                                if (rule->assign) {
                                        result = build_ok;
                                        do_assign(rule->command_line, target);
```

```
 */
                                if (0) {
                                } else if (report_dependencies_level == 0) {
                                        if (rule->ignore_error) {
                                                cmd_options |= ignore_mask;
                                        }
                                        if (rule->silent) {
                                                cmd_options |= silent_mask;
                                        }
                                        if (rule->command_line->meta) {
                                                cmd_options |= meta_mask;
                                        }
                                        if (rule->make_refd) {
                                                cmd_options |= make_refd_mask;
                                        }
                                        if (do_not_exec_rule) {
                                                cmd_options |= do_not_exec_mask;
                                        }
                                        append_dmake_cmd(dmake_job_msg,
                                                         rule->command_line->str
                                                         cmd_options);
                                        /* Copying dosys()... */
                                        if (rule->make_refd) {
                                                if (waitflg) {
                                                        dmake_job_msg->setImmedi
                                                }
                                                called_make = true;
                                                if (command_changed &&
                                                    !wrote_state_file) {
                                                        write_state_file(0, fals
                                                        wrote_state_file = true;
                                                }
                                        }
                                }
                        } else
#endif
                        {
                                if (rule->silent && !silent) {
                                        silent_flag = true;
                                }
                                if (rule->ignore_error) {
                                        ignore++;
                                }
                                /* XXX - need to add support for + prefix */
                                if (silent_flag || ignore) {
                                        *p = getmem((silent_flag ? 1 : 0) +
                                                    ignore +
                                                    (strlen(rule->
                                                           command_line->
                                                           string_mb)) +
                                                    1);
                                        cp = *p++;
                                        if (silent_flag) {
                                                *cp++ = (int) at_char;
                                        }
                                        if (ignore) {
                                                *cp++ = (int) hyphen_char;
                                        }
                                        (void) strcpy(cp, rule->command_line->st
                                } else {
                                        *p++ = rule->command_line->string_mb;
                                }
                        }
                }
        }
        if ((argcnt == 0) ||
```



```
            (report_dependencies_level > 0)) {
#ifdef DISTRIBUTED
                if (dmake_job_msg) {
                        delete dmake_job_msg;
                }
#endif
                return build_ok;
        }
#ifdef DISTRIBUTED
        if (dmake_mode_type == distributed_mode) {
                // Send  a DoJob message to the rxm process.
                distribute_rxm(dmake_job_msg);

                // Wait for an acknowledgement.
                Avo_AcknowledgeMsg *ackMsg = getAcknowledgeMsg();
                if (ackMsg) {
                        delete ackMsg;
                }

                if (waitflg) {
                        // Wait for, and process a job result.
                        result = await_dist(waitflg);
                        if (called_make) {
                                maybe_reread_make_state();
                        }
                        check_state(temp_file_name);
                        if (result == build_failed) {
                                if (!continue_after_error) {

#ifdef PRINT_EXIT_STATUS
                                        warning(NOCATGETS("I'm in execute_parall
#endif

                                        fatal(catgets(catd, 1, 252, "Command fai
                                              target->string_mb);
                                }
                                /*
                                 * Make sure a failing command is not
                                 * saved in .make.state.
                                 */
                                line->body.line.command_used = NULL;
                        }
                        if (temp_file_name != NULL) {
                                free_name(temp_file_name);
                        }
                        temp_file_name = NULL;
                        Property spro = get_prop(sunpro_dependencies->prop, macr
                        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) {
                                        retmem_mb(val);
                                        spro->body.env_mem.value = NULL;
                                }
                        }
                        return result;
                } else {
                        parallel_process_cnt++;
                        return build_running;
```

```
392                 }
393         } else
394 #endif
395         {
396                 *p = NULL;

398                 Doname res = distribute_process(commands, line);
399                 if (res == build_running) {
400                         parallel_process_cnt++;
401                 }

403                 /*
404                  * Return only those memory that were specially allocated
405                  * for part of commands.
406                  */
407                 for (int i = 0; commands[i] != NULL; i++) {
408                         if ((commands[i][0] == (int) at_char) ||
409                             (commands[i][0] == (int) hyphen_char)) {
410                                 retmem_mb(commands[i]);
411                         }
412                 }
413                 return res;
414         }
415 }

417 #ifdef DISTRIBUTED
418 /*
419  *      append_dmake_cmd()
420  *
421  *      Replaces all escaped newline's (\<cr>)
422  *        in the original command line with space's,
423  *        then append the new command line to the DoJobMsg object.
424  */
425 static void
426 append_dmake_cmd(Avo_DoJobMsg *dmake_job_msg,
427                  char *orig_cmd_line,
428                  int cmd_options)
429 {
430 /*
431         Avo_DmakeCommand        *tmp_dmake_command;

433         tmp_dmake_command = new Avo_DmakeCommand(orig_cmd_line, cmd_options);
434         dmake_job_msg->appendCmd(tmp_dmake_command);
435         delete tmp_dmake_command;
436  */
437         dmake_job_msg->appendCmd(new Avo_DmakeCommand(orig_cmd_line, cmd_options
438 }
439 #endif

441 #ifdef TEAMWARE_MAKE_CMN
442 #define MAXJOBS_ADJUST_RFE4694000

444 #ifdef MAXJOBS_ADJUST_RFE4694000

446 #include <unistd.h>     /* sysconf(_SC_NPROCESSORS_ONLN) */
447 #include <sys/ipc.h>            /* ftok() */
448 #include <sys/shm.h>            /* shmget(), shmat(), shmdt(), shmctl() */
449 #include <semaphore.h>          /* sem_init(), sem_trywait(), sem_post(), sem_de
450 #if defined(linux)
451 #define LOADAVG_1MIN    0
452 #else
453 #include <sys/loadavg.h>        /* getloadavg() */
454 #endif /* linux */

456 /*
457  *      adjust_pmake_max_jobs (int pmake_max_jobs)
```



```
458  *
459  *      Parameters:
460  *              pmake_max_jobs  - max jobs limit set by user
461  *
462  *      External functions used:
463  *              sysconf()
464  *              getloadavg()
465  */
466 static int
467 adjust_pmake_max_jobs (int pmake_max_jobs)
468 {
469         static int      ncpu = 0;
470         double          loadavg[3];
471         int             adjustment;
472         int             adjusted_max_jobs;

474         if (ncpu <= 0) {
475                 if ((ncpu = sysconf(_SC_NPROCESSORS_ONLN)) <= 0) {
476                         ncpu = 1;
477                 }
478         }
479         if (getloadavg(loadavg, 3) != 3) return(pmake_max_jobs);
480         adjustment = ((int)loadavg[LOADAVG_1MIN]);
481         if (adjustment < 2) return(pmake_max_jobs);
482         if (ncpu > 1) {
483                 adjustment = adjustment / ncpu;
484         }
485         adjusted_max_jobs = pmake_max_jobs - adjustment;
486         if (adjusted_max_jobs < 1) adjusted_max_jobs = 1;
487         return(adjusted_max_jobs);
488 }

490 /*
491  *  M2 adjust mode data and functions
492  *
493  *  m2_init()           - initializes M2 shared semaphore
494  *  m2_acquire_job()    - decrements M2 semaphore counter
495  *  m2_release_job()    - increments M2 semaphore counter
496  *  m2_fini()           - destroys M2 semaphore and shared memory*
497  *
498  *  Environment variables:
499  *      __DMAKE_M2_FILE__
500  *
501  *  External functions:
502  *      ftok(), shmget(), shmat(), shmdt(), shmctl()
503  *      sem_init(), sem_trywait(), sem_post(), sem_destroy()
504  *      creat(), close(), unlink()
505  *      getenv(), putenv()
506  *
507  *  Static variables:
508  *      m2_file         - tmp file name to create ipc key for shared memory
509  *      m2_shm_id       - shared memory id
510  *      m2_shm_sem      - shared memory semaphore
511  */

513 static char     m2_file[MAXPATHLEN];
514 static int      m2_shm_id = -1;
515 static sem_t*   m2_shm_sem = 0;

517 static int
518 m2_init() {
519         char    *var;
520         key_t   key;

522         if ((var = getenv(NOCATGETS("__DMAKE_M2_FILE__"))) == 0) {
523                 /* compose tmp file name */
```

```
524                 sprintf(m2_file, NOCATGETS("%s/dmake.m2.%d.XXXXXX"), tmpdir, get

526                 /* create tmp file */
527                 int fd = mkstemp(m2_file);
528                 if (fd < 0) {
529                         return -1;
530                 } else {
531                         close(fd);
532                 }
533         } else {
534                 /* using existing semaphore */
535                 strcpy(m2_file, var);
536         }

538         /* combine IPC key */
539         if ((key = ftok(m2_file, 38)) == (key_t) -1) {
540                 return -1;
541         }

543         /* create shared memory */
544         if ((m2_shm_id = shmget(key, sizeof(*m2_shm_sem), 0666 | (var ? 0 : IPC_
545                 return -1;
546         }

548         /* attach shared memory */
549         if ((m2_shm_sem = (sem_t*) shmat(m2_shm_id, 0, 0666)) == (sem_t*)-1) {
550                 return -1;
551         }

553         /* root process */
554         if (var == 0) {
555                 /* initialize semaphore */
556                 if (sem_init(m2_shm_sem, 1, pmake_max_jobs)) {
557                         return -1;
558                 }

560                 /* alloc memory for env variable */
561                 if ((var = (char*) malloc(MAXPATHLEN)) == 0) {
562                         return -1;
563                 }

565                 /* put key to env */
566                 sprintf(var, NOCATGETS("__DMAKE_M2_FILE__=%s"), m2_file);
567                 if (putenv(var)) {
568                         return -1;
569                 }
570         }
571         return 0;
572 }

574 static void
575 m2_fini() {
576         if (m2_shm_id >= 0) {
577                 struct shmid_ds stat;

579                 /* determine the number of attached processes */
580                 if (shmctl(m2_shm_id, IPC_STAT, &stat) == 0) {
581                         if (stat.shm_nattch <= 1) {
582                                 /* destroy semaphore */
583                                 if (m2_shm_sem != 0) {
584                                         (void) sem_destroy(m2_shm_sem);
585                                 }

587                                 /* destroy shared memory */
588                                 (void) shmctl(m2_shm_id, IPC_RMID, &stat);
```



```
590                                 /* remove tmp file created for the key */
591                                 (void) unlink(m2_file);
592                         } else {
593                                 /* detach shared memory */
594                                 if (m2_shm_sem != 0) {
595                                         (void) shmdt((char*) m2_shm_sem);
596                                 }
597                         }
598                 }

600                 m2_shm_id = -1;
601                 m2_shm_sem = 0;
602         }
603 }

605 static int
606 m2_acquire_job() {
607         if ((m2_shm_id >= 0) && (m2_shm_sem != 0)) {
608                 if (sem_trywait(m2_shm_sem) == 0) {
609                         return 1;
610                 }
611                 if (errno == EAGAIN) {
612                         return 0;
613                 }
614         }
615         return -1;
616 }

618 static int
619 m2_release_job() {
620         if ((m2_shm_id >= 0) && (m2_shm_sem != 0)) {
621                 if (sem_post(m2_shm_sem) == 0) {
622                         return 0;
623                 }
624         }
625         return -1;
626 }

628 /*
629  *  job adjust mode
630  *
631  *  Possible values:
632  *    ADJUST_M1         - adjustment by system load (default)
633  *    ADJUST_M2         - fixed limit of jobs for the group of nested dmakes
634  *    ADJUST_NONE       - no adjustment - fixed limit of jobs for the current dm
635  */
636 static enum {
637         ADJUST_UNKNOWN,
638         ADJUST_M1,
639         ADJUST_M2,
640         ADJUST_NONE
641 } job_adjust_mode = ADJUST_UNKNOWN;

643 /*
644  *  void job_adjust_fini()
645  *
646  *  Description:
647  *      Cleans up job adjust data.
648  *
649  *  Static variables:
650  *      job_adjust_mode Current job adjust mode
651  */
652 void
653 job_adjust_fini() {
654         if (job_adjust_mode == ADJUST_M2) {
655                 m2_fini();
```

```
656         }
657 }

659 /*
660  *  void job_adjust_error()
661  *
662  *  Description:
663  *      Prints warning message, cleans up job adjust data, and disables job adju
664  *
665  *  Environment:
666  *      DMAKE_ADJUST_MAX_JOBS
667  *
668  *  External functions:
669  *      putenv()
670  *
671  *  Static variables:
672  *      job_adjust_mode Current job adjust mode
673  */
674 static void
675 job_adjust_error() {
676         if (job_adjust_mode != ADJUST_NONE) {
677                 /* cleanup internals */
678                 job_adjust_fini();

680                 /* warning message for the user */
681                 warning(catgets(catd, 1, 339, "Encountered max jobs auto adjustm

683                 /* switch off job adjustment for the children */
684                 putenv(NOCATGETS("DMAKE_ADJUST_MAX_JOBS=NO"));

686                 /* and for this dmake */
687                 job_adjust_mode = ADJUST_NONE;
688         }
689 }

691 /*
692  *  void job_adjust_init()
693  *
694  *  Description:
695  *      Parses DMAKE_ADJUST_MAX_JOBS env variable
696  *      and performs appropriate initializations.
697  *
698  *  Environment:
699  *      DMAKE_ADJUST_MAX_JOBS
700  *        DMAKE_ADJUST_MAX_JOBS == "NO" - no adjustment
701  *        DMAKE_ADJUST_MAX_JOBS == "M2" - M2 adjust mode
702  *        other                         - M1 adjust mode
703  *
704  *  External functions:
705  *      getenv()
706  *
707  *  Static variables:
708  *      job_adjust_mode Current job adjust mode
709  */
710 static void
711 job_adjust_init() {
712         if (job_adjust_mode == ADJUST_UNKNOWN) {
713                 /* default mode */
714                 job_adjust_mode = ADJUST_M1;

716                 /* determine adjust mode */
717                 if (char *var = getenv(NOCATGETS("DMAKE_ADJUST_MAX_JOBS"))) {
718                         if (strcasecmp(var, NOCATGETS("NO")) == 0) {
719                                 job_adjust_mode = ADJUST_NONE;
720                         } else if (strcasecmp(var, NOCATGETS("M2")) == 0) {
721                                 job_adjust_mode = ADJUST_M2;
```



```
722                         }
723                 }

725                 /* M2 specific initialization */
726                 if (job_adjust_mode == ADJUST_M2) {
727                         if (m2_init()) {
728                                 job_adjust_error();
729                         }
730                 }
731         }
732 }

734 #endif /* MAXJOBS_ADJUST_RFE4694000 */
735 #endif /* TEAMWARE_MAKE_CMN */

737 /*
738  *      distribute_process(char **commands, Property line)
739  *
740  *      Parameters:
741  *              commands        argv vector of commands to execute
742  *
743  *      Return value:
744  *                              The result of the execution
745  *
746  *      Static variables used:
747  *              process_running Set to the pid of the process set running
748  * #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
749  *              job_adjust_mode Current job adjust mode
750  * #endif
751  */
752 static Doname
753 distribute_process(char **commands, Property line)
754 {
755         static unsigned file_number = 0;
756         wchar_t         string[MAXPATHLEN];
757         char            mbstring[MAXPATHLEN];
758         int             filed;
759         int             res;
760         int             tmp_index;
761         char            *tmp_index_str_ptr;

763 #if !defined (TEAMWARE_MAKE_CMN) || !defined (MAXJOBS_ADJUST_RFE4694000)
764         while (parallel_process_cnt >= pmake_max_jobs) {
765                 await_parallel(false);
766                 finish_children(true);
767         }
768 #else /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
769         /* initialize adjust mode, if not initialized */
770         if (job_adjust_mode == ADJUST_UNKNOWN) {
771                 job_adjust_init();
772         }

774         /* actions depend on adjust mode */
775         switch (job_adjust_mode) {
776         case ADJUST_M1:
777                 while (parallel_process_cnt >= adjust_pmake_max_jobs (pmake_max_
778                         await_parallel(false);
779                         finish_children(true);
780                 }
781                 break;
782         case ADJUST_M2:
783                 if ((res = m2_acquire_job()) == 0) {
784                         if (parallel_process_cnt > 0) {
785                                 await_parallel(false);
786                                 finish_children(true);
```

```
788                                 if ((res = m2_acquire_job()) == 0) {
789                                         return build_serial;
790                                 }
791                         } else {
792                                 return build_serial;
793                         }
794                 }
795                 if (res < 0) {
796                         /* job adjustment error */
797                         job_adjust_error();
798 
799                         /* no adjustment */
800                         while (parallel_process_cnt >= pmake_max_jobs) {
801                                 await_parallel(false);
802                                 finish_children(true);
803                         }
804                 }
805                 break;
806         default:
807                 while (parallel_process_cnt >= pmake_max_jobs) {
808                         await_parallel(false);
809                         finish_children(true);
810                 }
811         }
812 #endif /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
813 #ifdef DISTRIBUTED
814         if (send_mtool_msgs) {
815                 send_job_start_msg(line);
816         }
817 #endif
818 #ifdef DISTRIBUTED
819         setvar_envvar((Avo_DoJobMsg *)NULL);
820 #else
821         setvar_envvar();
822 #endif
823         /*
824          * Tell the user what DMake is doing.
825          */
826         if (!silent && output_mode != txt2_mode) {
827                 /*
828                  * Print local_host --> x job(s).
829                  */
830                 (void) fprintf(stdout,
831                                 catgets(catd, 1, 325, "%s --> %d %s\n"),
832                                 local_host,
833                                 parallel_process_cnt + 1,
834                                 (parallel_process_cnt == 0) ? catgets(catd, 1, 12
835 
836                 /* Print command line(s). */
837                 tmp_index = 0;
838                 while (commands[tmp_index] != NULL) {
839                     /* No @ char. */
840                     /* XXX - need to add [2] when + prefix is added */
841                     if ((commands[tmp_index][0] != (int) at_char) &&
842                         (commands[tmp_index][1] != (int) at_char)) {
843                         tmp_index_str_ptr = commands[tmp_index];
844                         if (*tmp_index_str_ptr == (int) hyphen_char) {
845                                 tmp_index_str_ptr++;
846                         }
847                         (void) fprintf(stdout, "%s\n", tmp_index_str_ptr);
848                     }
849                     tmp_index++;
850                 }
851                 (void) fflush(stdout);
852         }
```



```
854         (void) sprintf(mbstring,
855                         NOCATGETS("%s/dmake.stdout.%d.%d.XXXXXX"),
856                         tmpdir,
857                         getpid(),
858                         file_number++);
859 
860         mktemp(mbstring);
861 
862         stdout_file = strdup(mbstring);
863         stderr_file = NULL;
864 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
865         if (!out_err_same) {
866                 (void) sprintf(mbstring,
867                                 NOCATGETS("%s/dmake.stderr.%d.%d.XXXXXX"),
868                                 tmpdir,
869                                 getpid(),
870                                 file_number++);
871 
872                 mktemp(mbstring);
873 
874                 stderr_file = strdup(mbstring);
875         }
876 #endif
877 
878 #ifdef SGE_SUPPORT
879         if (grid) {
880                 static char *dir4gridscripts = NULL;
881                 static char *hostName = NULL;
882                 if (dir4gridscripts == NULL) {
883                         Name            dmakeOdir_name, dmakeOdir_value;
884                         Property        prop;
885                         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
886                         dmakeOdir_name = GETNAME(wcs_buffer, FIND_LENGTH);
887                         if (((prop = get_prop(dmakeOdir_name->prop, macro_prop))
888                             ((dmakeOdir_value = prop->body.macro.value) != NULL)
889                                 dir4gridscripts = dmakeOdir_value->string_mb;
890                         }
891                         dir4gridscripts = Avo_PathNames::pathname_output_directo
892                         hostName = Avo_PathNames::pathname_local_host();
893                 }
894                 (void) sprintf(script_file,
895                                 NOCATGETS("%s/dmake.script.%s.%d.%d.XXXXXX"),
896                                 dir4gridscripts,
897                                 hostName,
898                                 getpid(),
899                                 file_number++);
900         }
901 #endif /* SGE_SUPPORT */
902         process_running = run_rule_commands(local_host, commands);
903 
904         return build_running;
905 }
906 
907 /*
908  *      doname_parallel(target, do_get, implicit)
909  *
910  *      Processes the given target and finishes up any parallel
911  *      processes left running.
912  *
913  *      Return value:
914  *                              Result of target build
915  *
916  *      Parameters:
917  *              target          Target to build
918  *              do_get          True if sccs get to be done
919  *              implicit        True if this is an implicit target
```

```
 */
Doname
doname_parallel(Name target, Boolean do_get, Boolean implicit)
{
        Doname          result;

        result = doname_check(target, do_get, implicit, false);
        if (result == build_ok || result == build_failed) {
                return result;
        }
        finish_running();
        return (Doname) target->state;
}

/*
 *      doname_subtree(target, do_get, implicit)
 *
 *      Completely computes an object and its dependents for a
 *      serial subtree build.
 *
 *      Parameters:
 *              target          Target to build
 *              do_get          True if sccs get to be done
 *              implicit        True if this is an implicit target
 *
 *      Static variables used:
 *              running_tail    Tail of the list of running processes
 *
 *      Global variables used:
 *              running_list    The list of running processes
 */
static void
doname_subtree(Name target, Boolean do_get, Boolean implicit)
{
        Running         save_running_list;
        Running         *save_running_tail;

        save_running_list = running_list;
        save_running_tail = running_tail;
        running_list = NULL;
        running_tail = &running_list;
        target->state = build_subtree;
        target->checking_subtree = true;
        while(doname_check(target, do_get, implicit, false) == build_running) {
                target->checking_subtree = false;
                finish_running();
                target->state = build_subtree;
        }
        target->checking_subtree = false;
        running_list = save_running_list;
        running_tail = save_running_tail;
}

/*
 *      finish_running()
 *
 *      Keeps processing until the running_list is emptied out.
 *
 *      Parameters:
 *
 *      Global variables used:
 *              running_list    The list of running processes
 */
void
finish_running(void)
{
```



```
        while (running_list != NULL) {
#ifdef DISTRIBUTED
                if (dmake_mode_type == distributed_mode) {
                        if ((just_did_subtree) ||
                            (parallel_process_cnt == 0)) {
                                just_did_subtree = false;
                        } else {
                                (void) await_dist(false);
                                finish_children(true);
                        }
                } else
#endif
                {
                        await_parallel(false);
                        finish_children(true);
                }
                if (running_list != NULL) {
                        process_next();
                }
        }
}

/*
 *      process_next()
 *
 *      Searches the running list for any targets which can start processing.
 *      This can be a pending target, a serial target, or a subtree target.
 *
 *      Parameters:
 *
 *      Static variables used:
 *              running_tail            The end of the list of running procs
 *              subtree_conflict        A target which conflicts with a subtree
 *              subtree_conflict2       The other target which conflicts
 *
 *      Global variables used:
 *              commands_done           True if commands executed
 *              debug_level             Controls debug output
 *              parallel_process_cnt    Number of parallel process running
 *              recursion_level         Indentation for debug output
 *              running_list            List of running processes
 */
static void
process_next(void)
{
        Running         rp;
        Running         *rp_prev;
        Property        line;
        Chain           target_group;
        Dependency      dep;
        Boolean         quiescent = true;
        Running         *subtree_target;
        Boolean         saved_commands_done;
        Property        *conditionals;

        subtree_target = NULL;
        subtree_conflict = NULL;
        subtree_conflict2 = NULL;
        /*
         * If nothing currently running, build a serial target, if any.
         */
start_loop_1:
        for (rp_prev = &running_list, rp = running_list;
             rp != NULL && parallel_process_cnt == 0;
             rp = rp->next) {
                if (rp->state == build_serial) {
```

```
1052                        *rp_prev = rp->next;
1053                        if (rp->next == NULL) {
1054                                running_tail = rp_prev;
1055                        }
1056                        recursion_level = rp->recursion_level;
1057                        rp->target->state = build_pending;
1058                        (void) doname_check(rp->target,
1059                                            rp->do_get,
1060                                            rp->implicit,
1061                                            false);
1062                        quiescent = false;
1063                        delete_running_struct(rp);
1064                        goto start_loop_1;
1065                } else {
1066                        rp_prev = &rp->next;
1067                }
1068        }
1069        /*
1070         * Find a target to build.  The target must be pending, have all
1071         * its dependencies built, and not be in a target group with a target
1072         * currently building.
1073         */
1074 start_loop_2:
1075        for (rp_prev = &running_list, rp = running_list;
1076             rp != NULL;
1077             rp = rp->next) {
1078                if (!(rp->state == build_pending ||
1079                      rp->state == build_subtree)) {
1080                        quiescent = false;
1081                        rp_prev = &rp->next;
1082                } else if (rp->state == build_pending) {
1083                        line = get_prop(rp->target->prop, line_prop);
1084                        for (dep = line->body.line.dependencies;
1085                             dep != NULL;
1086                             dep = dep->next) {
1087                                if (dep->name->state == build_running ||
1088                                    dep->name->state == build_pending ||
1089                                    dep->name->state == build_serial) {
1090                                        break;
1091                                }
1092                        }
1093                        if (dep == NULL) {
1094                                for (target_group = line->body.line.target_group
1095                                     target_group != NULL;
1096                                     target_group = target_group->next) {
1097                                        if (is_running(target_group->name)) {
1098                                                break;
1099                                        }
1100                                }
1101                                if (target_group == NULL) {
1102                                        *rp_prev = rp->next;
1103                                        if (rp->next == NULL) {
1104                                                running_tail = rp_prev;
1105                                        }
1106                                        recursion_level = rp->recursion_level;
1107                                        rp->target->state = rp->redo ?
1108                                          build_dont_know : build_pending;
1109                                        saved_commands_done = commands_done;
1110                                        conditionals =
1111                                                set_conditionals
1112                                                    (rp->conditional_cnt,
1113                                                     rp->conditional_targets);
1114                                        rp->target->dont_activate_cond_values =
1115                                        if ((doname_check(rp->target,
1116                                                          rp->do_get,
1117                                                          rp->implicit,
```



```
1118                                                                rp->target->has_target
1119                                                 build_running) &&
1120                                                !commands_done) {
1121                                                    commands_done =
1122                                                      saved_commands_done;
1123                                        }
1124                                        rp->target->dont_activate_cond_values =
1125                                        reset_conditionals
1126                                                (rp->conditional_cnt,
1127                                                 rp->conditional_targets,
1128                                                 conditionals);
1129                                        quiescent = false;
1130                                        delete_running_struct(rp);
1131                                        goto start_loop_2;
1132                                } else {
1133                                        rp_prev = &rp->next;
1134                                }
1135                        } else {
1136                                rp_prev = &rp->next;
1137                        }
1138                } else {
1139                        rp_prev = &rp->next;
1140                }
1141        }
1142        /*
1143         * If nothing has been found to build and there exists a subtree
1144         * target with no dependency conflicts, build it.
1145         */
1146        if (quiescent) {
1147 start_loop_3:
1148                for (rp_prev = &running_list, rp = running_list;
1149                     rp != NULL;
1150                     rp = rp->next) {
1151                        if (rp->state == build_subtree) {
1152                                if (!dependency_conflict(rp->target)) {
1153                                        *rp_prev = rp->next;
1154                                        if (rp->next == NULL) {
1155                                                running_tail = rp_prev;
1156                                        }
1157                                        recursion_level = rp->recursion_level;
1158                                        doname_subtree(rp->target,
1159                                                       rp->do_get,
1160                                                       rp->implicit);
1161 #ifdef DISTRIBUTED
1162                                        just_did_subtree = true;
1163 #endif
1164                                        quiescent = false;
1165                                        delete_running_struct(rp);
1166                                        goto start_loop_3;
1167                                } else {
1168                                        subtree_target = rp_prev;
1169                                        rp_prev = &rp->next;
1170                                }
1171                        } else {
1172                                rp_prev = &rp->next;
1173                        }
1174                }
1175        }
1176        /*
1177         * If still nothing found to build, we either have a deadlock
1178         * or a subtree with a dependency conflict with something waiting
1179         * to build.
1180         */
1181        if (quiescent) {
1182                if (subtree_target == NULL) {
1183                        fatal(catgets(catd, 1, 126, "Internal error: deadlock de
```

```
1184                 } else {
1185                         rp = *subtree_target;
1186                         if (debug_level > 0) {
1187                                 warning(catgets(catd, 1, 127, "Conditional macro
1188                                         subtree_conflict2->string_mb,
1189                                         rp->target->string_mb,
1190                                         subtree_conflict->string_mb);
1191                         }
1192                         *subtree_target = (*subtree_target)->next;
1193                         if (rp->next == NULL) {
1194                                 running_tail = subtree_target;
1195                         }
1196                         recursion_level = rp->recursion_level;
1197                         doname_subtree(rp->target, rp->do_get, rp->implicit);
1198 #ifdef DISTRIBUTED
1199                         just_did_subtree = true;
1200 #endif
1201                         delete_running_struct(rp);
1202                 }
1203         }
1204 }

1206 /*
1207  *      set_conditionals(cnt, targets)
1208  *
1209  *      Sets the conditional macros for the targets given in the array of
1210  *      targets.  The old macro values are returned in an array of
1211  *      Properties for later resetting.
1212  *
1213  *      Return value:
1214  *                                      Array of conditional macro settings
1215  *
1216  *      Parameters:
1217  *              cnt                     Number of targets
1218  *              targets                 Array of targets
1219  */
1220 static Property *
1221 set_conditionals(int cnt, Name *targets)
1222 {
1223         Property        *locals, *lp;
1224         Name            *tp;

1226         locals = (Property *) getmem(cnt * sizeof(Property));
1227         for (lp = locals, tp = targets;
1228              cnt > 0;
1229              cnt--, lp++, tp++) {
1230                 *lp = (Property) getmem((*tp)->conditional_cnt *
1231                                         sizeof(struct _Property));
1232                 set_locals(*tp, *lp);
1233         }
1234         return locals;
1235 }

1237 /*
1238  *      reset_conditionals(cnt, targets, locals)
1239  *
1240  *      Resets the conditional macros as saved in the given array of
1241  *      Properties.  The resets are done in reverse order.  Afterwards the
1242  *      data structures are freed.
1243  *
1244  *      Parameters:
1245  *              cnt                     Number of targets
1246  *              targets                 Array of targets
1247  *              locals                  Array of dependency macro settings
1248  */
1249 static void
```



```
1250 reset_conditionals(int cnt, Name *targets, Property *locals)
1251 {
1252         Name            *tp;
1253         Property        *lp;

1255         for (tp = targets + (cnt - 1), lp = locals + (cnt - 1);
1256              cnt > 0;
1257              cnt--, tp--, lp--) {
1258                 reset_locals(*tp,
1259                              *lp,
1260                              get_prop((*tp)->prop, conditional_prop),
1261                              0);
1262                 retmem_mb((caddr_t) *lp);
1263         }
1264         retmem_mb((caddr_t) locals);
1265 }

1267 /*
1268  *      dependency_conflict(target)
1269  *
1270  *      Returns true if there is an intersection between
1271  *      the subtree of the target and any dependents of the pending targets.
1272  *
1273  *      Return value:
1274  *                                      True if conflict found
1275  *
1276  *      Parameters:
1277  *              target                  Subtree target to check
1278  *
1279  *      Static variables used:
1280  *              subtree_conflict        Target conflict found
1281  *              subtree_conflict2       Second conflict found
1282  *
1283  *      Global variables used:
1284  *              running_list            List of running processes
1285  *              wait_name               .WAIT, not a real dependency
1286  */
1287 static Boolean
1288 dependency_conflict(Name target)
1289 {
1290         Property        line;
1291         Property        pending_line;
1292         Dependency      dp;
1293         Dependency      pending_dp;
1294         Running         rp;

1296         /* Return if we are already checking this target */
1297         if (target->checking_subtree) {
1298                 return false;
1299         }
1300         target->checking_subtree = true;
1301         line = get_prop(target->prop, line_prop);
1302         if (line == NULL) {
1303                 target->checking_subtree = false;
1304                 return false;
1305         }
1306         /* Check each dependency of the target for conflicts */
1307         for (dp = line->body.line.dependencies; dp != NULL; dp = dp->next) {
1308                 /* Ignore .WAIT dependency */
1309                 if (dp->name == wait_name) {
1310                         continue;
1311                 }
1312                 /*
1313                  * For each pending target, look for a dependency which
1314                  * is the same as a dependency of the subtree target.  Since
1315                  * we can't build the subtree until all pending targets have
```

```
1316                 * finished which depend on the same dependency, this is
1317                 * a conflict.
1318                 */
1319                for (rp = running_list; rp != NULL; rp = rp->next) {
1320                        if (rp->state == build_pending) {
1321                                pending_line = get_prop(rp->target->prop,
1322                                                        line_prop);
1323                                if (pending_line == NULL) {
1324                                        continue;
1325                                }
1326                                for(pending_dp = pending_line->
1327                                                        body.line.dependencies;
1328                                    pending_dp != NULL;
1329                                    pending_dp = pending_dp->next) {
1330                                        if (dp->name == pending_dp->name) {
1331                                                target->checking_subtree
1332                                                                = false;
1333                                                subtree_conflict = rp->target;
1334                                                subtree_conflict2 = dp->name;
1335                                                return true;
1336                                        }
1337                                }
1338                        }
1339                }
1340                if (dependency_conflict(dp->name)) {
1341                        target->checking_subtree = false;
1342                        return true;
1343                }
1344        }
1345        target->checking_subtree = false;
1346        return false;
1347 }

1349 /*
1350  *      await_parallel(waitflg)
1351  *
1352  *      Waits for parallel children to exit and finishes their processing.
1353  *      If waitflg is false, the function returns after update_delay.
1354  *
1355  *      Parameters:
1356  *              waitflg         dwight
1357  */
1358 void
1359 await_parallel(Boolean waitflg)
1360 {
1361 #ifdef _CHECK_UPDATE_H
1362        static int number_of_unknown_children = 0;
1363 #endif /* _CHECK_UPDATE_H */
1364        Boolean         nohang;
1365        pid_t           pid;
1366        int             status;
1367        Running         rp;
1368        int             waiterr;

1370        nohang = false;
1371        for ( ; ; ) {
1372                if (!nohang) {
1373                        (void) alarm((int) update_delay);
1374                }
1375                pid = waitpid((pid_t)-1,
1376                              &status,
1377                              nohang ? WNOHANG : 0);
1378                waiterr = errno;
1379                if (!nohang) {
1380                        (void) alarm(0);
1381                }
```



```
1382                if (pid <= 0) {
1383                        if (waiterr == EINTR) {
1384                                if (waitflg) {
1385                                        continue;
1386                                } else {
1387                                        return;
1388                                }
1389                        } else {
1390                                return;
1391                        }
1392                }
1393                for (rp = running_list;
1394                     (rp != NULL) && (rp->pid != pid);
1395                     rp = rp->next) {
1396                        ;
1397                }
1398                if (rp == NULL) {
1399 #ifdef _CHECK_UPDATE_H
1400                        /* Ignore first child - it is check_update */
1401                        if (number_of_unknown_children <= 0) {
1402                                number_of_unknown_children = 1;
1403                                return;
1404                        }
1405 #endif /* _CHECK_UPDATE_H */
1406                        if (send_mtool_msgs) {
1407                                continue;
1408                        } else {
1409                                fatal(catgets(catd, 1, 128, "Internal error: ret
1410                        }
1411                } else {
1412                        rp->state = (WIFEXITED(status) && WEXITSTATUS(status) ==
1413                }
1414                nohang = true;
1415                parallel_process_cnt--;

1417 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
1418                if (job_adjust_mode == ADJUST_M2) {
1419                        if (m2_release_job()) {
1420                                job_adjust_error();
1421                        }
1422                }
1423 #endif
1424        }
1425 }

1427 /*
1428  *      finish_children(docheck)
1429  *
1430  *      Finishes the processing for all targets which were running
1431  *      and have now completed.
1432  *
1433  *      Parameters:
1434  *              docheck         Completely check the finished target
1435  *
1436  *      Static variables used:
1437  *              running_tail    The tail of the running list
1438  *
1439  *      Global variables used:
1440  *              continue_after_error  -k flag
1441  *              fatal_in_progress  True if we are finishing up after fatal err
1442  *              running_list    List of running processes
1443  */
1444 void
1445 finish_children(Boolean docheck)
1446 {
1447        int             cmds_length;
```

```
1448         Property        line;
1449         Property        line2;
1450         struct stat     out_buf;
1451         Running         rp;
1452         Running         *rp_prev;
1453         Cmd_line        rule;
1454         Boolean         silent_flag;

1456         for (rp_prev = &running_list, rp = running_list;
1457              rp != NULL;
1458              rp = rp->next) {
1459 bypass_for_loop_inc_4:
1460                 /*
1461                  * If the state is ok or failed, then this target has
1462                  * finished building.
1463                  * In parallel_mode, output the accumulated stdout/stderr.
1464                  * Read the auto dependency_stuff, handle a failed build,
1465                  * update the target, then finish the doname process for
1466                  * that target.
1467                  */
1468                 if (rp->state == build_ok || rp->state == build_failed) {
1469                         *rp_prev = rp->next;
1470                         if (rp->next == NULL) {
1471                                 running_tail = rp_prev;
1472                         }
1473                         if ((line2 = rp->command) == NULL) {
1474                                 line2 = get_prop(rp->target->prop, line_prop);
1475                         }
1476                         if (dmake_mode_type == distributed_mode) {
1477                                 if (rp->make_refd) {
1478                                         maybe_reread_make_state();
1479                                 }
1480                         } else {
1481                                 /*
1482                                  * Send an Avo_MToolJobResultMsg to maketool.
1483                                  */
1484 #ifdef DISTRIBUTED
1485                                 if (send_mtool_msgs) {
1486                                         send_job_result_msg(rp);
1487                                 }
1488 #endif
1489                                 /*
1490                                  * Check if there were any job output
1491                                  * from the parallel build.
1492                                  */
1493                                 if (rp->stdout_file != NULL) {
1494                                         if (stat(rp->stdout_file, &out_buf) < 0)
1495                                                 fatal(catgets(catd, 1, 130, "sta
1496                                                       rp->stdout_file,
1497                                                       errmsg(errno));
1498                                         }
1499                                         if ((line2 != NULL) &&
1500                                             (out_buf.st_size > 0)) {
1501                                                 cmds_length = 0;
1502                                                 for (rule = line2->body.line.com
1503                                                      silent_flag = silent;
1504                                                      rule != NULL;
1505                                                      rule = rule->next) {
1506                                                         cmds_length += rule->com
1507                                                         silent_flag = BOOLEAN(si
1508                                                 }
1509                                                 if (out_buf.st_size != cmds_leng
1510                                                     output_mode == txt2_mode) {
1511                                                         dump_out_file(rp->stdout
1512                                                 }
1513                                         }
```



```
1514                                         (void) unlink(rp->stdout_file);
1515                                         retmem_mb(rp->stdout_file);
1516                                         rp->stdout_file = NULL;
1517                                 }
1518 #if defined(REDIRECT_ERR)
1519                                 if (!out_err_same && (rp->stderr_file != NULL))
1520                                         if (stat(rp->stderr_file, &out_buf) < 0)
1521                                                 fatal(catgets(catd, 1, 130, "sta
1522                                                       rp->stderr_file,
1523                                                       errmsg(errno));
1524                                         }
1525                                         if ((line2 != NULL) &&
1526                                             (out_buf.st_size > 0)) {
1527                                                 dump_out_file(rp->stderr_file, t
1528                                         }
1529                                         (void) unlink(rp->stderr_file);
1530                                         retmem_mb(rp->stderr_file);
1531                                         rp->stderr_file = NULL;
1532                                 }
1533 #endif
1534                         }
1535                         check_state(rp->temp_file);
1536                         if (rp->temp_file != NULL) {
1537                                 free_name(rp->temp_file);
1538                         }
1539                         rp->temp_file = NULL;
1540                         if (rp->state == build_failed) {
1541                                 line = get_prop(rp->target->prop, line_prop);
1542                                 if (line != NULL) {
1543                                         line->body.line.command_used = NULL;
1544                                 }
1545                                 if (continue_after_error ||
1546                                     fatal_in_progress ||
1547                                     !docheck) {
1548                                         warning(catgets(catd, 1, 256, "Command f
1549                                                 rp->command ? line2->body.line.t
1550                                         build_failed_seen = true;
1551                                 } else {
1552                                         /*
1553                                          * XXX??? - DMake needs to exit(),
1554                                          * but shouldn't call fatal().
1555                                          */
1556 #ifdef PRINT_EXIT_STATUS
1557                                         warning(NOCATGETS("I'm in finish_childre
1558 #endif

1560                                         fatal(catgets(catd, 1, 258, "Command fai
1561                                                 rp->command ? line2->body.line.t
1562                                 }
1563                         }
1564                         if (!docheck) {
1565                                 delete_running_struct(rp);
1566                                 rp = *rp_prev;
1567                                 if (rp == NULL) {
1568                                         break;
1569                                 } else {
1570                                         goto bypass_for_loop_inc_4;
1571                                 }
1572                         }
1573                         update_target(get_prop(rp->target->prop, line_prop),
1574                                       rp->state);
1575                         finish_doname(rp);
1576                         delete_running_struct(rp);
1577                         rp = *rp_prev;
1578                         if (rp == NULL) {
1579                                 break;
```

```
1580                        } else {
1581                                goto bypass_for_loop_inc_4;
1582                        }
1583                } else {
1584                        rp_prev = &rp->next;
1585                }
1586        }
1587 }

1589 /*
1590  *      dump_out_file(filename, err)
1591  *
1592  *      Write the contents of the file to stdout, then unlink the file.
1593  *
1594  *      Parameters:
1595  *              filename        Name of temp file containing output
1596  *
1597  *      Global variables used:
1598  */
1599 static void
1600 dump_out_file(char *filename, Boolean err)
1601 {
1602         int             chars_read;
1603         char            copybuf[BUFSIZ];
1604         int             fd;
1605         int             out_fd = (err ? 2 : 1);

1607         if ((fd = open(filename, O_RDONLY)) < 0) {
1608                 fatal(catgets(catd, 1, 141, "open failed for output file %s: %s"
1609                       filename,
1610                       errmsg(errno));
1611         }
1612         if (!silent && output_mode != txt2_mode) {
1613                 (void) fprintf(err ? stderr : stdout,
1614                                err ?
1615                                 catgets(catd, 1, 338, "%s --> Job errors\n") :
1616                                 catgets(catd, 1, 259, "%s --> Job output\n"),
1617                                local_host);
1618                 (void) fflush(err ? stderr : stdout);
1619         }
1620         for (chars_read = read(fd, copybuf, BUFSIZ);
1621              chars_read > 0;
1622              chars_read = read(fd, copybuf, BUFSIZ)) {
1623                 /*
1624                  * Read buffers from the source file until end or error.
1625                  */
1626                 if (write(out_fd, copybuf, chars_read) < 0) {
1627                         fatal(catgets(catd, 1, 260, "write failed for output fil
1628                               filename,
1629                               errmsg(errno));
1630                 }
1631         }
1632         (void) close(fd);
1633         (void) unlink(filename);
1634 }

1636 /*
1637  *      finish_doname(rp)
1638  *
1639  *      Completes the processing for a target which was left running.
1640  *
1641  *      Parameters:
1642  *              rp              Running list entry for target
1643  *
1644  *      Global variables used:
1645  *              debug_level     Debug flag
```



```
1646  *              recursion_level Indentation for debug output
1647  */
1648 static void
1649 finish_doname(Running rp)
1650 {
1651         int             auto_count = rp->auto_count;
1652         Name            *automatics = rp->automatics;
1653         Doname          result = rp->state;
1654         Name            target = rp->target;
1655         Name            true_target = rp->true_target;
1656         Property        *conditionals;

1658         recursion_level = rp->recursion_level;
1659         if (result == build_ok) {
1660                 if (true_target == NULL) {
1661                         (void) printf(NOCATGETS("Target = %s\n"), target->string
1662                         (void) printf(NOCATGETS(" State = %d\n"), result);
1663                         fatal(NOCATGETS("Internal error: NULL true_target in fin
1664                 }
1665                 /* If all went OK, set a nice timestamp */
1666                 if (true_target->stat.time == file_doesnt_exist) {
1667                         true_target->stat.time = file_max_time;
1668                 }
1669         }
1670         target->state = result;
1671         if (target->is_member) {
1672                 Property member;

1674                 /* Propagate the timestamp from the member file to the member */
1675                 if ((target->stat.time != file_max_time) &&
1676                     ((member = get_prop(target->prop, member_prop)) != NULL) &&
1677                     (exists(member->body.member.member) > file_doesnt_exist)) {
1678                         target->stat.time =
1679 /*
1680                           exists(member->body.member.member);
1681  */
1682                           member->body.member.member->stat.time;
1683                 }
1684         }
1685         /*
1686          * Check if we found any new auto dependencies when we
1687          * built the target.
1688          */
1689         if ((result == build_ok) && check_auto_dependencies(target,
1690                                                             auto_count,
1691                                                             automatics)) {
1692                 if (debug_level > 0) {
1693                         (void) printf(catgets(catd, 1, 261, "%*sTarget `%s' acqu
1694                                       recursion_level,
1695                                       "",
1696                                       true_target->string_mb);
1697                 }
1698                 target->rechecking_target = true;
1699                 target->state = build_running;

1701                 /* [tolik, Tue Mar 25 1997]
1702                  * Fix for bug 4038824:
1703                  *       command line options set by conditional macros get drop
1704                  * rp->conditional_cnt and rp->conditional_targets must be copie
1705                  * to new 'rp' during add_pending(). Set_conditionals() stores
1706                  * rp->conditional_targets to the global variable 'conditional_t
1707                  * Add_pending() will use this variable to set up 'rp'.
1708                  */
1709                 conditionals = set_conditionals(rp->conditional_cnt, rp->conditi
1710                 add_pending(target,
1711                             recursion_level,
```

```
1712                            rp->do_get,
1713                            rp->implicit,
1714                            true);
1715                    reset_conditionals(rp->conditional_cnt, rp->conditional_targets,
1716            }
1717    }

1719    /*
1720     *      new_running_struct()
1721     *
1722     *      Constructor for Running struct. Creates a structure and initializes
1723     *      its fields.
1724     *
1725     */
1726    static Running new_running_struct()
1727    {
1728            Running         rp;

1730            rp = ALLOC(Running);
1731            rp->target = NULL;
1732            rp->true_target = NULL;
1733            rp->command = NULL;
1734            rp->sprodep_value = NULL;
1735            rp->sprodep_env = NULL;
1736            rp->auto_count = 0;
1737            rp->automatics = NULL;
1738            rp->pid = -1;
1739            rp->job_msg_id = -1;
1740            rp->stdout_file = NULL;
1741            rp->stderr_file = NULL;
1742            rp->temp_file = NULL;
1743            rp->next = NULL;
1744            return rp;
1745    }

1747    /*
1748     *      add_running(target, true_target, command, recursion_level, auto_count,
1749     *                                      automatics, do_get, implicit)
1750     *
1751     *      Adds a record on the running list for this target, which
1752     *      was just spawned and is running.
1753     *
1754     *      Parameters:
1755     *              target          Target being built
1756     *              true_target     True target for target
1757     *              command         Running command.
1758     *              recursion_level Debug indentation level
1759     *              auto_count      Count of automatic dependencies
1760     *              automatics      List of automatic dependencies
1761     *              do_get          Sccs get flag
1762     *              implicit        Implicit flag
1763     *
1764     *      Static variables used:
1765     *              running_tail    Tail of running list
1766     *              process_running PID of process
1767     *
1768     *      Global variables used:
1769     *              current_line    Current line for target
1770     *              current_target  Current target being built
1771     *              stderr_file     Temporary file for stdout
1772     *              stdout_file     Temporary file for stdout
1773     *              temp_file_name  Temporary file for auto dependencies
1774     */
1775    void
1776    add_running(Name target, Name true_target, Property command, int recursion_level
1777    {
```



```
1778            Running         rp;
1779            Name            *p;

1781            rp = new_running_struct();
1782            rp->state = build_running;
1783            rp->target = target;
1784            rp->true_target = true_target;
1785            rp->command = command;
1786            Property spro_val = get_prop(sunpro_dependencies->prop, macro_prop);
1787            if(spro_val) {
1788                    rp->sprodep_value = spro_val->body.macro.value;
1789                    spro_val->body.macro.value = NULL;
1790                    spro_val = get_prop(sunpro_dependencies->prop, env_mem_prop);
1791                    if(spro_val) {
1792                            rp->sprodep_env = spro_val->body.env_mem.value;
1793                            spro_val->body.env_mem.value = NULL;
1794                    }
1795            }
1796            rp->recursion_level = recursion_level;
1797            rp->do_get = do_get;
1798            rp->implicit = implicit;
1799            rp->auto_count = auto_count;
1800            if (auto_count > 0) {
1801                    rp->automatics = (Name *) getmem(auto_count * sizeof (Name));
1802                    for (p = rp->automatics; auto_count > 0; auto_count--) {
1803                            *p++ = *automatics++;
1804                    }
1805            } else {
1806                    rp->automatics = NULL;
1807            }
1808    #ifdef DISTRIBUTED
1809            if (dmake_mode_type == distributed_mode) {
1810                    rp->make_refd = called_make;
1811                    called_make = false;
1812            } else
1813    #endif
1814            {
1815                    rp->pid = process_running;
1816                    process_running = -1;
1817                    childPid = -1;
1818            }
1819            rp->job_msg_id = job_msg_id;
1820            rp->stdout_file = stdout_file;
1821            rp->stderr_file = stderr_file;
1822            rp->temp_file = temp_file_name;
1823            rp->redo = false;
1824            rp->next = NULL;
1825            store_conditionals(rp);
1826            stdout_file = NULL;
1827            stderr_file = NULL;
1828            temp_file_name = NULL;
1829            current_target = NULL;
1830            current_line = NULL;
1831            *running_tail = rp;
1832            running_tail = &rp->next;
1833    }

1835    /*
1836     *      add_pending(target, recursion_level, do_get, implicit, redo)
1837     *
1838     *      Adds a record on the running list for a pending target
1839     *      (waiting for its dependents to finish running).
1840     *
1841     *      Parameters:
1842     *              target          Target being built
1843     *              recursion_level Debug indentation level
```

```
 *              do_get          Sccs get flag
 *              implicit        Implicit flag
 *              redo            True if this target is being redone
 *
 *      Static variables used:
 *              running_tail    Tail of running list
 */
void
add_pending(Name target, int recursion_level, Boolean do_get, Boolean implicit,
{
        Running         rp;
        rp = new_running_struct();
        rp->state = build_pending;
        rp->target = target;
        rp->recursion_level = recursion_level;
        rp->do_get = do_get;
        rp->implicit = implicit;
        rp->redo = redo;
        store_conditionals(rp);
        *running_tail = rp;
        running_tail = &rp->next;
}

/*
 *      add_serial(target, recursion_level, do_get, implicit)
 *
 *      Adds a record on the running list for a target which must be
 *      executed in serial after others have finished.
 *
 *      Parameters:
 *              target          Target being built
 *              recursion_level Debug indentation level
 *              do_get          Sccs get flag
 *              implicit        Implicit flag
 *
 *      Static variables used:
 *              running_tail    Tail of running list
 */
void
add_serial(Name target, int recursion_level, Boolean do_get, Boolean implicit)
{
        Running         rp;

        rp = new_running_struct();
        rp->target = target;
        rp->recursion_level = recursion_level;
        rp->do_get = do_get;
        rp->implicit = implicit;
        rp->state = build_serial;
        rp->redo = false;
        store_conditionals(rp);
        *running_tail = rp;
        running_tail = &rp->next;
}

/*
 *      add_subtree(target, recursion_level, do_get, implicit)
 *
 *      Adds a record on the running list for a target which must be
 *      executed in isolation after others have finished.
 *
 *      Parameters:
 *              target          Target being built
 *              recursion_level Debug indentation level
 *              do_get          Sccs get flag
 *              implicit        Implicit flag
```



```
 *
 *      Static variables used:
 *              running_tail    Tail of running list
 */
void
add_subtree(Name target, int recursion_level, Boolean do_get, Boolean implicit)
{
        Running         rp;

        rp = new_running_struct();
        rp->target = target;
        rp->recursion_level = recursion_level;
        rp->do_get = do_get;
        rp->implicit = implicit;
        rp->state = build_subtree;
        rp->redo = false;
        store_conditionals(rp);
        *running_tail = rp;
        running_tail = &rp->next;
}

/*
 *      store_conditionals(rp)
 *
 *      Creates an array of the currently active targets with conditional
 *      macros (found in the chain conditional_targets) and puts that
 *      array in the Running struct.
 *
 *      Parameters:
 *              rp              Running struct for storing chain
 *
 *      Global variables used:
 *              conditional_targets  Chain of current dynamic conditionals
 */
static void
store_conditionals(Running rp)
{
        int             cnt;
        Chain           cond_name;

        if (conditional_targets == NULL) {
                rp->conditional_cnt = 0;
                rp->conditional_targets = NULL;
                return;
        }
        cnt = 0;
        for (cond_name = conditional_targets;
             cond_name != NULL;
             cond_name = cond_name->next) {
                cnt++;
        }
        rp->conditional_cnt = cnt;
        rp->conditional_targets = (Name *) getmem(cnt * sizeof(Name));
        for (cond_name = conditional_targets;
             cond_name != NULL;
             cond_name = cond_name->next) {
                rp->conditional_targets[--cnt] = cond_name->name;
        }
}

/*
 *      parallel_ok(target, line_prop_must_exists)
 *
 *      Returns true if the target can be run in parallel
 *
 *      Return value:
```

```
 *                              True if can run in parallel
 *
 *      Parameters:
 *              target          Target being tested
 *
 *      Global variables used:
 *              all_parallel    True if all targets default to parallel
 *              only_parallel   True if no targets default to parallel
 */
Boolean
parallel_ok(Name target, Boolean line_prop_must_exists)
{
        Boolean         assign;
        Boolean         make_refd;
        Property        line;
        Cmd_line        rule;

        assign = make_refd = false;
        if (((line = get_prop(target->prop, line_prop)) == NULL) &&
            line_prop_must_exists) {
                return false;
        }
        if (line != NULL) {
                for (rule = line->body.line.command_used;
                     rule != NULL;
                     rule = rule->next) {
                        if (rule->assign) {
                                assign = true;
                        } else if (rule->make_refd) {
                                make_refd = true;
                        }
                }
        }
        if (assign) {
                return false;
        } else if (target->parallel) {
                return true;
        } else if (target->no_parallel) {
                return false;
        } else if (all_parallel) {
                return true;
        } else if (only_parallel) {
                return false;
        } else if (make_refd) {
                return false;
        } else {
                return true;
        }
}

/*
 *      is_running(target)
 *
 *      Returns true if the target is running.
 *
 *      Return value:
 *                              True if target is running
 *
 *      Parameters:
 *              target          Target to check
 *
 *      Global variables used:
 *              running_list    List of running processes
 */
Boolean
is_running(Name target)
```



```
{
        Running         rp;

        if (target->state != build_running) {
                return false;
        }
        for (rp = running_list;
             rp != NULL && target != rp->target;
             rp = rp->next);
        if (rp == NULL) {
                return false;
        } else {
                return (rp->state == build_running) ? true : false;
        }
}

/*
 * This function replaces the makesh binary.
 */

#ifdef SGE_SUPPORT
#define DO_CHECK(f)     if (f <= 0) { \
                                fprintf(stderr, \
                                        catgets(catd, 1, 347, "Could not write t
                                                script_file, errmsg(errno)); \
                                _exit(1); \
                        }
#endif /* SGE_SUPPORT */

static pid_t
run_rule_commands(char *host, char **commands)
{
        Boolean         always_exec;
        Name            command;
        Boolean         ignore;
        int             length;
        Doname          result;
        Boolean         silent_flag;
#ifdef SGE_SUPPORT
        wchar_t         *wcmd, *tmp_wcs_buffer = NULL;
        char            *cmd, *tmp_mbs_buffer = NULL;
        FILE            *scrfp;
        Name            shell = getvar(shell_name);
#else
        wchar_t         *tmp_wcs_buffer;
#endif /* SGE_SUPPORT */

        childPid = fork();
        switch (childPid) {
        case -1:        /* Error */
                fatal(catgets(catd, 1, 337, "Could not fork child process for dm
                      errmsg(errno));
                break;
        case 0:         /* Child */
                /* To control the processed targets list is not the child's busi
                running_list = NULL;
#if defined(REDIRECT_ERR)
                if(out_err_same) {
                        redirect_io(stdout_file, (char*)NULL);
                } else {
                        redirect_io(stdout_file, stderr_file);
                }
#else
                redirect_io(stdout_file, (char*)NULL);
#endif
#ifdef SGE_SUPPORT
```

```
2108                 if (grid) {
2109                         int fdes = mkstemp(script_file);
2110                         if ((fdes < 0) || (scrfp = fdopen(fdes, "w")) == NULL) {
2111                                 fprintf(stderr,
2112                                         catgets(catd, 1, 341, "Could not create
2113                                         script_file, errmsg(errno));
2114                                 _exit(1);
2115                         }
2116                         if (IS_EQUAL(shell->string_mb, "")) {
2117                                 shell = shell_name;
2118                         }
2119                 }
2120 #endif /* SGE_SUPPORT */
2121                 for (commands = commands;
2122                      (*commands != (char *)NULL);
2123                      commands++) {
2124                         silent_flag = silent;
2125                         ignore = false;
2126                         always_exec = false;
2127                         while ((**commands == (int) at_char) ||
2128                                (**commands == (int) hyphen_char) ||
2129                                (**commands == (int) plus_char)) {
2130                                 if (**commands == (int) at_char) {
2131                                         silent_flag = true;
2132                                 }
2133                                 if (**commands == (int) hyphen_char) {
2134                                         ignore = true;
2135                                 }
2136                                 if (**commands == (int) plus_char) {
2137                                         always_exec = true;
2138                                 }
2139                                 (*commands)++;
2140                         }
2141 #ifdef SGE_SUPPORT
2142                         if (grid) {
2143                                 if ((length = strlen(*commands)) >= MAXPATHLEN /
2144                                         wcmd = tmp_wcs_buffer = ALLOC_WC(length
2145                                         (void) mbstowcs(tmp_wcs_buffer, *command
2146                                 } else {
2147                                         MBSTOWCS(wcs_buffer, *commands);
2148                                         wcmd = wcs_buffer;
2149                                         cmd = mbs_buffer;
2150                                 }
2151                                 wchar_t *from = wcmd + wslen(wcmd);
2152                                 wchar_t *to = from + (from - wcmd);
2153                                 *to = (int) nul_char;
2154                                 while (from > wcmd) {
2155                                         *--to = *--from;
2156                                         if (*from == (int) newline_char) { // ne
2157                                                 *--to = *--from;
2158                                         } else if (wschr(char_semantics_char, *f
2159                                                 *--to = (int) backslash_char;
2160                                         }
2161                                 }
2162                                 if (length >= MAXPATHLEN*MB_LEN_MAX/2) { // size
2163                                         cmd = tmp_mbs_buffer = getmem((length *
2164                                         (void) wcstombs(tmp_mbs_buffer, to, (len
2165                                 } else {
2166                                         WCSTOMBS(mbs_buffer, to);
2167                                         cmd = mbs_buffer;
2168                                 }
2169                                 char *mbst, *mbend;
2170                                 if ((length > 0) &&
2171                                     !silent_flag) {
2172                                         for (mbst = cmd; (mbend = strstr(mbst, "
2173                                                 *mbend = '\0';
```



```
2174                                                 DO_CHECK(fprintf(scrfp, NOCATGET
2175                                                 *mbend = '\\';
2176                                         }
2177                                         DO_CHECK(fprintf(scrfp, NOCATGETS("/usr/
2178                                 }
2179                                 if (!do_not_exec_rule ||
2180                                     !working_on_targets ||
2181                                     always_exec) {
2182 #if defined(linux)
2183                                         if (0 != strcmp(shell->string_mb, (char*
2184                                                 DO_CHECK(fprintf(scrfp, NOCATGET
2185                                         } else
2186 #endif
2187                                         DO_CHECK(fprintf(scrfp, NOCATGETS("%s -c
2188                                         DO_CHECK(fputs(NOCATGETS("__DMAKECMDEXIT
2189                                         if (ignore) {
2190                                                 DO_CHECK(fprintf(scrfp, NOCATGET
2191                                                         catgets(catd, 1, 343, "\
2192                                                         catgets(catd, 1, 344, "(
2193                                         } else {
2194                                                 DO_CHECK(fprintf(scrfp, NOCATGET
2195                                                         catgets(catd, 1, 342, "\
2196                                         }
2197                                         if (silent_flag) {
2198                                                 DO_CHECK(fprintf(scrfp, NOCATGET
2199                                                         catgets(catd, 1, 345, "T
2200                                                 for (mbst = cmd; (mbend = strstr
2201                                                         *mbend = '\0';
2202                                                         DO_CHECK(fprintf(scrfp,
2203                                                         *mbend = '\\';
2204                                                 }
2205                                                 DO_CHECK(fprintf(scrfp, NOCATGET
2206                                         }
2207                                         if (!ignore) {
2208                                                 DO_CHECK(fputs(NOCATGETS("\texit
2209                                         }
2210                                         DO_CHECK(fputs(NOCATGETS("fi\n"), scrfp)
2211                                 }
2212                                 if (tmp_wcs_buffer) {
2213                                         retmem_mb(tmp_mbs_buffer);
2214                                         tmp_mbs_buffer = NULL;
2215                                 }
2216                                 if (tmp_wcs_buffer) {
2217                                         retmem(tmp_wcs_buffer);
2218                                         tmp_wcs_buffer = NULL;
2219                                 }
2220                                 continue;
2221                         }
2222 #endif /* SGE_SUPPORT */
2223                         if ((length = strlen(*commands)) >= MAXPATHLEN) {
2224                                 tmp_wcs_buffer = ALLOC_WC(length + 1);
2225                                 (void) mbstowcs(tmp_wcs_buffer, *commands, lengt
2226                                 command = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2227                                 retmem(tmp_wcs_buffer);
2228                         } else {
2229                                 MBSTOWCS(wcs_buffer, *commands);
2230                                 command = GETNAME(wcs_buffer, FIND_LENGTH);
2231                         }
2232                         if ((command->hash.length > 0) &&
2233                             !silent_flag) {
2234                                 (void) printf("%s\n", command->string_mb);
2235                         }
2236                         result = dosys(command,
2237                                        ignore,
2238                                        false,
2239                                        false, /* bugs #4085164 & #4990057 */
```

```
2240                                    /* BOOLEAN(silent_flag && ignore), */
2241                                    always_exec,
2242                                    (Name) NULL,
2243                                    false);
2244                            if (result == build_failed) {
2245                                    if (silent_flag) {
2246                                            (void) printf(catgets(catd, 1, 152, "The
2247                                    }
2248                                    if (!ignore) {
2249                                            _exit(1);
2250                                    }
2251                            }
2252                    }
2253 #ifndef SGE_SUPPORT
2254                    _exit(0);
2255 #else
2256                    if (!grid) {
2257                            _exit(0);
2258                    }
2259                    DO_CHECK(fputs(NOCATGETS("exit 0\n"), scrfp));
2260                    if (fclose(scrfp) != 0) {
2261                            fprintf(stderr,
2262                            catgets(catd, 1, 346, "Could not close file: %s:
2263                            script_file, errmsg(errno));
2264                            _exit(1);
2265                    }
2266                    {

2268 #define DEFAULT_QRSH_TRIES_NUMBER       1
2269 #define DEFAULT_QRSH_TIMEOUT            0

2271                    static char     *sge_env_var = NULL;
2272                    static int      qrsh_tries_number = DEFAULT_QRSH_TRIES_NUMBER;
2273                    static int      qrsh_timeout = DEFAULT_QRSH_TIMEOUT;
2274 #define SGE_DEBUG
2275 #ifdef SGE_DEBUG
2276                    static  Boolean do_not_remove = false;
2277 #endif /* SGE_DEBUG */
2278                    if (sge_env_var == NULL) {
2279                            sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_TRIES")
2280                            if (sge_env_var != NULL) {
2281                                    qrsh_tries_number = atoi(sge_env_var);
2282                                    if (qrsh_tries_number < 1 || qrsh_tries_number >
2283                                            qrsh_tries_number = DEFAULT_QRSH_TRIES_N
2284                                    }
2285                            }
2286                            sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_TIMEOUT
2287                            if (sge_env_var != NULL) {
2288                                    qrsh_timeout = atoi(sge_env_var);
2289                                    if (qrsh_timeout <= 0) {
2290                                            qrsh_timeout = DEFAULT_QRSH_TIMEOUT;
2291                                    }
2292                            } else {
2293                                    sge_env_var = "";
2294                            }
2295 #ifdef SGE_DEBUG
2296                            sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_DEBUG")
2297                            if (sge_env_var == NULL) {
2298                                    sge_env_var = "";
2299                            }
2300                            if (strstr(sge_env_var, NOCATGETS("noqrsh")) != NULL)
2301                                    qrsh_tries_number = 0;
2302                            if (strstr(sge_env_var, NOCATGETS("donotremove")) != NUL
2303                                    do_not_remove = true;
2304 #endif /* SGE_DEBUG */
2305                    }
```



```
2306                    for (int i = qrsh_tries_number; ; i--)
2307                    if ((childPid = fork()) < 0) {
2308                            fatal(catgets(catd, 1, 348, "Could not fork child proces
2309                                    errmsg(errno));
2310                            _exit(1);
2311                    } else if (childPid == 0) {
2312                            enable_interrupt((void (*) (int))SIG_DFL);
2313                            if (i > 0) {
2314                                    static char qrsh_cmd[50+MAXPATHLEN] = NOCATGETS(
2315                                    static char *fname_ptr = NULL;
2316                                    static char *argv[] = { NOCATGETS("sh"),
2317                                                            NOCATGETS("-fce"),
2318                                                            qrsh_cmd,
2319                                                            NULL};
2320                                    if (fname_ptr == NULL) {
2321                                            fname_ptr = qrsh_cmd + strlen(qrsh_cmd);
2322                                    }
2323                                    strcpy(fname_ptr, script_file);
2324                                    (void) execve(NOCATGETS("/bin/sh"), argv, enviro
2325                            } else {
2326                                    static char *argv[] = { NOCATGETS("sh"),
2327                                                            script_file,
2328                                                            NULL};
2329                                    (void) execve(NOCATGETS("/bin/sh"), argv, enviro
2330                            }
2331                            fprintf(stderr,
2332                                    catgets(catd, 1, 349, "Could not load ‘qrsh’: %s
2333                                    errmsg(errno));
2334                            _exit(1);
2335                    } else {
2336 #if defined (HP_UX) || defined (linux) || defined (SUN5_0)
2337                            int             status;
2338 #else
2339                            union wait      status;
2340 #endif
2341                            pid_t pid;
2342                            while ((pid = wait(&status)) != childPid) {
2343                                    if (pid == -1) {
2344                                            fprintf(stderr,
2345                                                    catgets(catd, 1, 350, "wait() fa
2346                                                    errmsg(errno));
2347                                            _exit(1);
2348                                    }
2349                            }
2350                            if (status != 0 && i > 0) {
2351                                    if (i > 1) {
2352                                            sleep(qrsh_timeout);
2353                                    }
2354                                    continue;
2355                            }
2356 #ifdef SGE_DEBUG
2357                            if (do_not_remove) {
2358                                    if (status) {
2359                                            fprintf(stderr,
2360                                                    NOCATGETS("SGE script failed: %s
2361                                                    script_file);
2362                                    }
2363                                    _exit(status ? 1 : 0);
2364                            }
2365 #endif /* SGE_DEBUG */
2366                            (void) unlink(script_file);
2367                            _exit(status ? 1 : 0);
2368                    }
2369                    }
2370 #endif /* SGE_SUPPORT */
2371                    break;
```

```
2372         default:
2373                 break;
2374         }
2375         return childPid;
2376 }

2378 static void
2379 maybe_reread_make_state(void)
2380 {
2381         /* Copying dosys()... */
2382         if (report_dependencies_level == 0) {
2383                 make_state->stat.time = file_no_time;
2384                 (void) exists(make_state);
2385                 if (make_state_before == make_state->stat.time) {
2386                         return;
2387                 }
2388                 makefile_type = reading_statefile;
2389                 if (read_trace_level > 1) {
2390                         trace_reader = true;
2391                 }
2392                 temp_file_number++;
2393                 (void) read_simple_file(make_state,
2394                                         false,
2395                                         false,
2396                                         false,
2397                                         false,
2398                                         false,
2399                                         true);
2400                 trace_reader = false;
2401         }
2402 }

2404 #ifdef DISTRIBUTED
2405 /*
2406  * Create and send an Avo_MToolJobResultMsg.
2407  */
2408 static void
2409 send_job_result_msg(Running rp)
2410 {
2411         Avo_MToolJobResultMsg   *msg;
2412         RWCollectable           *xdr_msg;

2414         msg = new Avo_MToolJobResultMsg();
2415         msg->setResult(rp->job_msg_id,
2416                        (rp->state == build_ok) ? 0 : 1,
2417                        DONE);
2418         append_job_result_msg(msg,
2419                               rp->stdout_file,
2420                               rp->stderr_file);

2422         xdr_msg = (RWCollectable *)msg;
2423         xdr(get_xdrs_ptr(), xdr_msg);
2424         (void) fflush(get_mtool_msgs_fp());

2426         delete msg;
2427 }

2429 /*
2430  * Append the stdout/err to Avo_MToolJobResultMsg.
2431  */
2432 static void
2433 append_job_result_msg(Avo_MToolJobResultMsg *msg, char *outFn, char *errFn)
2434 {
2435         FILE            *fp;
2436         char            line[MAXPATHLEN];
```



```
2438         fp = fopen(outFn, "r");
2439         if (fp == NULL) {
2440                 /* Hmmm... what should we do here? */
2441                 return;
2442         }
2443         while (fgets(line, MAXPATHLEN, fp) != NULL) {
2444                 if (line[strlen(line) - 1] == '\n') {
2445                         line[strlen(line) - 1] = '\0';
2446                 }
2447                 msg->appendOutput(AVO_STRDUP(line));
2448         }
2449         (void) fclose(fp);
2450 }
2451 #endif

2453 static void
2454 delete_running_struct(Running rp)
2455 {
2456         if ((rp->conditional_cnt > 0) &&
2457             (rp->conditional_targets != NULL)) {
2458                 retmem_mb((char *) rp->conditional_targets);
2459         }
2460 /**/
2461         if ((rp->auto_count > 0) &&
2462             (rp->automatics != NULL)) {
2463                 retmem_mb((char *) rp->automatics);
2464         }
2465 /**/
2466         if(rp->sprodep_value) {
2467                 free_name(rp->sprodep_value);
2468         }
2469         if(rp->sprodep_env) {
2470                 retmem_mb(rp->sprodep_env);
2471         }
2472         retmem_mb((char *) rp);

2474 }

2476 #endif

2478 #endif /* ! codereview */
```

```
**********************************************************
   11515 Wed May 20 11:04:10 2015
new/usr/src/cmd/make/bin/make/common/pmake.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)pmake.cc 1.9 06/12/12
 */

#pragma ident   "@(#)pmake.cc   1.9     06/12/12"

#ifdef TEAMWARE_MAKE_CMN

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

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
```



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


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


        MBSTOWCS(wcs_buffer, NOCATGETS("MAKE_MACHINES"));
        MAKE_MACHINES = GETNAME(wcs_buffer, FIND_LENGTH);
        /* Did the user specify a .make.machines file on the command line? */
        default_make_machines = false;
        if (make_machines_name == NULL) {
                /* Try reading the default .make.machines file, in $(HOME). */
                homedir = getenv(NOCATGETS("HOME"));
                if ((homedir != NULL) && (strlen(homedir) < (sizeof(mb_make_mach
                        sprintf(mb_make_machines_path,
                         NOCATGETS("%s/.make.machines"), homedir);
                        MBSTOWCS(make_machines_path, mb_make_machines_path);
                        make_machines_name = GETNAME(make_machines_path, FIND_LE
                        default_make_machines = true;
                }
                if (make_machines_name == NULL) {
                        /*
                         * No $(HOME)/.make.machines file.
                         * Return 0 for PMake max jobs.
```

```
128                          */
129                         return(0);
130                 }
131         }
132 /*
133         make_machines_list_mb = getenv(MAKE_MACHINES->string_mb);
134  */
135         /* Open the .make.machines file. */
136         if ((make_machines_file = fopen(make_machines_name->string_mb, "r")) ==
137                 if (!default_make_machines) {
138                         /* Error opening .make.machines file. */
139                         fatal(catgets(catd, 1, 314, "Open of %s failed: %s"),
140                               make_machines_name->string_mb,
141                               errmsg(errno));
142                 } else {
143                         /*
144                          * No $(HOME)/.make.machines file.
145                          * Return 0 for PMake max jobs.
146                          */
147                         return(0);
148                 }
149         /* Stat the .make.machines file to get the size of the file.  */
150         } else if (fstat(fileno(make_machines_file), &make_machines_buf) < 0) {
151                 /* Error stat'ing .make.machines file. */
152                 fatal(catgets(catd, 1, 315, "Stat of %s failed: %s"),
153                       make_machines_name->string_mb,
154                       errmsg(errno));
155         } else {
156                 /* Allocate memory for "MAKE_MACHINES=<contents of .m.m>" */
157                 make_machines_list_mb =
158                   (char *) getmem((int) (strlen(MAKE_MACHINES->string_mb) +
159                                          2 +
160                                          make_machines_buf.st_size));
161                 sprintf(make_machines_list_mb,
162                         "%s=",
163                         MAKE_MACHINES->string_mb);
164                 /* Read in the .make.machines file. */
165                 if (fread(make_machines_list_mb + strlen(MAKE_MACHINES->string_m
166                           sizeof(char),
167                           (int) make_machines_buf.st_size,
168                           make_machines_file) != make_machines_buf.st_size) {
169                         /*
170                          * Error reading .make.machines file.
171                          * Return 0 for PMake max jobs.
172                          */
173                         warning(catgets(catd, 1, 316, "Unable to read %s"),
174                                 make_machines_name->string_mb);
175                         (void) fclose(make_machines_file);
176                         retmem_mb((caddr_t) make_machines_list_mb);
177                         return(0);
178                 } else {
179                         (void) fclose(make_machines_file);
180                         /* putenv "MAKE_MACHINES=<contents of .m.m>" */
181                         *(make_machines_list_mb +
182                           strlen(MAKE_MACHINES->string_mb) +
183                           1 +
184                           make_machines_buf.st_size) = (int) nul_char;
185                         if (putenv(make_machines_list_mb) != 0) {
186                                 warning(catgets(catd, 1, 317, "Couldn't put cont
187                                         make_machines_name->string_mb);
188                         } else {
189                                 make_machines_list_mb += strlen(MAKE_MACHINES->s
190                                 make_machines_list = ALLOC_WC(strlen(make_machin
191                                 (void) mbstowcs(make_machines_list,
192                                                 make_machines_list_mb,
193                                                 (strlen(make_machines_list_mb) +
```



```
194                         }
195                 }
196         }

198         uname(&uts_info);
199         strcpy(local_host_mb, &uts_info.nodename[0]);
200         MBSTOWCS(local_host, local_host_mb);
201         local_host_wslen = wslen(local_host);

203         // There is no getdomainname() function on Solaris.
204         // And netname2host() function does not work on Linux.
205         // So we have to use different APIs.
206 #ifdef linux
207         if (getdomainname(mbs_buffer, MAXNETNAMELEN+1) == 0) {
208                 sprintf(mbs_buffer2, "%s.%s", local_host_mb, mbs_buffer);
209 #else
210         if (host2netname(mbs_buffer, NULL, NULL) &&
211             netname2host(mbs_buffer, mbs_buffer2, MAXNETNAMELEN+1)) {
212 #endif
213                 MBSTOWCS(full_host, mbs_buffer2);
214                 full_host_wslen = wslen(full_host);
215         }

217         for (ms = make_machines_list;
218              (ms) && (*ms );
219              ) {
220                 /*
221                  * Skip white space and comments till you reach
222                  * a machine name.
223                  */
224                 pskip_till_next_word(&ms);

226                 /*
227                  * If we haven't reached the end of file, process the
228                  * machine name.
229                  */
230                 if (*ms) {
231                         /*
232                          * If invalid machine name decrement counter
233                          * and skip line.
234                          */
235                         mp = ms;
236                         SKIPWORD(ms);
237                         c = *ms;
238                         *ms++ = '\0'; /* Append null to machine name. */
239                         /*
240                          * If this was the beginning of a comment
241                          * (we overwrote a # sign) and it's not
242                          * end of line yet, shift the # sign.
243                          */
244                         if ((c == '#') && (*ms != '\n') && (*ms)) {
245                                 *ms = '#';
246                         }
247                         WCSTOMBS(mbs_buffer, mp);
248                         /*
249                          * Print "Ignoring unknown host" if:
250                          * 1) hostname is longer than MAX_HOSTNAMELEN, or
251                          * 2) hostname is unknown
252                          */
253                         if ((wslen(mp) > MAX_HOSTNAMELEN) ||
254                             ((hp = gethostbyname(mbs_buffer)) == NULL)) {
255                                 warning(catgets(catd, 1, 318, "Ignoring unknown
256                                         mbs_buffer);
257                                 SKIPTOEND(ms);
258                                 /* Increment ptr if not end of file. */
259                                 if (*ms) {
```

```
260                                     ms++;
261                             }
262                     } else {
263                             /* Compare current hostname with local_host. */
264                             if (wslen(mp) == local_host_wslen &&
265                                 IS_WEQUALN(mp, local_host, local_host_wslen)
266                                     /*
267                                      * Bingo, local_host is in .make.machine
268                                      * Continue reading.
269                                      */
270                                     pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
271                             /* Compare current hostname with full_host. */
272                             } else if (wslen(mp) == full_host_wslen &&
273                                        IS_WEQUALN(mp, full_host, full_host_w
274                                     /*
275                                      * Bingo, full_host is in .make.machines
276                                      * Continue reading.
277                                      */
278                                     pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
279                             } else {
280                                     if (c != '\n') {
281                                         SKIPTOEND(ms);
282                                         if (*ms) {
283                                             ms++;
284                                         }
285                                     }
286                                     continue;
287                             }
288                             /* If we get here, local_host is in .make.machin
289                             if (c != '\n')  {
290                                     /* Now look for keyword 'max'. */
291                                     MBSTOWCS(wcs_buffer, NOCATGETS("max"));
292                                     SKIPSPACE(ms);
293                                     while ((*ms != '\n') && (*ms)) {
294                                             if (*ms == '#') {
295                                                     pskip_comment(&ms);
296                                             } else if (IS_WEQUALN(ms, wcs_bu
297                                                     /* Skip "max". */
298                                                     ms += 3;
299                                                     pmake_max_jobs = get_max
300                                                     SKIPSPACE(ms);
301                                             } else {
302                                                     warning(catgets(catd, 1,
303                                                     SKIPTOEND(ms);
304                                                     break;
305                                             }
306                                     }
307                             }
308                             break; /* out of outermost for() loop. */
309                     }
310             }
311         }
312         retmem(make_machines_list);
313         return(pmake_max_jobs);
314 }

315 /*
316  *      pskip_till_next_word(cp)
317  *
318  *      Parameters:
319  *              cp              the address of the string pointer.
320  *
321  *      On return:
322  *              cp              points to beginning of machine name.
323  *
324  *
325  */
```



```
326 static void
327 pskip_till_next_word(wchar_t **cp)
328 {
329         /*
330          * Keep recursing until all combinations of white spaces
331          * and comments have been skipped.
332          */
333         if (pskip_white_space(cp) || pskip_comment(cp)) {
334                 pskip_till_next_word(cp);
335         }
336 }

337 /*
338  *      pskip_white_space(cp_address)
339  *
340  *      Advances the string pointer so that it points to the first
341  *      non white character (space/tab/linefeed).
342  *
343  *      Parameters:
344  *              cp_address      the address of the string pointer.
345  *
346  *      Return Value:
347  *                              True if the pointer was changed.
348  *
349  */
350 static Boolean
351 pskip_white_space(wchar_t **cp_address)
352 {
353         wchar_t         *cp = *cp_address;

354         while (*cp && iswspace(*cp)) {
355                 cp++;
356         }
357         /* Have we skipped any characters? */
358         if (cp != *cp_address) {
359                 *cp_address = cp;
360                 return(true);
361         } else {
362                 return(false);
363         }
364 }

365 /*
366  *      pskip_comment(cp_address)
367  *
368  *      If cp_address is pointing to '#' (the beginning of a comment),
369  *      increment the pointer till you reach end of line.
370  *
371  *      Parameters:
372  *              cp_address      the address of the string pointer.
373  *
374  *      Return Value:
375  *                              True if the pointer was changed.
376  *
377  */
378 static Boolean
379 pskip_comment(wchar_t **cp_address)
380 {
381         wchar_t         *cp = *cp_address;

382         /* Is this the beginning of a comment? Skip till end of line. */
383         if (*cp == '#') {
384                 SKIPTOEND(cp);
385         }
386         /* Have we skipped a comment line? */
387         if (cp != *cp_address)  {
388                 return(true);
389         } else {
390                 return(false);
391         }
```

```
392                 *cp_address = cp;
393                 return(true);
394         } else {
395                 return(false);
396         }
397 }

399 static int
400 get_max(wchar_t **ms_address, wchar_t *hostname)
401 {
402         wchar_t         *ms = *ms_address;
403         int             limit = PMAKE_DEF_MAX_JOBS; /* Default setting. */

405         WCSTOMBS(mbs_buffer, hostname);
406         /* Look for '='. */
407         SKIPSPACE(ms);
408         if ((!*ms) || (*ms == '\n') || (*ms != '=')) {
409                 SKIPTOEND(ms);
410                 warning(catgets(catd, 1, 319, "expected '=' after max, ignoring
411                         mbs_buffer);
412                 *ms_address = ms;
413                 return((int) limit);
414         } else {
415                 ms++;
416                 SKIPSPACE(ms);
417                 if ((*ms != '\n') && (*ms != '\0')) {
418                         /* We've found, hopefully, a valid "max" value. */
419                         limit = (int) wcstol(ms, &ms, 10);
420                         if (limit < 1) {
421                                 limit = PMAKE_DEF_MAX_JOBS;
422                                 warning(catgets(catd, 1, 320, "max value cannot
423                         }
424                 } else {
425                         /* No "max" value after "max=". */
426                         warning(catgets(catd, 1, 321, "no max value specified fo
427                 }
428                 *ms_address = ms;
429                 return(limit);
430         }
431 }

433 #endif

435 #endif /* ! codereview */
```

```
**********************************************************
   58689 Wed May 20 11:04:10 2015
new/usr/src/cmd/make/bin/make/common/read.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)read.cc 1.64 06/12/12
 */

#pragma ident   "@(#)read.cc    1.64    06/12/12"

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

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

#if defined(HP_UX) || defined(linux)
#include <avo/types.h>
extern "C" Avo_err *avo_find_run_dir(char **dirp);
#endif

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
```



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

/*
 * File table of contents
 */
static  void            parse_makefile(register Name true_makefile_name, registe
static  Source          push_macro_value(register Source bp, register wchar_t *b
extern  void            enter_target_groups_and_dependencies(Name_vector target,
extern  Name            normalize_name(register wchar_t *name_string, register i

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


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


#if defined(HP_UX) || defined(linux)
        Avo_err                 *findrundir_err;
        char                    *run_dir, makerules_dir[BUFSIZ];
```

```
#endif

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

#ifdef NSE
        if (report_file){
                wscpy(previous_current_makefile, current_makefile);
                wscpy(current_makefile, wcb);
        }
#endif
        if (max_include_depth++ >= 40) {
                fatal(catgets(catd, 1, 66, "Too many nested include statements")
        }
        if (makefile->body.makefile.contents != NULL) {
                retmem(makefile->body.makefile.contents);
        }
        source->inp_buf =
          source->inp_buf_ptr =
            source->inp_buf_end = NULL;
        source->error_converting = false;
        makefile->body.makefile.contents = NULL;
        makefile->body.makefile.size = 0;
        if ((makefile_name->hash.length != 1) ||
            (wcb[0] != (int) hyphen_char)) {
                if ((makefile->body.makefile.contents == NULL) &&
                    (doname_it)) {
                        if (makefile_path == NULL) {
                                add_dir_to_path(".",
                                                &makefile_path,
                                                -1);
#ifdef SUN5_0
                                add_dir_to_path(NOCATGETS("/usr/share/lib/make")
                                                &makefile_path,
                                                -1);
                                add_dir_to_path(NOCATGETS("/etc/default"),
                                                &makefile_path,
                                                -1);
#elif defined(HP_UX)
                                findrundir_err = avo_find_run_dir(&run_dir);
                                if (! findrundir_err) {
                                        (void) sprintf(makerules_dir, NOCATGETS(
                                        add_dir_to_path(makerules_dir,
                                                        &makefile_path,
                                                        -1);
                                }

                                add_dir_to_path(NOCATGETS("/opt/SUNWspro/share/l
                                                &makefile_path,
                                                -1);
                                add_dir_to_path(NOCATGETS("/usr/share/lib/make")
                                                &makefile_path,
                                                -1);
#elif defined(linux)
                                findrundir_err = avo_find_run_dir(&run_dir);
                                if (! findrundir_err) {
                                        (void) sprintf(makerules_dir, NOCATGETS(
                                        add_dir_to_path(makerules_dir,
                                                        &makefile_path,
                                                        -1);
                                }

                                add_dir_to_path(NOCATGETS("/usr/SUNWspro/lib"),
                                                &makefile_path,
                                                -1);
                                add_dir_to_path(NOCATGETS("/opt/SUNWspro/share/l
                                                &makefile_path,
```



```
                                                -1);
                                add_dir_to_path(NOCATGETS("/usr/share/lib/make")
                                                &makefile_path,
                                                -1);
#else
                                add_dir_to_path(NOCATGETS("/usr/include/make"),
                                                &makefile_path,
                                                -1);
#endif
                        }
                        save_makefile_type = makefile_type;
                        makefile_type = reading_nothing;
                        if (doname(makefile_name, true, false) == build_dont_kno
                                /* Try normalized filename */
                                string_start=get_wstring(makefile_name->string_m
                                for (string_end=string_start+1; *string_end != N
                                normalized_makefile_name=normalize_name(string_s
                                if ((strcmp(makefile_name->string_mb, normalized
                                        (doname(normalized_makefile_name, true,
                                        n = access_vroot(makefile_name->string_m
                                                 4,
                                                 chase_path ?
                                                 makefile_path : NULL,
                                                 VROOT_DEFAULT);
                                        if (n == 0) {
                                                get_vroot_path((char **) NULL,
                                                       &path,
                                                       (char **) NULL);
                                                if ((path[0] == (int) period_cha
                                                    (path[1] == (int) slash_char
                                                        path += 2;
                                                }
                                                MBSTOWCS(wcs_buffer, path);
                                                makefile_name = GETNAME(wcs_buff
                                                                FIND_LENGTH);
                                        }
                                }
                                retmem(string_start);
                                /*
                                 * Commented out: retmem_mb(normalized_makefile_
                                 * We have to return this memory, but it seems t
                                 * in dmake or in Sun C++ 5.7 compiler (it works
                                 * is compiled using Sun C++ 5.6).
                                 */
                                // retmem_mb(normalized_makefile_name->string_mb
                        }
                        makefile_type = save_makefile_type;
                }
                source->string.free_after_use = false;
                source->previous = NULL;
                source->already_expanded = false;
                /* Lock the file for read, but not when -n. */
                if (lock_makefile &&
                    !do_not_exec_rule) {

                        make_state_lockfile = getmem(strlen(make_state->string_
                        (void) sprintf(make_state_lockfile,
                                                NOCATGETS("%s.lock"),
                                                make_state->string_mb);
                        (void) file_lock(make_state->string_mb,
                                         make_state_lockfile,
                                         (int *) &make_state_locked,
                                         0);
                        if(!make_state_locked) {
                                printf(NOCATGETS("-- NO LOCKING for read\n"));
                                retmem_mb(make_state_lockfile);
```

```
260                                 make_state_lockfile = 0;
261                                 return failed;
262                         }
263                 }
264                 if (makefile->body.makefile.contents == NULL) {
265                         save_makefile_type = makefile_type;
266                         makefile_type = reading_nothing;
267                         if ((doname_it) &&
268                             (doname(makefile_name, true, false) == build_failed)
269                                 if (complain) {
270                                         (void) fprintf(stderr,
271 #ifdef DISTRIBUTED
272                                                        catgets(catd, 1, 67, "dma
273 #else
274                                                        catgets(catd, 1, 237, "ma
275 #endif
276                                                        makefile_name->string_mb)
277                                 }
278                                 max_include_depth--;
279                                 makefile_type = save_makefile_type;
280                                 return failed;
281                         }
282                         makefile_type = save_makefile_type;
283                         //
284                         // Before calling exists() make sure that we have the ri
285                         //
286                         makefile_name->stat.time = file_no_time;

288                         if (exists(makefile_name) == file_doesnt_exist) {
289                                 if (complain ||
290                                     (makefile_name->stat.stat_errno != ENOENT))
291                                         if (must_exist) {
292                                                 fatal(catgets(catd, 1, 68, "Can'
293                                                       makefile_name->string_mb,
294                                                       errmsg(makefile_name->
295                                                              stat.stat_errno));
296                                         } else {
297                                                 warning(catgets(catd, 1, 69, "Ca
298                                                         makefile_name->string_mb
299                                                         errmsg(makefile_name->
300                                                                stat.stat_errno))
301                                         }
302                                 }
303                                 max_include_depth--;
304                                 if(make_state_locked && (make_state_lockfile !=
305                                         (void) unlink(make_state_lockfile);
306                                         retmem_mb(make_state_lockfile);
307                                         make_state_lockfile = NULL;
308                                         make_state_locked = false;
309                                 }
310                                 retmem(wcb);
311                                 retmem_mb((char *)source);
312                                 return failed;
313                         }
314                         /*
315                          * These values are the size and bytes of
316                          * the MULTI-BYTE makefile.
317                          */
318                         orig_makefile->body.makefile.size =
319                           makefile->body.makefile.size =
320                             source->bytes_left_in_file =
321                               makefile_name->stat.size;
322                         if (report_file) {
323                                 for (dpp = &makefiles_used;
324                                      *dpp != NULL;
325                                      dpp = &(*dpp)->next);
```



```
326                                 dp = ALLOC(Dependency);
327                                 dp->next = NULL;
328                                 dp->name = makefile_name;
329                                 dp->automatic = false;
330                                 dp->stale = false;
331                                 dp->built = false;
332                                 *dpp = dp;
333                         }
334                         source->fd = open_vroot(makefile_name->string_mb,
335                                                 O_RDONLY,
336                                                 0,
337                                                 NULL,
338                                                 VROOT_DEFAULT);
339                         if (source->fd < 0) {
340                                 if (complain || (errno != ENOENT)) {
341                                         if (must_exist) {
342                                                 fatal(catgets(catd, 1, 70, "Can'
343                                                       makefile_name->string_mb,
344                                                       errmsg(errno));
345                                         } else {
346                                                 warning(catgets(catd, 1, 71, "Ca
347                                                         makefile_name->string_mb
348                                                         errmsg(errno));
349                                         }
350                                 }
351                                 max_include_depth--;
352                                 return failed;
353                         }
354                         (void) fcntl(source->fd, F_SETFD, 1);
355                         orig_makefile->body.makefile.contents =
356                           makefile->body.makefile.contents =
357                             source->string.text.p =
358                               source->string.buffer.start =
359                                 ALLOC_WC((int) (makefile_name->stat.size + 2));
360                         if (makefile_type == reading_cpp_file) {
361                                 forget_after_parse = true;
362                         }
363                         source->string.text.end = source->string.text.p;
364                         source->string.buffer.end =
365                           source->string.text.p + makefile_name->stat.size;
366                 } else {
367                         /* Do we ever reach here? */
368                         source->fd = -1;
369                         source->string.text.p =
370                           source->string.buffer.start =
371                             makefile->body.makefile.contents;
372                         source->string.text.end =
373                           source->string.buffer.end =
374                             source->string.text.p + makefile->body.makefile.size
375                         source->bytes_left_in_file =
376                           makefile->body.makefile.size;
377                 }
378                 file_being_read = wcb;
379         } else {
380                 char            *stdin_text_p;
381                 char            *stdin_text_end;
382                 char            *stdin_buffer_start;
383                 char            *stdin_buffer_end;
384                 char            *p_mb;
385                 int             num_mb_chars;
386                 size_t          num_wc_chars;

388                 MBSTOWCS(wcs_buffer, NOCATGETS("Standard in"));
389                 makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
390                 /*
391                  * Memory to read standard in, then convert it
```

```
392                  * to wide char strings.
393                  */
394                 stdin_buffer_start =
395                   stdin_text_p = getmem(length = 1024);
396                 stdin_buffer_end = stdin_text_p + length;
397                 MBSTOWCS(wcs_buffer, NOCATGETS("standard input"));
398                 file_being_read = (wchar_t *) wsdup(wcs_buffer);
399                 line_number = 0;
400                 while ((n = read(fileno(stdin),
401                                  stdin_text_p,
402                                  length)) > 0) {
403                         length -= n;
404                         stdin_text_p += n;
405                         if (length == 0) {
406                                 p_mb = getmem(length = 1024 +
407                                               (stdin_buffer_end -
408                                                stdin_buffer_start));
409                                 (void) strncpy(p_mb,
410                                                stdin_buffer_start,
411                                                (stdin_buffer_end -
412                                                 stdin_buffer_start));
413                                 retmem_mb(stdin_buffer_start);
414                                 stdin_text_p = p_mb +
415                                   (stdin_buffer_end - stdin_buffer_start);
416                                 stdin_buffer_start = p_mb;
417                                 stdin_buffer_end =
418                                   stdin_buffer_start + length;
419                                 length = 1024;
420                         }
421                 }
422                 if (n < 0) {
423                         fatal(catgets(catd, 1, 72, "Error reading standard input
424                               errmsg(errno));
425                 }
426                 stdin_text_p = stdin_buffer_start;
427                 stdin_text_end = stdin_buffer_end - length;
428                 num_mb_chars = stdin_text_end - stdin_text_p;

430                 /*
431                  * Now, convert the sequence of multibyte chars into
432                  * a sequence of corresponding wide character codes.
433                  */
434                 source->string.free_after_use = false;
435                 source->previous = NULL;
436                 source->bytes_left_in_file = 0;
437                 source->fd = -1;
438                 source->already_expanded = false;
439                 source->string.buffer.start =
440                   source->string.text.p = ALLOC_WC(num_mb_chars + 1);
441                 source->string.buffer.end =
442                     source->string.text.p + num_mb_chars;
443                 num_wc_chars = mbstowcs(source->string.text.p,
444                                         stdin_text_p,
445                                         num_mb_chars);
446                 if ((int) num_wc_chars >= 0) {
447                         source->string.text.end =
448                           source->string.text.p + num_wc_chars;
449                 }
450                 (void) retmem_mb(stdin_text_p);
451         }
452         line_number = 1;
453         if (trace_reader) {
454                 (void) printf(catgets(catd, 1, 73, ">>>>>>>>>>>>>>>> Reading mak
455                               makefile_name->string_mb);
456         }
457         parse_makefile(makefile_name, source);
```



```
458         if (trace_reader) {
459                 (void) printf(catgets(catd, 1, 74, ">>>>>>>>>>>>>>>> End of make
460                               makefile_name->string_mb);
461         }
462 #ifdef NSE
463         if (report_file && (previous_current_makefile[0] != NULL)) {
464                 wscpy(current_makefile, previous_current_makefile);
465         }
466 #endif
467         if(file_being_read) {
468                 retmem(file_being_read);
469         }
470         file_being_read = previous_file_being_read;
471         line_number = previous_line_number;
472         makefile_type = reading_nothing;
473         max_include_depth--;
474         if (make_state_locked) {
475                 /* Unlock .make.state. */
476                 unlink(make_state_lockfile);
477                 make_state_locked = false;
478                 retmem_mb(make_state_lockfile);
479         }
480         if (forget_after_parse) {
481                 retmem(makefile->body.makefile.contents);
482                 makefile->body.makefile.contents = NULL;
483         }
484         retmem_mb((char *)source);
485         return succeeded;
486 }

488 /*
489  *      parse_makefile(true_makefile_name, source)
490  *
491  *      Strings are read from Sources.
492  *      When macros are found, their values are represented by a
493  *      Source that is pushed on a stack. At end of string
494  *      (that is returned from GET_CHAR() as 0), the block is popped.
495  *
496  *      Parameters:
497  *              true_makefile_name      The name of makefile we are parsing
498  *              source                  The source block to read from
499  *
500  *      Global variables used:
501  *              do_not_exec_rule Is -n on?
502  *              line_number     The number of the current makefile line
503  *              makefile_type   What kind of makefile are we reading?
504  *              empty_name      The Name ""
505  */
506 static void
507 parse_makefile(register Name true_makefile_name, register Source source)
508 {
509 /*
510         char                    mb_buffer[MB_LEN_MAX];
511  */
512         register wchar_t        *source_p;
513         register wchar_t        *source_end;
514         register wchar_t        *string_start;
515         wchar_t                 *string_end;
516         register Boolean        macro_seen_in_string;
517         Boolean                 append;
518         String_rec              name_string;
519         wchar_t                 name_buffer[STRING_BUFFER_LENGTH];
520         register int            distance;
521         register int            paren_count;
522         int                     brace_count;
523         int                     char_number;
```

```
524         Cmd_line                command;
525         Cmd_line                command_tail;
526         Name                    macro_value;

528         Name_vector_rec         target;
529         Name_vector_rec         depes;
530         Name_vector_rec         extra_name_vector;
531         Name_vector             current_names;
532         Name_vector             extra_names = &extra_name_vector;
533         Name_vector             nvp;
534         Boolean                 target_group_seen;

536         register Reader_state   state;
537         register Reader_state   on_eoln_state;
538         register Separator      separator;

540         wchar_t                 buffer[4 * STRING_BUFFER_LENGTH];
541         Source                  extrap;

543         Boolean                 save_do_not_exec_rule = do_not_exec_rule;
544         Name                    makefile_name;

546         static Name             sh_name;
547         static Name             shell_name;
548         int                     i;

550         static wchar_t          include_space[10];
551         static wchar_t          include_tab[10];
552         int                     tmp_bytes_left_in_string;
553         Boolean                 tmp_maybe_include = false;
554         int                     emptycount = 0;
555         Boolean                 first_target;

557         String_rec              include_name;
558         wchar_t                 include_buffer[STRING_BUFFER_LENGTH];

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

564  start_new_line:
565         /*
566          * Read whitespace on old line. Leave pointer on first char on
567          * next line.
568          */
569         first_target = true;
570         on_eoln_state = exit_state;
571 /*
572         for (WCTOMB(mb_buffer, GET_CHAR());
573              1;
574              source_p++, WCTOMB(mb_buffer, GET_CHAR()))
575                 switch (mb_buffer[0]) {
576  */
577         for (char_number=0; 1; source_p++,char_number++) switch (GET_CHAR()) {
578         case nul_char:
579                 /* End of this string. Pop it and return to the previous one */
580                 GET_NEXT_BLOCK(source);
581                 source_p--;
582                 if (source == NULL) {
583                         GOTO_STATE(on_eoln_state);
584                 }
585                 break;
586         case newline_char:
587         end_of_line:
588                 source_p++;
589                 if (source->fd >= 0) {
```



```
590                         line_number++;
591                 }
592                 switch (GET_CHAR()) {
593                 case nul_char:
594                         GET_NEXT_BLOCK(source);
595                         if (source == NULL) {
596                                 GOTO_STATE(on_eoln_state);
597                         }
598                         /* Go back to the top of this loop */
599                         goto start_new_line;
600                 case newline_char:
601                 case numbersign_char:
602                 case dollar_char:
603                 case space_char:
604                 case tab_char:
605                         /*
606                          * Go back to the top of this loop since the
607                          * new line does not start with a regular char.
608                          */
609                         goto start_new_line;
610                 default:
611                         /* We found the first proper char on the new line */
612                         goto start_new_line_no_skip;
613                 }
614         case space_char:
615                 if (char_number == 0)
616                         line_started_with_space=line_number;
617         case tab_char:
618                 /* Whitespace. Just keep going in this loop */
619                 break;
620         case numbersign_char:
621                 /* Comment. Skip over it */
622                 for (; 1; source_p++) {
623                         switch (GET_CHAR()) {
624                         case nul_char:
625                                 GET_NEXT_BLOCK_NOCHK(source);
626                                 if (source == NULL) {
627                                         GOTO_STATE(on_eoln_state);
628                                 }
629                                 if (source->error_converting) {
630                                 // Illegal byte sequence - skip its first byte
631                                         source->inp_buf_ptr++;
632                                 }
633                                 source_p--;
634                                 break;
635                         case backslash_char:
636                                 /* Comments can be continued */
637                                 if (*++source_p == (int) nul_char) {
638                                         GET_NEXT_BLOCK_NOCHK(source);
639                                         if (source == NULL) {
640                                                 GOTO_STATE(on_eoln_state);
641                                         }
642                                         if (source->error_converting) {
643                                         // Illegal byte sequence - skip its firs
644                                                 source->inp_buf_ptr++;
645                                                 source_p--;
646                                                 break;
647                                         }
648                                 }
649                                 if(*source_p == (int) newline_char) {
650                                         if (source->fd >= 0) {
651                                                 line_number++;
652                                         }
653                                 }
654                                 break;
655                         case newline_char:
```

```
656                                 /*
657                                  * After we skip the comment we go to
658                                  * the end of line handler since end of
659                                  * line terminates comments.
660                                  */
661                                 goto end_of_line;
662                         }
663                 }
664         case dollar_char:
665                 /* Macro reference */
666                 if (source->already_expanded) {
667                         /*
668                          * If we are reading from the expansion of a
669                          * macro we already expanded everything enough.
670                          */
671                         goto start_new_line_no_skip;
672                 }
673                 /*
674                  * Expand the value and push the Source on the stack of
675                  * things being read.
676                  */
677                 source_p++;
678                 UNCACHE_SOURCE();
679                 {
680                         Source t = (Source) alloca((int) sizeof (Source_rec));
681                         source = push_macro_value(t,
682                                                   buffer,
683                                                   sizeof buffer,
684                                                   source);
685                 }
686                 CACHE_SOURCE(1);
687                 break;
688         default:
689                 /* We found the first proper char on the new line */
690                 goto start_new_line_no_skip;
691         }

693         /*
694          * We found the first normal char (one that starts an identifier)
695          * on the newline.
696          */
697 start_new_line_no_skip:
698         /* Inspect that first char to see if it maybe is special anyway */
699         switch (GET_CHAR()) {
700         case nul_char:
701                 GET_NEXT_BLOCK(source);
702                 if (source == NULL) {
703                         GOTO_STATE(on_eoln_state);
704                 }
705                 goto start_new_line_no_skip;
706         case newline_char:
707                 /* Just in case */
708                 goto start_new_line;
709         case exclam_char:
710                 /* Evaluate the line before it is read */
711                 string_start = source_p + 1;
712                 macro_seen_in_string = false;
713                 /* Stuff the line in a string so we can eval it. */
714                 for (; 1; source_p++) {
715                         switch (GET_CHAR()) {
716                         case newline_char:
717                                 goto eoln_1;
718                         case nul_char:
719                                 if (source->fd > 0) {
720                                         if (!macro_seen_in_string) {
721                                                 macro_seen_in_string = true;
```



```
722                                                 INIT_STRING_FROM_STACK(
723                                                         name_string, name_buffer);
724                                         }
725                                         append_string(string_start,
726                                                       &name_string,
727                                                       source_p - string_start);
728                                         GET_NEXT_BLOCK(source);
729                                         string_start = source_p;
730                                         source_p--;
731                                         break;
732                                 }
733                         eoln_1:
734                                 if (!macro_seen_in_string) {
735                                         INIT_STRING_FROM_STACK(name_string,
736                                                                name_buffer);
737                                 }
738                                 append_string(string_start,
739                                               &name_string,
740                                               source_p - string_start);
741                                 extrap = (Source)
742                                   alloca((int) sizeof (Source_rec));
743                                 extrap->string.buffer.start = NULL;
744                                 extrap->inp_buf =
745                                   extrap->inp_buf_ptr =
746                                     extrap->inp_buf_end = NULL;
747                                 extrap->error_converting = false;
748                                 if (*source_p == (int) nul_char) {
749                                         source_p++;
750                                 }
751                                 /* Eval the macro */
752                                 expand_value(GETNAME(name_string.buffer.start,
753                                                      FIND_LENGTH),
754                                              &extrap->string,
755                                              false);
756                                 if (name_string.free_after_use) {
757                                         retmem(name_string.buffer.start);
758                                 }
759                                 UNCACHE_SOURCE();
760                                 extrap->string.text.p =
761                                   extrap->string.buffer.start;
762                                 extrap->fd = -1;
763                                 /* And push the value */
764                                 extrap->previous = source;
765                                 source = extrap;
766                                 CACHE_SOURCE(0);
767                                 goto line_evald;
768                         }
769                 }
770         default:
771                 goto line_evald;
772         }

774         /* We now have a line we can start reading */
775 line_evald:
776         if (source == NULL) {
777                 GOTO_STATE(exit_state);
778         }
779         /* Check if this is an include command */
780         if ((makefile_type == reading_makefile) &&
781             !source->already_expanded) {
782             if (include_space[0] == (int) nul_char) {
783                 MBSTOWCS(include_space, NOCATGETS("include "));
784                 MBSTOWCS(include_tab, NOCATGETS("include\t"));
785             }
786             if ((IS_WEQUALN(source_p, include_space, 8)) ||
787                 (IS_WEQUALN(source_p, include_tab, 8))) {
```

```
788                 source_p += 7;
789                 if (iswspace(*source_p)) {
790                         Makefile_type save_makefile_type;
791                         wchar_t         *name_start;
792                         int             name_length;
793
794                         /*
795                          * Yes, this is an include.
796                          * Skip spaces to get to the filename.
797                          */
798                         while (iswspace(*source_p) ||
799                                (*source_p == (int) nul_char)) {
800                                 switch (GET_CHAR()) {
801                                 case nul_char:
802                                         GET_NEXT_BLOCK(source);
803                                         if (source == NULL) {
804                                                 GOTO_STATE(on_eoln_state);
805                                         }
806                                         break;
807
808                                 default:
809                                         source_p++;
810                                         break;
811                                 }
812                         }
813
814                         string_start = source_p;
815                         /* Find the end of the filename */
816                         macro_seen_in_string = false;
817                         while (!iswspace(*source_p) ||
818                                (*source_p == (int) nul_char)) {
819                                 switch (GET_CHAR()) {
820                                 case nul_char:
821                                         if (!macro_seen_in_string) {
822                                                 INIT_STRING_FROM_STACK(name_stri
823                                                                        name_buff
824                                         }
825                                         append_string(string_start,
826                                                       &name_string,
827                                                       source_p - string_start);
828                                         macro_seen_in_string = true;
829                                         GET_NEXT_BLOCK(source);
830                                         string_start = source_p;
831                                         if (source == NULL) {
832                                                 GOTO_STATE(on_eoln_state);
833                                         }
834                                         break;
835
836                                 default:
837                                         source_p++;
838                                         break;
839                                 }
840                         }
841
842                         source->string.text.p = source_p;
843                         if (macro_seen_in_string) {
844                                 append_string(string_start,
845                                               &name_string,
846                                               source_p - string_start);
847                                 name_start = name_string.buffer.start;
848                                 name_length = name_string.text.p - name_start;
849                         } else {
850                                 name_start = string_start;
851                                 name_length = source_p - string_start;
852                         }
```



```
853
854                         /* Strip "./" from the head of the name */
855                         if ((name_start[0] == (int) period_char) &&
856                            (name_start[1] == (int) slash_char)) {
857                                 name_start += 2;
858                                 name_length -= 2;
859                         }
860                         /* if include file name is surrounded by double quotes *
861                         if ((name_start[0] == (int) doublequote_char) &&
862                             (name_start[name_length - 1] == (int) doublequote_ch
863                                 name_start += 1;
864                                 name_length -= 2;
865
866                                 /* if name does not begin with a slash char */
867                                 if (name_start[0] != (int) slash_char) {
868                                         if ((name_start[0] == (int) period_char)
869                                             (name_start[1] == (int) slash_char))
870                                                 name_start += 2;
871                                                 name_length -= 2;
872                                         }
873
874                                         INIT_STRING_FROM_STACK(include_name, inc
875                                         APPEND_NAME(true_makefile_name,
876                                                     &include_name,
877                                                     true_makefile_name->hash.l
878
879                                         wchar_t *slash = wsrchr(include_name.buf
880                                         if (slash != NULL) {
881                                                 include_name.text.p = slash + 1;
882                                                 append_string(name_start,
883                                                               &include_name,
884                                                               name_length);
885
886                                                 name_start = include_name.buffer
887                                                 name_length = include_name.text.
888                                         }
889                                 }
890                         }
891
892                         /* Even when we run -n we want to create makefiles */
893                         do_not_exec_rule = false;
894                         makefile_name = GETNAME(name_start, name_length);
895                         if (makefile_name->dollar) {
896                                 String_rec      destination;
897                                 wchar_t         buffer[STRING_BUFFER_LENGTH];
898                                 wchar_t         *p;
899                                 wchar_t         *q;
900
901                                 INIT_STRING_FROM_STACK(destination, buffer);
902                                 expand_value(makefile_name,
903                                              &destination,
904                                              false);
905                                 for (p = destination.buffer.start;
906                                      (*p != (int) nul_char) && iswspace(*p);
907                                      p++);
908                                 for (q = p;
909                                      (*q != (int) nul_char) && !iswspace(*q);
910                                      q++);
911                                 makefile_name = GETNAME(p, q-p);
912                                 if (destination.free_after_use) {
913                                         retmem(destination.buffer.start);
914                                 }
915                         }
916                         source_p++;
917                         UNCACHE_SOURCE();
918                         /* Read the file */
919                         save_makefile_type = makefile_type;
```

```
920                         if (read_simple_file(makefile_name,
921                                              true,
922                                              true,
923                                              true,
924                                              false,
925                                              true,
926                                              false) == failed) {
927                                 fatal_reader(catgets(catd, 1, 75, "Read of inclu
928                                              makefile_name->string_mb);
929                         }
930                         makefile_type = save_makefile_type;
931                         do_not_exec_rule = save_do_not_exec_rule;
932                         CACHE_SOURCE(0);
933                         goto start_new_line;
934                 } else {
935                         source_p -= 7;
936                 }
937             } else {
938                 /* Check if the word include was split across 8K boundary. */
939
940                 tmp_bytes_left_in_string = source->string.text.end - source_p;
941                 if (tmp_bytes_left_in_string < 8) {
942                         tmp_maybe_include = false;
943                         if (IS_WEQUALN(source_p,
944                                        include_space,
945                                        tmp_bytes_left_in_string)) {
946                                 tmp_maybe_include = true;
947                         }
948                         if (tmp_maybe_include) {
949                                 GET_NEXT_BLOCK(source);
950                                 tmp_maybe_include = false;
951                                 goto line_evald;
952                         }
953                 }
954             }
955         }

957         /* Reset the status in preparation for the new line */
958         for (nvp = &target; nvp != NULL; nvp = nvp->next) {
959                 nvp->used = 0;
960         }
961         for (nvp = &depes; nvp != NULL; nvp = nvp->next) {
962                 nvp->used = 0;
963         }
964         target_group_seen = false;
965         command = command_tail = NULL;
966         macro_value = NULL;
967         append = false;
968         current_names = &target;
969         SET_STATE(scan_name_state);
970         on_eoln_state = illegal_eoln_state;
971         separator = none_seen;

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

977 /*************************************************************
978  *      Scan name state
979  */
980 case scan_name_state:
981         /* Scan an identifier. We skip over chars until we find a break char */
982         /* First skip white space. */
983         for (; 1; source_p++) switch (GET_CHAR()) {
984         case nul_char:
985                 GET_NEXT_BLOCK(source);
```



```
986                 source_p--;
987                 if (source == NULL) {
988                         GOTO_STATE(on_eoln_state);
989                 }
990                 break;
991         case newline_char:
992                 /* We found the end of the line. */
993                 /* Do postprocessing or return error */
994                 source_p++;
995                 if (source->fd >= 0) {
996                         line_number++;
997                 }
998                 GOTO_STATE(on_eoln_state);
999         case backslash_char:
1000                /* Continuation */
1001                if (*++source_p == (int) nul_char) {
1002                        GET_NEXT_BLOCK(source);
1003                        if (source == NULL) {
1004                                GOTO_STATE(on_eoln_state);
1005                        }
1006                }
1007                if (*source_p == (int) newline_char) {
1008                        if (source->fd >= 0) {
1009                                line_number++;
1010                        }
1011                } else {
1012                        source_p--;
1013                }
1014                break;
1015        case tab_char:
1016        case space_char:
1017                /* Whitespace is skipped */
1018                break;
1019        case numbersign_char:
1020                /* Comment. Skip over it */
1021                for (; 1; source_p++) {
1022                        switch (GET_CHAR()) {
1023                        case nul_char:
1024                                GET_NEXT_BLOCK_NOCHK(source);
1025                                if (source == NULL) {
1026                                        GOTO_STATE(on_eoln_state);
1027                                }
1028                                if (source->error_converting) {
1029                                // Illegal byte sequence - skip its first byte
1030                                        source->inp_buf_ptr++;
1031                                }
1032                                source_p--;
1033                                break;
1034                        case backslash_char:
1035                                if (*++source_p == (int) nul_char) {
1036                                        GET_NEXT_BLOCK_NOCHK(source);
1037                                        if (source == NULL) {
1038                                                GOTO_STATE(on_eoln_state);
1039                                        }
1040                                        if (source->error_converting) {
1041                                        // Illegal byte sequence - skip its firs
1042                                                source->inp_buf_ptr++;
1043                                                source_p--;
1044                                                break;
1045                                        }
1046                                }
1047                                if(*source_p == (int) newline_char) {
1048                                        if (source->fd >= 0) {
1049                                                line_number++;
1050                                        }
1051                                }
```

```
1052                                break;
1053                        case newline_char:
1054                                source_p++;
1055                                if (source->fd >= 0) {
1056                                        line_number++;
1057                                }
1058                                GOTO_STATE(on_eoln_state);
1059                        }
1060                }
1061        case dollar_char:
1062                /* Macro reference. Expand and push value */
1063                if (source->already_expanded) {
1064                        goto scan_name;
1065                }
1066                source_p++;
1067                UNCACHE_SOURCE();
1068                {
1069                        Source t = (Source) alloca((int) sizeof (Source_rec));
1070                        source = push_macro_value(t,
1071                                                  buffer,
1072                                                  sizeof buffer,
1073                                                  source);
1074                }
1075                CACHE_SOURCE(1);
1076                break;
1077        default:
1078                /* End of white space */
1079                goto scan_name;
1080        }

1082        /* First proper identifier character */
1083  scan_name:

1085        string_start = source_p;
1086        paren_count = brace_count = 0;
1087        macro_seen_in_string = false;
1088        resume_name_scan:
1089        for (; 1; source_p++) {
1090                switch (GET_CHAR()) {
1091                case nul_char:
1092                        /* Save what we have seen so far of the identifier */
1093                        if (source_p != string_start) {
1094                                if (!macro_seen_in_string) {
1095                                        INIT_STRING_FROM_STACK(name_string,
1096                                                               name_buffer);
1097                                }
1098                                append_string(string_start,
1099                                              &name_string,
1100                                              source_p - string_start);
1101                                macro_seen_in_string = true;
1102                        }
1103                        /* Get more text to read */
1104                        GET_NEXT_BLOCK(source);
1105                        string_start = source_p;
1106                        source_p--;
1107                        if (source == NULL) {
1108                                GOTO_STATE(on_eoln_state);
1109                        }
1110                        break;
1111                case newline_char:
1112                        if (paren_count > 0) {
1113                                fatal_reader(catgets(catd, 1, 76, "Unmatched ‘(’
1114                        }
1115                        if (brace_count > 0) {
1116                                fatal_reader(catgets(catd, 1, 77, "Unmatched ‘{’
1117                        }
```



```
1118                        source_p++;
1119                        /* Enter name */
1120                        current_names = enter_name(&name_string,
1121                                                   macro_seen_in_string,
1122                                                   string_start,
1123                                                   source_p - 1,
1124                                                   current_names,
1125                                                   &extra_names,
1126                                                   &target_group_seen);
1127                        first_target = false;
1128                        if (extra_names == NULL) {
1129                                extra_names = (Name_vector)
1130                                  alloca((int) sizeof (Name_vector_rec));
1131                        }
1132                        /* Do postprocessing or return error */
1133                        if (source->fd >= 0) {
1134                                line_number++;
1135                        }
1136                        GOTO_STATE(on_eoln_state);
1137                case backslash_char:
1138                        /* Check if this is a quoting backslash */
1139                        if (!macro_seen_in_string) {
1140                                INIT_STRING_FROM_STACK(name_string,
1141                                                       name_buffer);
1142                                macro_seen_in_string = true;
1143                        }
1144                        append_string(string_start,
1145                                      &name_string,
1146                                      source_p - string_start);
1147                        if (*++source_p == (int) nul_char) {
1148                                GET_NEXT_BLOCK(source);
1149                                if (source == NULL) {
1150                                        GOTO_STATE(on_eoln_state);
1151                                }
1152                        }
1153                        if (*source_p == (int) newline_char) {
1154                                if (source->fd >= 0) {
1155                                        line_number++;
1156                                }
1157                                *source_p = (int) space_char;
1158                                string_start = source_p;
1159                                goto resume_name_scan;
1160                        } else {
1161                                string_start = source_p;
1162                                break;
1163                        }
1164                        break;
1165                case numbersign_char:
1166                        if (paren_count + brace_count > 0) {
1167                                break;
1168                        }
1169                        fatal_reader(catgets(catd, 1, 78, "Unexpected comment se
1170                case dollar_char:
1171                        if (source->already_expanded) {
1172                                break;
1173                        }
1174                        /* Save the identifier so far */
1175                        if (source_p != string_start) {
1176                                if (!macro_seen_in_string) {
1177                                        INIT_STRING_FROM_STACK(name_string,
1178                                                               name_buffer);
1179                                }
1180                                append_string(string_start,
1181                                              &name_string,
1182                                              source_p - string_start);
1183                                macro_seen_in_string = true;
```

```
1184                        }
1185                        /* Eval and push the macro */
1186                        source_p++;
1187                        UNCACHE_SOURCE();
1188                        {
1189                                Source t =
1190                                  (Source) alloca((int) sizeof (Source_rec));
1191                                source = push_macro_value(t,
1192                                                          buffer,
1193                                                          sizeof buffer,
1194                                                          source);
1195                        }
1196                        CACHE_SOURCE(1);
1197                        string_start = source_p + 1;
1198                        break;
1199                case parenleft_char:
1200                        paren_count++;
1201                        break;
1202                case parenright_char:
1203                        if (--paren_count < 0) {
1204                                fatal_reader(catgets(catd, 1, 79, "Unmatched `)'
1205                        }
1206                        break;
1207                case braceleft_char:
1208                        brace_count++;
1209                        break;
1210                case braceright_char:
1211                        if (--brace_count < 0) {
1212                                fatal_reader(catgets(catd, 1, 80, "Unmatched `}'
1213                        }
1214                        break;
1215                case ampersand_char:
1216                case greater_char:
1217                case bar_char:
1218                        if (paren_count + brace_count == 0) {
1219                                source_p++;
1220                        }
1221                        /* Fall into */
1222                case tab_char:
1223                case space_char:
1224                        if (paren_count + brace_count > 0) {
1225                                break;
1226                        }
1227                        current_names = enter_name(&name_string,
1228                                                   macro_seen_in_string,
1229                                                   string_start,
1230                                                   source_p,
1231                                                   current_names,
1232                                                   &extra_names,
1233                                                   &target_group_seen);
1234                        first_target = false;
1235                        if (extra_names == NULL) {
1236                                extra_names = (Name_vector)
1237                                  alloca((int) sizeof (Name_vector_rec));
1238                        }
1239                        goto enter_state;
1240                case colon_char:
1241                        if (paren_count + brace_count > 0) {
1242                                break;
1243                        }
1244                        if (separator == conditional_seen) {
1245                                break;
1246                        }
1247 /** POSIX **/
1248 #if 0
1249                        if(posix) {
```



```
1250                          emptycount = 0;
1251                        }
1252 #endif
1253 /** END POSIX **/
1254                        /* End of the target list. We now start reading */
1255                        /* dependencies or a conditional assignment */
1256                        if (separator != none_seen) {
1257                                fatal_reader(catgets(catd, 1, 81, "Extra `:', `:
1258                        }
1259                        /* Enter the last target */
1260                        if ((string_start != source_p) ||
1261                            macro_seen_in_string) {
1262                                current_names =
1263                                  enter_name(&name_string,
1264                                             macro_seen_in_string,
1265                                             string_start,
1266                                             source_p,
1267                                             current_names,
1268                                             &extra_names,
1269                                             &target_group_seen);
1270                                first_target = false;
1271                                if (extra_names == NULL) {
1272                                        extra_names = (Name_vector)
1273                                          alloca((int)
1274                                                 sizeof (Name_vector_rec));
1275                                }
1276                        }
1277                        /* Check if it is ":" "::" or ":=" */
1278                scan_colon_label:
1279                        switch (*++source_p) {
1280                        case nul_char:
1281                                GET_NEXT_BLOCK(source);
1282                                source_p--;
1283                                if (source == NULL) {
1284                                        GOTO_STATE(enter_dependencies_state);
1285                                }
1286                                goto scan_colon_label;
1287                        case equal_char:
1288                                if(svr4) {
1289                                  fatal_reader(catgets(catd, 1, 82, "syntax erro
1290                                }
1291                                separator = conditional_seen;
1292                                source_p++;
1293                                current_names = &depes;
1294                                GOTO_STATE(scan_name_state);
1295                        case colon_char:
1296                                separator = two_colon;
1297                                source_p++;
1298                                break;
1299                        default:
1300                                separator = one_colon;
1301                        }
1302                        current_names = &depes;
1303                        on_eoln_state = enter_dependencies_state;
1304                        GOTO_STATE(scan_name_state);
1305                case semicolon_char:
1306                        if (paren_count + brace_count > 0) {
1307                                break;
1308                        }
1309                        /* End of reading names. Start reading the rule */
1310                        if ((separator != one_colon) &&
1311                            (separator != two_colon)) {
1312                                fatal_reader(catgets(catd, 1, 83, "Unexpected co
1313                        }
1314                        /* Enter the last dependency */
1315                        if ((string_start != source_p) ||
```

```
1316                            macro_seen_in_string) {
1317                                current_names =
1318                                  enter_name(&name_string,
1319                                             macro_seen_in_string,
1320                                             string_start,
1321                                             source_p,
1322                                             current_names,
1323                                             &extra_names,
1324                                             &target_group_seen);
1325                                first_target = false;
1326                                if (extra_names == NULL) {
1327                                        extra_names = (Name_vector)
1328                                          alloca((int)
1329                                                 sizeof (Name_vector_rec));
1330                                }
1331                        }
1332                        source_p++;
1333                        /* Make sure to enter a rule even if the is */
1334                        /* no text here */
1335                        command = command_tail = ALLOC(Cmd_line);
1336                        command->next = NULL;
1337                        command->command_line = empty_name;
1338                        command->make_refd = false;
1339                        command->ignore_command_dependency = false;
1340                        command->assign = false;
1341                        command->ignore_error = false;
1342                        command->silent = false;

1344                        GOTO_STATE(scan_command_state);
1345                case plus_char:
1346                        /*
1347                        ** following code drops the target separator plus char i
1348                        ** a line.
1349                        */
1350                        if(first_target && !macro_seen_in_string &&
1351                                source_p == string_start) {
1352                                for (; 1; source_p++)
1353                                switch (GET_CHAR()) {
1354                                case nul_char:
1355                                        if (source_p != string_start) {
1356                                                if (!macro_seen_in_string) {
1357                                                        INIT_STRING_FROM_STACK(n
1358                                                                               n
1359                                                }
1360                                                append_string(string_start,
1361                                                              &name_string,
1362                                                              source_p - string_
1363                                                macro_seen_in_string = true;
1364                                        }
1365                                        GET_NEXT_BLOCK(source);
1366                                        string_start = source_p;
1367                                        source_p--;
1368                                        if (source == NULL) {
1369                                                GOTO_STATE(on_eoln_state);
1370                                        }
1371                                        break;
1372                                case plus_char:
1373                                        source_p++;
1374                                        while (*source_p == (int) nul_char) {
1375                                                if (source_p != string_start) {
1376                                                        if (!macro_seen_in_strin
1377                                                                INIT_STRING_FROM
1378                                                                               n
1379                                                        }
1380                                                        append_string(string_sta
1381                                                                      &name_stri
```



```
1382                                                                              source_p -
1383                                                        macro_seen_in_string = t
1384                                                }
1385                                                GET_NEXT_BLOCK(source);
1386                                                string_start = source_p;
1387                                                if (source == NULL) {
1388                                                        GOTO_STATE(on_eoln_state
1389                                                }
1390                                        }
1391                                        if (*source_p == (int) tab_char ||
1392                                                         *source_p == (int) space
1393                                                macro_seen_in_string = false;
1394                                                string_start = source_p + 1;
1395                                        } else {
1396                                                goto resume_name_scan;
1397                                        }
1398                                        break;
1399                                case tab_char:
1400                                case space_char:
1401                                        string_start = source_p + 1;
1402                                        break;
1403                                default:
1404                                        goto resume_name_scan;
1405                                }
1406                        }
1407                        if (paren_count + brace_count > 0) {
1408                                break;
1409                        }
1410                        /* We found "+=" construct */
1411                        if (source_p != string_start) {
1412                                /* "+" is not a break char. */
1413                                /* Ignore it if it is part of an identifier */
1414                                source_p++;
1415                                goto resume_name_scan;
1416                        }
1417                        /* Make sure the "+" is followed by a "=" */
1418                scan_append:
1419                        switch (*++source_p) {
1420                        case nul_char:
1421                                if (!macro_seen_in_string) {
1422                                        INIT_STRING_FROM_STACK(name_string,
1423                                                               name_buffer);
1424                                }
1425                                append_string(string_start,
1426                                              &name_string,
1427                                              source_p - string_start);
1428                                GET_NEXT_BLOCK(source);
1429                                source_p--;
1430                                string_start = source_p;
1431                                if (source == NULL) {
1432                                        GOTO_STATE(illegal_eoln_state);
1433                                }
1434                                goto scan_append;
1435                        case equal_char:
1436                                if(!svr4) {
1437                                  append = true;
1438                                } else {
1439                                  fatal_reader(catgets(catd, 1, 84, "Must be a s
1440                                }
1441                                break;
1442                        default:
1443                                /* The "+" just starts a regular name. */
1444                                /* Start reading that name */
1445                                goto resume_name_scan;
1446                        }
1447                        /* Fall into */
```

```
1448                case equal_char:
1449                        if (paren_count + brace_count > 0) {
1450                                break;
1451                        }
1452                        /* We found macro assignment. */
1453                        /* Check if it is legal and if it is appending */
1454                        switch (separator) {
1455                        case none_seen:
1456                                separator = equal_seen;
1457                                on_eoln_state = enter_equal_state;
1458                                break;
1459                        case conditional_seen:
1460                                on_eoln_state = enter_conditional_state;
1461                                break;
1462                        default:
1463                                /* Reader must special check for "MACRO:sh=" */
1464                                /* notation */
1465                                if (sh_name == NULL) {
1466                                        MBSTOWCS(wcs_buffer, NOCATGETS("sh"));
1467                                        sh_name = GETNAME(wcs_buffer, FIND_LENGT
1468                                        MBSTOWCS(wcs_buffer, NOCATGETS("shell"))
1469                                        shell_name = GETNAME(wcs_buffer, FIND_LE
1470                                }
1471
1472                                if (!macro_seen_in_string) {
1473                                        INIT_STRING_FROM_STACK(name_string,
1474                                                               name_buffer);
1475                                }
1476                                append_string(string_start,
1477                                              &name_string,
1478                                              source_p - string_start
1479                                );
1480
1481                                if ( (((target.used == 1) &&
1482                                     (depes.used == 1) &&
1483                                     (depes.names[0] == sh_name)) ||
1484                                    ((target.used == 1) &&
1485                                     (depes.used == 0) &&
1486                                     (separator == one_colon) &&
1487                                     (GETNAME(name_string.buffer.start,FIND_LENG
1488                                    (!svr4)) {
1489                                        String_rec      macro_name;
1490                                        wchar_t         buffer[100];
1491
1492                                        INIT_STRING_FROM_STACK(macro_name,
1493                                                               buffer);
1494                                        APPEND_NAME(target.names[0],
1495                                                    &macro_name,
1496                                                    FIND_LENGTH);
1497                                        append_char((int) colon_char,
1498                                                    &macro_name);
1499                                        APPEND_NAME(sh_name,
1500                                                    &macro_name,
1501                                                    FIND_LENGTH);
1502                                        target.names[0] =
1503                                          GETNAME(macro_name.buffer.start,
1504                                                  FIND_LENGTH);
1505                                        separator = equal_seen;
1506                                        on_eoln_state = enter_equal_state;
1507                                        break;
1508                                } else if ( (((target.used == 1) &&
1509                                              (depes.used == 1) &&
1510                                              (depes.names[0] == shell_name)) ||
1511                                             ((target.used == 1) &&
1512                                              (depes.used == 0) &&
1513                                              (separator == one_colon) &&
```



```
1514                                                  (GETNAME(name_string.buffer.start,FI
1515                                                 (!svr4)) {
1516                                              String_rec      macro_name;
1517                                              wchar_t         buffer[100];
1518
1519                                              INIT_STRING_FROM_STACK(macro_name,
1520                                                                     buffer);
1521                                              APPEND_NAME(target.names[0],
1522                                                          &macro_name,
1523                                                          FIND_LENGTH);
1524                                              append_char((int) colon_char,
1525                                                          &macro_name);
1526                                              APPEND_NAME(shell_name,
1527                                                          &macro_name,
1528                                                          FIND_LENGTH);
1529                                              target.names[0] =
1530                                                GETNAME(macro_name.buffer.start,
1531                                                        FIND_LENGTH);
1532                                              separator = equal_seen;
1533                                              on_eoln_state = enter_equal_state;
1534                                              break;
1535                                      }
1536                                      if(svr4) {
1537                                        fatal_reader(catgets(catd, 1, 85, "syntax erro
1538                                      }
1539                                      else {
1540                                        fatal_reader(catgets(catd, 1, 86, "Macro assig
1541                                      }
1542                              }
1543                              if (append) {
1544                                      source_p--;
1545                              }
1546                              /* Enter the macro name */
1547                              if ((string_start != source_p) ||
1548                                  macro_seen_in_string) {
1549                                      current_names =
1550                                        enter_name(&name_string,
1551                                                   macro_seen_in_string,
1552                                                   string_start,
1553                                                   source_p,
1554                                                   current_names,
1555                                                   &extra_names,
1556                                                   &target_group_seen);
1557                                      first_target = false;
1558                                      if (extra_names == NULL) {
1559                                              extra_names = (Name_vector)
1560                                                alloca((int)
1561                                                       sizeof (Name_vector_rec));
1562                                      }
1563                              }
1564                              if (append) {
1565                                      source_p++;
1566                              }
1567                              macro_value = NULL;
1568                              source_p++;
1569                              distance = 0;
1570                              /* Skip whitespace to the start of the value */
1571                              macro_seen_in_string = false;
1572                              for (; 1; source_p++) {
1573                                      switch (GET_CHAR()) {
1574                                      case nul_char:
1575                                              GET_NEXT_BLOCK(source);
1576                                              source_p--;
1577                                              if (source == NULL) {
1578                                                      GOTO_STATE(on_eoln_state);
1579                                              }
```

```
1580                                    break;
1581                            case backslash_char:
1582                                    if (*++source_p == (int) nul_char) {
1583                                            GET_NEXT_BLOCK(source);
1584                                            if (source == NULL) {
1585                                                    GOTO_STATE(on_eoln_state
1586                                            }
1587                                    }
1588                                    if (*source_p != (int) newline_char) {
1589                                            if (!macro_seen_in_string) {
1590                                                    macro_seen_in_string =
1591                                                      true;
1592                                                    INIT_STRING_FROM_STACK(n
1593                                                                           n
1594                                            }
1595                                            append_char((int)
1596                                                        backslash_char,
1597                                                        &name_string);
1598                                            append_char(*source_p,
1599                                                        &name_string);
1600                                            string_start = source_p+1;
1601                                            goto macro_value_start;
1602                                    } else {
1603                                        if (source->fd >= 0) {
1604                                            line_number++;
1605                                        }
1606                                    }
1607                                    break;
1608                            case newline_char:
1609                            case numbersign_char:
1610                                    string_start = source_p;
1611                                    goto macro_value_end;
1612                            case tab_char:
1613                            case space_char:
1614                                    break;
1615                            default:
1616                                    string_start = source_p;
1617                                    goto macro_value_start;
1618                            }
1619                    }
1620            macro_value_start:
1621                    /* Find the end of the value */
1622                    for (; 1; source_p++) {
1623                            if (distance != 0) {
1624                                    *source_p = *(source_p + distance);
1625                            }
1626                            switch (GET_CHAR()) {
1627                            case nul_char:
1628                                    if (!macro_seen_in_string) {
1629                                            macro_seen_in_string = true;
1630                                            INIT_STRING_FROM_STACK(name_stri
1631                                                                   name_buff
1632                                    }
1633                                    append_string(string_start,
1634                                                  &name_string,
1635                                                  source_p - string_start);
1636                                    GET_NEXT_BLOCK(source);
1637                                    string_start = source_p;
1638                                    source_p--;
1639                                    if (source == NULL) {
1640                                            GOTO_STATE(on_eoln_state);
1641                                    }
1642                                    break;
1643                            case backslash_char:
1644                                    source_p++;
1645                                    if (distance != 0) {
```



```
1646                                            *source_p =
1647                                              *(source_p + distance);
1648                                    }
1649                                    if (*source_p == (int) nul_char) {
1650                                            if (!macro_seen_in_string) {
1651                                                    macro_seen_in_string =
1652                                                      true;
1653                                                    INIT_STRING_FROM_STACK(n
1654                                                                           n
1655                                            }

1657 /*  BID_1225561 */
1658                                            *(source_p - 1) = (int) space_ch
1659                                            append_string(string_start,
1660                                                          &name_string,
1661                                                          source_p -
1662                                                          string_start - 1);
1663                                            GET_NEXT_BLOCK(source);
1664                                            string_start = source_p;
1665                                            if (source == NULL) {
1666                                                    GOTO_STATE(on_eoln_state
1667                                            }
1668                                            if (distance != 0) {
1669                                                    *source_p =
1670                                                      *(source_p +
1671                                                        distance);
1672                                            }
1673                                            if (*source_p == (int) newline_c
1674                                                    append_char((int) space_
1675                                            } else {
1676                                                    append_char((int) backsl
1677                                            }
1678 /***************/
1679                                    }
1680                                    if (*source_p == (int) newline_char) {
1681                                            source_p--;
1682                                            line_number++;
1683                                            distance++;
1684                                            *source_p = (int) space_char;
1685                                            while ((*(source_p +
1686                                                      distance + 1) ==
1687                                                     (int) tab_char) ||
1688                                                    (*(source_p +
1689                                                       distance + 1) ==
1690                                                     (int) space_char)) {
1691                                                    distance++;
1692                                            }
1693                                    }
1694                                    break;
1695                            case newline_char:
1696                            case numbersign_char:
1697                                    goto macro_value_end;
1698                            }
1699                    }
1700            macro_value_end:
1701                    /* Complete the value in the string */
1702                    if (!macro_seen_in_string) {
1703                            macro_seen_in_string = true;
1704                            INIT_STRING_FROM_STACK(name_string,
1705                                                   name_buffer);
1706                    }
1707                    append_string(string_start,
1708                                  &name_string,
1709                                  source_p - string_start);
1710                    if (name_string.buffer.start != name_string.text.p) {
1711                                    macro_value =
```

```
1712                                        GETNAME(name_string.buffer.start,
1713                                                FIND_LENGTH);
1714                        }
1715                        if (name_string.free_after_use) {
1716                                retmem(name_string.buffer.start);
1717                        }
1718                        for (; distance > 0; distance--) {
1719                                *source_p++ = (int) space_char;
1720                        }
1721                        GOTO_STATE(on_eoln_state);
1722                }
1723        }

1725 /*************************************************************
1726  *      enter dependencies state
1727  */
1728  case enter_dependencies_state:
1729  enter_dependencies_label:
1730 /* Expects pointer on first non whitespace char after last dependency. (On */
1731 /* next line.) We end up here after having read a "targets : dependencies" */
1732 /* line. The state checks if there is a rule to read and if so dispatches */
1733 /* to scan_command_state scan_command_state reads one rule line and the */
1734 /* returns here */

1736        /* First check if the first char on the next line is special */
1737        switch (GET_CHAR()) {
1738        case nul_char:
1739                GET_NEXT_BLOCK(source);
1740                if (source == NULL) {
1741                        break;
1742                }
1743                goto enter_dependencies_label;
1744        case exclam_char:
1745                /* The line should be evaluate before it is read */
1746                macro_seen_in_string = false;
1747                string_start = source_p + 1;
1748                for (; 1; source_p++) {
1749                        switch (GET_CHAR()) {
1750                        case newline_char:
1751                                goto eoln_2;
1752                        case nul_char:
1753                                if (source->fd > 0) {
1754                                        if (!macro_seen_in_string) {
1755                                                macro_seen_in_string = true;
1756                                                INIT_STRING_FROM_STACK(name_stri
1757                                                                       name_buff
1758                                        }
1759                                        append_string(string_start,
1760                                                      &name_string,
1761                                                      source_p - string_start);
1762                                        GET_NEXT_BLOCK(source);
1763                                        string_start = source_p;
1764                                        source_p--;
1765                                        break;
1766                                }
1767                        eoln_2:
1768                                if (!macro_seen_in_string) {
1769                                        INIT_STRING_FROM_STACK(name_string,
1770                                                               name_buffer);
1771                                }
1772                                append_string(string_start,
1773                                              &name_string,
1774                                              source_p - string_start);
1775                                extrap = (Source)
1776                                  alloca((int) sizeof (Source_rec));
1777                                extrap->string.buffer.start = NULL;
```



```
1778                                extrap->inp_buf =
1779                                  extrap->inp_buf_ptr =
1780                                    extrap->inp_buf_end = NULL;
1781                                extrap->error_converting = false;
1782                                expand_value(GETNAME(name_string.buffer.start,
1783                                                     FIND_LENGTH),
1784                                             &extrap->string,
1785                                             false);
1786                                if (name_string.free_after_use) {
1787                                        retmem(name_string.buffer.start);
1788                                }
1789                                UNCACHE_SOURCE();
1790                                extrap->string.text.p =
1791                                  extrap->string.buffer.start;
1792                                extrap->fd = -1;
1793                                extrap->previous = source;
1794                                source = extrap;
1795                                CACHE_SOURCE(0);
1796                                goto enter_dependencies_label;
1797                        }
1798                }
1799        case dollar_char:
1800                if (source->already_expanded) {
1801                        break;
1802                }
1803                source_p++;
1804                UNCACHE_SOURCE();
1805                {
1806                        Source t = (Source) alloca((int) sizeof (Source_rec));
1807                        source = push_macro_value(t,
1808                                                  buffer,
1809                                                  sizeof buffer,
1810                                                  source);
1811                }
1812                CACHE_SOURCE(0);
1813                goto enter_dependencies_label;
1814        case numbersign_char:
1815                if (makefile_type != reading_makefile) {
1816                        source_p++;
1817                        GOTO_STATE(scan_command_state);
1818                }
1819                for (; 1; source_p++) {
1820                        switch (GET_CHAR()) {
1821                        case nul_char:
1822                                GET_NEXT_BLOCK_NOCHK(source);
1823                                if (source == NULL) {
1824                                        GOTO_STATE(on_eoln_state);
1825                                }
1826                                if (source->error_converting) {
1827                                // Illegal byte sequence - skip its first byte
1828                                        source->inp_buf_ptr++;
1829                                }
1830                                source_p--;
1831                                break;
1832                        case backslash_char:
1833                                if (*++source_p == (int) nul_char) {
1834                                        GET_NEXT_BLOCK_NOCHK(source);
1835                                        if (source == NULL) {
1836                                                GOTO_STATE(on_eoln_state);
1837                                        }
1838                                        if (source->error_converting) {
1839                                        // Illegal byte sequence - skip its firs
1840                                                source->inp_buf_ptr++;
1841                                                source_p--;
1842                                                break;
1843                                        }
```

```
1844                                }
1845                                if(*source_p == (int) newline_char) {
1846                                        if (source->fd >= 0) {
1847                                                line_number++;
1848                                        }
1849                                }
1850                                break;
1851                        case newline_char:
1852                                source_p++;
1853                                if (source->fd >= 0) {
1854                                        line_number++;
1855                                }
1856                                goto enter_dependencies_label;
1857                        }
1858                }

1860        case tab_char:
1861                GOTO_STATE(scan_command_state);
1862        }

1864        /* We read all the command lines for the target/dependency line. */
1865        /* Enter the stuff */
1866        enter_target_groups_and_dependencies( &target, &depes, command,
1867                                              separator, target_group_seen);

1869        goto start_new_line;

1871 /*************************************************************
1872  *      scan command state
1873  */
1874 case scan_command_state:
1875        /* We need to read one rule line. Do that and return to */
1876        /* the enter dependencies state */
1877        string_start = source_p;
1878        macro_seen_in_string = false;
1879        for (; 1; source_p++) {
1880                switch (GET_CHAR()) {
1881                case backslash_char:
1882                        if (!macro_seen_in_string) {
1883                                INIT_STRING_FROM_STACK(name_string,
1884                                                       name_buffer);
1885                        }
1886                        append_string(string_start,
1887                                      &name_string,
1888                                      source_p - string_start);
1889                        macro_seen_in_string = true;
1890                        if (*++source_p == (int) nul_char) {
1891                                GET_NEXT_BLOCK(source);
1892                                if (source == NULL) {
1893                                        string_start = source_p;
1894                                        goto command_newline;
1895                                }
1896                        }
1897                        append_char((int) backslash_char, &name_string);
1898                        append_char(*source_p, &name_string);
1899                        if (*source_p == (int) newline_char) {
1900                                if (source->fd >= 0) {
1901                                        line_number++;
1902                                }
1903                                if (*++source_p == (int) nul_char) {
1904                                        GET_NEXT_BLOCK(source);
1905                                        if (source == NULL) {
1906                                                string_start = source_p;
1907                                                goto command_newline;
1908                                        }
1909                                }
```



```
1910                                if (*source_p == (int) tab_char) {
1911                                        source_p++;
1912                                }
1913                        } else {
1914                                if (*++source_p == (int) nul_char) {
1915                                        GET_NEXT_BLOCK(source);
1916                                        if (source == NULL) {
1917                                                string_start = source_p;
1918                                                goto command_newline;
1919                                        }
1920                                }
1921                        }
1922                        string_start = source_p;
1923                        if ((*source_p == (int) newline_char) ||
1924                            (*source_p == (int) backslash_char) ||
1925                            (*source_p == (int) nul_char)) {
1926                                source_p--;
1927                        }
1928                        break;
1929                case newline_char:
1930                command_newline:
1931                        if ((string_start != source_p) ||
1932                            macro_seen_in_string) {
1933                                if (macro_seen_in_string) {
1934                                        append_string(string_start,
1935                                                      &name_string,
1936                                                      source_p - string_start);
1937                                        string_start =
1938                                          name_string.buffer.start;
1939                                        string_end = name_string.text.p;
1940                                } else {
1941                                        string_end = source_p;
1942                                }
1943                                while ((*string_start != (int) newline_char) &&
1944                                       iswspace(*string_start)){
1945                                        string_start++;
1946                                }
1947                                if ((string_end > string_start) ||
1948                                    (makefile_type == reading_statefile)) {
1949                                        if (command_tail == NULL) {
1950                                                command =
1951                                                  command_tail =
1952                                                    ALLOC(Cmd_line);
1953                                        } else {
1954                                                command_tail->next =
1955                                                  ALLOC(Cmd_line);
1956                                                command_tail =
1957                                                  command_tail->next;
1958                                        }
1959                                        command_tail->next = NULL;
1960                                        command_tail->make_refd = false;
1961                                        command_tail->ignore_command_dependency
1962                                        command_tail->assign = false;
1963                                        command_tail->ignore_error = false;
1964                                        command_tail->silent = false;
1965                                        command_tail->command_line =
1966                                          GETNAME(string_start,
1967                                                  string_end - string_start);
1968                                        if (macro_seen_in_string &&
1969                                            name_string.free_after_use) {
1970                                                retmem(name_string.
1971                                                       buffer.start);
1972                                        }
1973                                }
1974                        }
1975                        do {
```

```
1976                                if ((source != NULL) && (source->fd >= 0)) {
1977                                        line_number++;
1978                                }
1979                                if ((source != NULL) &&
1980                                    (*++source_p == (int) nul_char)) {
1981                                        GET_NEXT_BLOCK(source);
1982                                        if (source == NULL) {
1983                                                GOTO_STATE(on_eoln_state);
1984                                        }
1985                                }
1986                        } while (*source_p == (int) newline_char);

1988                        GOTO_STATE(enter_dependencies_state);
1989                case nul_char:
1990                        if (!macro_seen_in_string) {
1991                                INIT_STRING_FROM_STACK(name_string,
1992                                                       name_buffer);
1993                        }
1994                        append_string(string_start,
1995                                      &name_string,
1996                                      source_p - string_start);
1997                        macro_seen_in_string = true;
1998                        GET_NEXT_BLOCK(source);
1999                        string_start = source_p;
2000                        source_p--;
2001                        if (source == NULL) {
2002                                GOTO_STATE(enter_dependencies_state);
2003                        }
2004                        break;
2005                }
2006        }

2008 /*************************************************************
2009  *      enter equal state
2010  */
2011 case enter_equal_state:
2012        if (target.used != 1) {
2013                GOTO_STATE(poorly_formed_macro_state);
2014        }
2015        enter_equal(target.names[0], macro_value, append);
2016        goto start_new_line;

2018 /*************************************************************
2019  *      enter conditional state
2020  */
2021 case enter_conditional_state:
2022        if (depes.used != 1) {
2023                GOTO_STATE(poorly_formed_macro_state);
2024        }
2025        for (nvp = &target; nvp != NULL; nvp = nvp->next) {
2026                for (i = 0; i < nvp->used; i++) {
2027                        enter_conditional(nvp->names[i],
2028                                          depes.names[0],
2029                                          macro_value,
2030                                          append);
2031                }
2032        }
2033        goto start_new_line;

2035 /*************************************************************
2036  *      Error states
2037  */
2038 case illegal_bytes_state:
2039        fatal_reader(catgets(catd, 1, 340, "Invalid byte sequence"));
2040 case illegal_eoln_state:
2041        if (line_number > 1) {
```



```
2042                if (line_started_with_space == (line_number - 1)) {
2043                        line_number--;
2044                        fatal_reader(catgets(catd, 1, 90, "Unexpected end of lin
2045                }
2046        }
2047        fatal_reader(catgets(catd, 1, 87, "Unexpected end of line seen"));
2048 case poorly_formed_macro_state:
2049        fatal_reader(catgets(catd, 1, 88, "Badly formed macro assignment"));
2050 case exit_state:
2051        return;
2052 default:
2053        fatal_reader(catgets(catd, 1, 89, "Internal error. Unknown reader state"
2054 }
2055 }

2057 /*
2058  *      push_macro_value(bp, buffer, size, source)
2059  *
2060  *      Macro and function that evaluates one macro
2061  *      and makes the reader read from the value of it
2062  *
2063  *      Return value:
2064  *                              The source block to read the macro from
2065  *
2066  *      Parameters:
2067  *              bp              The new source block to fill in
2068  *              buffer          Buffer to read from
2069  *              size            size of the buffer
2070  *              source          The old source block
2071  *
2072  *      Global variables used:
2073  */
2074 static Source
2075 push_macro_value(register Source bp, register wchar_t *buffer, int size, registe
2076 {
2077        bp->string.buffer.start = bp->string.text.p = buffer;
2078        bp->string.text.end = NULL;
2079        bp->string.buffer.end = buffer + (size/SIZEOFWCHAR_T);
2080        bp->string.free_after_use = false;
2081        bp->inp_buf =
2082          bp->inp_buf_ptr =
2083            bp->inp_buf_end = NULL;
2084        bp->error_converting = false;
2085        expand_macro(source, &bp->string, (wchar_t *) NULL, false);
2086        bp->string.text.p = bp->string.buffer.start;

2088        /* 4209588: 'make' doesn't understand a macro with whitespaces in the he
2089         * strip whitespace from the begining of the macro value
2090         */
2091        while (iswspace(*bp->string.text.p)) {
2092                bp->string.text.p++;
2093        }

2095        bp->fd = -1;
2096        bp->already_expanded = true;
2097        bp->previous = source;
2098        return bp;
2099 }

2101 /*
2102  *      enter_target_groups_and_dependencies(target, depes, command, separator,
2103  *                                           target_group_seen)
2104  *
2105  *      Parameters:
2106  *              target          Structure that shows the target(s) on the line
2107  *                              we are currently parsing. This can looks like
```

```
 *                              target1 .. targetN : dependencies
 *                                                   commands
 *                              or
 *                              target1 + .. + targetN : dependencies
 *                                                       commands
 *              depes           Dependencies
 *              command         Points to the command(s) to be executed for
 *                              this target.
 *              separator       : or :: or :=
 *              target_group_seen       Set if we have target1 + .. + targetN
 *
 *
 *      After reading the command lines for a target, this routine
 *      is called to setup the dependencies and the commands for it.
 *      If the target is a % pattern or part of a target group, then
 *      the appropriate routines are called.
 */

void
enter_target_groups_and_dependencies(Name_vector target, Name_vector depes, Cmd_
{
        int                     i;
        Boolean                 reset= true;
        Chain                   target_group_member;
        Percent                 percent_ptr;

        for (; target != NULL; target = target->next) {
                for (i = 0; i < target->used; i++) {
                        if (target->names[i] != NULL) {
                                if (target_group_seen) {
                                        target_group_member =
                                          find_target_groups(target, i, reset);
                                        if(target_group_member == NULL) {
                                                fatal_reader(catgets(catd, 1, 32
                                        }
                                }
                                reset = false;

                                /* If we saw it in the makefile it must be
                                 * a file */
                                target->names[i]->stat.is_file = true;
                                /* Make sure that we use dependencies
                                 * entered for makefiles */
                                target->names[i]->state = build_dont_know;

                                /* If the target is special we delegate
                                 * the processing */
                                if (target->names[i]->special_reader
                                    != no_special) {
                                        special_reader(target->names[i],
                                                       depes,
                                                       command);
                                }
                                /* Check if this is a "a%b : x%y" type rule */
                                else if (target->names[i]->percent) {
                                        percent_ptr =
                                          enter_percent(target->names[i],
                                                        target->target_group[i],
                                                        depes, command);
                                        if (target_group_seen) {
                                                target_group_member->percent_mem
                                                  percent_ptr;
                                        }
                                } else if (target->names[i]->dollar) {
                                        enter_dyntarget(target->names[i]);
                                        enter_dependencies
```



```
                                                (target->names[i],
                                                 target->target_group[i],
                                                 depes,
                                                 command,
                                                 separator);
                                } else {
                                        if (target_group_seen) {
                                                target_group_member->percent_mem
                                                  NULL;
                                        }

                                        enter_dependencies
                                          (target->names[i],
                                           target->target_group[i],
                                           depes,
                                           command,
                                           separator);
                                }
                        }
                }
        }
}


#endif /* ! codereview */
```

```
**********************************************************
   52492 Wed May 20 11:04:11 2015
new/usr/src/cmd/make/bin/make/common/read2.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)read2.cc 1.53 06/12/12
 */

#pragma ident   "@(#)read2.cc   1.53    06/12/12"

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

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

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static  Boolean         built_last_make_run_seen;

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



```
static  Name_vector     enter_member_name(register wchar_t *lib_start, register
extern  Name            normalize_name(register wchar_t *name_string, register i
static  void            read_suffixes_list(register Name_vector depes);
static  void            make_relative(wchar_t *to, wchar_t *result);
static  void            print_rule(register Cmd_line command);
static  void            sh_transform(Name *name, Name *value);


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

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

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

```
128                 }
129         }
130         *string_end = ch;

132         if (makefile_type == reading_cpp_file) {
133                 /* Remove extra ../ constructs if we are reading from a report f
134                 name = normalize_name(string_start, string_end - string_start);
135         } else {
136                 /*
137                  * /tolik, fix bug 1197477/
138                  * Normalize every target name before entering.
139                  * ..//obj/a.o and ../obj//a.o are not two different targets.
140                  * There is only one target ../obj/a.o
141                  */
142                 /*name = GETNAME(string_start, string_end - string_start);*/
143                 name = normalize_name(string_start, string_end - string_start);
144         }

146         /* Internalize the name. Detect the name "+" (target group here) */
147 if(current_names->used != 0 && current_names->names[current_names->used-1] == pl
148         if(name == plus) {
149                 return current_names;
150         }
151 }
152         /* If the current_names vector is full we patch in the one from */
153         /* extra_names */
154         if (current_names->used == VSIZEOF(current_names->names)) {
155                 if (current_names->next != NULL) {
156                         current_names = current_names->next;
157                 } else {
158                         current_names->next = *extra_names;
159                         *extra_names = NULL;
160                         current_names = current_names->next;
161                         current_names->used = 0;
162                         current_names->next = NULL;
163                 }
164         }
165         current_names->target_group[current_names->used] = NULL;
166         current_names->names[current_names->used++] = name;
167         if (name == plus) {
168                 *target_group_seen = true;
169         }
170         if (tail_present && string->free_after_use) {
171                 retmem(string->buffer.start);
172         }
173         return current_names;
174 }

176 /*
177  *      enter_member_name(lib_start, member_start, string_end,
178  *                current_names, extra_names)
179  *
180  *      A string has been found to contain member names.
181  *      (The "lib.a[members]" and "lib.a(members)" notation)
182  *      Handle it pretty much as enter_name() does for simple names.
183  *
184  *      Return value:
185  *                              The name vector that was used
186  *
187  *      Parameters:
188  *              lib_start       Points to the of start of "lib.a(member.o)"
189  *              member_start    Points to "member.o" from above string.
190  *              string_end      Points to char after last of above string.
191  *              current_names   Vector to deposit the name in
192  *              extra_names     Where to get next name vector if we run out
193  *
```



```
194  *      Global variables used:
195  */
196 static Name_vector
197 enter_member_name(register wchar_t *lib_start, register wchar_t *member_start, r
198 {
199         register Boolean        entry = false;
200         wchar_t                 buffer[STRING_BUFFER_LENGTH];
201         Name                    lib;
202         Name                    member;
203         Name                    name;
204         Property                prop;
205         wchar_t                 *memberp;
206         wchar_t                 *q;
207         register int            paren_count;
208         register Boolean        has_dollar;
209         register wchar_t        *cq;
210         Name                    long_member_name = NULL;

212         /* Internalize the name of the library */
213         lib = GETNAME(lib_start, member_start - lib_start);
214         lib->is_member = true;
215         member_start++;
216         if (*member_start == (int) parenleft_char) {
217                 /* This is really the "lib.a((entries))" format */
218                 entry = true;
219                 member_start++;
220         }
221         /* Move the library name to the buffer where we intend to build the */
222         /* "lib.a(member)" for each member */
223         (void) wsncpy(buffer, lib_start, member_start - lib_start);
224         memberp = buffer + (member_start-lib_start);
225         while (1) {
226                 long_member_name = NULL;
227                 /* Skip leading spaces */
228                 for (;
229                      (member_start < string_end) && iswspace(*member_start);
230                      member_start++);
231                 /* Find the end of the member name. Allow nested (). Detect $*/
232                 for (cq = memberp, has_dollar = false, paren_count = 0;
233                      (member_start < string_end) &&
234                      ((*member_start != (int) parenright_char) ||
235                       (paren_count > 0)) &&
236                      !iswspace(*member_start);
237                      *cq++ = *member_start++) {
238                         switch (*member_start) {
239                         case parenleft_char:
240                                 paren_count++;
241                                 break;
242                         case parenright_char:
243                                 paren_count--;
244                                 break;
245                         case dollar_char:
246                                 has_dollar = true;
247                         }
248                 }
249                 /* Internalize the member name */
250                 member = GETNAME(memberp, cq - memberp);
251                 *cq = 0;
252                 if ((q = (wchar_t *) wsrchr(memberp, (int) slash_char)) == NULL)
253                         q = memberp;
254                 }
255                 if ((cq - q > (int) ar_member_name_len) &&
256                     !has_dollar) {
257                         *cq++ = (int) parenright_char;
258                         if (entry) {
259                                 *cq++ = (int) parenright_char;
```

```
260                         }
261                         long_member_name = GETNAME(buffer, cq - buffer);
262                         cq = q + (int) ar_member_name_len;
263                 }
264                 *cq++ = (int) parenright_char;
265                 if (entry) {
266                         *cq++ = (int) parenright_char;
267                 }
268                 /* Internalize the "lib.a(member)" notation for this member */
269                 name = GETNAME(buffer, cq - buffer);
270                 name->is_member = lib->is_member;
271                 if (long_member_name != NULL) {
272                         prop = append_prop(name, long_member_name_prop);
273                         name->has_long_member_name = true;
274                         prop->body.long_member_name.member_name =
275                           long_member_name;
276                 }
277                 /* And add the member prop */
278                 prop = append_prop(name, member_prop);
279                 prop->body.member.library = lib;
280                 if (entry) {
281                         /* "lib.a((entry))" notation */
282                         prop->body.member.entry = member;
283                         prop->body.member.member = NULL;
284                 } else {
285                         /* "lib.a(member)" Notation */
286                         prop->body.member.entry = NULL;
287                         prop->body.member.member = member;
288                 }
289                 /* Handle overflow of current_names */
290                 if (current_names->used == VSIZEOF(current_names->names)) {
291                         if (current_names->next != NULL) {
292                                 current_names = current_names->next;
293                         } else {
294                                 if (*extra_names == NULL) {
295                                         current_names =
296                                           current_names->next =
297                                             ALLOC(Name_vector);
298                                 } else {
299                                         current_names =
300                                           current_names->next =
301                                             *extra_names;
302                                         *extra_names = NULL;
303                                 }
304                                 current_names->used = 0;
305                                 current_names->next = NULL;
306                         }
307                 }
308                 current_names->target_group[current_names->used] = NULL;
309                 current_names->names[current_names->used++] = name;
310                 while (iswspace(*member_start)) {
311                         member_start++;
312                 }
313                 /* Check if there are more members */
314                 if ((*member_start == (int) parenright_char) ||
315                     (member_start >= string_end)) {
316                         return current_names;
317                 }
318         }
319         /* NOTREACHED */
320 }
321
322 /*
323  *      normalize_name(name_string, length)
324  *
325  *      Take a namestring and remove redundant ../, // and ./ constructs
```



```
326  *
327  *      Return value:
328  *                              The normalized name
329  *
330  *      Parameters:
331  *              name_string     Path string to normalize
332  *              length          Length of that string
333  *
334  *      Global variables used:
335  *              dot             The Name ".", compared against
336  *              dotdot          The Name "..", compared against
337  */
338 Name
339 normalize_name(register wchar_t *name_string, register int length)
340 {
341         static Name             dotdot;
342         register wchar_t        *string = ALLOC_WC(length + 1);
343         register wchar_t        *string2;
344         register wchar_t        *cdp;
345         wchar_t                 *current_component;
346         Name                    name;
347         register int            count;
348
349         if (dotdot == NULL) {
350                 MBSTOWCS(wcs_buffer, "..");
351                 dotdot = GETNAME(wcs_buffer, FIND_LENGTH);
352         }
353
354         /*
355          * Copy string removing ./ and //.
356          * First strip leading ./
357          */
358         while ((length > 1) &&
359                (name_string[0] == (int) period_char) &&
360                (name_string[1] == (int) slash_char)) {
361                 name_string += 2;
362                 length -= 2;
363                 while ((length > 0) && (name_string[0] == (int) slash_char)) {
364                         name_string++;
365                         length--;
366                 }
367         }
368         /* Then copy the rest of the string removing /./ & // */
369         cdp = string;
370         while (length > 0) {
371                 if (((length > 2) &&
372                      (name_string[0] == (int) slash_char) &&
373                      (name_string[1] == (int) period_char) &&
374                      (name_string[2] == (int) slash_char)) ||
375                     ((length == 2) &&
376                      (name_string[0] == (int) slash_char) &&
377                      (name_string[1] == (int) period_char))) {
378                         name_string += 2;
379                         length -= 2;
380                         continue;
381                 }
382                 if ((length > 1) &&
383                     (name_string[0] == (int) slash_char) &&
384                     (name_string[1] == (int) slash_char)) {
385                         name_string++;
386                         length--;
387                         continue;
388                 }
389                 *cdp++ = *name_string++;
390                 length--;
391         }
```

```
392         *cdp = (int) nul_char;
393         /*
394          * Now scan for <name>/../ and remove such combinations iff <name>
395          * is not another ..
396          * Each time something is removed, the whole process is restarted.
397          */
398 removed_one:
399         name_string = string;
400         string2 = name_string;          /*save for free*/
401         current_component =
402           cdp =
403             string =
404               ALLOC_WC((length = wslen(name_string)) + 1);
405         while (length > 0) {
406                 if (((length > 3) &&
407                      (name_string[0] == (int) slash_char) &&
408                      (name_string[1] == (int) period_char) &&
409                      (name_string[2] == (int) period_char) &&
410                      (name_string[3] == (int) slash_char)) ||
411                     ((length == 3) &&
412                      (name_string[0] == (int) slash_char) &&
413                      (name_string[1] == (int) period_char) &&
414                      (name_string[2] == (int) period_char))) {
415                         /* Positioned on the / that starts a /.. sequence */
416                         if (((count = cdp - current_component) != 0) &&
417                             (exists(name = GETNAME(string, cdp - string)) > file
418                             (!name->stat.is_sym_link)) {
419                                 name = GETNAME(current_component, count);
420                                 if(name != dotdot) {
421                                         cdp = current_component;
422                                         name_string += 3;
423                                         length -= 3;
424                                         if (length > 0) {
425                                                 name_string++;  /* skip slash */
426                                                 length--;
427                                                 while (length > 0) {
428                                                         *cdp++ = *name_string++;
429                                                         length--;
430                                                 }
431                                         }
432                                         *cdp = (int) nul_char;
433                                         retmem(string2);
434                                         goto removed_one;
435                                 }
436                         }
437                 }
438                 if ((*cdp++ = *name_string++) == (int) slash_char) {
439                         current_component = cdp;
440                 }
441                 length--;
442         }
443         *cdp = (int) nul_char;
444         if (string[0] == (int) nul_char) {
445                 name = dot;
446         } else {
447                 name = GETNAME(string, FIND_LENGTH);
448         }
449         retmem(string);
450         retmem(string2);
451         return name;
452 }

454 /*
455  *      find_target_groups(target_list)
456  *
457  *      If a "+" was seen when the target list was scanned we need to extract
```



```
458  *      the groups. Each target in the name vector that is a member of a
459  *      group gets a pointer to a chain of all the members stuffed in its
460  *      target_group vector slot
461  *
462  *      Parameters:
463  *              target_list     The list of targets that contains "+"
464  *
465  *      Global variables used:
466  *              plus            The Name "+", compared against
467  */
468 Chain
469 find_target_groups(register Name_vector target_list, register int i, Boolean res
470 {
471         static Chain            target_group = NULL;
472         static Chain            tail_target_group = NULL;
473         static Name             *next;
474         static Boolean  clear_target_group = false;

476         if (reset) {
477                 target_group = NULL;
478                 tail_target_group = NULL;
479                 clear_target_group = false;
480         }

482         /* Scan the list of targets */
483         /* If the previous target terminated a group */
484         /* we flush the pointer to that member chain */
485         if (clear_target_group) {
486                 clear_target_group = false;
487                 target_group = NULL;
488         }
489         /* Pick up a pointer to the cell with */
490         /* the next target */
491         if (i + 1 != target_list->used) {
492                 next = &target_list->names[i + 1];
493         } else {
494                 next = (target_list->next != NULL) ?
495                   &target_list->next->names[0] : NULL;
496         }
497         /* We have four states here :
498          *      0:      No target group started and next element is not "+"
499          *              This is not interesting.
500          *      1:      A target group is being built and the next element
501          *              is not "+". This terminates the group.
502          *      2:      No target group started and the next member is "+"
503          *              This is the first target in a group.
504          *      3:      A target group started and the next member is a "+"
505          *              The group continues.
506          */
507         switch ((target_group ? 1 : 0) +
508                 (next && (*next == plus) ?
509                  2 : 0)) {
510               case 0:   /* Not target_group */
511                 break;
512               case 1:   /* Last group member */
513                 /* We need to keep this pointer so */
514                 /* we can stuff it for last member */
515                 clear_target_group = true;
516                 /* fall into */
517               case 3:   /* Middle group member */
518                 /* Add this target to the */
519                 /* current chain */
520                 tail_target_group->next = ALLOC(Chain);
521                 tail_target_group = tail_target_group->next;
522                 tail_target_group->next = NULL;
523                 tail_target_group->name = target_list->names[i];
```

```
524                 break;
525               case 2:   /* First group member */
526                 /* Start a new chain */
527                 target_group = tail_target_group = ALLOC(Chain);
528                 target_group->next = NULL;
529                 target_group->name = target_list->names[i];
530                 break;
531         }
532         /* Stuff the current chain, if any, in the */
533         /* targets group slot */
534         target_list->target_group[i] = target_group;
535         if ((next != NULL) &&
536             (*next == plus)) {
537                 *next = NULL;
538         }
539         return (tail_target_group);
540 }

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



```
591         /* Check if this is a .RECURSIVE line */
592         if ((depes->used >= 3) &&
593             (depes->names[0] == recursive_name)) {
594 #ifdef NSE
595                 nse_did_recursion= true;
596 #endif
597                 target->has_recursive_dependency = true;
598                 depes->names[0] = NULL;
599                 recursive_state = 0;
600                 dp = NULL;
601                 dpp = &dp;
602                 /* Read the dependencies. They are "<directory> <target-made>*/
603                 /* <makefile>*" */
604                 for (; depes != NULL; depes = depes->next) {
605                         for (i = 0; i < depes->used; i++) {
606                                 if (depes->names[i] != NULL) {
607                                         switch (recursive_state++) {
608                                         case 0: /* Directory */
609                                         {
610                                                 depstr.init(depes->names[i]);
611                                                 make_relative(depstr.get_string(
612                                                                 relative);
613                                                 directory =
614                                                   GETNAME(relative,
615                                                           FIND_LENGTH);
616                                         }
617                                                 break;
618                                         case 1: /* Target */
619                                                 name = depes->names[i];
620                                                 break;
621                                         default:        /* Makefiles */
622                                                 *dpp = ALLOC(Dependency);
623                                                 (*dpp)->next = NULL;
624                                                 (*dpp)->name = depes->names[i];
625                                                 (*dpp)->automatic = false;
626                                                 (*dpp)->stale = false;
627                                                 (*dpp)->built = false;
628                                                 dpp = &((*dpp)->next);
629                                                 break;
630                                         }
631                                 }
632                         }
633                 }
634                 /* Check if this recursion already has been reported else */
635                 /* enter the recursive prop for the target */
636                 /* The has_built flag is used to tell if this .RECURSIVE */
637                 /* was discovered from this run (read from a tmp file) */
638                 /* or was from discovered from the original .make.state */
639                 /* file */
640                 for (line = get_prop(target->prop, recursive_prop);
641                      line != NULL;
642                      line = get_prop(line->next, recursive_prop)) {
643                         if ((line->body.recursive.directory == directory) &&
644                             (line->body.recursive.target == name)) {
645                                 line->body.recursive.makefiles = dp;
646                                 line->body.recursive.has_built =
647                                   (Boolean)
648                                     (makefile_type == reading_cpp_file);
649                                 return;
650                         }
651                 }
652                 line2 = append_prop(target, recursive_prop);
653                 line2->body.recursive.directory = directory;
654                 line2->body.recursive.target = name;
655                 line2->body.recursive.makefiles = dp;
```

```
656                 line2->body.recursive.has_built =
657                     (Boolean) (makefile_type == reading_cpp_file);
658                 line2->body.recursive.in_depinfo = false;
659                 return;
660         }
661         /* If this is the first target that doesnt start with a "." in the */
662         /* makefile we remember that */
663         Wstring tstr(target);
664         wchar_t * wcb = tstr.get_string();
665         if ((makefile_type == reading_makefile) &&
666             (default_target_to_build == NULL) &&
667             ((wcb[0] != (int) period_char) ||
668              wschr(wcb, (int) slash_char))) {

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



```
722                         if (trace_reader) {
723                                 (void) printf("%s:\t", target->string_mb);
724                         }
725                         /* Make "target" depend on "<number>%target */
726                         line2 = maybe_append_prop(target, line_prop);
727                         enter_dependency(line2, name, true);
728                         line2->body.line.target = target;
729                         /* Put a prop on "<number>%target that makes */
730                         /* appear as "target" */
731                         /* when it is processed */
732                         maybe_append_prop(name, target_prop)->
733                           body.target.target = target;
734                         target->is_double_colon_parent = true;
735                         name->is_double_colon = true;
736                         name->has_target_prop = true;
737                         if (trace_reader) {
738                                 (void) printf("\n");
739                         }
740                         (target = name)->stat.is_file = true;
741                 }
742         }
743         /* This really is a regular dependency line. Just enter it */
744         line = maybe_append_prop(target, line_prop);
745         line->body.line.target = target;
746         /* Depending on what kind of makefile we are reading we have to */
747         /* treat things differently */
748         switch (makefile_type) {
749         case reading_makefile:
750                 /* Reading regular makefile. Just notice whether this */
751                 /* redefines the rule for the  target */
752                 if (command != NULL) {
753                         if (line->body.line.command_template != NULL) {
754                                 line->body.line.command_template_redefined =
755                                   true;
756                                 if ((wcb[0] == (int) period_char) &&
757                                     !wschr(wcb, (int) slash_char)) {
758                                         line->body.line.command_template =
759                                           command;
760                                 }
761                         } else {
762                                 line->body.line.command_template = command;
763                         }
764                 } else {
765                         if ((wcb[0] == (int) period_char) &&
766                             !wschr(wcb, (int) slash_char)) {
767                                 line->body.line.command_template = command;
768                         }
769                 }
770                 break;
771         case rereading_statefile:
772                 /* Rereading the statefile. We only enter thing that changed */
773                 /* since the previous time we read it */
774                 if (!built_last_make_run_seen) {
775                         for (Cmd_line next, cmd = command; cmd != NULL; cmd = ne
776                                 next = cmd->next;
777                                 free(cmd);
778                         }
779                         return;
780                 }
781                 built_last_make_run_seen = false;
782                 command_changed = true;
783                 target->ran_command = true;
784         case reading_statefile:
785                 /* Reading the statefile for the first time. Enter the rules */
786                 /* as "Commands used" not "templates to use" */
787                 if (command != NULL) {
```

```
788                         for (Cmd_line next, cmd = line->body.line.command_used;
789                              cmd != NULL; cmd = next) {
790                                 next = cmd->next;
791                                 free(cmd);
792                         }
793                         line->body.line.command_used = command;
794                 }
795         case reading_cpp_file:
796                 /* Reading report file from programs that reports */
797                 /* dependencies. If this is the first time the target is */
798                 /* read from this reportfile we clear all old */
799                 /* automatic depes */
800                 if (target->temp_file_number == temp_file_number) {
801                         break;
802                 }
803                 target->temp_file_number = temp_file_number;
804                 command_changed = true;
805                 if (line != NULL) {
806                         for (dp = line->body.line.dependencies;
807                              dp != NULL;
808                              dp = dp->next) {
809                                 if (dp->automatic) {
810                                         dp->stale = true;
811                                 }
812                         }
813                 }
814                 break;
815         default:
816                 fatal_reader(catgets(catd, 1, 93, "Internal error. Unknown makef
817                         makefile_type);
818         }
819         /* A target may only be involved in one target group */
820         if (line->body.line.target_group != NULL) {
821                 if (target_group != NULL) {
822                         fatal_reader(catgets(catd, 1, 94, "Too many target group
823                                 target->string_mb);
824                 }
825         } else {
826                 line->body.line.target_group = target_group;
827         }

829         if (trace_reader) {
830                 (void) printf("%s:\t", target->string_mb);
831         }
832         /* Enter the dependencies */
833         register_as_auto = BOOLEAN(makefile_type != reading_makefile);
834         not_auto_found = false;
835         for (;
836              (depes != NULL) && !not_auto_found;
837              depes = depes->next) {
838                 for (i = 0; i < depes->used; i++) {
839                         /* the dependency .NOT_AUTO signals beginning of
840                          * explicit dependancies which were put at end of
841                          * list in .make.state file - we stop entering
842                          * dependencies at this point
843                          */
844                         if (depes->names[i] == not_auto) {
845                                 not_auto_found = true;
846                                 break;
847                         }
848                         enter_dependency(line,
849                                          depes->names[i],
850                                          register_as_auto);
851                 }
852         }
853         if (trace_reader) {
```



```
854                 (void) printf("\n");
855                 print_rule(command);
856         }
857 }

859 /*
860  *      enter_dependency(line, depe, automatic)
861  *
862  *      Enter one dependency. Do not enter duplicates.
863  *
864  *      Parameters:
865  *              line            The line block that the dependeny is
866  *                              entered for
867  *              depe            The dependency to enter
868  *              automatic       Used to set the field "automatic"
869  *
870  *      Global variables used:
871  *              makefile_type   We do different things for makefile vs. report
872  *              trace_reader    Indicates that we should echo stuff we read
873  *              wait_name       The Name ".WAIT", compared against
874  */
875 void
876 enter_dependency(Property line, register Name depe, Boolean automatic)
877 {
878         register Dependency     dp;
879         register Dependency     *insert;

881         if (trace_reader) {
882                 (void) printf("%s ", depe->string_mb);
883         }
884         /* Find the end of the list and check for duplicates */
885         for (insert = &line->body.line.dependencies, dp = *insert;
886              dp != NULL;
887              insert = &dp->next, dp = *insert) {
888                 if ((dp->name == depe) && (depe != wait_name)) {
889                         if (dp->automatic) {
890                                 dp->automatic = automatic;
891                                 if (automatic) {
892                                         dp->built = false;
893                                         depe->stat.is_file = true;
894 #ifdef NSE
895                                         depe->has_parent= true;
896                                         depe->is_target= true;
897 #endif
898                                 }
899                         }
900                         dp->stale = false;
901                         return;
902                 }
903         }
904         /* Insert the new dependency since we couldnt find it */
905         dp = *insert = ALLOC(Dependency);
906         dp->name = depe;
907         dp->next = NULL;
908         dp->automatic = automatic;
909         dp->stale = false;
910         dp->built = false;
911         depe->stat.is_file = true;
912 #ifdef NSE
913         depe->has_parent= true;
914         depe->is_target= true;
915 #endif

917         if ((makefile_type == reading_makefile) &&
918             (line != NULL) &&
919             (line->body.line.target != NULL)) {
```

```
920                 line->body.line.target->has_regular_dependency = true;
921 #ifdef NSE
922                 line->body.line.target->is_target= true;
923 #endif
924         }
925 }

927 /*
928  *      enter_percent(target, depes, command)
929  *
930  *      Enter "x%y : a%b" type lines
931  *      % patterns are stored in four parts head and tail for target and source
932  *
933  *      Parameters:
934  *              target          Left hand side of pattern
935  *              depes           The dependency list with the rh pattern
936  *              command         The command for the pattern
937  *
938  *      Global variables used:
939  *              empty_name      The Name "", compared against
940  *              percent_list    The list of all percent rules, added to
941  *              trace_reader    Indicates that we should echo stuff we read
942  */
943 Percent
944 enter_percent(register Name target, Chain target_group, register Name_vector dep
945 {
946         register Percent        result = ALLOC(Percent);
947         register Percent        depe;
948         register Percent        *depe_tail = &result->dependencies;
949         register Percent        *insert;
950         register wchar_t        *cp, *cp1;
951         Name_vector             nvp;
952         int                     i;
953         int                     pattern;

955         result->next = NULL;
956         result->patterns = NULL;
957         result->patterns_total = 0;
958         result->command_template = command;
959         result->being_expanded = false;
960         result->name = target;
961         result->dependencies = NULL;
962         result->target_group = target_group;

964         /* get patterns count */
965         Wstring wcb(target);
966         cp = wcb.get_string();
967         while (true) {
968                 cp = (wchar_t *) wschr(cp, (int) percent_char);
969                 if (cp != NULL) {
970                         result->patterns_total++;
971                         cp++;
972                 } else {
973                         break;
974                 }
975         }
976         result->patterns_total++;

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

981         /* then create patterns */
982         cp = wcb.get_string();
983         pattern = 0;
984         while (true) {
985                 cp1 = (wchar_t *) wschr(cp, (int) percent_char);
```



```
986                 if (cp1 != NULL) {
987                         result->patterns[pattern] = GETNAME(cp, cp1 - cp);
988                         cp = cp1 + 1;
989                         pattern++;
990                 } else {
991                         result->patterns[pattern] = GETNAME(cp, (int) target->ha
992                         break;
993                 }
994         }

996         Wstring wcb1;

998         /* build dependencies list */
999         for (nvp = depes; nvp != NULL; nvp = nvp->next) {
1000                for (i = 0; i < nvp->used; i++) {
1001                        depe = ALLOC(Percent);
1002                        depe->next = NULL;
1003                        depe->patterns = NULL;
1004                        depe->patterns_total = 0;
1005                        depe->name = nvp->names[i];
1006                        depe->dependencies = NULL;
1007                        depe->command_template = NULL;
1008                        depe->being_expanded = false;
1009                        depe->target_group = NULL;

1011                        *depe_tail = depe;
1012                        depe_tail = &depe->next;

1014                        if (depe->name->percent) {
1015                                /* get patterns count */
1016                                wcb1.init(depe->name);
1017                                cp = wcb1.get_string();
1018                                while (true) {
1019                                        cp = (wchar_t *) wschr(cp, (int) percent
1020                                        if (cp != NULL) {
1021                                                depe->patterns_total++;
1022                                                cp++;
1023                                        } else {
1024                                                break;
1025                                        }
1026                                }
1027                                depe->patterns_total++;

1029                                /* allocate storage for patterns */
1030                                depe->patterns = (Name *) getmem(sizeof(Name) *

1032                                /* then create patterns */
1033                                cp = wcb1.get_string();
1034                                pattern = 0;
1035                                while (true) {
1036                                        cp1 = (wchar_t *) wschr(cp, (int) percen
1037                                        if (cp1 != NULL) {
1038                                                depe->patterns[pattern] = GETNAM
1039                                                cp = cp1 + 1;
1040                                                pattern++;
1041                                        } else {
1042                                                depe->patterns[pattern] = GETNAM
1043                                                break;
1044                                        }
1045                                }
1046                        }
1047                }
1048        }

1050        /* Find the end of the percent list and append the new pattern */
1051        for (insert = &percent_list; (*insert) != NULL; insert = &(*insert)->nex
```

```
1052         *insert = result;

1054         if (trace_reader) {
1055                 (void) printf("%s:", result->name->string_mb);

1057                 for (depe = result->dependencies; depe != NULL; depe = depe->nex
1058                         (void) printf(" %s", depe->name->string_mb);
1059                 }

1061                 (void) printf("\n");

1063                 print_rule(command);
1064         }

1066         return result;
1067 }

1069 /*
1070  *      enter_dyntarget(target)
1071  *
1072  *      Enter "$$(MACRO) : b" type lines
1073  *
1074  *      Parameters:
1075  *              target          Left hand side of pattern
1076  *
1077  *      Global variables used:
1078  *              dyntarget_list  The list of all percent rules, added to
1079  *              trace_reader    Indicates that we should echo stuff we read
1080  */
1081 Dyntarget
1082 enter_dyntarget(register Name target)
1083 {
1084         register Dyntarget      result = ALLOC(Dyntarget);
1085         Dyntarget               p;
1086         Dyntarget               *insert;
1087         int                             i;

1089         result->next = NULL;
1090         result->name = target;


1093         /* Find the end of the dyntarget list and append the new pattern */
1094         for (insert = &dyntarget_list, p = *insert;
1095              p != NULL;
1096              insert = &p->next, p = *insert);
1097         *insert = result;

1099         if (trace_reader) {
1100                 (void) printf(NOCATGETS("Dynamic target %s:\n"), result->name->s
1101         }
1102         return( result);
1103 }


1106 /*
1107  *      special_reader(target, depes, command)
1108  *
1109  *      Read the pseudo targets make knows about
1110  *      This handles the special targets that should not be entered as regular
1111  *      target/dependency sets.
1112  *
1113  *      Parameters:
1114  *              target          The special target
1115  *              depes           The list of dependencies it was entered with
1116  *              command         The command it was entered with
1117  *
```



```
1118  *      Static variables used:
1119  *              built_last_make_run_seen Set to indicate .BUILT_LAST... seen
1120  *
1121  *      Global variables used:
1122  *              all_parallel    Set to indicate that everything runs parallel
1123  *              svr4            Set when ".SVR4" target is read
1124  *              svr4_name       The Name ".SVR4"
1125  *              posix           Set when ".POSIX" target is read
1126  *              posix_name      The Name ".POSIX"
1127  *              current_make_version The Name "<current version number>"
1128  *              default_rule    Set when ".DEFAULT" target is read
1129  *              default_rule_name The Name ".DEFAULT", used for tracing
1130  *              dot_keep_state  The Name ".KEEP_STATE", used for tracing
1131  *              ignore_errors   Set if ".IGNORE" target is read
1132  *              ignore_name     The Name ".IGNORE", used for tracing
1133  *              keep_state      Set if ".KEEP_STATE" target is read
1134  *              no_parallel_name The Name ".NO_PARALLEL", used for tracing
1135  *              only_parallel   Set to indicate only some targets runs parallel
1136  *              parallel_name   The Name ".PARALLEL", used for tracing
1137  *              precious        The Name ".PRECIOUS", used for tracing
1138  *              sccs_get_name   The Name ".SCCS_GET", used for tracing
1139  *              sccs_get_posix_name The Name ".SCCS_GET_POSIX", used for tracing
1140  *              get_name        The Name ".GET", used for tracing
1141  *              sccs_get_rule   Set when ".SCCS_GET" target is read
1142  *              silent          Set when ".SILENT" target is read
1143  *              silent_name     The Name ".SILENT", used for tracing
1144  *              trace_reader    Indicates that we should echo stuff we read
1145  */
1146 void
1147 special_reader(Name target, register Name_vector depes, Cmd_line command)
1148 {
1149         register int            n;

1151         switch (target->special_reader) {

1153         case svr4_special:
1154                 if (depes->used != 0) {
1155                         fatal_reader(catgets(catd, 1, 98, "Illegal dependencies
1156                                         target->string_mb);
1157                 }
1158                 svr4  = true;
1159                 posix  = false;
1160                 keep_state = false;
1161                 all_parallel = false;
1162                 only_parallel = false;
1163                 if (trace_reader) {
1164                         (void) printf("%s:\n", svr4_name->string_mb);
1165                 }
1166                 break;

1168         case posix_special:
1169                 if(svr4)
1170                   break;
1171                 if (depes->used != 0) {
1172                         fatal_reader(catgets(catd, 1, 99, "Illegal dependencies
1173                                         target->string_mb);
1174                 }
1175                 posix  = true;
1176                         /* with posix on, use the posix get rule */
1177                 sccs_get_rule = sccs_get_posix_rule;
1178                         /* turn keep state off being SunPro make specific */
1179                 keep_state = false;
1180                 #if defined(SUN5_0)
1181                 /* Use /usr/xpg4/bin/sh on Solaris */
1182                 MBSTOWCS(wcs_buffer, NOCATGETS("/usr/xpg4/bin/sh"));
1183                 (void) SETVAR(shell_name, GETNAME(wcs_buffer, FIND_LENGTH), fals
```

```
1184                #endif
1185                if (trace_reader) {
1186                        (void) printf("%s:\n", posix_name->string_mb);
1187                }
1188                break;

1190        case built_last_make_run_special:
1191                built_last_make_run_seen = true;
1192                break;

1194        case default_special:
1195                if (depes->used != 0) {
1196                        warning(catgets(catd, 1, 100, "Illegal dependency list f
1197                                target->string_mb);
1198                }
1199                default_rule = command;
1200                if (trace_reader) {
1201                        (void) printf("%s:\n",
1202                                        default_rule_name->string_mb);
1203                        print_rule(command);
1204                }
1205                break;

1207 #ifdef NSE
1208        case derived_src_special:
1209                for (; depes != NULL; depes= depes->next)
1210                        for (n= 0; n < depes->used; n++) {
1211                                if (trace_reader)
1212                                        (void)printf("%s:\t%s\n",
1213                                        precious->string_mb,
1214                                        depes->names[n]->string_mb);
1215                                depes->names[n]->stat.is_derived_src= true;
1216                        };
1217                break;
1218 #endif

1220        case ignore_special:
1221                if ((depes->used != 0) &&(!posix)){
1222                        fatal_reader(catgets(catd, 1, 101, "Illegal dependencies
1223                                        target->string_mb);
1224                }
1225                if (depes->used == 0)
1226                {
1227                   ignore_errors_all = true;
1228                }
1229                if(svr4) {
1230                  ignore_errors_all = true;
1231                  break;
1232                }
1233                for (; depes != NULL; depes = depes->next) {
1234                        for (n = 0; n < depes->used; n++) {
1235                                depes->names[n]->ignore_error_mode = true;
1236                        }
1237                }
1238                if (trace_reader) {
1239                        (void) printf("%s:\n", ignore_name->string_mb);
1240                }
1241                break;

1243        case keep_state_special:
1244                if(svr4)
1245                  break;
1246                        /* ignore keep state, being SunPro make specific */
1247                if(posix)
1248                  break;
1249                if (depes->used != 0) {
```



```
1250                        fatal_reader(catgets(catd, 1, 102, "Illegal dependencies
1251                                        target->string_mb);
1252                }
1253                keep_state = true;
1254                if (trace_reader) {
1255                        (void) printf("%s:\n",
1256                                        dot_keep_state->string_mb);
1257                }
1258                break;

1260        case keep_state_file_special:
1261                if(svr4)
1262                  break;
1263                if(posix)
1264                  break;
1265                        /* it's not necessary to specify KEEP_STATE, if this
1266                        ** is given, so set the keep_state.
1267                        */
1268                keep_state = true;
1269                if (depes->used != 0) {
1270                   if((!make_state) ||(!strcmp(make_state->string_mb,NOCATGETS("
1271                     make_state = depes->names[0];
1272                   }
1273                }
1274                break;
1275        case make_version_special:
1276                if(svr4)
1277                  break;
1278                if (depes->used != 1) {
1279                        fatal_reader(catgets(catd, 1, 103, "Illegal dependency l
1280                                        target->string_mb);
1281                }
1282                if (depes->names[0] != current_make_version) {
1283                        /*
1284                         * Special case the fact that version 1.0 and 1.1
1285                         * are identical.
1286                         */
1287                        if (!IS_EQUAL(depes->names[0]->string_mb,
1288                                        NOCATGETS("VERSION-1.1")) ||
1289                            !IS_EQUAL(current_make_version->string_mb,
1290                                        NOCATGETS("VERSION-1.0"))) {
1291                                /*
1292                                 * Version mismatches should cause the
1293                                 * .make.state file to be skipped.
1294                                 * This is currently not true - it is read
1295                                 * anyway.
1296                                 */
1297                                warning(catgets(catd, 1, 104, "Version mismatch
1298                                        current_make_version->string_mb,
1299                                        depes->names[0]->string_mb);
1300                        }
1301                }
1302                break;

1304        case no_parallel_special:
1305                if(svr4)
1306                  break;
1307                /* Set the no_parallel bit for all the targets on */
1308                /* the dependency list */
1309                if (depes->used == 0) {
1310                        /* only those explicitly made parallel */
1311                        only_parallel = true;
1312                        all_parallel = false;
1313                }
1314                for (; depes != NULL; depes = depes->next) {
1315                        for (n = 0; n < depes->used; n++) {
```

```
1316                                if (trace_reader) {
1317                                        (void) printf("%s:\t%s\n",
1318                                                      no_parallel_name->string_m
1319                                                      depes->names[n]->string_mb
1320                                }
1321                                depes->names[n]->no_parallel = true;
1322                                depes->names[n]->parallel = false;
1323                        }
1324                }
1325                break;

1327        case parallel_special:
1328                if(svr4)
1329                  break;
1330                if (depes->used == 0) {
1331                        /* everything runs in parallel */
1332                        all_parallel = true;
1333                        only_parallel = false;
1334                }
1335                /* Set the parallel bit for all the targets on */
1336                /* the dependency list */
1337                for (; depes != NULL; depes = depes->next) {
1338                        for (n = 0; n < depes->used; n++) {
1339                                if (trace_reader) {
1340                                        (void) printf("%s:\t%s\n",
1341                                                      parallel_name->string_mb,
1342                                                      depes->names[n]->string_mb
1343                                }
1344                                depes->names[n]->parallel = true;
1345                                depes->names[n]->no_parallel = false;
1346                        }
1347                }
1348                break;

1350        case localhost_special:
1351                if(svr4)
1352                  break;
1353                /* Set the no_parallel bit for all the targets on */
1354                /* the dependency list */
1355                if (depes->used == 0) {
1356                        /* only those explicitly made parallel */
1357                        only_parallel = true;
1358                        all_parallel = false;
1359                }
1360                for (; depes != NULL; depes = depes->next) {
1361                        for (n = 0; n < depes->used; n++) {
1362                                if (trace_reader) {
1363                                        (void) printf("%s:\t%s\n",
1364                                                      localhost_name->string_mb,
1365                                                      depes->names[n]->string_mb
1366                                }
1367                                depes->names[n]->no_parallel = true;
1368                                depes->names[n]->parallel = false;
1369                                depes->names[n]->localhost = true;
1370                        }
1371                }
1372                break;

1374        case precious_special:
1375                if (depes->used == 0) {
1376                        /* everything is precious      */
1377                        all_precious = true;
1378                } else {
1379                        all_precious = false;
1380                }
1381                if(svr4) {
```



```
1382                  all_precious = true;
1383                  break;
1384                }
1385                /* Set the precious bit for all the targets on */
1386                /* the dependency list */
1387                for (; depes != NULL; depes = depes->next) {
1388                        for (n = 0; n < depes->used; n++) {
1389                                if (trace_reader) {
1390                                        (void) printf("%s:\t%s\n",
1391                                                      precious->string_mb,
1392                                                      depes->names[n]->string_mb
1393                                }
1394                                depes->names[n]->stat.is_precious = true;
1395                        }
1396                }
1397                break;

1399        case sccs_get_special:
1400                if (depes->used != 0) {
1401                        fatal_reader(catgets(catd, 1, 105, "Illegal dependencies
1402                                     target->string_mb);
1403                }
1404                sccs_get_rule = command;
1405                sccs_get_org_rule = command;
1406                if (trace_reader) {
1407                        (void) printf("%s:\n", sccs_get_name->string_mb);
1408                        print_rule(command);
1409                }
1410                break;

1412        case sccs_get_posix_special:
1413                if (depes->used != 0) {
1414                        fatal_reader(catgets(catd, 1, 106, "Illegal dependencies
1415                                     target->string_mb);
1416                }
1417                sccs_get_posix_rule = command;
1418                if (trace_reader) {
1419                        (void) printf("%s:\n", sccs_get_posix_name->string_mb);
1420                        print_rule(command);
1421                }
1422                break;

1424        case get_posix_special:
1425                if (depes->used != 0) {
1426                        fatal_reader(catgets(catd, 1, 107, "Illegal dependencies
1427                                     target->string_mb);
1428                }
1429                get_posix_rule = command;
1430                if (trace_reader) {
1431                        (void) printf("%s:\n", get_posix_name->string_mb);
1432                        print_rule(command);
1433                }
1434                break;

1436        case get_special:
1437                if(!svr4) {
1438                  break;
1439                }
1440                if (depes->used != 0) {
1441                        fatal_reader(catgets(catd, 1, 108, "Illegal dependencies
1442                                     target->string_mb);
1443                }
1444                get_rule = command;
1445                sccs_get_rule = command;
1446                if (trace_reader) {
1447                        (void) printf("%s:\n", get_name->string_mb);
```

```
1448                        print_rule(command);
1449                }
1450                break;

1452        case silent_special:
1453                if ((depes->used != 0) && (!posix)){
1454                        fatal_reader(catgets(catd, 1, 109, "Illegal dependencies
1455                                target->string_mb);
1456                }
1457                if (depes->used == 0)
1458                {
1459                   silent_all = true;
1460                }
1461                if(svr4) {
1462                  silent_all = true;
1463                  break;
1464                }
1465                for (; depes != NULL; depes = depes->next) {
1466                        for (n = 0; n < depes->used; n++) {
1467                                depes->names[n]->silent_mode = true;
1468                        }
1469                }
1470                if (trace_reader) {
1471                        (void) printf("%s:\n", silent_name->string_mb);
1472                }
1473                break;

1475        case suffixes_special:
1476                read_suffixes_list(depes);
1477                break;

1479        default:

1481                fatal_reader(catgets(catd, 1, 110, "Internal error: Unknown spec
1482        }
1483 }

1485 /*
1486  *      read_suffixes_list(depes)
1487  *
1488  *      Read the special list .SUFFIXES. If it is empty the old list is
1489  *      cleared. Else the new one is appended. Suffixes with ~ are extracted
1490  *      and marked.
1491  *
1492  *      Parameters:
1493  *              depes           The list of suffixes
1494  *
1495  *      Global variables used:
1496  *              hashtab         The central hashtable for Names.
1497  *              suffixes        The list of suffixes, set or appended to
1498  *              suffixes_name   The Name ".SUFFIXES", used for tracing
1499  *              trace_reader    Indicates that we should echo stuff we read
1500  */
1501 static void
1502 read_suffixes_list(register Name_vector depes)
1503 {
1504        register int            n;
1505        register Dependency     dp;
1506        register Dependency     *insert_dep;
1507        register Name           np;
1508        Name                    np2;
1509        register Boolean        first = true;

1511        if (depes->used == 0) {
1512                /* .SUFFIXES with no dependency list clears the */
1513                /* suffixes list */
```



```
1514                for (Name_set::iterator np = hashtab.begin(), e = hashtab.end();
1515                                np->with_squiggle =
1516                                  np->without_squiggle =
1517                                    false;
1518                }
1519                suffixes = NULL;
1520                if (trace_reader) {
1521                        (void) printf("%s:\n", suffixes_name->string_mb);
1522                }
1523                return;
1524        }
1525        Wstring str;
1526        /* Otherwise we append to the list */
1527        for (; depes != NULL; depes = depes->next) {
1528                for (n = 0; n < depes->used; n++) {
1529                        np = depes->names[n];
1530                        /* Find the end of the list and check if the */
1531                        /* suffix already has been entered */
1532                        for (insert_dep = &suffixes, dp = *insert_dep;
1533                             dp != NULL;
1534                             insert_dep = &dp->next, dp = *insert_dep) {
1535                                if (dp->name == np) {
1536                                        goto duplicate_suffix;
1537                                }
1538                        }
1539                        if (trace_reader) {
1540                                if (first) {
1541                                        (void) printf("%s:\t",
1542                                                        suffixes_name->string_mb);
1543                                        first = false;
1544                                }
1545                                (void) printf("%s ", depes->names[n]->string_mb)
1546                        }
1547                if(!(posix|svr4)) {
1548                        /* If the suffix is suffixed with "~" we */
1549                        /* strip that and mark the suffix nameblock */
1550                        str.init(np);
1551                        wchar_t * wcb = str.get_string();
1552                        if (wcb[np->hash.length - 1] ==
1553                            (int) tilde_char) {
1554                                np2 = GETNAME(wcb,
1555                                                (int)(np->hash.length - 1));
1556                                np2->with_squiggle = true;
1557                                if (np2->without_squiggle) {
1558                                        continue;
1559                                }
1560                                np = np2;
1561                        }
1562                }
1563                        np->without_squiggle = true;
1564                        /* Add the suffix to the list */
1565                        dp = *insert_dep = ALLOC(Dependency);
1566                        insert_dep = &dp->next;
1567                        dp->next = NULL;
1568                        dp->name = np;
1569                        dp->built = false;
1570                duplicate_suffix:;
1571                }
1572        }
1573        if (trace_reader) {
1574                (void) printf("\n");
1575        }
1576 }

1578 /*
1579  *      make_relative(to, result)
```

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

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

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

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

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

        /*
         * Now for some special cases. Check for exact matches and
         * for either name terminating exactly.
         */
        if (*from == (int) nul_char) {
                if (*to == (int) nul_char) {
                        MBSTOWCS(wcs_buffer, ".");
                        (void) wscpy(result, wcs_buffer);
                        retmem(allocated);
                        return;
```



```
                }
                if (*to == (int) slash_char) {
                        ncomps--;
                        tocomp = &to[1];
                }
        } else if ((*from == (int) slash_char) && (*to == (int) nul_char)) {
                ncomps--;
                tocomp = to;
        }
        /* Add on the ".."s. */
        for (i = 0; i < ncomps; i++) {
                MBSTOWCS(wcs_buffer, "../");
                (void) wscat(result, wcs_buffer);
        }

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

/*
 *      print_rule(command)
 *
 *      Used when tracing the reading of rules
 *
 *      Parameters:
 *              command         Command to print
 *
 *      Global variables used:
 */
static void
print_rule(register Cmd_line command)
{
        for (; command != NULL; command = command->next) {
                (void) printf("\t%s\n", command->command_line->string_mb);
        }
}

/*
 *      enter_conditional(target, name, value, append)
 *
 *      Enter "target := MACRO= value" constructs
 *
 *      Parameters:
 *              target          The target the macro is for
 *              name            The name of the macro
 *              value           The value for the macro
 *              append          Indicates if the assignment is appending or not
 *
 *      Global variables used:
 *              conditionals    A special Name that stores all conditionals
 *                              where the target is a % pattern
 *              trace_reader    Indicates that we should echo stuff we read
 */
void
enter_conditional(register Name target, Name name, Name value, register Boolean
{
        register Property       conditional;
        static int              sequence;
        Name                    orig_target = target;
```

```
1713         if (name == target_arch) {
1714                 enter_conditional(target, virtual_root, virtual_root, false);
1715         }
1716
1717         if (target->percent) {
1718                 target = conditionals;
1719         }
1720
1721         if (name->colon) {
1722                 sh_transform(&name, &value);
1723         }
1724
1725         /* Count how many conditionals we must activate before building the */
1726         /* target */
1727         if (target->percent) {
1728                 target = conditionals;
1729         }
1730
1731         target->conditional_cnt++;
1732         maybe_append_prop(name, macro_prop)->body.macro.is_conditional = true;
1733         /* Add the property for the target */
1734         conditional = append_prop(target, conditional_prop);
1735         conditional->body.conditional.target = orig_target;
1736         conditional->body.conditional.name = name;
1737         conditional->body.conditional.value = value;
1738         conditional->body.conditional.sequence = sequence++;
1739         conditional->body.conditional.append = append;
1740         if (trace_reader) {
1741                 if (value == NULL) {
1742                         (void) printf("%s := %s %c=\n",
1743                                       target->string_mb,
1744                                       name->string_mb,
1745                                       append ?
1746                                       (int) plus_char : (int) space_char);
1747                 } else {
1748                         (void) printf("%s := %s %c= %s\n",
1749                                       target->string_mb,
1750                                       name->string_mb,
1751                                       append ?
1752                                       (int) plus_char : (int) space_char,
1753                                       value->string_mb);
1754                 }
1755         }
1756 }
1757
1758 /*
1759  *      enter_equal(name, value, append)
1760  *
1761  *      Enter "MACRO= value" constructs
1762  *
1763  *      Parameters:
1764  *              name            The name of the macro
1765  *              value           The value for the macro
1766  *              append          Indicates if the assignment is appending or not
1767  *
1768  *      Global variables used:
1769  *              trace_reader    Indicates that we should echo stuff we read
1770  */
1771 void
1772 enter_equal(Name name, Name value, register Boolean append)
1773 {
1774         wchar_t         *string;
1775         Name            temp;
1776
1777         if (name->colon) {
```



```
1778                 sh_transform(&name, &value);
1779         }
1780         (void) SETVAR(name, value, append);
1781
1782         /* if we're setting FC, we want to set F77 to the same value. */
1783         Wstring nms(name);
1784         wchar_t * wcb = nms.get_string();
1785         string = wcb;
1786         if (string[0]=='F' &&
1787             string[1]=='C' &&
1788             string[2]=='\0') {
1789                 MBSTOWCS(wcs_buffer, NOCATGETS("F77"));
1790                 temp = GETNAME(wcs_buffer, FIND_LENGTH);
1791                 (void) SETVAR(temp, value, append);
1792 /*
1793                 fprintf(stderr, catgets(catd, 1, 111, "warning: FC is obsolete,
1794  */
1795         }
1796
1797         if (trace_reader) {
1798                 if (value == NULL) {
1799                         (void) printf("%s %c=\n",
1800                                       name->string_mb,
1801                                       append ?
1802                                       (int) plus_char : (int) space_char);
1803                 } else {
1804                         (void) printf("%s %c= %s\n",
1805                                       name->string_mb,
1806                                       append ?
1807                                       (int) plus_char : (int) space_char,
1808                                       value->string_mb);
1809                 }
1810         }
1811 }
1812
1813 /*
1814  *      sh_transform(name, value)
1815  *
1816  *      Parameters:
1817  *              name    The name of the macro we might transform
1818  *              value   The value to transform
1819  *
1820  */
1821 static void
1822 sh_transform(Name *name, Name *value)
1823 {
1824         /* Check if we need :sh transform */
1825         wchar_t         *colon;
1826         String_rec      command;
1827         String_rec      destination;
1828         wchar_t         buffer[1000];
1829         wchar_t         buffer1[1000];
1830
1831         static wchar_t  colon_sh[4];
1832         static wchar_t  colon_shell[7];
1833
1834         if (colon_sh[0] == (int) nul_char) {
1835                 MBSTOWCS(colon_sh, NOCATGETS(":sh"));
1836                 MBSTOWCS(colon_shell, NOCATGETS(":shell"));
1837         }
1838         Wstring nms((*name));
1839         wchar_t * wcb = nms.get_string();
1840
1841         colon = (wchar_t *) wsrchr(wcb, (int) colon_char);
1842         if ((colon != NULL) && (IS_WEQUAL(colon, colon_sh) || IS_WEQUAL(colon, c
1843                 INIT_STRING_FROM_STACK(destination, buffer);
```

```
1845                 if(*value == NULL) {
1846                         buffer[0] = 0;
1847                 } else {
1848                         Wstring wcb1((*value));
1849                         if (IS_WEQUAL(colon, colon_shell)) {
1850                                 INIT_STRING_FROM_STACK(command, buffer1);
1851                                 expand_value(*value, &command, false);
1852                         } else {
1853                                 command.text.p = wcb1.get_string() + (*value)->h
1854                                 command.text.end = command.text.p;
1855                                 command.buffer.start = wcb1.get_string();
1856                                 command.buffer.end = command.text.p;
1857                         }
1858                         sh_command2string(&command, &destination);
1859                 }

1861                 (*value) = GETNAME(destination.buffer.start, FIND_LENGTH);
1862                 *colon = (int) nul_char;
1863                 (*name) = GETNAME(wcb, FIND_LENGTH);
1864                 *colon = (int) colon_char;
1865         }
1866 }

1868 /*
1869  *      fatal_reader(format, args...)
1870  *
1871  *      Parameters:
1872  *              format          printf style format string
1873  *              args            arguments to match the format
1874  *
1875  *      Global variables used:
1876  *              file_being_read Name of the makefile being read
1877  *              line_number     Line that is being read
1878  *              report_pwd      Indicates whether current path should be shown
1879  *              temp_file_name  When reading tempfile we report that name
1880  */
1881 /*VARARGS*/
1882 void
1883 fatal_reader(char * pattern, ...)
1884 {
1885         va_list args;
1886         char    message[1000];

1888         va_start(args, pattern);
1889         if (file_being_read != NULL) {
1890                 WCSTOMBS(mbs_buffer, file_being_read);
1891                 if (line_number != 0) {
1892                         (void) sprintf(message,
1893                                        catgets(catd, 1, 112, "%s, line %d: %s"),
1894                                        mbs_buffer,
1895                                        line_number,
1896                                        pattern);
1897                 } else {
1898                         (void) sprintf(message,
1899                                        "%s: %s",
1900                                        mbs_buffer,
1901                                        pattern);
1902                 }
1903                 pattern = message;
1904         }

1906         (void) fflush(stdout);
1907 #ifdef DISTRIBUTED
1908         (void) fprintf(stderr, catgets(catd, 1, 113, "dmake: Fatal error in read
1909 #else
```



```
1910         (void) fprintf(stderr, catgets(catd, 1, 238, "make: Fatal error in reade
1911 #endif
1912         (void) vfprintf(stderr, pattern, args);
1913         (void) fprintf(stderr, "\n");
1914         va_end(args);

1916         if (temp_file_name != NULL) {
1917                 (void) fprintf(stderr,
1918 #ifdef DISTRIBUTED
1919                                catgets(catd, 1, 114, "dmake: Temp-file %s not re
1920 #else
1921                                catgets(catd, 1, 239, "make: Temp-file %s not rem
1922 #endif
1923                                temp_file_name->string_mb);
1924                 temp_file_name = NULL;
1925         }

1927         if (report_pwd) {
1928                 (void) fprintf(stderr,
1929                                catgets(catd, 1, 115, "Current working directory
1930                                get_current_path());
1931         }
1932         (void) fflush(stderr);
1933 #if defined(SUN5_0) || defined(HP_UX)
1934         exit_status = 1;
1935 #endif
1936         exit(1);
1937 }

1939 #endif /* ! codereview */
```

```
**********************************************************
   12815 Wed May 20 11:04:11 2015
new/usr/src/cmd/make/bin/make/common/rep.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2003 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)rep.cc 1.25 06/12/12
 */

#pragma ident   "@(#)rep.cc     1.25    06/12/12"

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

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

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

/*
 * File table of contents
 */



/*
 *      report_recursive_init()
 *
 *      Read the .nse_depinfo file and make a list of all the
 *      .RECURSIVE entries.
 *
```



```
 *      Parameters:
 *
 *      Static variables used:
 *              bpatch          Points to slot where next cell should be added
 *
 *      Global variables used:
 *              recursive_name  The Name ".RECURSIVE", compared against
 */

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

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

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

                colon = (wchar_t *) wschr(line, (int) colon_char);
                if (colon == NULL) {
                        continue;
                }
                dollar = (wchar_t *) wschr(line, (int) dollar_char);
```

```
128                 line[wslen(line) - 1] = (int) nul_char;
129                 if (IS_WEQUALN(&colon[2], wcb,
130                     (int) recursive_name->hash.length)) {
131                         /*
132                          * If this entry is an old entry, ignore it
133                          */
134                         MBSTOWCS(wcs_buffer, DEPINFO_FMT_VERSION);
135                         if (dollar == NULL ||
136                             !IS_WEQUALN(wcs_buffer, (dollar+1) - VER_LEN, VER_LE
137                                 continue;
138                             }
139                         rp = ALLOC(Recursive_make);
140                         (void) memset((char *) rp, 0, sizeof (Recursive_make_rec
141                         /*
142                          * set conditional_macro_string if string is present
143                          */
144                         rp->oldline = (wchar_t *) wsdup(line);
145                         if ( dollar != NULL ){
146                                 rp->cond_macrostring =
147                                     (wchar_t *) wsdup(dollar - VER_LEN + 1);
148                         }
149                         /*
150                          * get target name into recursive struct
151                          */
152                         *colon = (int) nul_char;
153                         rp->target = (wchar_t *) wsdup(line);
154                         *bpatch = rp;
155                         bpatch = &rp->next;
156                 }
157         }
158         (void) fclose(fp);
159 }

161 /*
162  *      report_recursive_dep(target, line)
163  *
164  *      Report a target as recursive.
165  *
166  *      Parameters:
167  *              line            Dependency line reported
168  *
169  *      Static variables used:
170  *              bpatch          Points to slot where next cell should be added
171  *              changed         Written if report set changed
172  */
173 void
174 report_recursive_dep(Name target, wchar_t *line)
175 {
176         Recursive_make  rp;
177         wchar_t         rec_buf[STRING_BUFFER_LENGTH];
178         String_rec      string;

180         INIT_STRING_FROM_STACK(string, rec_buf);
181         cond_macros_into_string(target, &string);
182         /*
183          * find an applicable recursive entry, if there isn't one, create it
184          */
185         rp = find_recursive_target(target);
186         if (rp == NULL) {
187                 rp = ALLOC(Recursive_make);
188                 (void) memset((char *) rp, 0, sizeof (Recursive_make_rec));
189                 wchar_t * wcb = get_wstring(target->string_mb); // XXX Tolik: ne
190                 rp->target = wcb;
191                 rp->newline = (wchar_t *) wsdup(line);
192                 rp->cond_macrostring = (wchar_t *) wsdup(rec_buf);
193                 *bpatch = rp;
```



```
194                 bpatch = &rp->next;
195                 changed = true;
196         } else {
197                 if ((rp->oldline != NULL) && !IS_WEQUAL(rp->oldline, line)) {
198                         rp->newline = (wchar_t *) wsdup(line);
199                         changed = true;
200                 }
201                 rp->removed = false;
202         }
203 }

205 /*
206  *      find_recursive_target(target)
207  *
208  *      Search the list for a given target.
209  *
210  *      Return value:
211  *                              The target cell
212  *
213  *      Parameters:
214  *              target          The target we need
215  *              top_level_target more info used to determinde the
216  *                               target we need
217  *
218  *      Static variables used:
219  *              recursive_list  The list of targets
220  */
221 Recursive_make
222 find_recursive_target(Name target)
223 {
224         Recursive_make  rp;
225         String_rec      string;
226         wchar_t         rec_buf[STRING_BUFFER_LENGTH];

228         INIT_STRING_FROM_STACK(string, rec_buf);
229         cond_macros_into_string(target, &string);

231         Wstring tstr(target);
232         wchar_t * wcb = tstr.get_string();
233         for (rp = recursive_list; rp != NULL; rp = rp->next) {
234                 /*
235                  * If this entry has already been removed, ignore it.
236                  */
237                 if (rp->removed)
238                         continue;
239                 /*
240                  * If this target, and the target on the list are the same
241                  * and if one of them contains conditional macro info, while
242                  * the other doesn't,  remove this entry from the list of
243                  * recursive entries.  This can only happen if the Makefile
244                  * has changed to no longer contain conditional macros.
245                  */
246                 if (IS_WEQUAL(rp->target, wcb)) {
247                         if (rp->cond_macrostring[VER_LEN] == '\0' &&
248                             string.buffer.start[VER_LEN] != '\0'){
249                                 rp->removed = true;
250                                 continue;
251                         } else if (rp->cond_macrostring[VER_LEN] != '\0' &&
252                             string.buffer.start[VER_LEN] == '\0'){
253                                 rp->removed = true;
254                                 continue;
255                         }
256                 }
257                 /*
258                  * If this is not a VERS2 entry,  only need to match
259                  * the target name.  toptarg information from VERS1 entries
```

```
260                  * are ignored.
261                  */
262                 MBSTOWCS(wcs_buffer, DEPINFO_FMT_VERSION);
263                 if (IS_WEQUALN(wcs_buffer, string.buffer.start, VER_LEN)) {
264                         if (IS_WEQUAL(rp->cond_macrostring,
265                             string.buffer.start) &&
266                             IS_WEQUAL(rp->target, wcb)) {
267                                 return rp;
268                         }
269                 } else {
270                         if (IS_WEQUAL(rp->target, wcb)) {
271                                 return rp;
272                         }
273                 }
274         }
275         return NULL;
276 }

278 /*
279  *      remove_recursive_dep(target, top_level_target)
280  *
281  *      Mark a target as no longer recursive.
282  *
283  *      Parameters:
284  *              target          The target we want to remove
285  *              top_level_target target we want to remove must be built from
286  *                               the same top level target
287  *
288  *      Static variables used:
289  *              changed         Written if report set changed
290  */
291 void
292 remove_recursive_dep(Name target)
293 {
294         Recursive_make  rp;

296         rp = find_recursive_target(target);

298         if ( rp != NULL ) {
299                 rp->removed = true;
300                 changed = true;
301                 if(rp->target) {
302                         retmem(rp->target);
303                         rp->target = NULL;
304                 }
305                 if(rp->newline) {
306                         retmem(rp->newline);
307                         rp->newline = NULL;
308                 }
309                 if(rp->oldline) {
310                         retmem(rp->oldline);
311                         rp->oldline = NULL;
312                 }
313                 if(rp->cond_macrostring) {
314                         retmem(rp->cond_macrostring);
315                         rp->cond_macrostring = NULL;
316                 }
317         }
318 }

320 #ifdef NSE
321 /*
322  *      report_recursive_done()
323  *
324  *      Write the .nse_depinfo file.
325  *
```



```
326  *      Parameters:
327  *
328  *      Static variables used:
329  *              recursive_list  The list of targets
330  *              changed         Written if report set changed
331  *
332  *      Global variables used:
333  *              recursive_name  The Name ".RECURSIVE", compared against
334  */
335 void
336 report_recursive_done(void)
337 {
338         char            *search_dir;
339         char            nse_depinfo[MAXPATHLEN];
340         char            tmpfile[MAXPATHLEN];
341         FILE            *ofp;
342         FILE            *ifp;
343         wchar_t         *space;
344         wchar_t         *data;
345         wchar_t         *line;
346         wchar_t         *bigger_line;
347         int             line_size, line_index;
348         int             lock_err;
349         Recursive_make  rp;

351         if (changed == false) {
352                 return;
353         }

355         search_dir = getenv(NOCATGETS("NSE_DEP"));
356         if (search_dir == NULL) {
357                 return;
358         }
359         (void) sprintf(nse_depinfo, "%s/%s", search_dir, NSE_DEPINFO);
360         (void) sprintf(tmpfile, "%s.%d", nse_depinfo, getpid());
361         ofp = fopen(tmpfile, "w");
362         if (ofp == NULL) {
363                 (void) fprintf(stderr,
364                                 catgets(catd, 1, 116, "Cannot open ‘%s’ for writi
365                                 tmpfile);
366                 return;
367         }
368         (void) sprintf(nse_depinfo_lockfile,
369                         "%s/%s", search_dir, NSE_DEPINFO_LOCK);
370         if (lock_err = file_lock(nse_depinfo,
371                                  nse_depinfo_lockfile,
372                                  (int *) &nse_depinfo_locked, 0)) {
373                 (void) fprintf(stderr,
374                                 catgets(catd, 1, 117, "writing .RECURSIVE lines t
375                                 tmpfile);
376                 (void) fprintf(stderr,
377                                 catgets(catd, 1, 118, "To recover, merge .nse_dep
378                                 getpid(),
379                                 catgets(catd, 1, 119, "with .nse_depinfo"));
380         }

382         if (nse_depinfo_locked) {
383                 ifp = fopen(nse_depinfo, "r");
384                 if (ifp != NULL) {
385                         /*
386                          * Copy all the non-.RECURSIVE lines from
387                          * the old file to the new one.
388                          */
389                         line_size = MAXPATHLEN;
390                         line_index = line_size - 1;
391                         line = ALLOC_WC(line_size);
```

```
392                         while (fgetws(line, line_size, ifp) != NULL) {
393                                 while (wslen(line) == line_index) {
394                                         if (line[wslen(line) - 1] == '\n') {
395                                                 continue;
396                                         }
397                                         bigger_line = ALLOC_WC(2 * line_size);
398                                         wscpy(bigger_line, line);
399                                         retmem(line);
400                                         line = bigger_line;
401                                         if (fgetws(&line[line_index],
402                                                 line_size, ifp) == NULL)
403                                           continue;
404                                         line_index = 2 * line_index;
405                                         line_size = 2 * line_size;
406                                 }

408                                 space = wschr(line, (int) space_char);
409                                 if (space != NULL &&
410                                     IS_WEQUALN(&space[1],
411                                             recursive_name->string,
412                                             (int) recursive_name->hash.length)
413                                         continue;
414                                 }
415                                 WCSTOMBS(mbs_buffer, line);
416                                 (void) fprintf(ofp, "%s", mbs_buffer);
417                         }
418                         (void) fclose(ifp);
419                 }
420         }

422         /*
423          * Write out the .RECURSIVE lines.
424          */
425         for (rp = recursive_list; rp != NULL; rp = rp->next) {
426                 if (rp->removed) {
427                         continue;
428                 }
429                 if (rp->newline != NULL) {
430                         data = rp->newline;
431                 } else {
432                         data = rp->oldline;
433                 }
434                 if (data != NULL) {
435                         WCSTOMBS(mbs_buffer, data);
436                         (void) fprintf(ofp, "%s\n", mbs_buffer);
437                 }
438         }
439         (void) fclose(ofp);

441         if (nse_depinfo_locked) {
442                 (void) rename(tmpfile, nse_depinfo);
443                 (void) unlink(nse_depinfo_lockfile);
444                 nse_depinfo_locked = false;
445                 nse_depinfo_lockfile[0] = '\0';
446                 (void) chmod(nse_depinfo, 0666);
447         }
448 }
449 #endif // NSE

451 /* gather_recursive_deps()
452  *
453  *      Create or update list of recursive targets.
454  */
455 void
456 gather_recursive_deps(void)
457 {
```



```
458         Name_set::iterator      np, e;
459         String_rec              rec;
460         wchar_t                 rec_buf[STRING_BUFFER_LENGTH];
461         register Property       lines;
462         Boolean                 has_recursive;
463         Dependency              dp;

465         report_recursive_init();

467         /* Go thru all targets and dump recursive dependencies */
468         for (np = hashtab.begin(), e = hashtab.end(); np != e; np++) {
469                 if (np->has_recursive_dependency){
470                         has_recursive = false;
471                         /*
472                          * start .RECURSIVE line with target:
473                          */
474                         INIT_STRING_FROM_STACK(rec, rec_buf);
475                         APPEND_NAME(np, &rec, FIND_LENGTH);
476                         append_char((int) colon_char, &rec);
477                         append_char((int) space_char, &rec);
478
479                         for (lines = get_prop(np->prop,recursive_prop);
480                             lines != NULL;
481                             lines = get_prop(lines->next, recursive_prop)) {
482                                 /*
483                                  * if entry is already in depinfo
484                                  * file or entry was not built, ignore it
485                                  */
486                                 if (lines->body.recursive.in_depinfo)
487                                         continue;
488                                 if (!lines->body.recursive.has_built)
489                                         continue;
490                                 has_recursive = true;
491                                 lines->body.recursive.in_depinfo=true;
492
493                                 /*
494                                 * Write the remainder of the
495                                 * .RECURSIVE line
496                                 */
497                                 APPEND_NAME(recursive_name, &rec,
498                                     FIND_LENGTH);
499                                 append_char((int) space_char, &rec);
500                                 APPEND_NAME(lines->body.recursive.directory,
501                                         &rec, FIND_LENGTH);
502                                 append_char((int) space_char, &rec);
503                                 APPEND_NAME(lines->body.recursive.target,
504                                         &rec, FIND_LENGTH);
505                                 append_char((int) space_char, &rec);
506
507                                 /* Complete list of makefiles used */
508                                 for (dp = lines->body.recursive.makefiles;
509                                     dp != NULL;
510                                     dp = dp->next) {
511                                         APPEND_NAME(dp->name, &rec,  FIND_LENGTH
512                                         append_char((int) space_char, &rec);
513                                 }
514                         }
515                         /*
516                          * dump list of conditional targets,
517                          * and report recursive entry, if needed
518                          */
519                         cond_macros_into_string(np, &rec);
520                         if (has_recursive){
521                                 report_recursive_dep(np, rec.buffer.start);
522                         }
```

```
524                 } else if ( np->has_built ) {
525                         remove_recursive_dep(np);
526                 }
527         }
528 }

530 #endif /* ! codereview */
```

```
**********************************************************
   12783 Wed May 20 11:04:11 2015
new/usr/src/cmd/make/bin/make/common/state.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)state.cc 1.27 06/12/12
 */

#pragma ident   "@(#)state.cc   1.27    06/12/12"

/*
 *      state.c
 *
 *      This file contains the routines that write the .make.state file
 */

/*
 * Included files
 */
#include <mk/defs.h>
#include <mksh/misc.h>          /* errmsg() */
#include <setjmp.h>             /* setjmp() */
#include <unistd.h>             /* getpid() */
#include <errno.h>              /* errno    */
#include <locale.h>             /* MB_CUR_MAX    */

/*
 * Defined macros
 */
#define LONGJUMP_VALUE 17
#define XFWRITE(string, length, fd) {if (fwrite(string, 1, length, fd) == 0) \
                                        longjmp(long_jump, LONGJUMP_VALUE);}
#define XPUTC(ch, fd) { \
        if (putc((int) ch, fd) == EOF) \
                longjmp(long_jump, LONGJUMP_VALUE); \
        }
#define XFPUTS(string, fd) fputs(string, fd)

/*
 * typedefs & structs
 */
```



```
/*
 * Static variables
 */

/*
 * File table of contents
 */
static char * escape_target_name(Name np)
{
        if(np->dollar) {
                int len = strlen(np->string_mb);
                char * buff = (char*)malloc(2 * len);
                int pos = 0;
                wchar_t wc;
                int pp = 0;
                while(pos < len) {
                        int n = mbtowc(&wc, np->string_mb + pos, MB_CUR_MAX);
                        if(n < 0) { // error - this shouldn't happen
                                (void)free(buff);
                                return strdup(np->string_mb);
                        }
                        if(wc == dollar_char) {
                                buff[pp] = '\\'; pp++;
                                buff[pp] = '$'; pp++;
                        } else {
                                for(int j=0;j<n;j++) {
                                        buff[pp] = np->string_mb[pos+j]; pp++;
                                }
                        }
                        pos += n;
                }
                buff[pp] = '\0';
                return buff;
        } else {
                return strdup(np->string_mb);
        }
}

static  void            print_auto_depes(register Dependency dependency, registe

/*
 *      write_state_file(report_recursive, exiting)
 *
 *      Write a new version of .make.state
 *
 *      Parameters:
 *              report_recursive        Should only be done at end of run
 *              exiting                 true if called from the exit handler
 *
 *      Global variables used:
 *              built_last_make_run The Name ".BUILT_LAST_MAKE_RUN", written
 *              command_changed If no command changed we do not need to write
 *              current_make_version The Name "<current version>", written
 *              do_not_exec_rule If -n is on we do not write statefile
 *              hashtab         The hashtable that contains all names
 *              keep_state      If .KEEP_STATE is no on we do not write file
 *              make_state      The Name ".make.state", used for opening file
 *              make_version    The Name ".MAKE_VERSION", written
 *              recursive_name  The Name ".RECURSIVE", written
 *              rewrite_statefile Indicates that something changed
 */

void
#ifdef NSE
write_state_file(int report_recursive, Boolean exiting)
```

```
#else
write_state_file(int, Boolean exiting)
#endif
{
        register FILE           *fd;
        int                     lock_err;
        char                    buffer[MAXPATHLEN];
        char                    make_state_tempfile[MAXPATHLEN];
        jmp_buf                 long_jump;
        register int            attempts = 0;
        Name_set::iterator      np, e;
        register Property       lines;
        register int            m;
        Dependency              dependency;
        register Boolean        name_printed;
        Boolean                 built_this_run = false;
        char                    *target_name;
        int                     line_length;
        register Cmd_line       cp;


        if (!rewrite_statefile ||
            !command_changed ||
            !keep_state ||
            do_not_exec_rule ||
            (report_dependencies_level > 0)) {
                return;
        }
        /* Lock the file for writing. */
        make_state_lockfile = getmem(strlen(make_state->string_mb) + strlen(NOCA
        (void) sprintf(make_state_lockfile,
                       NOCATGETS("%s.lock"),
                       make_state->string_mb);
        if (lock_err = file_lock(make_state->string_mb,
                                 make_state_lockfile,
                                 (int *) &make_state_locked, 0)) {
                retmem_mb(make_state_lockfile);
                make_state_lockfile = NULL;

                /*
                 * We need to make sure that we are not being
                 * called by the exit handler so we don't call
                 * it again.
                 */

                if (exiting) {
                        (void) sprintf(buffer, NOCATGETS("%s/.make.state.%d.XXXX
                        report_pwd = true;
                        warning(catgets(catd, 1, 60, "Writing to %s"), buffer);
                        int fdes = mkstemp(buffer);
                        if ((fdes < 0) || (fd = fdopen(fdes, "w")) == NULL) {
                                fprintf(stderr,
                                        catgets(catd, 1, 61, "Could not open sta
                                        buffer,
                                        errmsg(errno));
                                return;
                        }
                } else {
                        report_pwd = true;
                        fatal(catgets(catd, 1, 62, "Can't lock .make.state"));
                }
        }

        (void) sprintf(make_state_tempfile,
                       NOCATGETS("%s.tmp"),
                       make_state->string_mb);
```



```
        /* Delete old temporary statefile (in case it exists) */
        (void) unlink(make_state_tempfile);
        if ((fd = fopen(make_state_tempfile, "w")) == NULL) {
                lock_err = errno; /* Save it! unlink() can change errno */
                (void) unlink(make_state_lockfile);
                retmem_mb(make_state_lockfile);
                make_state_lockfile = NULL;
                make_state_locked = false;
                fatal(catgets(catd, 1, 59, "Could not open temporary statefile '
                      make_state_tempfile,
                      errmsg(lock_err));
        }
#ifdef NSE
        if (nse) {
                (void) fchmod(fileno(fd), 0666);
        }
#endif
        /*
         * Set a trap for failed writes. If a write fails, the routine
         * will try saving the .make.state file under another name in /tmp.
         */
        if (setjmp(long_jump)) {
                (void) fclose(fd);
                if (attempts++ > 5) {
                        if ((make_state_lockfile != NULL) &&
                            make_state_locked) {
                                (void) unlink(make_state_lockfile);
                                retmem_mb(make_state_lockfile);
                                make_state_lockfile = NULL;
                                make_state_locked = false;
                        }
                        fatal(catgets(catd, 1, 63, "Giving up on writing statefi
                }
                sleep(10);
                (void) sprintf(buffer, NOCATGETS("%s/.make.state.%d.XXXXXX"), tm
                int fdes = mkstemp(buffer);
                if ((fdes < 0) || (fd = fdopen(fdes, "w")) == NULL) {
                        fatal(catgets(catd, 1, 64, "Could not open statefile '%s
                              buffer,
                              errmsg(errno));
                }
                warning(catgets(catd, 1, 65, "Initial write of statefile failed.
                        buffer);
        }

        /* Write the version stamp. */
        XFWRITE(make_version->string_mb,
                strlen(make_version->string_mb),
                fd);
        XPUTC(colon_char, fd);
        XPUTC(tab_char, fd);
        XFWRITE(current_make_version->string_mb,
                strlen(current_make_version->string_mb),
                fd);
        XPUTC(newline_char, fd);

        /*
         * Go through all the targets, dump their dependencies and
         * command used.
         */
        for (np = hashtab.begin(), e = hashtab.end(); np != e; np++) {
                /*
                 * If the target has no command used nor dependencies,
                 * we can go to the next one.
                 */
                if ((lines = get_prop(np->prop, line_prop)) == NULL) {
```

```
260                         continue;
261                 }
262                 /* If this target is a special target, don't print. */
263                 if (np->special_reader != no_special) {
264                         continue;
265                 }
266                 /*
267                  * Find out if any of the targets dependencies should
268                  * be written to .make.state.
269                  */
270                 for (m = 0, dependency = lines->body.line.dependencies;
271                      dependency != NULL;
272                      dependency = dependency->next) {
273                         if (m = !dependency->stale
274                             && (dependency->name != force)
275 #ifndef PRINT_EXPLICIT_DEPEN
276                             && dependency->automatic
277 #endif
278                             ) {
279                                 break;
280                         }
281                 }
282                 /* Only print if dependencies listed. */
283                 if (m || (lines->body.line.command_used != NULL)) {
284                         name_printed = false;
285                         /*
286                          * If this target was built during this make run,
287                          * we mark it.
288                          */
289                         built_this_run = false;
290                         if (np->has_built) {
291                                 built_this_run = true;
292                                 XFWRITE(built_last_make_run->string_mb,
293                                         strlen(built_last_make_run->string_mb),
294                                         fd);
295                                 XPUTC(colon_char, fd);
296                                 XPUTC(newline_char, fd);
297                         }
298                         /* If the target has dependencies, we dump them. */
299                         target_name = escape_target_name(np);
300                         if (np->has_long_member_name) {
301                                 target_name =
302                                   get_prop(np->prop, long_member_name_prop)
303                                     ->body.long_member_name.member_name->
304                                       string_mb;
305                         }
306                         if (m) {
307                                 XFPUTS(target_name, fd);
308                                 XPUTC(colon_char, fd);
309                                 XFPUTS("\t", fd);
310                                 name_printed = true;
311                                 line_length = 0;
312                                 for (dependency =
313                                      lines->body.line.dependencies;
314                                      dependency != NULL;
315                                      dependency = dependency->next) {
316                                         print_auto_depes(dependency,
317                                                          fd,
318                                                          built_this_run,
319                                                          &line_length,
320                                                          target_name,
321                                                          long_jump);
322                                 }
323                                 XFPUTS("\n", fd);
324                         }
325                         /* If there is a command used, we dump it. */
```



```
326                         if (lines->body.line.command_used != NULL) {
327                                 /*
328                                  * Only write the target name if it
329                                  * wasn't done for the dependencies.
330                                  */
331                                 if (!name_printed) {
332                                         XFPUTS(target_name, fd);
333                                         XPUTC(colon_char, fd);
334                                         XPUTC(newline_char, fd);
335                                 }
336                                 /*
337                                  * Write the command lines.
338                                  * Prefix each textual line with a tab.
339                                  */
340                                 for (cp = lines->body.line.command_used;
341                                      cp != NULL;
342                                      cp = cp->next) {
343                                         char            *csp;
344                                         int             n;
345
346                                         XPUTC(tab_char, fd);
347                                         if (cp->command_line != NULL) {
348                                                 for (csp = cp->
349                                                         command_line->
350                                                         string_mb,
351                                                      n = strlen(cp->
352                                                              command_line->
353                                                              string_mb);
354                                                      n > 0;
355                                                      n--, csp++) {
356                                                         XPUTC(*csp, fd);
357                                                         if (*csp ==
358                                                             (int) newline_char)
359                                                                 XPUTC(tab_char,
360                                                                       fd);
361                                                 }
362                                         }
363                                         XPUTC(newline_char, fd);
364                                 }
365                         }
366                         (void)free(target_name);
367                 }
368         }
369         if (fclose(fd) == EOF) {
370                 longjmp(long_jump, LONGJUMP_VALUE);
371         }
372         if (attempts == 0) {
373                 if (unlink(make_state->string_mb) != 0 && errno != ENOENT) {
374                         lock_err = errno; /* Save it! unlink() can change errno
375                         /* Delete temporary statefile */
376                         (void) unlink(make_state_tempfile);
377                         (void) unlink(make_state_lockfile);
378                         retmem_mb(make_state_lockfile);
379                         make_state_lockfile = NULL;
380                         make_state_locked = false;
381                         fatal(catgets(catd, 1, 356, "Could not delete old statef
382                               make_state->string_mb,
383                               errmsg(lock_err));
384                 }
385                 if (rename(make_state_tempfile, make_state->string_mb) != 0) {
386                         lock_err = errno; /* Save it! unlink() can change errno
387                         /* Delete temporary statefile */
388                         (void) unlink(make_state_tempfile);
389                         (void) unlink(make_state_lockfile);
390                         retmem_mb(make_state_lockfile);
391
```

```
392                         make_state_lockfile = NULL;
393                         make_state_locked = false;
394                         fatal(catgets(catd, 1, 357, "Could not rename ‘%s’ to ‘%
395                               make_state_tempfile,
396                               make_state->string_mb,
397                               errmsg(lock_err));
398                 }
399         }
400         if ((make_state_lockfile != NULL) && make_state_locked) {
401                 (void) unlink(make_state_lockfile);
402                 retmem_mb(make_state_lockfile);
403                 make_state_lockfile = NULL;
404                 make_state_locked = false;
405         }
406 #ifdef NSE
407         if (report_recursive) {
408                 report_recursive_done();
409         }
410 #endif
411 }

413 /*
414  *      print_auto_depes(dependency, fd, built_this_run,
415  *                       line_length, target_name, long_jump)
416  *
417  *      Will print a dependency list for automatic entries.
418  *
419  *      Parameters:
420  *              dependency      The dependency to print
421  *              fd              The file to print it to
422  *              built_this_run  If on we prefix each line with .BUILT_THIS...
423  *              line_length     Pointer to line length var that we update
424  *              target_name     We need this when we restart line
425  *              long_jump       setjmp/longjmp buffer used for IO error action
426  *
427  *      Global variables used:
428  *              built_last_make_run The Name ".BUILT_LAST_MAKE_RUN", written
429  *              force           The Name " FORCE", compared against
430  */
431 static void
432 print_auto_depes(register Dependency dependency, register FILE *fd, register Boo
433 {
434         if (!dependency->automatic ||
435             dependency->stale ||
436             (dependency->name == force)) {
437                 return;
438         }
439         XFWRITE(dependency->name->string_mb,
440                 strlen(dependency->name->string_mb),
441                 fd);
442         /*
443          * Check if the dependency line is too long.
444          * If so, break it and start a new one.
445          */
446         if ((*line_length += (int) strlen(dependency->name->string_mb) + 1) > 45
447                 *line_length = 0;
448                 XPUTC(newline_char, fd);
449                 if (built_this_run) {
450                         XFPUTS(built_last_make_run->string_mb, fd);
451                         XPUTC(colon_char, fd);
452                         XPUTC(newline_char, fd);
453                 }
454                 XFPUTS(target_name, fd);
455                 XPUTC(colon_char, fd);
456                 XPUTC(tab_char, fd);
457         } else {
```



```
458                 XFPUTS(" ", fd);
459         }
460         return;
461 }


464 #endif /* ! codereview */
```

```
**********************************************************
    4637 Wed May 20 11:04:11 2015
new/usr/src/cmd/make/bin/make/common/svr4.make.rules.file
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 1994 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)svr4.make.rules.file 1.4 06/12/12
#
.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f .f~ \
.C .C~ .Y .Y~ .L .L~

MAKE=make
BUILD=build
AR=ar
ARFLAGS=rv
AS=as
ASFLAGS=
CC=cc
CFLAGS=-O
F77=f77
FFLAGS=-O
GET=get
GFLAGS=
LD=ld
LDFLAGS=
LEX=lex
LFLAGS=
YACC=yacc
YFLAGS=
C++C=CC
C++FLAGS=-O


.c:
        $(CC) $(CFLAGS) $< -o $@ $(LDFLAGS)
.c~:
        $(GET) $(GFLAGS) $<
        $(CC) $(CFLAGS) $*.c -o $@ $(LDFLAGS)
        -rm -f $*.c
.f:
        $(F77) $(FFLAGS) $< -o $@ $(LDFLAGS)
.f~:
        $(GET) $(GFLAGS) $<
        $(F77) $(FFLAGS) $*.f -o $@ $(LDFLAGS)
```



```
        -rm -f $*.f
.s:
        $(AS) $(ASFLAGS) $< -o $@ $(LDFLAGS)
.s~:
        $(GET) $(GFLAGS) $<
        $(AS) $(ASFLAGS) $*.s -o $* $(LDFLAGS)
        -rm -f $*.s
.sh:
        cp $< $@; chmod 0777 $@
.sh~:
        $(GET) $(GFLAGS) $<
        cp $*.sh $*; chmod 0777 $@
        -rm -f $*.sh
.C:
        $(C++C) $(C++FLAGS) $< -o $@ $(LDFLAGS)
.C~:
        $(GET) $(GFLAGS) $<
        $(C++C) $(C++FLAGS) $*.C -o $@ $(LDFLAGS)
        -rm -f $*.C

.c.a:
        $(CC) $(CFLAGS) -c $<
        $(AR) $(ARFLAGS) $@ $*.o
        -rm -f $*.o
.c.o:
        $(CC) $(CFLAGS) -c $<
.c~.a:
        $(GET) $(GFLAGS) $<
        $(CC) $(CFLAGS) -c $*.c
        $(AR) $(ARFLAGS) $@ $*.o
        -rm -f $*.[co]
.c~.c:
        $(GET) $(GFLAGS) $<
.c~.o:
        $(GET) $(GFLAGS) $<
        $(CC) $(CFLAGS) -c $*.c
        -rm -f $*.c
.f.a:
        $(F77) $(FFLAGS) -c $*.f
        $(AR) $(ARFLAGS) $@ $*.o
        -rm -f $*.o
.f.o:
        $(F77) $(FFLAGS) -c $*.f
.f~.a:
        $(GET) $(GFLAGS) $<
        $(F77) $(FFLAGS) -c $*.f
        $(AR) $(ARFLAGS) $@ $*.o
        -rm -f $*.[fo]
.f~.f:
        $(GET) $(GFLAGS) $<
.f~.o:
        $(GET) $(GFLAGS) $<
        $(F77) $(FFLAGS) -c $*.f
        -rm -f $*.f
.h~.h:
        $(GET) $(GFLAGS) $<
.l.c:
        $(LEX) $(LFLAGS) $<
        mv lex.yy.c $@
.l.o:
        $(LEX) $(LFLAGS) $<
        $(CC) $(CFLAGS) -c lex.yy.c
        -rm lex.yy.c; mv lex.yy.o $@
.l~.c:
        $(GET) $(GFLAGS) $<
        $(LEX) $(LFLAGS) $*.l
```

```
128         mv lex.yy.c $@
129         -rm -f $*.l
130 .l~.l:
131         $(GET) $(GFLAGS) $<
132 .l~.o:
133         $(GET) $(GFLAGS) $<
134         $(LEX) $(LFLAGS) $*.l
135         $(CC) $(CFLAGS) -c lex.yy.c
136         -rm -f lex.yy.c $*.l
137         mv lex.yy.o $@
138 .s.a:
139         $(AS) $(ASFLAGS) -o $*.o $*.s
140         $(AR) $(ARFLAGS) $@ $*.o
141 .s.o:
142         $(AS) $(ASFLAGS) -o $@ $<
143 .s~.a:
144         $(GET) $(GFLAGS) $<
145         $(AS) $(ASFLAGS) -o $*.o $*.s
146         $(AR) $(ARFLAGS) $@ $*.o
147         -rm -f $*.[so]
148 .s~.o:
149         $(GET) $(GFLAGS) $<
150         $(AS) $(ASFLAGS) -o $*.o $*.s
151         -rm -f $*.s
152 .s~.s:
153         $(GET) $(GFLAGS) $<
154 .sh~.sh:
155         $(GET) $(GFLAGS) $<
156 .y.c:
157         $(YACC) $(YFLAGS) $<
158         mv y.tab.c $@
159 .y.o:
160         $(YACC) $(YFLAGS) $<
161         $(CC) $(CFLAGS) -c y.tab.c
162         -rm y.tab.c
163         mv y.tab.o $@
164 .y~.c:
165         $(GET) $(GFLAGS) $<
166         $(YACC) $(YFLAGS) $*.y
167         mv y.tab.c $*.c
168         -rm -f $*.y
169 .y~.o:
170         $(GET) $(GFLAGS) $<
171         $(YACC) $(YFLAGS) $*.y
172         $(CC) $(CFLAGS) -c y.tab.c
173         -rm -f y.tab.c $*.y
174         mv y.tab.o $*.o
175 .y~.y :
176         $(GET) $(GFLAGS) $<
177 .C.a:
178         $(C++C) $(C++FLAGS) -c $<
179         $(AR) $(ARFLAGS) $@ $*.o
180         -rm -f $*.o
181 .C.o:
182         $(C++C) $(C++FLAGS) -c $<
183 .C~.a:
184         $(GET) $(GFLAGS) $<
185         $(C++C) $(C++FLAGS) -c $*.C
186         $(AR) $(ARFLAGS) $@ $*.o
187         -rm -f $*.[Co]
188 .C~.C:
189         $(GET) $(GFLAGS) $<
190 .C~.o:
191         $(GET) $(GFLAGS) $<
192         $(C++C) $(C++FLAGS) -c $*.C
193         -rm -f $*.C
```



```
194 .L.C:
195         $(LEX) $(LFLAGS) $<
196         mv lex.yy.c $@
197 .L.o:
198         $(LEX) $(LFLAGS) $<
199         $(C++C) $(C++FLAGS) -c lex.yy.c
200         -rm lex.yy.c; mv lex.yy.o $@
201 .L~.C:
202         $(GET) $(GFLAGS) $<
203         $(LEX) $(LFLAGS) $*.L
204         mv lex.yy.c $@
205         -rm -f $*.L
206 .L~.L:
207         $(GET) $(GFLAGS) $<
208 .L~.o:
209         $(GET) $(GFLAGS) $<
210         $(LEX) $(LFLAGS) $*.L
211         $(C++C) $(C++FLAGS) -c lex.yy.c
212         -rm -f lex.yy.c $*.L
213         mv lex.yy.c $@
214 .Y.C:
215         $(YACC) $(YFLAGS) $<
216         mv y.tab.c $@
217 .Y.o:
218         $(YACC) $(YFLAGS) $<
219         $(C++C) $(C++FLAGS) -c y.tab.c
220         -rm y.tab.c
221         mv y.tab.o $@
222 .Y~.C:
223         $(GET) $(GFLAGS) $<
224         $(YACC) $(YFLAGS) $*.Y
225         mv y.tab.c $*.C
226         -rm -f $*.Y
227 .Y~.o:
228         $(GET) $(GFLAGS) $<
229         $(YACC) $(YFLAGS) $*.Y
230         $(C++C) $(C++FLAGS) -c y.tab.c
231         -rm -f y.tab.c $*.Y
232         mv y.tab.o $*.o
233 .Y~.Y :
234         $(GET) $(GFLAGS) $<

236 markfile.o:     markfile
237         echo "static char _sccsid[] = \"‘grep @’(#)’ markfile‘\";" > markfile.c
238         $(CC) -c markfile.c
239         -rm -f markfile.c

241 .SCCS_GET:
242         $(GET) $(GFLAGS)  s.$@
243 #endif /* ! codereview */
```

****
1293 Wed May 20 11:04:11 2015
new/usr/src/cmd/make/bin/make/make.svr4/Makefile
make: initial Sun make source, disconnected from the build
****

```
#
# 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 2001 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.6 06/12/12
#

TOP=../../../..
include $(TOP)/rules/master.mk

all clean clobber l10n_install:

install:
        $(INSTALL) -d $(DESTDIR)/usr/lib
        $(INSTALL) -d $(DESTDIR)/usr/ccs/lib
        rm -f $(DESTDIR)/usr/lib/svr4.make
        rm -f $(DESTDIR)/usr/ccs/lib/svr4.make
        ln $(DESTDIR)/usr/ccs/bin/make $(DESTDIR)/usr/lib/svr4.make
        (cd $(DESTDIR)/usr/ccs/lib; ln -s ../../lib/svr4.make svr4.make)
#endif /* ! codereview */
```

****
1186 Wed May 20 11:04:12 2015
new/usr/src/cmd/make/bin/make/make.xpg4/Makefile
make: initial Sun make source, disconnected from the build
****

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.2 06/12/12
#
# @(#)Makefile  1.1     95/06/29 SMI

TOP=../../../..
include $(TOP)/rules/master.mk

all clean clobber l10n_install:

install:
        $(INSTALL) -d $(DESTDIR)/usr/xpg4/bin
        rm -f $(DESTDIR)/usr/xpg4/bin/make
        ln $(DESTDIR)/usr/ccs/bin/make $(DESTDIR)/usr/xpg4/bin/make
#endif /* ! codereview */
```

```
**********************************************************
    1012 Wed May 20 11:04:12 2015
new/usr/src/cmd/make/bin/make/smake/Makefile
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.5 06/12/12
#

TOP             = ../../../..
include $(TOP)/rules/variant.mk
include $(TOP)/rules/derived.mk

#endif /* ! codereview */
```

```
**********************************************************
    1587 Wed May 20 11:04:13 2015
new/usr/src/cmd/make/bin/make/smake/src/Makefile
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.3 06/12/12
#

# Generic makefile for use in src directories.  Knows how to make common things
# in the right $(VARIANT) directory.

#TOP =          ../../../../..
include $(TOP)/rules/variant.mk

all :=          TARG = all
install :=              TARG = install
clean :=                TARG = clean
test :=         TARG = test
l10n_install := TARG = l10n_install
i18n_install := TARG = i18n_install

SRC =           ../src
MFLAGS +=       SRC=$(SRC)

# See $(TOP)/rules/master.mk for how these are built.
%.h %.cc %.C %.E %.o all install clean test l10n_install i18n_install: FRC
        @ if [ ! -d ../$(VARIANT) ]; then \
                mkdir ../$(VARIANT) ; \
        fi
        cd ../$(VARIANT); $(MAKE) $(MFLAGS) -f $(SRC)/Variant.mk DESTDIR=$(DESTD

FRC:
#endif /* ! codereview */
```

```
**********************************************************
    3792 Wed May 20 11:04:13 2015
new/usr/src/cmd/make/bin/make/smake/src/Variant.mk
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2005 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Variant.mk 1.36 06/12/12
#

TOP             = ../../../../..
include $(TOP)/rules/master.mk
include $(TOP)/Make/lib/Lib.mk

# RELEASE_VER should be "Generic" for FCS. Otherwise it should be overridden
# to display build number - "Build XX", patch number - "Patch XXXXXX-XX", etc.

PKG_TOP         = ../../../..
PROG            = make
PACKAGE         = SUNWspro
PRODVER         = 11
PRODVER_V       = SunOS 5.11
RELEASE_VER     = Generic
DATE:sh         = date '+%B %Y'

VERSTRING       = RELEASE VERSION $(PRODVER_V) $(RELEASE_VER) $(DATE)

MORE_SRC        = \
                ar.cc \
                depvar.cc \
                doname.cc \
                dosys.cc \
                files.cc \
                globals.cc \
                implicit.cc \
                macro.cc \
                main.cc \
                misc.cc \
                nse_printdep.cc \
                read.cc \
                read2.cc \
                rep.cc \
                state.cc

CPPFLAGS        += -I$(PKG_TOP)/include
```



```
LDLIB           += \
                -lnsl \
                -lsocket \
                -lw

HDRS_DIR        = $(PKG_TOP)/include/mk
HDRS_LIST       = $(HDRS_DIR)/copyright.h $(HDRS_DIR)/defs.h

.INIT: $(HDRS_LIST)

SRCS = $(PROG).cc $(MORE_SRC)
OBJS = $(SRCS:%.cc=%.o)

LIB +=          -lintl -lm

all install:: $(PROG)

$(PROG): $(OBJS)
        $(CCC) $(CCFLAGS) $(LDFLAGS) -o $@ \
                $(OBJS) $(LIB) $(I18LIB) $(LDLIB)
        /bin/sh $(TOP)/exe/sanity-check.sh $(TOP) $(PROG)

clean:
        $(RM) $(OBJS) $(PROG)


include $(TOP)/rules/computed-options.mk

#
# This LIB macro must be declared after the include's above
#
LIB             = \
                $(LIBMKSH) \
                $(LIBMKSDMSI18N) \
                $(LIBBSD) \
                $(LIBVROOT)

%.o: ../../common/%.cc
        $(COMPILE.cc) $(OUTPUT_OPTION) $<

depvar.o nse.o nse_printdep.o := CPPFLAGS += -DSUNOS4_AND_AFTER

LDFLAGS +=      -xildoff -norunpath

install:: install-make-hdrs install-make-bin

IHDR            = make.rules.file
VIHDR           = svr4.make.rules.file
HDR             = ${PKG_TOP}/bin/make/common/make.rules.file
VHDR            = ${PKG_TOP}/bin/make/common/svr4.make.rules.file
HDRSDIR         = $(DESTDIR)/usr/share/lib/make
HDRFILE         = make.rules
VHDRFILE        = svr4.make.rules
SMAKEDIR        = $(DESTDIR)/usr/ccs/bin

install-make-bin: make
        $(INSTALL) -d $(SMAKEDIR)
        $(INSTALL) make $(SMAKEDIR)
        mcs -d $(SMAKEDIR)/make
        mcs -a '@(#)$(VERSTRING)' $(SMAKEDIR)/make

install-make-hdrs: $(HDR) $(VHDR)
        $(INSTALL) -d $(HDRSDIR)
        $(INSTALL) -m 0444 $(HDR) $(HDRSDIR)
        mv -f $(HDRSDIR)/$(IHDR) $(HDRSDIR)/$(HDRFILE)
```

```
128         $(INSTALL) -m 0444 $(VHDR) $(HDRSDIR)
129         mv -f $(HDRSDIR)/$(VIHDR) $(HDRSDIR)/$(VHDRFILE)


132 #
133 # i18n stuff
134 #
135 MAKE_MSG        = SUNW_SPRO_MAKE.msg
136 #I18N_DIRS      = ../../common ../../../../lib/bsd/src ../../../../lib/mksh/src
137 I18N_DIRS       = ../../common
138 TEXTDOMAIN      = SUNW_SPRO_MAKE
139 APPPATH         = $(PKG_TOP)/bin/make/smake/$(VARIANT)
140 LIB_DESTDIR     = $(DESTDIR)/usr/lib
141 CAT_DESTDIR     = $(LIB_DESTDIR)/locale/C/LC_MESSAGES

143 $(CAT_DESTDIR):
144         $(INSTALL) -d $@

146 msg_catalogs: $(CAT_DESTDIR) .WAIT MAKE_MSG
147         cp $(APPPATH)/$(TEXTDOMAIN).msg $(CAT_DESTDIR)

149 MAKE_MSG:
150         $(GENMSG) -l $(SRC)/genmsg.project -o $(TEXTDOMAIN).msg `find $(I18N_DIR
151         rm -f *.cc.new

153 i18n_install: msg_catalogs

155 FRC:
156 #endif /* ! codereview */
```

****
1624 Wed May 20 11:04:13 2015
new/usr/src/cmd/make/include/avo/avo_alloca.h
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1998 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)avo_alloca.h 1.4 06/12/12
 */

#pragma ident   "@(#)avo_alloca.h 1.4     06/12/12"

#ifndef _AVO_ALLOCA_H
#define _AVO_ALLOCA_H

#include <alloca.h>

#ifdef __SunOS_5_4
// The following prototype declaration is necessary when compiling on Solaris
// 2.4 using 5.0 compilers. On Solaris 2.4 the necessary prototypes are not
// included in alloca.h. The 4.x compilers provide a workaround by declaring the
// prototype as a pre-defined type. The 5.0 compilers do not implement this work
// This can be removed when support for 2.4 is dropped

#include <stdlib.h>     // for size_t
extern "C" void *__builtin_alloca(size_t);

#endif  // ifdef __SunOS_5_4

#endif  // ifdef _AVO_ALLOCA_H

#endif /* ! codereview */
```

```
**********************************************************
    2187 Wed May 20 11:04:13 2015
new/usr/src/cmd/make/include/avo/intl.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 2001 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)intl.h 1.19 06/12/12
 */

#pragma ident   "@(#)intl.h     1.19    06/12/12"

#ifndef _AVO_INTL_H
#define _AVO_INTL_H

#if defined(SUN4_x) || defined(HP_UX)
#include <avo/widefake.h>
#endif

/*
 * For catgets
 */
#include <nl_types.h>

#ifdef HP_UX
#ifdef __cplusplus
#ifndef _STDLIB_INCLUDED
#include <stdlib.h>             /* for wchar_t definition and HP-UX - */
#endif                          /* wide character function prototypes. */
extern "C" {
char *gettext(char *msg);
char *dgettext(const char *, const char *);
char *bindtextdomain(const char *, const char *);
char *textdomain(char *);
}
#endif /* __cplusplus */
#endif

/*
 * NOCATGETS is a dummy macro that returns it argument.
 * It is used to identify strings that we consciously do not
 * want to apply catgets() to.  We have tools that check the
 * sources for strings that are not catgets'd and the tools
```



```
 * ignore strings that are NOCATGETS'd.
 */
#define NOCATGETS(str)  (str)

/*
 * Define the various text domains
 */
#define AVO_DOMAIN_CODEMGR      "codemgr"
#define AVO_DOMAIN_VERTOOL      "vertool"
#define AVO_DOMAIN_FILEMERGE    "filemerge"
#define AVO_DOMAIN_DMAKE        "dmake"
#define AVO_DOMAIN_PMAKE        "pmake"
#define AVO_DOMAIN_FREEZEPOINT  "freezept"
#define AVO_DOMAIN_MAKETOOL     "maketool"

#endif
#endif /* ! codereview */
```

```
**********************************************************
    1508 Wed May 20 11:04:13 2015
new/usr/src/cmd/make/include/bsd/bsd.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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.
 */
/*
 * @(#)bsd.h 1.6 06/12/12
 */

#pragma ident   "@(#)bsd.h      1.6     06/12/12"

/*
 * bsd/bsd.h: Interface definitions to BSD compatibility functions for SVR4.
 */

#ifndef _BSD_BSD_H
#define _BSD_BSD_H

#include <signal.h>

#if defined (HP_UX) || defined (linux)
typedef void SIG_FUNC_TYP(int);
typedef SIG_FUNC_TYP *SIG_TYP;
#define SIG_PF SIG_TYP
#endif

#ifndef __cplusplus
typedef void (*SIG_PF) (int);
#endif

#ifdef __cplusplus
extern "C" SIG_PF bsd_signal(int a, SIG_PF b);
#else
extern void (*bsd_signal(int, void (*) (int))) (int);
#endif
extern void bsd_signals(void);

#endif

#endif /* ! codereview */
```

```
**********************************************************
     965 Wed May 20 11:04:14 2015
new/usr/src/cmd/make/include/mk/copyright.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)copyright.h 1.11 06/12/12
 */
#endif /* ! codereview */
```

```
**********************************************************
   16641 Wed May 20 11:04:14 2015
new/usr/src/cmd/make/include/mk/defs.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#ifndef _MK_DEFS_H
#define _MK_DEFS_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)defs.h 1.61 06/12/12
 */

#pragma ident   "@(#)defs.h     1.61    06/12/12"

/*
 * Included files
 */
#ifdef DISTRIBUTED
#       include <dm/Avo_AcknowledgeMsg.h>
#       include <dm/Avo_DoJobMsg.h>
#       include <dm/Avo_JobResultMsg.h>
#endif

#include <mksh/defs.h>

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
#       include <rw/xdrstrea.h>
#endif


/*
 * Defined macros
 */

#define SKIPSPACE(x)    while (*x &&                            \
                               ((*x == (int) space_char) ||     \
                                (*x == (int) tab_char) ||       \
                                (*x == (int) comma_char))) {    \
                                       x++;                     \
                        }

#define SKIPWORD(x)     while (*x &&                            \
                               (*x != (int) space_char) &&      \
                               (*x != (int) tab_char) &&        \
                               (*x != (int) newline_char) &&    \
                               (*x != (int) comma_char) &&      \
                               (*x != (int) equal_char)) {      \
                                       x++;                     \
                        }

#define SKIPTOEND(x)    while (*x &&                            \
                               (*x != (int) newline_char)) {    \
                                       x++;                     \
                        }

#define PMAKE_DEF_MAX_JOBS      2       /* Default number of parallel jobs. */

#define OUT_OF_DATE(a,b) \
        (((a) < (b)) || (((a) == file_doesnt_exist) && ((b) == file_doesnt_exist

#define OUT_OF_DATE_SEC(a,b) \
        (((a).tv_sec < (b).tv_sec) || (((a).tv_sec == file_doesnt_exist.tv_sec)

#define SETVAR(name, value, append) \
                                setvar_daemon(name, value, append, no_daemon, \
                                              true, debug_level)
#ifdef SUN5_0
#define MAX(a,b)                (((a)>(b))?(a):(b))
/*
 * New feature added to SUN5_0 make,  invoke the vanilla svr4 make when
 * the USE_SVR4_MAKE environment variable is set.
 */
#define SVR4_MAKE               "/usr/ccs/lib/svr4.make"
#define USE_SVR4_MAKE           "USE_SVR4_MAKE"
#endif
/*
 * The standard MAXHOSTNAMELEN is 64. We want 32.
 */
#define MAX_HOSTNAMELEN         32


/*
 * typedefs & structs
 */
typedef enum {
        no_state,
        scan_name_state,
        scan_command_state,
        enter_dependencies_state,
        enter_conditional_state,
        enter_equal_state,
        illegal_bytes_state,
        illegal_eoln_state,
        poorly_formed_macro_state,
        exit_state
}                       Reader_state;

struct _Name_vector {
        struct _Name            *names[64];
        struct _Chain           *target_group[64];
        short                   used;
        struct _Name_vector     *next;
};

struct _Running {
        struct _Running         *next;
        Doname                  state;
        struct _Name            *target;
        struct _Name            *true_target;
```

```
128         struct _Property        *command;
129         struct _Name            *sprodep_value;
130         char                    *sprodep_env;
131         int                     recursion_level;
132         Boolean                 do_get;
133         Boolean                 implicit;
134         Boolean                 redo;
135         int                     auto_count;
136         struct _Name            **automatics;
137         pid_t                   pid;
138 #ifdef TEAMWARE_MAKE_CMN
139         int                     job_msg_id;
140 #else
141         int                     host;
142 #endif
143         char                    *stdout_file;
144         char                    *stderr_file;
145         struct _Name            *temp_file;
146         int                     conditional_cnt;
147         struct _Name            **conditional_targets;
148 #ifdef TEAMWARE_MAKE_CMN
149         Boolean                 make_refd;
150 #endif
151 };

153 typedef enum {
154         serial_mode,
155         parallel_mode,
156         distributed_mode
157 } DMake_mode;

159 typedef enum {
160         txt1_mode,
161         txt2_mode,
162         html1_mode
163 } DMake_output_mode;

165 struct _Recursive_make {
166         struct _Recursive_make  *next;  /* Linked list */
167         wchar_t                 *target;/* Name of target */
168         wchar_t                 *oldline;/* Original line in .nse_depinfo */
169         wchar_t                 *newline;/* New line in .nse_depinfo */
170         wchar_t                 *cond_macrostring;
171                                         /* string built from value of
172                                          * conditional macros used by
173                                          * this target
174                                          */
175         Boolean                 removed;/* This target is no longer recursive*/
176 };

178 struct _Dyntarget {
179         struct _Dyntarget       *next;
180         struct _Name            *name;
181 };


184 /*
185  * Typedefs for all structs
186  */
187 typedef struct _Cmd_line        *Cmd_line, Cmd_line_rec;
188 typedef struct _Dependency      *Dependency, Dependency_rec;
189 typedef struct _Macro           *Macro, Macro_rec;
190 typedef struct _Name_vector     *Name_vector, Name_vector_rec;
191 typedef struct _Percent         *Percent, Percent_rec;
192 typedef struct _Dyntarget       *Dyntarget;
193 typedef struct _Recursive_make  *Recursive_make, Recursive_make_rec;
```



```
194 typedef struct _Running         *Running, Running_rec;


197 /*
198  *      extern declarations for all global variables.
199  *      The actual declarations are in globals.cc
200  */
201 extern  Boolean         allrules_read;
202 extern  Name            posix_name;
203 extern  Name            svr4_name;
204 extern  Boolean         sdot_target;
205 extern  Boolean         all_parallel;
206 extern  Boolean         assign_done;
207 extern  Boolean         build_failed_seen;
208 #ifdef DISTRIBUTED
209 extern  Boolean         building_serial;
210 #endif
211 extern  Name            built_last_make_run;
212 extern  Name            c_at;
213 #ifdef DISTRIBUTED
214 extern  Boolean         called_make;
215 #endif
216 extern  Boolean         command_changed;
217 extern  Boolean         commands_done;
218 extern  Chain           conditional_targets;
219 extern  Name            conditionals;
220 extern  Boolean         continue_after_error;
221 extern  Property        current_line;
222 extern  Name            current_make_version;
223 extern  Name            current_target;
224 extern  short           debug_level;
225 extern  Cmd_line        default_rule;
226 extern  Name            default_rule_name;
227 extern  Name            default_target_to_build;
228 extern  Boolean         depinfo_already_read;
229 extern  Name            dmake_group;
230 extern  Name            dmake_max_jobs;
231 extern  Name            dmake_mode;
232 extern  DMake_mode      dmake_mode_type;
233 extern  Name            dmake_output_mode;
234 extern  DMake_output_mode       output_mode;
235 extern  Name            dmake_odir;
236 extern  Name            dmake_rcfile;
237 extern  Name            done;
238 extern  Name            dot;
239 extern  Name            dot_keep_state;
240 extern  Name            dot_keep_state_file;
241 extern  Name            empty_name;
242 extern  Boolean         fatal_in_progress;
243 extern  int             file_number;
244 extern  Name            force;
245 extern  Name            ignore_name;
246 extern  Boolean         ignore_errors;
247 extern  Boolean         ignore_errors_all;
248 extern  Name            init;
249 extern  int             job_msg_id;
250 extern  Boolean         keep_state;
251 extern  Name            make_state;
252 #ifdef TEAMWARE_MAKE_CMN
253 extern  timestruc_t     make_state_before;
254 #endif
255 extern  Boolean         make_state_locked;
256 extern  Dependency      makefiles_used;
257 extern  Name            makeflags;
258 extern  Name            make_version;
259 extern  char            mbs_buffer2[];
```

```
260 extern  char            *mbs_ptr;
261 extern  char            *mbs_ptr2;
262 extern  Boolean         no_action_was_taken;
263 extern  int             mtool_msgs_fd;
264 extern  Boolean         no_parallel;
265 #ifdef SGE_SUPPORT
266 extern  Boolean         grid;
267 #endif
268 extern  Name            no_parallel_name;
269 extern  Name            not_auto;
270 extern  Boolean         only_parallel;
271 extern  Boolean         parallel;
272 extern  Name            parallel_name;
273 extern  Name            localhost_name;
274 extern  int             parallel_process_cnt;
275 extern  Percent         percent_list;
276 extern  Dyntarget       dyntarget_list;
277 extern  Name            plus;
278 extern  Name            pmake_machinesfile;
279 extern  Name            precious;
280 extern  Name            primary_makefile;
281 extern  Boolean         quest;
282 extern  short           read_trace_level;
283 extern  Boolean         reading_dependencies;
284 extern  int             recursion_level;
285 extern  Name            recursive_name;
286 extern  short           report_dependencies_level;
287 extern  Boolean         report_pwd;
288 extern  Boolean         rewrite_statefile;
289 extern  Running         running_list;
290 extern  char            *sccs_dir_path;
291 extern  Name            sccs_get_name;
292 extern  Name            sccs_get_posix_name;
293 extern  Cmd_line        sccs_get_rule;
294 extern  Cmd_line        sccs_get_org_rule;
295 extern  Cmd_line        sccs_get_posix_rule;
296 extern  Name            get_name;
297 extern  Name            get_posix_name;
298 extern  Cmd_line        get_rule;
299 extern  Cmd_line        get_posix_rule;
300 extern  Boolean         send_mtool_msgs;
301 extern  Boolean         all_precious;
302 extern  Boolean         report_cwd;
303 extern  Boolean         silent_all;
304 extern  Boolean         silent;
305 extern  Name            silent_name;
306 extern  char            *stderr_file;
307 extern  char            *stdout_file;
308 #ifdef SGE_SUPPORT
309 extern  char            script_file[];
310 #endif
311 extern  Boolean         stdout_stderr_same;
312 extern  Dependency      suffixes;
313 extern  Name            suffixes_name;
314 extern  Name            sunpro_dependencies;
315 extern  Boolean         target_variants;
316 extern  char            *tmpdir;
317 extern  char            *temp_file_directory;
318 extern  Name            temp_file_name;
319 extern  short           temp_file_number;
320 extern  wchar_t         *top_level_target;
321 extern  Boolean         touch;
322 extern  Boolean         trace_reader;
323 extern  Boolean         build_unconditional;
324 extern  pathpt          vroot_path;
325 extern  Name            wait_name;
```



```
326 extern  wchar_t         wcs_buffer2[];
327 extern  wchar_t         *wcs_ptr;
328 extern  wchar_t         *wcs_ptr2;
329 extern  nl_catd         catd;
330 extern  long int        hostid;

332 /*
333  * Declarations of system defined variables
334  */
335 #if !defined(linux)
336 /* On linux this variable is defined in 'signal.h' */
337 extern  char            *sys_siglist[];
338 #endif

340 /*
341  * Declarations of system supplied functions
342  */
343 extern  int             file_lock(char *, char *, int *, int);

345 /*
346  * Declarations of functions declared and used by make
347  */
348 extern  void            add_pending(Name target, int recursion_level, Boolean do
349 extern  void            add_running(Name target, Name true_target, Property comm
350 extern  void            add_serial(Name target, int recursion_level, Boolean do_
351 extern  void            add_subtree(Name target, int recursion_level, Boolean do
352 extern  void            append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar
353 #ifdef DISTRIBUTED
354 extern  Doname          await_dist(Boolean waitflg);
355 #endif
356 #ifdef TEAMWARE_MAKE_CMN
357 extern  void            await_parallel(Boolean waitflg);
358 #endif
359 extern  void            build_suffix_list(Name target_suffix);
360 extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
361 extern  void            check_state(Name temp_file_name);
362 extern  void            cond_macros_into_string(Name np, String_rec *buffer);
363 extern  void            construct_target_string();
364 extern  void            create_xdrs_ptr(void);
365 extern  void            depvar_add_to_list (Name name, Boolean cmdline);
366 #ifdef DISTRIBUTED
367 extern  void            distribute_rxm(Avo_DoJobMsg *dmake_job_msg);
368 extern  int             getRxmMessage(void);
369 extern  Avo_JobResultMsg* getJobResultMsg(void);
370 extern  Avo_AcknowledgeMsg* getAcknowledgeMsg(void);
371 #endif
372 extern  Doname          doname(register Name target, register Boolean do_get, re
373 extern  Doname          doname_check(register Name target, register Boolean do_g
374 extern  Doname          doname_parallel(Name target, Boolean do_get, Boolean imp
375 extern  Doname          dosys(register Name command, register Boolean ignore_err
376 extern  void            dump_make_state(void);
377 extern  void            dump_target_list(void);
378 extern  void            enter_conditional(register Name target, Name name, Name
379 extern  void            enter_dependencies(register Name target, Chain target_gr
380 extern  void            enter_dependency(Property line, register Name depe, Bool
381 extern  void            enter_equal(Name name, Name value, register Boolean appe
382 extern  Percent         enter_percent(register Name target, Chain target_group,
383 extern  Dyntarget       enter_dyntarget(register Name target);
384 extern  Name_vector     enter_name(String string, Boolean tail_present, register
385 extern  Boolean         exec_vp(register char *name, register char **argv, char
386 extern  Doname          execute_parallel(Property line, Boolean waitflg, Boolean
387 extern  Doname          execute_serial(Property line);
388 extern  timestruc_t&    exists(register Name target);
389 extern  void            fatal(char *, ...);
390 extern  void            fatal_reader(char *, ...);
391 extern  Doname          find_ar_suffix_rule(register Name target, Name true_targ
```

```
392 extern  Doname          find_double_suffix_rule(register Name target, Property *
393 extern  Doname          find_percent_rule(register Name target, Property *comman
394 extern  int            find_run_directory (char *cmd, char *cwd, char *dir, cha
395 extern  Doname          find_suffix_rule(Name target, Name target_body, Name tar
396 extern  Chain          find_target_groups(register Name_vector target_list, reg
397 extern  void           finish_children(Boolean docheck);
398 extern  void           finish_running(void);
399 extern  void           free_chain(Name_vector ptr);
400 extern  void           gather_recursive_deps(void);
401 extern  char           *get_current_path(void);
402 extern  int            get_job_msg_id(void);
403 extern  FILE           *get_mtool_msgs_fp(void);
404 #ifdef DISTRIBUTED
405 extern  Boolean        get_dmake_group_specified(void);
406 extern  Boolean        get_dmake_max_jobs_specified(void);
407 extern  Boolean        get_dmake_mode_specified(void);
408 extern  Boolean        get_dmake_odir_specified(void);
409 extern  Boolean        get_dmake_rcfile_specified(void);
410 extern  Boolean        get_pmake_machinesfile_specified(void);
411 #endif
412 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
413 extern  XDR            *get_xdrs_ptr(void);
414 #endif
415 extern  wchar_t        *getmem_wc(register int size);
416 #if !defined(linux)
417 /* On linux getwd(char *) is defined in 'unistd.h' */
418 #ifdef __cplusplus
419 extern "C" {
420 #endif
421 extern  char           *getwd(char *);
422 #ifdef __cplusplus
423 }
424 #endif
425 #endif
426 extern  void           handle_interrupt(int);
427 extern  Boolean        is_running(Name target);
428 extern  void           load_cached_names(void);
429 extern  Boolean        parallel_ok(Name target, Boolean line_prop_must_exists);
430 extern  void           print_dependencies(register Name target, register Proper
431 extern  void           send_job_start_msg(Property line);
432 extern  void           send_rsrc_info_msg(int max_jobs, char *hostname, char *u
433 extern  void           print_value(register Name value, Daemon daemon);
434 extern  timestruc_t&   read_archive(register Name target);
435 extern  int            read_dir(Name dir, wchar_t *pattern, Property line, wcha
436 extern  void           read_directory_of_file(register Name file);
437 extern  int            read_make_machines(Name make_machines_name);
438 extern  Boolean        read_simple_file(register Name makefile_name, register B
439 extern  void           remove_recursive_dep(Name target);
440 extern  void           report_recursive_dep(Name target, char *line);
441 extern  void           report_recursive_done(void);
442 extern  void           report_recursive_init(void);
443 extern  Recursive_make find_recursive_target(Name target);
444 extern  void           reset_locals(register Name target, register Property old
445 extern  void           set_locals(register Name target, register Property old_l
446 extern  void           setvar_append(register Name name, register Name value);
447 #ifdef DISTRIBUTED
448 extern  void           setvar_envvar(Avo_DoJobMsg *dmake_job_msg);
449 #else
450 extern  void           setvar_envvar(void);
451 #endif
452 extern  void           special_reader(Name target, register Name_vector depes,
453 extern  void           startup_rxm();
454 extern  Doname         target_can_be_built(register Name target);
455 extern  char           *time_to_string(const timestruc_t &time);
456 extern  void           update_target(Property line, Doname result);
457 extern  void           warning(char *, ...);
```



```
458 extern  void           write_state_file(int report_recursive, Boolean exiting);
459 extern  Name           vpath_translation(register Name cmd);

461 #define DEPINFO_FMT_VERSION "VERS2$"
462 #define VER_LEN strlen(DEPINFO_FMT_VERSION)

464 #ifdef NSE

466 /*
467  *  NSE version for depinfo format
468  */
469 extern Boolean         nse;
470 extern Name            nse_backquote_seen;
471 extern Boolean         nse_did_recursion;
472 extern Name            nse_shell_var_used;
473 extern Boolean         nse_watch_vars;
474 extern wchar_t         current_makefile[MAXPATHLEN];
475 extern Boolean         nse_depinfo_locked;
476 extern char            nse_depinfo_lockfile[MAXPATHLEN];
477 extern Name            derived_src;

479 extern void            depvar_dep_macro_used(Name);
480 extern void            depvar_rule_macro_used(Name);
481 extern Boolean         nse_backquotes(wchar_t *);
482 extern void            nse_check_cd(Property);
483 extern void            nse_check_derived_src(Name, wchar_t *, Cmd_line);
484 extern void            nse_check_file_backquotes(wchar_t *);
485 extern void            nse_check_no_deps_no_rule(Name, Property, Property);
486 extern void            nse_check_sccs(wchar_t *, wchar_t *);
487 extern void            nse_dep_cmdmacro(wchar_t *);
488 extern int             nse_exit_status(void);
489 extern void            nse_init_source_suffixes(void);
490 extern void            nse_no_makefile(Name);
491 extern void            nse_rule_cmdmacro(wchar_t *);
492 extern void            nse_wildcard(wchar_t *, wchar_t *);
493 #endif

495 #endif
496 #endif /* ! codereview */
```

```
**********************************************************
    1214 Wed May 20 11:04:14 2015
new/usr/src/cmd/make/include/mksdmsi18n/mksdmsi18n.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 1996 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)mksdmsi18n.h 1.3 06/12/12
 */

#pragma ident   "@(#)mksdmsi18n.h       1.3     06/12/12"

#ifndef _AVO_MKSDMSI18N_H
#define _AVO_MKSDMSI18N_H

#ifndef _AVO_INTL_H
#include <avo/intl.h>
#endif

extern  nl_catd libmksdmsi18n_catd;

int     libmksdmsi18n_init();
void    libmksdmsi18n_fini();

#endif

#endif /* ! codereview */
```

```
**********************************************************
   24635 Wed May 20 11:04:14 2015
new/usr/src/cmd/make/include/mksh/defs.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#ifndef _MKSH_DEFS_H
#define _MKSH_DEFS_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)defs.h 1.35 06/12/12
 */

#pragma ident   "@(#)defs.h     1.35    06/12/12"

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

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

#if defined (HP_UX) || defined (linux)
#define  MAXNAMELEN          256
#define  RW_NO_OVERLOAD_WCHAR 1  /* Rogue Wave, belongs in <rw/compiler.h> */
#else
#include <wctype.h>
#include <widec.h>
#endif

#if defined (linux)
/*
 * Definition of wchar functions.
```



```
 */
#       include <wctype.h>
#       include <wchar.h>
#       define wsdup(x) wcsdup(x)
#       define wschr(x,y) wcschr(x,y)
#       define wscat(x,y) wcscat(x,y)
#       define wsrchr(x,y) wcsrchr(x,y)
#       define wslen(x) wcslen(x)
#       define wscpy(x,y) wcscpy(x,y)
#       define wsncpy(x,y,z) wcsncpy(x,y,z)
#       define wscmp(x,y) wcscmp(x,y)
#       define wsncmp(x,y,z) wcsncmp(x,y,z)
#endif

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

#define false   BOOLEAN_false
#define true    BOOLEAN_true

typedef enum {
        false =         0,
        true =          1,
        failed =        0,
        succeeded =     1
} Boolean;
#define BOOLEAN(expr)           ((expr) ? true : false)

/*
 * Some random constants (in an enum so dbx knows their values)
 */
enum {
        update_delay = 30,              /* time between rstat checks */
#ifdef sun386
        ar_member_name_len = 14,
#else
#if defined(SUN5_0) || defined(linux)
        ar_member_name_len = 1024,
#else
        ar_member_name_len = 15,
#endif
#endif

        hashsize = 2048                 /* size of hash table */
};


/*
 * Symbols that defines all the different char constants make uses
 */
enum {
        ampersand_char =        '&',
        asterisk_char =         '*',
        at_char =               '@',
        backquote_char =        '`',
        backslash_char =        '\\',
        bar_char =              '|',
        braceleft_char =        '{',
        braceright_char =       '}',
        bracketleft_char =      '[',
        bracketright_char =     ']',
        colon_char =            ':',
        comma_char =            ',',
        dollar_char =           '$',
        doublequote_char =      '"',
```

```
128         equal_char =            '=',
129         exclam_char =           '!',
130         greater_char =          '>',
131         hat_char =              '^',
132         hyphen_char =           '-',
133         less_char =             '<',
134         newline_char =          '\n',
135         nul_char =              '\0',
136         numbersign_char =       '#',
137         parenleft_char =        '(',
138         parenright_char =       ')',
139         percent_char =          '%',
140         period_char =           '.',
141         plus_char =             '+',
142         question_char =         '?',
143         quote_char =            '\'',
144         semicolon_char =        ';',
145         slash_char =            '/',
146         space_char =            ' ',
147         tab_char =              '\t',
148         tilde_char =            '~'
149 };

151 /*
152  * For make i18n. Codeset independent.
153  * Setup character semantics by identifying all the special characters
154  * of make, and assigning each an entry in the char_semantics[] vector.
155  */
156 enum {
157         ampersand_char_entry = 0,       /*  0 */
158         asterisk_char_entry,            /*  1 */
159         at_char_entry,                  /*  2 */
160         backquote_char_entry,           /*  3 */
161         backslash_char_entry,           /*  4 */
162         bar_char_entry,                 /*  5 */
163         bracketleft_char_entry,         /*  6 */
164         bracketright_char_entry,        /*  7 */
165         colon_char_entry,               /*  8 */
166         dollar_char_entry,              /*  9 */
167         doublequote_char_entry,         /* 10 */
168         equal_char_entry,               /* 11 */
169         exclam_char_entry,              /* 12 */
170         greater_char_entry,             /* 13 */
171         hat_char_entry,                 /* 14 */
172         hyphen_char_entry,              /* 15 */
173         less_char_entry,                /* 16 */
174         newline_char_entry,             /* 17 */
175         numbersign_char_entry,          /* 18 */
176         parenleft_char_entry,           /* 19 */
177         parenright_char_entry,          /* 20 */
178         percent_char_entry,             /* 21 */
179         plus_char_entry,                /* 22 */
180         question_char_entry,            /* 23 */
181         quote_char_entry,               /* 24 */
182         semicolon_char_entry,           /* 25 */
183 #ifdef SGE_SUPPORT
184         space_char_entry,               /* 26 */
185         tab_char_entry,                 /* 27 */
186         no_semantics_entry              /* 28 */
187 #else
188         no_semantics_entry              /* 26 */
189 #endif /* SGE_SUPPORT */
190 };

192 /*
193  * CHAR_SEMANTICS_ENTRIES should be the number of entries above.
```



```
194  * The last entry in char_semantics[] should be blank.
195  */
196 #ifdef SGE_SUPPORT
197 #define CHAR_SEMANTICS_ENTRIES  29
198 /*
199 #define CHAR_SEMANTICS_STRING   "&*@'\\|[]:$=!>-\n#()%+?;^<'\" \t"
200  */
201 #else
202 #define CHAR_SEMANTICS_ENTRIES  27
203 /*
204 #define CHAR_SEMANTICS_STRING   "&*@'\\|[]:$=!>-\n#()%+?;^<'\""
205  */
206 #endif /* SGE_SUPPORT */

208 /*
209  * Some utility macros
210  */
211 #define ALLOC(x)                ((struct _##x *)getmem(sizeof (struct _##x)))
212 #define ALLOC_WC(x)             ((wchar_t *)getmem((x) * SIZEOFWCHAR_T))
213 #define FIND_LENGTH             -1
214 #define GETNAME(a,b)            getname_fn((a), (b), false)
215 #define IS_EQUAL(a,b)           (!strcmp((a), (b)))
216 #define IS_EQUALN(a,b,n)        (!strncmp((a), (b), (n)))
217 #define IS_WEQUAL(a,b)          (!wscmp((a), (b)))
218 #define IS_WEQUALN(a,b,n)       (!wsncmp((a), (b), (n)))
219 #define MBLEN(a)                mblen((a), MB_LEN_MAX)
220 #define MBSTOWCS(a,b)           (void) mbstowcs_with_check((a), (b), MAXPATHLEN)
221 #define MBTOWC(a,b)             mbtowc((a), (b), MB_LEN_MAX)
222 #define SIZEOFWCHAR_T           (sizeof (wchar_t))
223 #define VSIZEOF(v)              (sizeof (v) / sizeof ((v)[0]))
224 #define WCSTOMBS(a,b)           (void) wcstombs((a), (b), (MAXPATHLEN * MB_LEN_M
225 #define WCTOMB(a,b)             (void) wctomb((a), (b))
226 #define HASH(v, c)              (v = (v)*31 + (unsigned int)(c))

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

230 /*
231  * Bits stored in funny vector to classify chars
232  */
233 enum {
234         dollar_sem =            0001,
235         meta_sem =              0002,
236         percent_sem =           0004,
237         wildcard_sem =          0010,
238         command_prefix_sem =    0020,
239         special_macro_sem =     0040,
240         colon_sem =             0100,
241         parenleft_sem =         0200
242 };

244 /*
245  * Type returned from doname class functions
246  */
247 typedef enum {
248         build_dont_know = 0,
249         build_failed,
250         build_ok,
251         build_in_progress,
252         build_running,          /* PARALLEL & DISTRIBUTED */
253         build_pending,          /* PARALLEL & DISTRIBUTED */
254         build_serial,           /* PARALLEL & DISTRIBUTED */
255         build_subtree           /* PARALLEL & DISTRIBUTED */
256 } Doname;

258 /*
259  * The String struct defines a string with the following layout
```

```
 *      "xxxxxxxxxxxxxxxCxxxxxxxxxxxxxxx________"
 *      ^               ^               ^               ^
 *      |               |               |               |
 *      buffer.start    text.p          text.end        buffer.end
 *      text.p points to the next char to read/write.
 */
struct _String {
        struct Text {
                wchar_t         *p;     /* Read/Write pointer */
                wchar_t         *end;   /* Read limit pointer */
        }               text;
        struct Physical_buffer {
                wchar_t         *start; /* Points to start of buffer */
                wchar_t         *end;   /* End of physical buffer */
        }               buffer;
        Boolean         free_after_use:1;
};

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

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

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

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

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

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

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

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

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



```
/*
 * Used for storing the $? list and also for the "target + target:"
 * construct.
 */
struct _Chain {
        struct _Chain           *next;
        struct _Name            *name;
        struct _Percent *percent_member;
};

/*
 * Stores one command line for a rule
 */
struct _Cmd_line {
        struct _Cmd_line        *next;
        struct _Name            *command_line;
        Boolean                 make_refd:1;    /* $(MAKE) referenced? */
        /*
         * Remember any command line prefixes given
         */
        Boolean                 ignore_command_dependency:1;    /* '?' */
        Boolean                 assign:1;                       /* '=' */
        Boolean                 ignore_error:1;                 /* '-' */
        Boolean                 silent:1;                       /* '@' */
        Boolean                 always_exec:1;                  /* '+' */
};

/*
 * Linked list of targets/files
 */
struct _Dependency {
        struct _Dependency      *next;
        struct _Name            *name;
        Boolean                 automatic:1;
        Boolean                 stale:1;
        Boolean                 built:1;
};

/*
 * The specials are markers for targets that the reader should special case
 */
typedef enum {
        no_special,
        built_last_make_run_special,
        default_special,
#ifdef NSE
        derived_src_special,
#endif
        get_posix_special,
        get_special,
        ignore_special,
        keep_state_file_special,
        keep_state_special,
        make_version_special,
        no_parallel_special,
        parallel_special,
        posix_special,
        precious_special,
        sccs_get_posix_special,
        sccs_get_special,
        silent_special,
        suffixes_special,
        svr4_special,
        localhost_special
} Special;
```

```
393 typedef enum {
394         no_colon,
395         one_colon,
396         two_colon,
397         equal_seen,
398         conditional_seen,
399         none_seen
400 } Separator;

402 /*
403  * Magic values for the timestamp stored with each name object
404  */

406 #if defined (linux)
407 typedef struct timespec timestruc_t;
408 #endif

410 extern const timestruc_t file_no_time;
411 extern const timestruc_t file_doesnt_exist;
412 extern const timestruc_t file_is_dir;
413 extern const timestruc_t file_min_time;
414 extern const timestruc_t file_max_time;

416 /*
417  * Each Name has a list of properties
418  * The properties are used to store information that only
419  * a subset of the Names need
420  */
421 typedef enum {
422         no_prop,
423         conditional_prop,
424         line_prop,
425         macro_prop,
426         makefile_prop,
427         member_prop,
428         recursive_prop,
429         sccs_prop,
430         suffix_prop,
431         target_prop,
432         time_prop,
433         vpath_alias_prop,
434         long_member_name_prop,
435         macro_append_prop,
436         env_mem_prop
437 } Property_id;

439 typedef enum {
440         no_daemon = 0,
441         chain_daemon
442 } Daemon;

444 struct _Env_mem {
445         char            *value;
446 };

448 struct _Macro_appendix {
449         struct _Name            *value;
450         struct _Name            *value_to_append;
451 };

453 struct _Macro {
454         /*
455         * For "ABC = xyz" constructs
456         * Name "ABC" get one macro prop
457         */
```



```
458         struct _Name            *value;
459 #ifdef NSE
460         Boolean                 imported:1;
461 #endif
462         Boolean                 exported:1;
463         Boolean                 read_only:1;
464         /*
465         * This macro is defined conditionally
466         */
467         Boolean                 is_conditional:1;
468         /*
469         * The list for $? is stored as a structured list that
470         * is translated into a string iff it is referenced.
471         * This is why  some macro values need a daemon.
472         */
473 #if defined(HP_UX) || defined(linux)
474         Daemon                  daemon;
475 #else
476         Daemon                  daemon:2;
477 #endif
478 };

480 struct _Macro_list {
481         struct _Macro_list      *next;
482         char                    *macro_name;
483         char                    *value;
484 };

486 enum sccs_stat {
487         DONT_KNOW_SCCS = 0,
488         NO_SCCS,
489         HAS_SCCS
490 };

492 struct _Name {
493         struct _Property        *prop;          /* List of properties */
494         char                    *string_mb;     /* Multi-byte name string */
495         struct {
496                 unsigned int            length;
497         }                       hash;
498         struct {
499                 timestruc_t             time;           /* Modification */
500                 int                     stat_errno;     /* error from "stat" */
501                 off_t                   size;           /* Of file */
502                 mode_t                  mode;           /* Of file */
503 #if defined(HP_UX) || defined(linux)
504                 Boolean                 is_file;
505                 Boolean                 is_dir;
506                 Boolean                 is_sym_link;
507                 Boolean                 is_precious;
508                 enum sccs_stat          has_sccs;
509 #else
510                 Boolean                 is_file:1;
511                 Boolean                 is_dir:1;
512                 Boolean                 is_sym_link:1;
513                 Boolean                 is_precious:1;
514 #ifdef NSE
515                 Boolean                 is_derived_src:1;
516 #endif
517                 enum sccs_stat          has_sccs:2;
518 #endif
519         }                       stat;
520         /*
521          * Count instances of :: definitions for this target
522          */
523         short                   colon_splits;
```

```
524         /*
525          * We only clear the automatic depes once per target per report
526          */
527         short                   temp_file_number;
528         /*
529          * Count how many conditional macros this target has defined
530          */
531         short                   conditional_cnt;
532         /*
533          * A conditional macro was used when building this target
534          */
535         Boolean                 depends_on_conditional:1;
536         /*
537          * Pointer to list of conditional macros which were used to build
538          * this target
539          */
540         struct _Macro_list      *conditional_macro_list;
541         Boolean                 has_member_depe:1;
542         Boolean                 is_member:1;
543         /*
544          * This target is a directory that has been read
545          */
546         Boolean                 has_read_dir:1;
547         /*
548          * This name is a macro that is now being expanded
549          */
550         Boolean                 being_expanded:1;
551         /*
552          * This name is a magic name that the reader must know about
553          */
554 #if defined(HP_UX) || defined(linux)
555         Special                 special_reader;
556         Doname                  state;
557         Separator               colons;
558 #else
559         Special                 special_reader:5;
560         Doname                  state:3;
561         Separator               colons:3;
562 #endif
563         Boolean                 has_depe_list_expanded:1;
564         Boolean                 suffix_scan_done:1;
565         Boolean                 has_complained:1;       /* For sccs */
566         /*
567          * This target has been built during this make run
568          */
569         Boolean                 ran_command:1;
570         Boolean                 with_squiggle:1;        /* for .SUFFIXES */
571         Boolean                 without_squiggle:1;     /* for .SUFFIXES */
572         Boolean                 has_read_suffixes:1;    /* Suffix list cached*/
573         Boolean                 has_suffixes:1;
574         Boolean                 has_target_prop:1;
575         Boolean                 has_vpath_alias_prop:1;
576         Boolean                 dependency_printed:1;   /* For dump_make_state()
577         Boolean                 dollar:1;               /* In namestring */
578         Boolean                 meta:1;                 /* In namestring */
579         Boolean                 percent:1;              /* In namestring */
580         Boolean                 wildcard:1;             /* In namestring */
581         Boolean                 has_parent:1;
582         Boolean                 is_target:1;
583         Boolean                 has_built:1;
584         Boolean                 colon:1;                /* In namestring */
585         Boolean                 parenleft:1;            /* In namestring */
586         Boolean                 has_recursive_dependency:1;
587         Boolean                 has_regular_dependency:1;
588         Boolean                 is_double_colon:1;
589         Boolean                 is_double_colon_parent:1;
```



```
590         Boolean                 has_long_member_name:1;
591         /*
592          * allowed to run in parallel
593          */
594         Boolean                 parallel:1;
595         /*
596          * not allowed to run in parallel
597          */
598         Boolean                 no_parallel:1;
599         /*
600          * used in dependency_conflict
601          */
602         Boolean                 checking_subtree:1;
603         Boolean                 added_pattern_conditionals:1;
604         /*
605          * rechecking target for possible rebuild
606          */
607         Boolean                 rechecking_target:1;
608         /*
609          * build this target in silent mode
610          */
611         Boolean                 silent_mode:1;
612         /*
613          * build this target in ignore error mode
614          */
615         Boolean                 ignore_error_mode:1;
616         Boolean                 dont_activate_cond_values:1;
617         /*
618          * allowed to run serially on local host
619          */
620         Boolean                 localhost:1;
621 };

623 /*
624  * Stores the % matched default rules
625  */
626 struct _Percent {
627         struct _Percent *next;
628         struct _Name            **patterns;
629         struct _Name            *name;
630         struct _Percent         *dependencies;
631         struct _Cmd_line        *command_template;
632         struct _Chain           *target_group;
633         int                     patterns_total;
634         Boolean                 being_expanded;
635 };

637 struct Conditional {
638         /*
639          * For "foo := ABC [+]= xyz" constructs
640          * Name "foo" gets one conditional prop
641          */
642         struct _Name            *target;
643         struct _Name            *name;
644         struct _Name            *value;
645         int                     sequence;
646         Boolean                 append:1;
647 };

649 struct Line {
650         /*
651          * For "target : dependencies" constructs
652          * Name "target" gets one line prop
653          */
654         struct _Cmd_line        *command_template;
655         struct _Cmd_line        *command_used;
```

```
656         struct _Dependency      *dependencies;
657         timestruc_t             dependency_time;
658         struct _Chain           *target_group;
659         Boolean                 is_out_of_date:1;
660         Boolean                 sccs_command:1;
661         Boolean                 command_template_redefined:1;
662         Boolean                 dont_rebuild_command_used:1;
663         /*
664          * Values for the dynamic macros
665          */
666         struct _Name            *target;
667         struct _Name            *star;
668         struct _Name            *less;
669         struct _Name            *percent;
670         struct _Chain           *query;
671 };

673 struct Makefile {
674         /*
675          * Names that reference makefiles gets one prop
676          */
677         wchar_t                 *contents;
678         off_t                   size;
679 };

681 struct Member {
682         /*
683          * For "lib(member)" and "lib((entry))" constructs
684          * Name "lib(member)" gets one member prop
685          * Name "lib((entry))" gets one member prop
686          * The member field is filled in when the prop is refd
687          */
688         struct _Name            *library;
689         struct _Name            *entry;
690         struct _Name            *member;
691 };

693 struct Recursive {
694         /*
695          * For "target: .RECURSIVE dir makefiles" constructs
696          * Used to keep track of recursive calls to make
697          * Name "target" gets one recursive prop
698          */
699         struct _Name            *directory;
700         struct _Name            *target;
701         struct _Dependency      *makefiles;
702         Boolean                 has_built;
703         Boolean                 in_depinfo;
704 };

706 struct Sccs {
707         /*
708          * Each file that has a SCCS s. file gets one prop
709          */
710         struct _Name            *file;
711 };

713 struct Suffix {
714         /*
715          * Cached list of suffixes that can build this target
716          * suffix is built from .SUFFIXES
717          */
718         struct _Name            *suffix;
719         struct _Cmd_line        *command_template;
720 };
```



```
722 struct Target {
723         /*
724          * For "target:: dependencies" constructs
725          * The "::" construct is handled by converting it to
726          * "foo: 1@foo" + "1@foo: dependecies"
727          * "1@foo" gets one target prop
728          * This target prop cause $@ to be bound to "foo"
729          * not "1@foo" when the rule is evaluated
730          */
731         struct _Name            *target;
732 };

734 struct STime {
735         /*
736          * Save the original time for :: targets
737          */
738         timestruc_t                     time;
739 };

741 struct Vpath_alias {
742         /*
743          * If a file was found using the VPATH it gets
744          * a vpath_alias prop
745          */
746         struct _Name            *alias;
747 };

749 struct Long_member_name {
750         /*
751          * Targets with a truncated member name carries
752          * the full lib(member) name for the state file
753          */
754         struct _Name            *member_name;
755 };

757 union Body {
758         struct _Macro           macro;
759         struct Conditional      conditional;
760         struct Line             line;
761         struct Makefile         makefile;
762         struct Member           member;
763         struct Recursive        recursive;
764         struct Sccs             sccs;
765         struct Suffix           suffix;
766         struct Target           target;
767         struct STime            time;
768         struct Vpath_alias      vpath_alias;
769         struct Long_member_name long_member_name;
770         struct _Macro_appendix  macro_appendix;
771         struct _Env_mem         env_mem;
772 };

774 #define PROPERTY_HEAD_SIZE (sizeof (struct _Property)-sizeof (union Body))
775 struct _Property {
776         struct _Property        *next;
777 #if defined(HP_UX) || defined(linux)
778         Property_id             type;
779 #else
780         Property_id             type:4;
781 #endif
782         union Body              body;
783 };

785 /* Structure for dynamic "ascii" arrays */
786 struct ASCII_Dyn_Array {
787         char                    *start;
```

```
788         size_t                  size;
789 };

791 struct _Envvar {
792         struct _Name            *name;
793         struct _Name            *value;
794         struct _Envvar          *next;
795         char                    *env_string;
796         Boolean                 already_put:1;
797 };

799 /*
800  * Macros for the reader
801  */
802 #define GOTO_STATE(new_state) { \
803                                   SET_STATE(new_state); \
804                                     goto enter_state; \
805                                 }
806 #define SET_STATE(new_state) state = (new_state)

808 #define UNCACHE_SOURCE()        if (source != NULL) { \
809                                         source->string.text.p = source_p; \
810                                   }
811 #define CACHE_SOURCE(comp)      if (source != NULL) { \
812                                         source_p = source->string.text.p - \
813                                           (comp); \
814                                         source_end = source->string.text.end; \
815                                   }
816 #define GET_NEXT_BLOCK_NOCHK(source)    { UNCACHE_SOURCE(); \
817                                  source = get_next_block_fn(source); \
818                                  CACHE_SOURCE(0) \
819                               }
820 #define GET_NEXT_BLOCK(source)  { GET_NEXT_BLOCK_NOCHK(source); \
821                                  if (source != NULL && source->error_converting) \
822                                         GOTO_STATE(illegal_bytes_state); \
823                                  } \
824                               }
825 #define GET_CHAR()              ((source == NULL) || \
826                                 (source_p >= source_end) ? 0 : *source_p)

828 struct _Source {
829         struct _String          string;
830         struct _Source          *previous;
831         off_t                   bytes_left_in_file;
832         short                   fd;
833         Boolean                 already_expanded:1;
834         Boolean                 error_converting:1;
835         char                    *inp_buf;
836         char                    *inp_buf_end;
837         char                    *inp_buf_ptr;
838 };

840 typedef enum {
841         reading_nothing,
842         reading_makefile,
843         reading_statefile,
844         rereading_statefile,
845         reading_cpp_file
846 } Makefile_type;

848 /*
849  * Typedefs for all structs
850  */
851 typedef struct _Chain           *Chain, Chain_rec;
852 typedef struct _Envvar          *Envvar, Envvar_rec;
853 typedef struct _Macro_list      *Macro_list, Macro_list_rec;
```



```
854 typedef struct _Name            *Name, Name_rec;
855 typedef struct _Property        *Property, Property_rec;
856 typedef struct _Source          *Source, Source_rec;
857 typedef struct _String          *String, String_rec;

859 /*
860  * name records hash table.
861  */
862 struct Name_set {
863 private:
864         // single node in a tree
865         struct entry {
866                 entry(Name name_, entry *parent_) :
867                         name(name_),
868                         parent(parent_),
869                         left(0),
870                         right(0),
871                         depth(1)
872                 {}

874                 Name            name;

876                 entry           *parent;
877                 entry           *left;
878                 entry           *right;
879                 unsigned        depth;

881                 void setup_depth() {
882                         unsigned rdepth = (right != 0) ? right->depth : 0;
883                         unsigned ldepth = (left != 0) ? left->depth : 0;
884                         depth = 1 + ((ldepth > rdepth) ? ldepth : rdepth);
885                 }
886         };

888 public:
889         // make iterator a friend of Name_set to have access to struct entry
890         struct iterator;
891         friend struct Name_set::iterator;

893         // iterator over tree nodes
894         struct iterator {
895         public:
896                 // constructors
897                 iterator() : node(0) {}
898                 iterator(entry *node_) : node(node_) {}

900                 // dereference operator
901                 Name operator->() const { return node->name; }

903                 // conversion operator
904                 operator Name() { return node->name; }

906                 // assignment operator
907                 iterator& operator=(const iterator &o) { node = o.node; return *

909                 // equality/inequality operators
910                 int operator==(const iterator &o) const { return (node == o.node
911                 int operator!=(const iterator &o) const { return (node != o.node

913                 // pre/post increment operators
914                 iterator& operator++();
915                 iterator  operator++(int) { iterator it = *this; ++*this; return

917         private:
918                 // the node iterator points to
919                 entry *node;
```

```
920         };

922 public:
923         // constructor
924         Name_set() : root(0) {}

926         // lookup, insert and remove operations
927         Name lookup(const char *key);
928         Name insert(const char *key, Boolean &found);
929         void insert(Name name);

931         // begin/end iterators
932         iterator begin() const;
933         iterator end() const { return iterator(); }

935 private:
936         // rebalance given node
937         void    rebalance(entry *node);

939 private:
940         // tree root
941         entry   *root;
942 };

944 /*
945  *      extern declarations for all global variables.
946  *      The actual declarations are in globals.cc
947  */
948 extern char             char_semantics[];
949 extern wchar_t          char_semantics_char[];
950 extern Macro_list       cond_macro_list;
951 extern Boolean          conditional_macro_used;
952 extern Boolean          do_not_exec_rule;               /* '-n' */
953 extern Boolean          dollarget_seen;
954 extern Boolean          dollarless_flag;
955 extern Name             dollarless_value;
956 extern char             **environ;
957 extern Envvar           envvar;
958 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
959 extern int              exit_status;
960 #endif
961 extern wchar_t          *file_being_read;
962 /* Variable gnu_style=true if env. var. SUN_MAKE_COMPAT_MODE=GNU (RFE 4866328) *
963 extern Boolean          gnu_style;
964 extern Name_set         hashtab;
965 extern Name             host_arch;
966 extern Name             host_mach;
967 extern int              line_number;
968 extern char             *make_state_lockfile;
969 extern Boolean          make_word_mentioned;
970 extern Makefile_type    makefile_type;
971 extern char             mbs_buffer[];
972 extern Name             path_name;
973 extern Boolean          posix;
974 extern Name             query;
975 extern Boolean          query_mentioned;
976 extern Name             hat;
977 extern Boolean          reading_environment;
978 extern Name             shell_name;
979 extern Boolean          svr4;
980 extern Name             target_arch;
981 extern Name             target_mach;
982 extern Boolean          tilde_rule;
983 extern wchar_t          wcs_buffer[];
984 extern Boolean          working_on_targets;
985 extern Name             virtual_root;
```



```
986 extern Boolean          vpath_defined;
987 extern Name             vpath_name;
988 extern Boolean          make_state_locked;
989 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
990 extern Boolean          out_err_same;
991 #endif
992 extern pid_t            childPid;
993 extern nl_catd          libmksh_catd;

995 /*
996  * RFE 1257407: make does not use fine granularity time info available from stat
997  * High resolution time comparison.
998  */

1000 inline int
1001 operator==(const timestruc_t &t1, const timestruc_t &t2) {
1002         return ((t1.tv_sec == t2.tv_sec) && (t1.tv_nsec == t2.tv_nsec));
1003 }

1005 inline int
1006 operator!=(const timestruc_t &t1, const timestruc_t &t2) {
1007         return ((t1.tv_sec != t2.tv_sec) || (t1.tv_nsec != t2.tv_nsec));
1008 }

1010 inline int
1011 operator>(const timestruc_t &t1, const timestruc_t &t2) {
1012         if (t1.tv_sec == t2.tv_sec) {
1013                 return (t1.tv_nsec > t2.tv_nsec);
1014         }
1015         return (t1.tv_sec > t2.tv_sec);
1016 }

1018 inline int
1019 operator>=(const timestruc_t &t1, const timestruc_t &t2) {
1020         if (t1.tv_sec == t2.tv_sec) {
1021                 return (t1.tv_nsec >= t2.tv_nsec);
1022         }
1023         return (t1.tv_sec > t2.tv_sec);
1024 }

1026 inline int
1027 operator<(const timestruc_t &t1, const timestruc_t &t2) {
1028         if (t1.tv_sec == t2.tv_sec) {
1029                 return (t1.tv_nsec < t2.tv_nsec);
1030         }
1031         return (t1.tv_sec < t2.tv_sec);
1032 }

1034 inline int
1035 operator<=(const timestruc_t &t1, const timestruc_t &t2) {
1036         if (t1.tv_sec == t2.tv_sec) {
1037                 return (t1.tv_nsec <= t2.tv_nsec);
1038         }
1039         return (t1.tv_sec < t2.tv_sec);
1040 }

1042 #endif
1043 #endif /* ! codereview */
```

```
**********************************************************
    2372 Wed May 20 11:04:14 2015
new/usr/src/cmd/make/include/mksh/dosys.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#ifndef _MKSH_DOSYS_H
#define _MKSH_DOSYS_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)dosys.h 1.9 06/12/12
 */

#pragma ident   "@(#)dosys.h    1.9     06/12/12"

#include <mksh/defs.h>
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
#       include <rw/xdrstrea.h>
#endif
#include <vroot/vroot.h>

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
extern Boolean  await(register Boolean ignore_error, register Boolean silent_err
#else
extern Boolean  await(register Boolean ignore_error, register Boolean silent_err
#endif
extern int      doexec(register wchar_t *command, register Boolean ignore_error,
extern int      doshell(wchar_t *command, register Boolean ignore_error, Boolean
extern Doname   dosys_mksh(register Name command, register Boolean ignore_error,
extern void     redirect_io(char *stdout_file, char *stderr_file);
extern void     sh_command2string(register String command, register String desti

#endif
#endif /* ! codereview */
```

```
**********************************************************
    1086 Wed May 20 11:04:15 2015
new/usr/src/cmd/make/include/mksh/globals.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#ifndef _MKSH_GLOBALS_H
#define _MKSH_GLOBALS_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 1994 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)globals.h 1.2 06/12/12
 */

#pragma ident   "@(#)globals.h  1.2     06/12/12"

#include <mksh/defs.h>

#endif
#endif /* ! codereview */
```

```
**********************************************************
    1174 Wed May 20 11:04:15 2015
new/usr/src/cmd/make/include/mksh/i18n.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#ifndef _MKSH_I18N_H
#define _MKSH_I18N_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 1994 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)i18n.h 1.2 06/12/12
 */

#pragma ident   "@(#)i18n.h     1.2     06/12/12"

#include <mksh/defs.h>

extern int      get_char_semantics_entry(wchar_t ch);
extern char     get_char_semantics_value(wchar_t ch);

#endif
#endif /* ! codereview */
```

```
**********************************************************
    1107 Wed May 20 11:04:15 2015
new/usr/src/cmd/make/include/mksh/libmksh_init.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#ifndef _MKSH_INIT_H
#define _MKSH_INIT_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 1995 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)libmksh_init.h 1.2 06/12/12
 */

#pragma ident   "@(#)libmksh_init.h     1.2     06/12/12"

int libmksh_init()
void libmksh_fini();

#endif
#endif /* ! codereview */
```

```
**********************************************************
    1470 Wed May 20 11:04:15 2015
new/usr/src/cmd/make/include/mksh/macro.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#ifndef _MKSH_MACRO_H
#define _MKSH_MACRO_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2002 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)macro.h 1.3 06/12/12
 */

#pragma ident   "@(#)macro.h    1.3     06/12/12"

#include <mksh/defs.h>

extern void     expand_macro(register Source source, register String destination
extern void     expand_value(Name value, register String destination, Boolean cm
extern Name     getvar(register Name name);

extern Property setvar_daemon(register Name name, register Name value, Boolean a

#endif
#endif /* ! codereview */
```

```
**********************************************************
    2316 Wed May 20 11:04:15 2015
new/usr/src/cmd/make/include/mksh/misc.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#ifndef _MKSH_MISC_H
#define _MKSH_MISC_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)misc.h 1.4 06/12/12
 */

#pragma ident   "@(#)misc.h     1.4     06/12/12"

#include <mksh/defs.h>

extern void     append_char(wchar_t from, register String to);
extern Property append_prop(register Name target, register Property_id type);
extern void     append_string(register wchar_t *from, register String to, regist
extern void     enable_interrupt(register void (*handler) (int));
extern char     *errmsg(int errnum);
extern void     fatal_mksh(char * message, ...);
extern void     fatal_reader_mksh(char * pattern, ...);
extern char     *get_current_path_mksh(void);
extern Property get_prop(register Property start, register Property_id type);
extern char     *getmem(register int size);
extern Name     getname_fn(wchar_t *name, register int len, register Boolean don
extern void     store_name(Name name);
extern void     free_name(Name name);
extern void     handle_interrupt_mksh(int);
extern Property maybe_append_prop(register Name target, register Property_id typ
extern void     retmem(wchar_t *p);
extern void     retmem_mb(caddr_t p);
extern void     setup_char_semantics(void);
extern void     setup_interrupt(register void (*handler) (int));
extern void     warning_mksh(char * message, ...);

extern void     append_string(register char *from, register String to, register
extern wchar_t  *get_wstring(char * from);


#endif
#endif /* ! codereview */
```

```
**********************************************************
    1510 Wed May 20 11:04:15 2015
new/usr/src/cmd/make/include/mksh/mksh.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#ifndef _MKSH_MKSH_H
#define _MKSH_MKSH_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2003 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)mksh.h 1.7 06/12/12
 */

#pragma ident   "@(#)mksh.h     1.7     06/12/12"

/*
 * Included files
 */
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
#       include <dm/Avo_DmakeCommand.h>
#endif

#include <mksh/defs.h>
#include <unistd.h>

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */

extern int      do_job(Avo_DmakeCommand *cmd_list[], char *env_list[], char *std

#endif /* TEAMWARE_MAKE_CMN */

#endif
#endif /* ! codereview */
```

```
**********************************************************
    1132 Wed May 20 11:04:16 2015
new/usr/src/cmd/make/include/mksh/read.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#ifndef _MKSH_READ_H
#define _MKSH_READ_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 1994 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)read.h 1.2 06/12/12
 */

#pragma ident   "@(#)read.h     1.2     06/12/12"

#include <mksh/defs.h>

extern Source   get_next_block_fn(register Source source);

#endif
#endif /* ! codereview */
```

```
**********************************************************
    2036 Wed May 20 11:04:16 2015
new/usr/src/cmd/make/include/vroot/args.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 1999 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)args.h 1.7 06/12/12
 */

#pragma ident   "@(#)args.h     1.7     06/12/12"

#ifndef _ARGS_H_
#define _ARGS_H_

#include <sys/syscall.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/param.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>

typedef enum { rw_read, rw_write} rwt, *rwpt;

extern  void    translate_with_thunk(register char *filename, int (*thunk) (char

union Args {
        struct { int mode;} access;
        struct { int mode;} chmod;
        struct { int user; int group;} chown;
        struct { int mode;} creat;
        struct { char **argv; char **environ;} execve;
        struct { struct stat *buffer;} lstat;
        struct { int mode;} mkdir;
        struct { char *name; int mode;} mount;
        struct { int flags; int mode;} open;
        struct { char *buffer; int buffer_size;} readlink;
        struct { struct stat *buffer;} stat;
#ifndef SUN5_0
        struct { struct statfs *buffer;} statfs;
```



```
#endif
        struct { int length;} truncate;
        struct { struct timeval *time;} utimes;
};

extern  union Args      vroot_args;
extern  int             vroot_result;

#endif
#endif /* ! codereview */
```

```
**********************************************************
    1837 Wed May 20 11:04:16 2015
new/usr/src/cmd/make/include/vroot/report.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 1994 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)report.h 1.6 06/12/12
 */

#pragma ident   "@(#)report.h   1.6     06/12/12"

#ifndef _REPORT_H_
#define _REPORT_H_

#include <stdio.h>

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

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

/* the following definitions define the interface between make and
 * NSE - the two systems must track each other.
 */
#define NSE_DEPINFO             ".nse_depinfo"
#define NSE_DEPINFO_LOCK        ".nse_depinfo.lock"
#define NSE_DEP_ENV             "NSE_DEP"
#define NSE_TFS_PUSH            "/usr/nse/bin/tfs_push"
#define NSE_TFS_PUSH_LEN        8
#define NSE_VARIANT_ENV         "NSE_VARIANT"
#define NSE_RT_SOURCE_NAME      "Shared_Source"

#endif
#endif /* ! codereview */
```

```
**********************************************************
    2403 Wed May 20 11:04:16 2015
new/usr/src/cmd/make/include/vroot/vroot.h
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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.
 */
/*
 * @(#)vroot.h 1.10 06/12/12
 */

#pragma ident   "@(#)vroot.h    1.10    06/12/12"

#ifndef _VROOT_H_
#define _VROOT_H_

#include <stdio.h>
#include <nl_types.h>

#define VROOT_DEFAULT ((pathpt)-1)

typedef struct {
        char            *path;
        short           length;
} pathcellt, *pathcellpt, patht;
typedef patht           *pathpt;

extern  void            add_dir_to_path(register char *path, register pathpt *po
extern  void            flush_path_cache(void);
extern  void            flush_vroot_cache(void);
extern  char            *get_path_name(void);
extern  char            *get_vroot_path(register char **vroot, register char **p
extern  char            *get_vroot_name(void);
extern  int             open_vroot(char *path, int flags, int mode, pathpt vroot
extern  pathpt          parse_path_string(register char *string, register int re
extern  void            scan_path_first(void);
extern  void            scan_vroot_first(void);
extern  void            set_path_style(int style);

extern  int             access_vroot(char *path, int mode, pathpt vroot_path, pa

extern  int             execve_vroot(char *path, char **argv, char **environ, pa

extern  int             lstat_vroot(char *path, struct stat *buffer, pathpt vroo
```



```
extern  int             stat_vroot(char *path, struct stat *buffer, pathpt vroot
extern  int             readlink_vroot(char *path, char *buffer, int buffer_size


extern nl_catd  libvroot_catd;
#endif
#endif /* ! codereview */
```

```
**********************************************************
    2131 Wed May 20 11:04:16 2015
new/usr/src/cmd/make/lib/Lib.mk
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 2001 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Lib.mk 1.13 06/12/12
#

# Definitions of the libraries supplied by make/dmake.
# You must correctly define TOP before including this file!

MAKE_TOP        = $(TOP)/Make

LIBBSD_DIR      = $(MAKE_TOP)/lib/bsd
LIBBSD          = $(LIBBSD_DIR)/$(VARIANT)/libbsd.a

LIBDMRXM_DIR    = $(MAKE_TOP)/lib/dmrxm
LIBDMRXM        = $(LIBDMRXM_DIR)/$(VARIANT)/libdmrxm.a

LIBDMRXS_DIR    = $(MAKE_TOP)/lib/dmrxs
LIBDMRXS        = $(LIBDMRXS_DIR)/$(VARIANT)/libdmrxs.a

LIBMAKESTATE_DIR = $(MAKE_TOP)/lib/makestate
LIBMAKESTATE    = $(LIBMAKESTATE_DIR)/$(VARIANT)/libmakestate.a

LIBMKSDMSI18N_DIR = $(MAKE_TOP)/lib/mksdmsi18n
LIBMKSDMSI18N   = $(LIBMKSDMSI18N_DIR)/$(VARIANT)/libmksdmsi18n.a

LIBMKSH_DIR     = $(MAKE_TOP)/lib/mksh
LIBMKSH         = $(LIBMKSH_DIR)/$(VARIANT)/libmksh.a

LIBVROOT_DIR    = $(MAKE_TOP)/lib/vroot
LIBVROOT        = $(LIBVROOT_DIR)/$(VARIANT)/libvroot.a

LIBDM_DIR       = $(MAKE_TOP)/lib/dm
LIBDM           = $(LIBDM_DIR)/$(VARIANT)/libdm.a

LIBDMCONF_DIR   = $(MAKE_TOP)/lib/dmconf
LIBDMCONF       = $(LIBDMCONF_DIR)/$(VARIANT)/libdmconf.a

LIBDMRC_DIR     = $(MAKE_TOP)/lib/dmrc
LIBDMRC         = $(LIBDMRC_DIR)/$(VARIANT)/libdmrc.a
```



```
LIBDMTHREAD_DIR = $(MAKE_TOP)/lib/dmthread
LIBDMTHREAD     = $(LIBDMTHREAD_DIR)/$(VARIANT)/libdmthread.a

LIBRX_DIR       = $(MAKE_TOP)/lib/rx
LIBRX           = $(LIBRX_DIR)/$(VARIANT)/librx.a

#endif /* ! codereview */
```

```
**********************************************************
    1092 Wed May 20 11:04:17 2015
new/usr/src/cmd/make/lib/Makefile
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.8 06/12/12
#
# @(#)Makefile  1.4     96/03/16 SMI

TOP =   ../..

SUBDIRS =       \
                makestate \
                bsd \
                dmrxm \
                dmrxs \
                mksdmsi18n \
                mksh \
                vroot

include $(TOP)/rules/recurse.mk
#endif /* ! codereview */
```

****
1008 Wed May 20 11:04:17 2015
new/usr/src/cmd/make/lib/bsd/Makefile
make: initial Sun make source, disconnected from the build
****

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.5 06/12/12
#

TOP =           ../../..
include $(TOP)/rules/variant.mk
include $(TOP)/rules/derived.mk
#endif /* ! codereview */
```

****
1564 Wed May 20 11:04:17 2015
new/usr/src/cmd/make/lib/bsd/src/Makefile
make: initial Sun make source, disconnected from the build
****

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.3 06/12/12
#

# Generic makefile for use in src directories.  Knows how to make common things
# in the right $(VARIANT) directory.

include $(TOP)/rules/variant.mk

all :=          TARG = all
install :=              TARG = install
clean :=                TARG = clean
test :=         TARG = test
l10n_install := TARG = l10n_install
i18n_install := TARG = i18n_install

SRC =           ../src
MFLAGS +=       SRC=$(SRC)

# See $(TOP)/rules/master.mk for how these are built.
%.h %.cc %.C %.E %.o all install clean test l10n_install i18n_install: FRC
        @ if [ ! -d ../$(VARIANT) ]; then \
                mkdir ../$(VARIANT) ; \
        fi
        cd ../$(VARIANT); $(MAKE) $(MFLAGS) -f $(SRC)/Variant.mk DESTDIR=$(DESTD

FRC:
#endif /* ! codereview */
```

****
1220 Wed May 20 11:04:18 2015
new/usr/src/cmd/make/lib/bsd/src/Variant.mk
make: initial Sun make source, disconnected from the build
****

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Variant.mk 1.13 06/12/12
#

TOP =            ../../../..
include ${TOP}/rules/master.mk

PKG_TOP =        $(TOP)/Make
CPPFLAGS +=      -I${PKG_TOP}/include
MSG_FILE = libbsd.msg
I18N_DIRS = $(SRC)

CCSRCS =                bsd.cc
CSRCS =

HDRS_DIR =       ${PKG_TOP}/include/bsd

.INIT: ${HDRS_DIR}/bsd.h

LIBNAME =        libbsd.a

include ${TOP}/rules/lib.mk

#endif /* ! codereview */
```

```
**********************************************************
    4343 Wed May 20 11:04:18 2015
new/usr/src/cmd/make/lib/bsd/src/bsd.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)bsd.cc 1.6 06/12/12
 */

#pragma ident   "@(#)bsd.cc     1.6     06/12/12"

#include <signal.h>

#include <bsd/bsd.h>

/* External references.
 */

/* Forward references.
 */

/* Static data.
 */
```



```
extern SIG_PF
bsd_signal (int Signal, SIG_PF Handler)
{
  auto SIG_PF                   previous_handler;
#ifdef SUN5_0
#ifdef sun
  previous_handler = sigset (Signal, Handler);
#else
  auto struct sigaction         new_action;
  auto struct sigaction         old_action;

  new_action.sa_flags = SA_SIGINFO;
  new_action.sa_handler = (void (*) ()) Handler;
  (void) sigemptyset (&new_action.sa_mask);
  (void) sigaddset (&new_action.sa_mask, Signal);

  (void) sigaction (Signal, &new_action, &old_action);

  previous_handler = (SIG_PF) old_action.sa_handler;
#endif
#elif defined(linux)
  previous_handler = sigset (Signal, Handler);
#else
  previous_handler = signal (Signal, Handler);
#endif
  return previous_handler;
}
```

```
 70 extern void
 71 bsd_signals (void)
 72 {
 73   static int                    initialized = 0;

 75   if (initialized == 0)
 76     {
 77       initialized = 1;
 78 #if !defined(SUN5_0) && !defined(linux)
 79 #if defined(SIGHUP)
 80       (void) bsd_signal (SIGHUP, SIG_DFL);
 81 #endif
 82 #if defined(SIGINT)
 83       (void) bsd_signal (SIGINT, SIG_DFL);
 84 #endif
 85 #if defined(SIGQUIT)
 86       (void) bsd_signal (SIGQUIT, SIG_DFL);
 87 #endif
 88 #if defined(SIGILL)
 89       (void) bsd_signal (SIGILL, SIG_DFL);
 90 #endif
 91 #if defined(SIGTRAP)
 92       (void) bsd_signal (SIGTRAP, SIG_DFL);
 93 #endif
 94 #if defined(SIGIOT)
 95       (void) bsd_signal (SIGIOT, SIG_DFL);
 96 #endif
 97 #if defined(SIGABRT)
 98       (void) bsd_signal (SIGABRT, SIG_DFL);
 99 #endif
100 #if defined(SIGEMT)
101       (void) bsd_signal (SIGEMT, SIG_DFL);
102 #endif
103 #if defined(SIGFPE)
104       (void) bsd_signal (SIGFPE, SIG_DFL);
105 #endif
106 #if defined(SIGBUS)
107       (void) bsd_signal (SIGBUS, SIG_DFL);
108 #endif
109 #if defined(SIGSEGV)
110       (void) bsd_signal (SIGSEGV, SIG_DFL);
111 #endif
112 #if defined(SIGSYS)
113       (void) bsd_signal (SIGSYS, SIG_DFL);
114 #endif
115 #if defined(SIGPIPE)
116       (void) bsd_signal (SIGPIPE, SIG_DFL);
117 #endif
118 #if defined(SIGALRM)
119       (void) bsd_signal (SIGALRM, SIG_DFL);
120 #endif
121 #if defined(SIGTERM)
122       (void) bsd_signal (SIGTERM, SIG_DFL);
123 #endif
124 #if defined(SIGUSR1)
125       (void) bsd_signal (SIGUSR1, SIG_DFL);
126 #endif
127 #if defined(SIGUSR2)
128       (void) bsd_signal (SIGUSR2, SIG_DFL);
129 #endif
130 #if defined(SIGCLD)
131       (void) bsd_signal (SIGCLD, SIG_DFL);
132 #endif
133 #if defined(SIGCHLD)
134       (void) bsd_signal (SIGCHLD, SIG_DFL);
135 #endif
```



```
136 #if defined(SIGPWR)
137       (void) bsd_signal (SIGPWR, SIG_DFL);
138 #endif
139 #if defined(SIGWINCH)
140       (void) bsd_signal (SIGWINCH, SIG_DFL);
141 #endif
142 #if defined(SIGURG)
143       (void) bsd_signal (SIGURG, SIG_DFL);
144 #endif
145 #if defined(SIGIO)
146       (void) bsd_signal (SIGIO, SIG_DFL);
147 #else
148 #if defined(SIGPOLL)
149       (void) bsd_signal (SIGPOLL, SIG_DFL);
150 #endif
151 #endif
152 #if defined(SIGTSTP)
153       (void) bsd_signal (SIGTSTP, SIG_DFL);
154 #endif
155 #if defined(SIGCONT)
156       (void) bsd_signal (SIGCONT, SIG_DFL);
157 #endif
158 #if defined(SIGTTIN)
159       (void) bsd_signal (SIGTTIN, SIG_DFL);
160 #endif
161 #if defined(SIGTTOU)
162       (void) bsd_signal (SIGTTOU, SIG_DFL);
163 #endif
164 #if defined(SIGVTALRM)
165       (void) bsd_signal (SIGVTALRM, SIG_DFL);
166 #endif
167 #if defined(SIGPROF)
168       (void) bsd_signal (SIGPROF, SIG_DFL);
169 #endif
170 #if defined(SIGXCPU)
171       (void) bsd_signal (SIGXCPU, SIG_DFL);
172 #endif
173 #if defined(SIGXFSZ)
174       (void) bsd_signal (SIGXFSZ, SIG_DFL);
175 #endif
176 #endif
177     }

179   return;
180 }
181 #endif /* ! codereview */
```

```
**********************************************************
    1286 Wed May 20 11:04:18 2015
new/usr/src/cmd/make/lib/makestate/Makefile
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2004 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.7 06/12/12
#

TOP =           ../../..
include $(TOP)/rules/variant.mk

V9_VARIANT :sh= \
if [ -x /usr/bin/isalist ] ; \
then \
    for f in `/usr/bin/isalist` ; \
    do \
        if [ "$f" = sparcv9 ] ; \
        then \
            echo sparcv9 ; \
            break ; \
        fi ; \
        if [ "$f" = amd64 ] ; \
        then \
            echo amd64-S2 ; \
            break ; \
        fi ; \
    done ; \
fi

include $(TOP)/rules/derived.mk
#endif /* ! codereview */
```

****
1596 Wed May 20 11:04:18 2015
new/usr/src/cmd/make/lib/makestate/src/Makefile
make: initial Sun make source, disconnected from the build
****

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.5 06/12/12
#
# @(#)Makefile  1.1     96/03/11 SMI

# Generic makefile for use in src directories.  Knows how to make common things
# in the right $(VARIANT) directory.

include $(TOP)/rules/variant.mk

all :=          TARG = all
install :=              TARG = install
clean :=                TARG = clean
test :=         TARG = test
l10n_install := TARG = l10n_install
i18n_install := TARG = i18n_install

SRC =           ../src
MFLAGS +=       SRC=$(SRC)

# See $(TOP)/rules/master.mk for how these are built.
%.h %.cc %.C %.E %.o all install clean test l10n_install i18n_install: FRC
        @ if [ ! -d ../$(VARIANT) ]; then \
                mkdir ../$(VARIANT) ; \
        fi
        cd ../$(VARIANT); $(MAKE) $(MFLAGS) -f $(SRC)/Variant.mk DESTDIR=$(DESTD

FRC:
#endif /* ! codereview */
```

```
**********************************************************
    1948 Wed May 20 11:04:18 2015
new/usr/src/cmd/make/lib/makestate/src/Variant.mk
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2005 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Variant.mk 1.12 06/12/12
#

TOP =           ../../../..

%.o: $(SRC)/%.c
        $(COMPILE.c) $(OUTPUT_OPTION) $<

include $(TOP)/rules/master.mk

PKG_TOP =       $(TOP)/Make

CSRCS = \
                ld_file.c \
                lock.c

LIBNAME =       libmakestate.a
DLIBNAME =      libmakestate.so.1
MSG_FILE = libmakestate.msg
I18N_DIRS = $(SRC)

CFLAGS += $(V9FLAGS) -KPIC -DTEXT_DOMAIN=\"SUNW_OST_OSLIB\"
CPPFLAGS=

#include $(TOP)/Make/lib/Lib.mk
include $(TOP)/rules/lib.mk

POUND_SIGN:sh=  echo \#
RELEASE=        5.11
VERSION=        $(RELEASE_VER)
PATCHID=        $(VERSION)
DATE:sh         = date '+%B %Y'
RELEASE_DATE=   $(DATE)
PATCH_DATE=     $(RELEASE_DATE)
RELEASE_CM=     "@($(POUND_SIGN))RELEASE VERSION SunOS $(RELEASE) $(PATCHID) $(P

PROCESS_COMMENT=        mcs -a $(RELEASE_CM)
POST_PROCESS_SO=        $(PROCESS_COMMENT) $@
```



```
$(DLIBNAME) : $(LIBNAME)
        $(CC) $(V9FLAGS) -o $@ -dy -G -ztext -h $@ ld_file.o lock.o -lelf
        mcs -d $@
        ${POST_PROCESS_SO}

all: $(DLIBNAME)

install: all
        ${INSTALL} -d ${DESTDIR}/usr/lib$(VAR_DIR)
        ${RM} ${DESTDIR}/usr/lib$(VAR_DIR)/$(DLIBNAME)
        ${INSTALL} $(DLIBNAME) ${DESTDIR}/usr/lib$(VAR_DIR)
#endif /* ! codereview */
```

```
**********************************************************
    3721 Wed May 20 11:04:19 2015
new/usr/src/cmd/make/lib/makestate/src/ld_file.c
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 1998 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)ld_file.c 1.7 06/12/12
 */
#pragma ident   "@(#)ld_file.c  1.7     06/12/12"

#pragma init(ld_support_init)

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <libelf.h>
#include <sys/param.h>
#include <link.h>

#define SUNPRO_DEPENDENCIES     "SUNPRO_DEPENDENCIES"

/*
 * Linked list of strings - used to keep lists of names
 * of directories or files.
 */

struct Stritem {
        char *          str;
        void *          next;
};

typedef struct Stritem  Stritem;

static char             * depend_file = NULL;
static Stritem          * list = NULL;


void mk_state_init()
{
        depend_file = getenv(SUNPRO_DEPENDENCIES);
} /* mk_state_init() */
```



```
static void
prepend_str(Stritem **list, const char * str)
{
        Stritem * new;
        char    * newstr;

        if (!(new = calloc(1, sizeof (Stritem)))) {
                perror("libmakestate.so");
                return;
        } /* if */

        if (!(newstr = malloc(strlen(str) + 1))) {
                perror("libmakestate.so");
                return;
        } /* if */

        new->str = strcpy(newstr, str);
        new->next = *list;
        *list = new;

} /* prepend_str() */


void
mk_state_collect_dep(const char * file)
{
        /*
         * SUNPRO_DEPENDENCIES wasn't set, we don't collect .make.state
         * information.
         */
        if (!depend_file)
                return;

        prepend_str(&list, file);

}  /* mk_state_collect_dep() */


void
mk_state_update_exit()
{
        Stritem         * cur;
        char              lockfile[MAXPATHLEN], * err, * space, * target;
        FILE            * ofp;
        extern char     * file_lock(char *, char *, int);

        if (!depend_file)
                return;

        if ((space = strchr(depend_file, ' ')) == NULL)
                return;
        *space = '\0';
        target = &space[1];

        (void) sprintf(lockfile, "%s.lock", depend_file);
        if ((err = file_lock(depend_file, lockfile, 0))) {
                (void) fprintf(stderr, "%s\n", err);
                return;
        } /* if */

        if (!(ofp = fopen(depend_file, "a")))
                return;
```

```
128         if (list)
129                 (void) fprintf(ofp, "%s: ", target);
130
131         for (cur = list; cur; cur = cur->next)
132                 (void) fprintf(ofp, " %s", cur->str);
133
134         (void) fputc('\n', ofp);
135
136         (void) fclose(ofp);
137         (void) unlink(lockfile);
138         *space = ' ';
139
140 } /* mk_state_update_exit() */
141
142 static void
143 /* LINTED static unused */
144 ld_support_init()
145 {
146         mk_state_init();
147
148 } /* ld_support_init() */
149
150 /* ARGSUSED */
151 void
152 ld_file(const char * file, const Elf_Kind ekind, int flags, Elf *elf)
153 {
154         if(! ((flags & LD_SUP_DERIVED) && !(flags & LD_SUP_EXTRACTED)))
155                 return;
156
157         mk_state_collect_dep(file);
158
159 } /* ld_file */
160
161 void
162 ld_atexit(int exit_code)
163 {
164         if (exit_code)
165            return;
166
167         mk_state_update_exit();
168
169 } /* ld_atexit() */
170
171 /*
172  * Supporting 64-bit objects
173  */
174 void
175 ld_file64(const char * file, const Elf_Kind ekind, int flags, Elf *elf)
176 {
177         if(! ((flags & LD_SUP_DERIVED) && !(flags & LD_SUP_EXTRACTED)))
178                 return;
179
180         mk_state_collect_dep(file);
181
182 } /* ld_file64 */
183
184 void
185 ld_atexit64(int exit_code)
186 {
187         if (exit_code)
188            return;
189
190         mk_state_update_exit();
191
192 } /* ld_atexit64() */
193 #endif /* ! codereview */
```

```
**********************************************************
    4872 Wed May 20 11:04:19 2015
new/usr/src/cmd/make/lib/makestate/src/lock.c
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)lock.c 1.5 06/12/12
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/stat.h>
#include <sys/errno.h>
#include <errno.h>              /* errno */

#if defined(_LP64)
/*
 * The symbols _sys_errlist and _sys_nerr are not visible in the
 * LP64 libc.  Use strerror(3C) instead.
 */
#else  /* #_LP64 */
extern  char *          sys_errlist[];
extern  int             sys_nerr;
#endif /* #_LP64 */

static  void            file_lock_error();

/*
 * This code stolen from the NSE library and changed to not depend
 * upon any NSE routines or header files.
 *
 * Simple file locking.
 * Create a symlink to a file.  The "test and set" will be
 * atomic as creating the symlink provides both functions.
 *
 * The timeout value specifies how long to wait for stale locks
 * to disappear.  If the lock is more than 'timeout' seconds old
```



```
 * then it is ok to blow it away.  This part has a small window
 * of vunerability as the operations of testing the time,
 * removing the lock and creating a new one are not atomic.
 * It would be possible for two processes to both decide to blow
 * away the lock and then have process A remove the lock and establish
 * its own, and then have process B remove the lock which accidentily
 * removes A's lock rather than the stale one.
 *
 * A further complication is with the NFS.  If the file in question is
 * being served by an NFS server, then its time is set by that server.
 * We can not use the time on the client machine to check for a stale
 * lock.  Therefore, a temp file on the server is created to get
 * the servers current time.
 *
 * Returns an error message.  NULL return means the lock was obtained.
 *
 */
char *
file_lock(char * name, char * lockname, int timeout)
{
        int             r;
        int             fd;
        struct  stat    statb;
        struct  stat    fs_statb;
        char            tmpname[MAXPATHLEN];
        static  char    msg[MAXPATHLEN];

        if (timeout <= 0) {
                timeout = 15;
        }
        for (;;) {
                r = symlink(name, lockname);
                if (r == 0) {
                        return (NULL);
                }
                if (errno != EEXIST) {
                        file_lock_error(msg, name,
                            (const char *)"symlink(%s, %s)", name, lockname);
                        return (msg);
                }
                for (;;) {
                        (void) sleep(1);
                        r = lstat(lockname, &statb);
                        if (r == -1) {
                                /*
                                 * The lock must have just gone away - try
                                 * again.
                                 */
                                break;
                        }

                        /*
                         * With the NFS the time given a file is the time on
                         * the file server.  This time may vary from the
                         * client's time.  Therefore, we create a tmpfile in
                         * the same directory to establish the time on the
                         * server and use this time to see if the lock has
                         * expired.
                         */
                        (void) sprintf(tmpname, "%s.XXXXXX", lockname);
                        (void) mktemp(tmpname);
                        fd = creat(tmpname, 0666);
                        if (fd != -1) {
                                (void) close(fd);
                        } else {
                                file_lock_error(msg, name,
```

```
128                                     (const char *)"creat(%s)", tmpname);
129                                 return (msg);
130                         }
131                         if (stat(tmpname, &fs_statb) == -1) {
132                                 file_lock_error(msg, name,
133                                     (const char *)"stat(%s)", tmpname);
134                                 return (msg);
135                         }
136                         (void) unlink(tmpname);
137                         if (statb.st_mtime + timeout < fs_statb.st_mtime) {
138                                 /*
139                                  * The lock has expired - blow it away.
140                                  */
141                                 (void) unlink(lockname);
142                                 break;
143                         }
144                 }
145         }
146         /* NOTREACHED */
147 }

149 /*
150  * Format a message telling why the lock could not be created.
151  */
152 /* VARARGS4 */
153 static  void
154 file_lock_error(char * msg, char * file, const char * str, char * arg1,
155         char * arg2)
156 {
157         int     len;

159         (void) sprintf(msg, "Could not lock file `%s'; ", file);
160         len = strlen(msg);
161         (void) sprintf(&msg[len], str, arg1, arg2);
162         (void) strcat(msg, " failed - ");
163 #if defined(_LP64)
164         /* Needs to be changed to use strerror(3C) instead. */
165         len = strlen(msg);
166         (void) sprintf(&msg[len], "errno %d", errno);
167 #else  /* #_LP64 */
168         if (errno < sys_nerr) {
169                 (void) strcat(msg, sys_errlist[errno]);
170         } else {
171                 len = strlen(msg);
172                 (void) sprintf(&msg[len], "errno %d", errno);
173         }
174 #endif /* #_LP64 */
175 }
176 #endif /* ! codereview */
```

```
**********************************************************
    1009 Wed May 20 11:04:19 2015
new/usr/src/cmd/make/lib/mksdmsi18n/Makefile
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #
  21 # Copyright 1996 Sun Microsystems, Inc. All rights reserved.
  22 # Use is subject to license terms.
  23 #
  24 # @(#)Makefile 1.5 06/12/12
  25 #

  27 TOP             = ../../..
  28 include $(TOP)/rules/variant.mk
  29 include $(TOP)/rules/derived.mk

  31 #endif /* ! codereview */
```

```
**********************************************************
    1327 Wed May 20 11:04:19 2015
new/usr/src/cmd/make/lib/mksdmsi18n/src/Makefile
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.10 06/12/12
#

TOP =           ../../../..
include $(TOP)/rules/master.mk

SUBDIRS =       lib
include $(TOP)/rules/recurse.mk
include $(SRC)/lib/version.mk

PKG_TOP =       $(TOP)/Make
CPPFLAGS +=     -I$(PKG_TOP)/include

CCSRCS =                libmksdmsi18n_init.cc

LIBNAME =       libmksdmsi18n.a

#
# Pass in the version number for the i18n message catalog.
#
libmksdmsi18n_init.o := CPPFLAGS += -DI18N_VERSION=$(VERSION)

include $(TOP)/rules/lib.mk
#endif /* ! codereview */
```

```
**********************************************************
    1401 Wed May 20 11:04:19 2015
new/usr/src/cmd/make/lib/mksdmsi18n/src/Variant.mk
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1997 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Variant.mk 1.4 06/12/12
#

TOP =           ../../../..
include $(TOP)/rules/master.mk
#include $(TOP)/rules/dmake.mk

#SUBDIRS =      $(SRC)/lib
#include $(TOP)/rules/recurse.mk

CCSRCS = \
                libmksdmsi18n_init.cc

CSRCS =

include $(SRC)/lib/version.mk

libmksdmsi18n_init.o := CPPFLAGS += -DI18N_VERSION=$(VERSION)

#PKG_TOP =      $(TOP)/Make
#CPPFLAGS +=    -I$(PKG_TOP)/include

LIBNAME =       libmksdmsi18n.a
MSG_FILE = libmksdmsi18n.msg
I18N_DIRS = $(SRC)

include $(TOP)/Make/lib/Lib.mk
include $(TOP)/rules/lib.mk

#endif /* ! codereview */
```

```
**********************************************************
    1687 Wed May 20 11:04:20 2015
new/usr/src/cmd/make/lib/mksdmsi18n/src/lib/Makefile
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.7 06/12/12
#

include version.mk

TOP             = ../../../../..
PKG_TOP         = $(TOP)/Make
PKG_LIB_TOP     = $(PKG_TOP)/lib
I18N_LIBS       = bsd dmrxm dmrxs mksh vroot
I18N_DIRS       = $(I18N_LIBS:%=$(PKG_LIB_TOP)/%/src)
SRC             = .
APPPATH         = .
LIBNAME         = libmksdmsi18n
TEXTDOMAIN      = $(LIBNAME)_$(VERSION)
MSG_FILE        = $(TEXTDOMAIN).msg

all clean:
install: catalogs .WAIT i18n_install

#
# This is a hack until the top level makefiles for i18n/l10n message catalogs
# are better organized and written.
# This "catalogs" target was cp'ed from $(TOP)/rules/lib.mk
#
catalogs:
        $(GENMSG) -l $(SRC)/genmsg.project -o $(MSG_FILE) `find $(I18N_DIRS) \(
        rm -f *.cc.new

include $(PKG_TOP)/prodver.mk
include $(TOP)/rules/i18n-install.mk

#endif /* ! codereview */
```

```
**********************************************************
    1006 Wed May 20 11:04:20 2015
new/usr/src/cmd/make/lib/mksdmsi18n/src/lib/version.mk
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1995 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)version.mk 1.2 06/12/12
#

# A version number is needed to version the i18n catalog file

VERSION = 1

#endif /* ! codereview */
```

```
**********************************************************
    1684 Wed May 20 11:04:20 2015
new/usr/src/cmd/make/lib/mksdmsi18n/src/libmksdmsi18n_init.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 1996 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)libmksdmsi18n_init.cc 1.5 06/12/12
 */

#pragma ident   "@(#)libmksdmsi18n_init.cc      1.5     06/12/12"

#include <avo/intl.h>
#include <stdio.h>
#include <stdlib.h>

nl_catd libmksdmsi18n_catd;

/*
 * Open the catalog file for libmksdmsi18n.  Users of this library must set
 * NSLPATH first.  See avo_18n_init().
 */
int
libmksdmsi18n_init()
{
        char            name[20];

        if (getenv(NOCATGETS("NLSPATH")) == NULL) {
                fprintf(stderr, NOCATGETS("Internal error: Set NLSPATH before op
                return 1;
        }
        sprintf(name, NOCATGETS("libmksdmsi18n_%d"), I18N_VERSION);
        libmksdmsi18n_catd = catopen(name, NL_CAT_LOCALE);
        return 0;
}

/*
 * Close the catalog file for libmksdmsi18n
 */
void
libmksdmsi18n_fini()
{
        catclose(libmksdmsi18n_catd);
```



```
}

#endif /* ! codereview */
```

****
1008 Wed May 20 11:04:20 2015
new/usr/src/cmd/make/lib/mksh/Makefile
make: initial Sun make source, disconnected from the build
****

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.5 06/12/12
#

TOP =           ../../..
include $(TOP)/rules/variant.mk
include $(TOP)/rules/derived.mk
#endif /* ! codereview */
```

****
1584 Wed May 20 11:04:20 2015
new/usr/src/cmd/make/lib/mksh/src/Makefile
make: initial Sun make source, disconnected from the build
****

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.3 06/12/12
#

# Generic makefile for use in src directories.  Knows how to make common things
# in the right $(VARIANT) directory.

#TOP =          ../../../..
include $(TOP)/rules/variant.mk

all :=          TARG = all
install :=              TARG = install
clean :=                TARG = clean
test :=         TARG = test
l10n_install := TARG = l10n_install
i18n_install := TARG = i18n_install

SRC =           ../src
MFLAGS +=       SRC=$(SRC)

# See $(TOP)/rules/master.mk for how these are built.
%.h %.cc %.C %.E %.o all install clean test l10n_install i18n_install: FRC
        @ if [ ! -d ../$(VARIANT) ]; then \
                mkdir ../$(VARIANT) ; \
        fi
        cd ../$(VARIANT); $(MAKE) $(MFLAGS) -f $(SRC)/Variant.mk DESTDIR=$(DESTD

FRC:
#endif /* ! codereview */
```

****
1495 Wed May 20 11:04:21 2015
new/usr/src/cmd/make/lib/mksh/src/Variant.mk
make: initial Sun make source, disconnected from the build
****

```
#
# 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 2002 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Variant.mk 1.19 06/12/12
#

TOP =            ../../../..
include $(TOP)/rules/master.mk
include $(TOP)/rules/dmake.mk

PKG_TOP =       $(TOP)/Make
CPPFLAGS +=     -I$(PKG_TOP)/include

CCSRCS =        \
                dosys.cc \
                globals.cc \
                i18n.cc \
                macro.cc \
                misc.cc \
                mksh.cc \
                read.cc

HDRS_DIR =      $(PKG_TOP)/include/mksh
HDRS_LIST =     $(HDRS_DIR)/defs.h \
                $(CCSRCS:%.cc=$(HDRS_DIR)/%.h) \
                $(CSRCS:%.c=$(HDRS_DIR)/%.h)

.INIT: $(HDRS_LIST)

LIBNAME =       libmksh.a
MSG_FILE = libmksh.msg
I18N_DIRS = $(SRC)

#CPPFLAGS       += -DTEAMWARE_MAKE_CMN -DDISTRIBUTED

include $(TOP)/Make/lib/Lib.mk
include $(TOP)/rules/lib.mk

#endif /* ! codereview */
```

```
**********************************************************
   23077 Wed May 20 11:04:21 2015
new/usr/src/cmd/make/lib/mksh/src/dosys.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
  1 /*
  2  * CDDL HEADER START
  3  *
  4  * The contents of this file are subject to the terms of the
  5  * Common Development and Distribution License (the "License").
  6  * You may not use this file except in compliance with the License.
  7  *
  8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  9  * or http://www.opensolaris.org/os/licensing.
 10  * See the License for the specific language governing permissions
 11  * and limitations under the License.
 12  *
 13  * When distributing Covered Code, include this CDDL HEADER in each
 14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 15  * If applicable, add the following below this CDDL HEADER, with the
 16  * fields enclosed by brackets "[]" replaced with your own identifying
 17  * information: Portions Copyright [yyyy] [name of copyright owner]
 18  *
 19  * CDDL HEADER END
 20  */
 21 /*
 22  * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
 23  * Use is subject to license terms.
 24  */
 25 /*
 26  * @(#)dosys.cc 1.38 06/12/12
 27  */

 29 #pragma ident   "@(#)dosys.cc   1.38    06/12/12"

 31 /*
 32  *      dosys.cc
 33  *
 34  *      Execute one commandline
 35  */

 37 /*
 38  * Included files
 39  */
 40 #include <sys/wait.h>                   /* WIFEXITED(status) */
 41 #include <avo/avo_alloca.h>             /* alloca() */

 43 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL) /* tolik */
 44 #       include <avo/strings.h> /* AVO_STRDUP() */
 45 #if defined(DISTRIBUTED)
 46 #       include <dm/Avo_CmdOutput.h>
 47 #       include <rw/xdrstrea.h>
 48 #endif
 49 #endif

 51 #include <stdio.h>              /* errno */
 52 #include <errno.h>              /* errno */
 53 #include <fcntl.h>              /* open() */
 54 #include <mksh/dosys.h>
 55 #include <mksh/macro.h>         /* getvar() */
 56 #include <mksh/misc.h>          /* getmem(), fatal_mksh(), errmsg() */
 57 #include <mksdmsi18n/mksdmsi18n.h>      /* libmksdmsi18n_init() */
 58 #include <sys/signal.h>         /* SIG_DFL */
 59 #include <sys/stat.h>           /* open() */
 60 #include <sys/wait.h>           /* wait() */
 61 #include <ulimit.h>             /* ulimit() */
```



```
 62 #include <unistd.h>             /* close(), dup2() */

 64 #if defined (HP_UX) || defined (linux)
 65 #       include <sys/param.h>
 66 #       include <wctype.h>
 67 #       include <wchar.h>
 68 #endif

 70 #if defined (linux)
 71 #       define wslen(x) wcslen(x)
 72 #       define wscpy(x,y) wcscpy(x,y)
 73 #endif

 75 /*
 76  * Defined macros
 77  */
 78 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
 79 #define SEND_MTOOL_MSG(cmds) \
 80         if (send_mtool_msgs) { \
 81                 cmds \
 82         }
 83 #else
 84 #define SEND_MTOOL_MSG(cmds)
 85 #endif

 87 /*
 88  * typedefs & structs
 89  */

 91 /*
 92  * Static variables
 93  */

 95 /*
 96  * File table of contents
 97  */
 98 static Boolean  exec_vp(register char *name, register char **argv, char **envp,

100 /*
101  * Workaround for NFS bug. Sometimes, when running 'open' on a remote
102  * dmake server, it fails with "Stale NFS file handle" error.
103  * The second attempt seems to work.
104  */
105 int
106 my_open(const char *path, int oflag, mode_t mode) {
107         int res = open(path, oflag, mode);
108 #ifdef linux
109 // Workaround for NFS problem: even when all directories in 'path'
110 // exist, 'open' (file creation) fails with ENOENT.
111         int nattempt = 0;
112         while (res < 0 && (errno == ESTALE || errno == EAGAIN || errno == ENOENT
113                 nattempt++;
114                 if(nattempt > 30) {
115                         break;
116                 }
117                 sleep(1);
118 #else
119         if (res < 0 && (errno == ESTALE || errno == EAGAIN)) {
120 #endif
121                 /* Stale NFS file handle. Try again */
122                 res = open(path, oflag, mode);
123         }
124         return res;
125 }

127 /*
```

```
 *      void
 *      redirect_io(char *stdout_file, char *stderr_file)
 *
 *      Redirects stdout and stderr for a child mksh process.
 */
void
redirect_io(char *stdout_file, char *stderr_file)
{
        long            descriptor_limit;
        int             i;

#if defined (HP_UX) || defined (linux)
        /*
         *  HP-UX does not support the UL_GDESLIM command for ulimit().
         *  NOFILE == max num open files per process (from <sys/param.h>)
         */
        descriptor_limit = NOFILE;
#else
        if ((descriptor_limit = ulimit(UL_GDESLIM)) < 0) {
                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 89, "ulimit() failed:
        }
#endif
        for (i = 3; i < descriptor_limit; i++) {
                (void) close(i);
        }
        if ((i = my_open(stdout_file,
                 O_WRONLY | O_CREAT | O_TRUNC | O_DSYNC,
                 S_IREAD | S_IWRITE)) < 0) {
                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 90, "Couldn't open sta
                      stdout_file,
                      errmsg(errno));
        } else {
                if (dup2(i, 1) == -1) {
                        fatal_mksh(NOCATGETS("*** Error: dup2(3, 1) failed: %s")
                                errmsg(errno));
                }
                close(i);
        }
        if (stderr_file == NULL) {
                if (dup2(1, 2) == -1) {
                        fatal_mksh(NOCATGETS("*** Error: dup2(1, 2) failed: %s")
                                errmsg(errno));
                }
        } else if ((i = my_open(stderr_file,
                        O_WRONLY | O_CREAT | O_TRUNC | O_DSYNC,
                        S_IREAD | S_IWRITE)) < 0) {
                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 91, "Couldn't open sta
                      stderr_file,
                      errmsg(errno));
        } else {
                if (dup2(i, 2) == -1) {
                        fatal_mksh(NOCATGETS("*** Error: dup2(3, 2) failed: %s")
                                errmsg(errno));
                }
                close(i);
        }
}

/*
 *      dosys_mksh(command, ignore_error, call_make, silent_error, target)
 *
 *      Check if command string contains meta chars and dispatch to
 *      the proper routine for executing one command line.
 *
 *      Return value:
 *                              Indicates if the command execution failed
```



```
 *
 *      Parameters:
 *              command         The command to run
 *              ignore_error    Should we abort when an error is seen?
 *              call_make       Did command reference $(MAKE) ?
 *              silent_error    Should error messages be suppressed for dmake?
 *              target          Target we are building
 *
 *      Global variables used:
 *              do_not_exec_rule Is -n on?
 *              working_on_targets We started processing real targets
 */
Doname
dosys_mksh(register Name command, register Boolean ignore_error, register Boolea
{
        register int            length = command->hash.length;
        register wchar_t        *p;
        register wchar_t        *q;
        register wchar_t        *cmd_string;
        struct stat             before;
        Doname                  result;
        Boolean                 working_on_targets_mksh = true;
        Wstring wcb(command);
        p = wcb.get_string();
        cmd_string = p;

        /* Strip spaces from head of command string */
        while (iswspace(*p)) {
                p++, length--;
        }
        if (*p == (int) nul_char) {
                return build_failed;
        }
        /* If we are faking it we just return */
        if (do_not_exec_rule &&
            working_on_targets_mksh &&
            !call_make &&
            !always_exec) {
                return build_ok;
        }

        /* Copy string to make it OK to write it. */
        q = ALLOC_WC(length + 1);
        (void) wscpy(q, p);
        /* Write the state file iff this command uses make. */
/* XXX - currently does not support recursive make's, $(MAKE)'s
        if (call_make && command_changed) {
                write_state_file(0, false);
        }
        (void) stat(make_state->string_mb, &before);
 */
        /*
         * Run command directly if it contains no shell meta chars,
         * else run it using the shell.
         */
        /* XXX - command->meta *may* not be set correctly */
        if (await(ignore_error,
                  silent_error,
                  target,
                  cmd_string,
                  command->meta ?
                    doshell(q, ignore_error, redirect_out_err, stdout_file, stde
                    doexec(q, ignore_error, redirect_out_err, stdout_file, stder
                  false,
                  NULL,
                  -1)) {
```

```
#ifdef PRINT_EXIT_STATUS
                warning_mksh(NOCATGETS("I'm in dosys_mksh(), and await() returne
#endif

                result = build_ok;
        } else {

#ifdef PRINT_EXIT_STATUS
                warning_mksh(NOCATGETS("I'm in dosys_mksh(), and await() returne
#endif

                result = build_failed;
        }
        retmem(q);

/* XXX - currently does not support recursive make's, $(MAKE)'s
        if ((report_dependencies_level == 0) &&
            call_make) {
                make_state->stat.time = (time_t)file_no_time;
                (void)exists(make_state);
                if (before.st_mtime == make_state->stat.time) {
                        return result;
                }
                makefile_type = reading_statefile;
                if (read_trace_level > 1) {
                        trace_reader = true;
                }
                (void) read_simple_file(make_state,
                                        false,
                                        false,
                                        false,
                                        false,
                                        false,
                                        true);
                trace_reader = false;
        }
 */
        return result;
}

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



```
        register Name           shell = getvar(shell_name);
        register char           *shellname;
        char                    *tmp_mbs_buffer;


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

        /*
         * Only prepend the /usr/bin/nice command to the original command
         * if the nice priority, nice_prio, is NOT zero (0).
         * Nice priorities can be a positive or a negative number.
         */
        if (nice_prio != 0) {
                argv[argv_index++] = NOCATGETS("nice");
                (void) sprintf(nice_prio_buf, NOCATGETS("-%d"), nice_prio);
                argv[argv_index++] = strdup(nice_prio_buf);
        }
        argv[argv_index++] = shellname;
#if defined(linux)
        if(0 == strcmp(shell->string_mb, (char*)NOCATGETS("/bin/sh"))) {
                argv[argv_index++] = (char*)(ignore_error ? NOCATGETS("-c") : NO
        } else {
                argv[argv_index++] = (char*)NOCATGETS("-c");
        }
#else
        argv[argv_index++] = (char*)(ignore_error ? NOCATGETS("-c") : NOCATGETS(
#endif
        if ((length = wslen(command)) >= MAXPATHLEN) {
                tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
                (void) wcstombs(tmp_mbs_buffer, command, (length * MB_LEN_MAX) +
                cmd_argv_index = argv_index;
                argv[argv_index++] = strdup(tmp_mbs_buffer);
                retmem_mb(tmp_mbs_buffer);
        } else {
                WCSTOMBS(mbs_buffer, command);
                cmd_argv_index = argv_index;
#if defined(linux)
                int mbl = strlen(mbs_buffer);
                if(mbl > 2) {
                        if(mbs_buffer[mbl-1] == '\n' && mbs_buffer[mbl-2] == '\\
                                mbs_buffer[mbl] = '\n';
                                mbs_buffer[mbl+1] = 0;
                        }
                }
#endif
                argv[argv_index++] = strdup(mbs_buffer);
        }
        argv[argv_index] = NULL;
        (void) fflush(stdout);
        if ((childPid = fork()) == 0) {
                enable_interrupt((void (*) (int)) SIG_DFL);
                if (redirect_out_err) {
                        redirect_io(stdout_file, stderr_file);
                }
#if 0
                if (filter_stderr) {
                        redirect_stderr();
                }
#endif
```

```
392                 if (nice_prio != 0) {
393                         (void) execve(NOCATGETS("/usr/bin/nice"), argv, environ)
394                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 92, "Could not
395                               errmsg(errno));
396                 } else {
397                         (void) execve(shell->string_mb, argv, environ);
398                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 93, "Could not
399                               shell->string_mb,
400                               errmsg(errno));
401                 }
402         }
403         if (childPid  == -1) {
404                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 94, "fork failed: %s")
405                       errmsg(errno));
406         }
407         retmem_mb(argv[cmd_argv_index]);
408         return childPid;
409 }

411 /*
412  *      exec_vp(name, argv, envp, ignore_error)
413  *
414  *      Like execve, but does path search.
415  *      This starts command when make invokes it directly (without a shell).
416  *
417  *      Return value:
418  *                              Returns false if the exec failed
419  *
420  *      Parameters:
421  *              name            The name of the command to run
422  *              argv            Arguments for the command
423  *              envp            The environment for it
424  *              ignore_error    Should we abort on error?
425  *
426  *      Global variables used:
427  *              shell_name      The Name "SHELL", used to get the path to shell
428  *              vroot_path      The path used by the vroot package
429  */
430 static Boolean
431 exec_vp(register char *name, register char **argv, char **envp, register Boolean
432 {
433         register Name           shell = getvar(shell_name);
434         register char           *shellname;
435         char                    *shargv[4];
436         Name                    tmp_shell;

438         if (IS_EQUAL(shell->string_mb, "")) {
439                 shell = shell_name;
440         }

442         for (int i = 0; i < 5; i++) {
443                 (void) execve_vroot(name,
444                                     argv + 1,
445                                     envp,
446                                     vroot_path,
447                                     VROOT_DEFAULT);
448                 switch (errno) {
449                 case ENOEXEC:
450                 case ENOENT:
451                         /* That failed. Let the shell handle it */
452                         shellname = strrchr(shell->string_mb, (int) slash_char);
453                         if (shellname == NULL) {
454                                 shellname = shell->string_mb;
455                         } else {
456                                 shellname++;
457                         }
```



```
458                         shargv[0] = shellname;
459                         shargv[1] = (char*)(ignore_error ? NOCATGETS("-c") : NOC
460                         shargv[2] = argv[0];
461                         shargv[3] = NULL;
462                         tmp_shell = getvar(shell_name);
463                         if (IS_EQUAL(tmp_shell->string_mb, "")) {
464                                 tmp_shell = shell_name;
465                         }
466                         (void) execve_vroot(tmp_shell->string_mb,
467                                             shargv,
468                                             envp,
469                                             vroot_path,
470                                             VROOT_DEFAULT);
471                         return failed;
472                 case ETXTBSY:
473                         /*
474                          * The program is busy (debugged?).
475                          * Wait and then try again.
476                          */
477                         (void) sleep((unsigned) i);
478                 case EAGAIN:
479                         break;
480                 default:
481                         return failed;
482                 }
483         }
484         return failed;
485 }

487 /*
488  *      doexec(command, ignore_error)
489  *
490  *      Will scan an argument string and split it into words
491  *      thus building an argument list that can be passed to exec_ve()
492  *
493  *      Return value:
494  *                              The pid of the process started here
495  *
496  *      Parameters:
497  *              command         The command to run
498  *              ignore_error    Should we abort on error?
499  *
500  *      Global variables used:
501  *              filter_stderr   If -X is on we redirect stderr
502  */
503 int
504 doexec(register wchar_t *command, register Boolean ignore_error, Boolean redirec
505 {
506         int                     arg_count = 5;
507         char                    **argv;
508         int                     length;
509         char                    nice_prio_buf[MAXPATHLEN];
510         register char           **p;
511         wchar_t                 *q;
512         register wchar_t        *t;
513         char                    *tmp_mbs_buffer;

515         /*
516          * Only prepend the /usr/bin/nice command to the original command
517          * if the nice priority, nice_prio, is NOT zero (0).
518          * Nice priorities can be a positive or a negative number.
519          */
520         if (nice_prio != 0) {
521                 arg_count += 2;
522         }
523         for (t = command; *t != (int) nul_char; t++) {
```

```
524                 if (iswspace(*t)) {
525                         arg_count++;
526                 }
527         }
528         argv = (char **)alloca(arg_count * (sizeof(char *)));
529         /*
530          * Reserve argv[0] for sh in case of exec_vp failure.
531          * Don't worry about prepending /usr/bin/nice command to argv[0].
532          * In fact, doing it may cause the sh command to fail!
533          */
534         p = &argv[1];
535         if ((length = wslen(command)) >= MAXPATHLEN) {
536                 tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
537                 (void) wcstombs(tmp_mbs_buffer, command, (length * MB_LEN_MAX) +
538                 argv[0] = strdup(tmp_mbs_buffer);
539                 retmem_mb(tmp_mbs_buffer);
540         } else {
541                 WCSTOMBS(mbs_buffer, command);
542                 argv[0] = strdup(mbs_buffer);
543         }

545         if (nice_prio != 0) {
546                 *p++ = strdup(NOCATGETS("/usr/bin/nice"));
547                 (void) sprintf(nice_prio_buf, NOCATGETS("-%d"), nice_prio);
548                 *p++ = strdup(nice_prio_buf);
549         }
550         /* Build list of argument words. */
551         for (t = command; *t;) {
552                 if (p >= &argv[arg_count]) {
553                         /* This should never happen, right? */
554                         WCSTOMBS(mbs_buffer, command);
555                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 95, "Command '
556                               mbs_buffer,
557                               arg_count);
558                 }
559                 q = t;
560                 while (!iswspace(*t) && (*t != (int) nul_char)) {
561                         t++;
562                 }
563                 if (*t) {
564                         for (*t++ = (int) nul_char; iswspace(*t); t++);
565                 }
566                 if ((length = wslen(q)) >= MAXPATHLEN) {
567                         tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
568                         (void) wcstombs(tmp_mbs_buffer, q, (length * MB_LEN_MAX)
569                         *p++ = strdup(tmp_mbs_buffer);
570                         retmem_mb(tmp_mbs_buffer);
571                 } else {
572                         WCSTOMBS(mbs_buffer, q);
573                         *p++ = strdup(mbs_buffer);
574                 }
575         }
576         *p = NULL;

578         /* Then exec the command with that argument list. */
579         (void) fflush(stdout);
580         if ((childPid = fork()) == 0) {
581                 enable_interrupt((void (*) (int)) SIG_DFL);
582                 if (redirect_out_err) {
583                         redirect_io(stdout_file, stderr_file);
584                 }
585 #if 0
586                 if (filter_stderr) {
587                         redirect_stderr();
588                 }
589 #endif
```



```
590                 (void) exec_vp(argv[1], argv, environ, ignore_error, vroot_path)
591                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 96, "Cannot load comma
592         }
593         if (childPid  == -1) {
594                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 97, "fork failed: %s")
595                       errmsg(errno));
596         }
597         for (int i = 0; argv[i] != NULL; i++) {
598                 retmem_mb(argv[i]);
599         }
600         return childPid;
601 }

603 /*
604  *      await(ignore_error, silent_error, target, command, running_pid)
605  *
606  *      Wait for one child process and analyzes
607  *      the returned status when the child process terminates.
608  *
609  *      Return value:
610  *                              Returns true if commands ran OK
611  *
612  *      Parameters:
613  *              ignore_error    Should we abort on error?
614  *              silent_error    Should error messages be suppressed for dmake?
615  *              target          The target we are building, for error msgs
616  *              command         The command we ran, for error msgs
617  *              running_pid     The pid of the process we are waiting for
618  *
619  *      Static variables used:
620  *              filter_file     The fd for the filter file
621  *              filter_file_name The name of the filter file
622  *
623  *      Global variables used:
624  *              filter_stderr   Set if -X is on
625  */
626 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
627 Boolean
628 await(register Boolean ignore_error, register Boolean silent_error, Name target,
629 #else
630 Boolean
631 await(register Boolean ignore_error, register Boolean silent_error, Name target,
632 #endif
633 {
634 #ifdef SUN5_0
635         int                     status;
636 #else
637 #ifndef WEXITSTATUS
638 #define WEXITSTATUS(stat)       stat.w_T.w_Retcode
639 #endif
640 #ifndef WTERMSIG
641 #define WTERMSIG(stat)          stat.w_T.w_Termsig
642 #endif
643 #ifndef WCOREDUMP
644 #define WCOREDUMP(stat)         stat.w_T.w_Coredump
645 #endif
646 #if defined (HP_UX) || defined (linux)
647         int                     status;
648 #else
649         union wait              status;
650 #endif
651 #endif
652         char                    *buffer;
653         int                     core_dumped;
654         int                     exit_status;
655 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
```

```
656         Avo_CmdOutput           *make_output_msg;
657 #endif
658         FILE                    *outfp;
659         register pid_t          pid;
660         struct stat             stat_buff;
661         int                     termination_signal;
662         char                    tmp_buf[MAXPATHLEN];
663 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
664         RWCollectable           *xdr_msg;
665 #endif

667         while ((pid = wait(&status)) != running_pid) {
668                 if (pid == -1) {
669                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 98, "wait() fa
670                 }
671         }
672         (void) fflush(stdout);
673         (void) fflush(stderr);

675 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
676         if (status == 0) {

678 #ifdef PRINT_EXIT_STATUS
679                 warning_mksh(NOCATGETS("I'm in await(), and status is 0."));
680 #endif

682                 return succeeded;
683         }

685 #ifdef PRINT_EXIT_STATUS
686         warning_mksh(NOCATGETS("I'm in await(), and status is *NOT* 0."));
687 #endif

689 #else
690         if (status.w_status == 0) {
691                 return succeeded;
692         }
693 #endif

695         exit_status = WEXITSTATUS(status);

697 #ifdef PRINT_EXIT_STATUS
698         warning_mksh(NOCATGETS("I'm in await(), and exit_status is %d."), exit_s
699 #endif

701         termination_signal = WTERMSIG(status);
702         core_dumped = WCOREDUMP(status);

704         /*
705          * If the child returned an error, we now try to print a
706          * nice message about it.
707          */
708         SEND_MTOOL_MSG(
709                 make_output_msg = new Avo_CmdOutput();
710                 (void) sprintf(tmp_buf, "%d", job_msg_id);
711                 make_output_msg->appendOutput(AVO_STRDUP(tmp_buf));
712         );

714         tmp_buf[0] = (int) nul_char;
715         if (!silent_error) {
716                 if (exit_status != 0) {
717                         (void) fprintf(stdout,
718                                 catgets(libmksdmsi18n_catd, 1, 103, "***
719                                 exit_status);
720                         SEND_MTOOL_MSG(
721                                 (void) sprintf(&tmp_buf[strlen(tmp_buf)],
```



```
722                                         catgets(libmksdmsi18n_catd, 1, 10
723                                         exit_status);
724                         );
725                 } else {
726 #if ! defined(SUN5_0) && ! defined(HP_UX) && ! defined(linux)
727                         if (termination_signal > NSIG) {
728 #endif
729                                 (void) fprintf(stdout,
730                                         catgets(libmksdmsi18n_catd, 1, 10
731                                         termination_signal);
732                                 SEND_MTOOL_MSG(
733                                         (void) sprintf(&tmp_buf[strlen(tmp_buf)]
734                                                 catgets(libmksdmsi18n_cat
735                                                 termination_signal);
736                                 );
737 #if ! defined(SUN5_0) && ! defined(HP_UX) && ! defined(linux)
738                         } else {
739                                 (void) fprintf(stdout,
740                                                 "*** %s",
741                                         sys_siglist[termination_signal]);
742                                 SEND_MTOOL_MSG(
743                                         (void) sprintf(&tmp_buf[strlen(tmp_buf)]
744                                                 "*** %s",
745                                                 sys_siglist[termination_s
746                                 );
747                         }
748 #endif
749                         if (core_dumped) {
750                                 (void) fprintf(stdout,
751                                         catgets(libmksdmsi18n_catd, 1, 10
752                                 SEND_MTOOL_MSG(
753                                         (void) sprintf(&tmp_buf[strlen(tmp_buf)]
754                                                 catgets(libmksdmsi18n_cat
755                                 );
756                         }
757                 }
758                 if (ignore_error) {
759                         (void) fprintf(stdout,
760                                 catgets(libmksdmsi18n_catd, 1, 109, " (ig
761                         SEND_MTOOL_MSG(
762                                 (void) sprintf(&tmp_buf[strlen(tmp_buf)],
763                                         catgets(libmksdmsi18n_catd, 1, 11
764                         );
765                 }
766                 (void) fprintf(stdout, "\n");
767                 (void) fflush(stdout);
768                 SEND_MTOOL_MSG(
769                         make_output_msg->appendOutput(AVO_STRDUP(tmp_buf));
770                 );
771         }
772         SEND_MTOOL_MSG(
773                 xdr_msg = (RWCollectable*) make_output_msg;
774                 xdr(xdrs_p, xdr_msg);
775                 delete make_output_msg;
776         );

778 #ifdef PRINT_EXIT_STATUS
779         warning_mksh(NOCATGETS("I'm in await(), returning failed."));
780 #endif

782         return failed;
783 }

785 /*
786  *      sh_command2string(command, destination)
787  *
```

```
 *      Run one sh command and capture the output from it.
 *
 *      Return value:
 *
 *      Parameters:
 *              command         The command to run
 *              destination     Where to deposit the output from the command
 *
 *      Static variables used:
 *
 *      Global variables used:
 */
void
sh_command2string(register String command, register String destination)
{
        register FILE           *fd;
        register int            chr;
        int                     status;
        Boolean                 command_generated_output = false;

        command->text.p = (int) nul_char;
        WCSTOMBS(mbs_buffer, command->buffer.start);
        if ((fd = popen(mbs_buffer, "r")) == NULL) {
                WCSTOMBS(mbs_buffer, command->buffer.start);
                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 111, "Could not run co
                      mbs_buffer);
        }
        while ((chr = getc(fd)) != EOF) {
                if (chr == (int) newline_char) {
                        chr = (int) space_char;
                }
                command_generated_output = true;
                append_char(chr, destination);
        }

        /*
         * We don't want to keep the last LINE_FEED since usually
         * the output of the 'sh:' command is used to evaluate
         * some MACRO. ( /bin/sh and other shell add a line feed
         * to the output so that the prompt appear in the right place.
         * We don't need that
         */
        if (command_generated_output){
                if ( *(destination->text.p-1) == (int) space_char) {
                        * (-- destination->text.p) = '\0';
                }
        } else {
                /*
                 * If the command didn't generate any output,
                 * set the buffer to a null string.
                 */
                *(destination->text.p) = '\0';
        }

        status = pclose(fd);
        if (status != 0) {
                WCSTOMBS(mbs_buffer, command->buffer.start);
                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 112, "The command '%s'
                      mbs_buffer,
                      WEXITSTATUS(status));
        }
}


#endif /* ! codereview */
```

```
**********************************************************
    3190 Wed May 20 11:04:21 2015
new/usr/src/cmd/make/lib/mksh/src/globals.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)globals.cc 1.16 06/12/12
 */

#pragma ident   "@(#)globals.cc 1.16    06/12/12"

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

/*
 * Included files
 */
#include <mksh/globals.h>

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Global variables
 */
char            char_semantics[CHAR_SEMANTICS_ENTRIES];
wchar_t         char_semantics_char[] = {
        ampersand_char,
        asterisk_char,
        at_char,
        backquote_char,
        backslash_char,
        bar_char,
        bracketleft_char,
        bracketright_char,
        colon_char,
        dollar_char,
        doublequote_char,
        equal_char,
        exclam_char,
        greater_char,
        hat_char,
        hyphen_char,
        less_char,
        newline_char,
        numbersign_char,
        parenleft_char,
        parenright_char,
        percent_char,
        plus_char,
        question_char,
        quote_char,
        semicolon_char,
#ifdef SGE_SUPPORT
        space_char,
        tab_char,
#endif
        nul_char
};
Macro_list      cond_macro_list;
Boolean         conditional_macro_used;
Boolean         do_not_exec_rule;               /* '-n' */
Boolean         dollarget_seen;
Boolean         dollarless_flag;
Name            dollarless_value;
Envvar          envvar;
#ifdef lint
char            **environ;
#endif
#ifdef SUN5_0
int             exit_status;
#endif
wchar_t         *file_being_read;
/* Variable gnu_style=true if env. var. SUN_MAKE_COMPAT_MODE=GNU (RFE 4866328) *
Boolean         gnu_style = false;
Name_set        hashtab;
Name            host_arch;
Name            host_mach;
int             line_number;
char            *make_state_lockfile;
Boolean         make_word_mentioned;
Makefile_type   makefile_type = reading_nothing;
char            mbs_buffer[(MAXPATHLEN * MB_LEN_MAX)];
Name            path_name;
Boolean         posix = true;
Name            hat;
Name            query;
Boolean         query_mentioned;
Boolean         reading_environment;
Name            shell_name;
Boolean         svr4 = false;
Name            target_arch;
Name            target_mach;
Boolean         tilde_rule;
Name            virtual_root;
Boolean         vpath_defined;
Name            vpath_name;
wchar_t         wcs_buffer[MAXPATHLEN];
Boolean         working_on_targets;
#if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
```

```
128 Boolean         out_err_same;
129 #endif
130 pid_t           childPid = -1;  // This variable is used for killing child's pro
131                                 // Such as qrsh, running command, etc.

133 /*
134  * timestamps defined in defs.h
135  */
136 const timestruc_t file_no_time          = { -1, 0 };
137 const timestruc_t file_doesnt_exist     = { 0, 0 };
138 const timestruc_t file_is_dir           = { 1, 0 };
139 const timestruc_t file_min_time         = { 2, 0 };
140 const timestruc_t file_max_time         = { INT_MAX, 0 };
141 #endif /* ! codereview */
```

```
**********************************************************
    2357 Wed May 20 11:04:21 2015
new/usr/src/cmd/make/lib/mksh/src/i18n.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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.
 */
/*
 * @(#)i18n.cc 1.3 06/12/12
 */

#pragma ident   "@(#)i18n.cc    1.3     06/12/12"

/*
 *      i18n.cc
 *
 *      Deal with internationalization conversions
 */

/*
 * Included files
 */
#include <mksh/i18n.h>
#include <mksh/misc.h>          /* setup_char_semantics() */
#if defined (linux)
#       include <wctype.h>
#       include <wchar.h>
#       define wschr(x,y) wcschr(x,y)
#endif

/*
 *      get_char_semantics_value(ch)
 *
 *      Return value:
 *              The character semantics of ch.
 *
 *      Parameters:
 *              ch              character we want semantics for.
 *
 */
char
get_char_semantics_value(wchar_t ch)
{
        static Boolean  char_semantics_setup;
```



```
        if (!char_semantics_setup) {
                setup_char_semantics();
                char_semantics_setup = true;
        }
        return char_semantics[get_char_semantics_entry(ch)];
}

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

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

#endif /* ! codereview */
```

```
**********************************************************
    2756 Wed May 20 11:04:21 2015
new/usr/src/cmd/make/lib/mksh/src/libmksh.msg
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
$quote "


$set    1
$
$ 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 1996 Sun Microsystems, Inc. All rights reserved.
$ Use is subject to license terms.
$
$ @(#)libmksh.msg 1.2 06/12/12
$
89      "ulimit() failed: %s"
90      "Couldn't open standard out temp file '%s': %s"
91      "Couldn't open standard error temp file '%s': %s"
92      "Could not load '/usr/bin/nice': %s"
93      "Could not load Shell from '%s': %s"
94      "fork failed: %s"
95      "Command '%s' has more than %d arguments"
96      "Cannot load command '%s': %s"
97      "fork failed: %s"
98      "wait() failed: %s"
99      "Could not open filter file for -X"
100     "Could not stat filter file for -X"
101     "\n**** Error: Directory %s Target %s:\n%s\n"
102     "**** Error: Directory %s Target %s\n"
103     "*** Error code %d"
104     "*** Error code %d"
105     "*** Signal %d"
106     "*** Signal %d"
107     " - core dumped"
108     " - core dumped"
109     " (ignored)"
110     " (ignored)"
111     "Could not run command '%s' for :sh transformation"
112     "The command '%s' returned status '%d'"
113     "Loop detected when expanding macro value '%s'"
114     "'$' at end of string '%s'"
115     "'$' at end of line"
116     "Unmatched '%c' in string '%s'"
117     "Premature EOF"
118     "Unmatched '%c' on line"
119     "Illegal macro reference '%s'"
```



```
120     "= missing from replacement macro reference"
121     "= missing from replacement macro reference"
122     "%% missing from replacement macro reference"
123     "%% missing from replacement macro reference"
124     "Too many %% in pattern"
125     "Conditional macro '%s' referenced on line %d"
126     "Out of memory"
127     "Error %d"
128     "mksh: Fatal error: "
129     "Current working directory %s\n"
131     "mksh: Fatal error in reader: "
133     "Current working directory %s\n"
134     "mksh: Warning: "
135     "Current working directory %s\n"
136     "Internal error. Unknown prop type %d"
137     "'cd %s' failed, and conversion of %s to automounter pathname also faile
138     "'cd %s' and 'cd %s' both failed"
139     "The following command caused the error:\n%s\n"
140     "Error reading '%s': Premature EOF"
141     "Error reading '%s': %s"
#endif /* ! codereview */
```

```
**********************************************************
   39346 Wed May 20 11:04:22 2015
new/usr/src/cmd/make/lib/mksh/src/macro.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)macro.cc 1.22 06/12/12
 */

#pragma ident   "@(#)macro.cc   1.22    06/12/12"

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

/*
 * Included files
 */
#include <mksh/dosys.h>         /* sh_command2string() */
#include <mksh/i18n.h>          /* get_char_semantics_value() */
#include <mksh/macro.h>
#include <mksh/misc.h>          /* retmem() */
#include <mksh/read.h>          /* get_next_block_fn() */
#include <mksdmsi18n/mksdmsi18n.h>      /* libmksdmsi18n_init() */

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

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



```
long env_alloc_num = 0;
long env_alloc_bytes = 0;

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

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

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

/*
 *      expand_value(value, destination, cmd)
 *
 *      Recursively expands all macros in the string value.
 *      destination is where the expanded value should be appended.
 *
 *      Parameters:
 *              value           The value we are expanding
 *              destination     Where to deposit the expansion
 *              cmd             If we are evaluating a command line we
 *                              turn \ quoting off
 *
 *      Global variables used:
 */
void
expand_value(Name value, register String destination, Boolean cmd)
{
        Source_rec              sourceb;
```

```
128         register Source         source = &sourceb;
129         register wchar_t        *source_p = NULL;
130         register wchar_t        *source_end = NULL;
131         wchar_t                 *block_start = NULL;
132         int                     quote_seen = 0;

134         if (value == NULL) {
135                 /*
136                  * Make sure to get a string allocated even if it
137                  * will be empty.
138                  */
139                 MBSTOWCS(wcs_buffer, "");
140                 append_string(wcs_buffer, destination, FIND_LENGTH);
141                 destination->text.end = destination->text.p;
142                 return;
143         }
144         if (!value->dollar) {
145                 /*
146                  * If the value we are expanding does not contain
147                  * any $, we don't have to parse it.
148                  */
149                 APPEND_NAME(value,
150                         destination,
151                         (int) value->hash.length
152                 );
153                 destination->text.end = destination->text.p;
154                 return;
155         }

157         if (value->being_expanded) {
158                 fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 113, "Loop dete
159                         value->string_mb);
160         }
161         value->being_expanded = true;
162         /* Setup the structure we read from */
163         Wstring vals(value);
164         sourceb.string.text.p = sourceb.string.buffer.start = wsdup(vals.get_str
165         sourceb.string.free_after_use = true;
166         sourceb.string.text.end =
167           sourceb.string.buffer.end =
168             sourceb.string.text.p + value->hash.length;
169         sourceb.previous = NULL;
170         sourceb.fd = -1;
171         sourceb.inp_buf =
172           sourceb.inp_buf_ptr =
173             sourceb.inp_buf_end = NULL;
174         sourceb.error_converting = false;
175         /* Lift some pointers from the struct to local register variables */
176         CACHE_SOURCE(0);
177 /* We parse the string in segments */
178 /* We read chars until we find a $, then we append what we have read so far */
179 /* (since last $ processing) to the destination. When we find a $ we call */
180 /* expand_macro() and let it expand that particular $ reference into dest */
181         block_start = source_p;
182         quote_seen = 0;
183         for (; 1; source_p++) {
184                 switch (GET_CHAR()) {
185                 case backslash_char:
186                         /* Quote $ in macro value */
187                         if (!cmd) {
188                                 quote_seen = ~quote_seen;
189                         }
190                         continue;
191                 case dollar_char:
192                         /* Save the plain string we found since */
193                         /* start of string or previous $ */
```



```
194                         if (quote_seen) {
195                                 append_string(block_start,
196                                         destination,
197                                         source_p - block_start - 1);
198                                 block_start = source_p;
199                                 break;
200                         }
201                         append_string(block_start,
202                                 destination,
203                                 source_p - block_start);
204                         source->string.text.p = ++source_p;
205                         UNCACHE_SOURCE();
206                         /* Go expand the macro reference */
207                         expand_macro(source, destination, sourceb.string.buffer.
208                         CACHE_SOURCE(1);
209                         block_start = source_p + 1;
210                         break;
211                 case nul_char:
212                         /* The string ran out. Get some more */
213                         append_string(block_start,
214                                 destination,
215                                 source_p - block_start);
216                         GET_NEXT_BLOCK_NOCHK(source);
217                         if (source == NULL) {
218                                 destination->text.end = destination->text.p;
219                                 value->being_expanded = false;
220                                 return;
221                         }
222                         if (source->error_converting) {
223                                 fatal_reader_mksh(NOCATGETS("Internal error: Inv
224                         }
225                         block_start = source_p;
226                         source_p--;
227                         continue;
228                 }
229                 quote_seen = 0;
230         }
231         retmem(sourceb.string.buffer.start);
232 }

234 /*
235  *      expand_macro(source, destination, current_string, cmd)
236  *
237  *      Should be called with source->string.text.p pointing to
238  *      the first char after the $ that starts a macro reference.
239  *      source->string.text.p is returned pointing to the first char after
240  *      the macro name.
241  *      It will read the macro name, expanding any macros in it,
242  *      and get the value. The value is then expanded.
243  *      destination is a String that is filled in with the expanded macro.
244  *      It may be passed in referencing a buffer to expand the macro into.
245  *      Note that most expansions are done on demand, e.g. right
246  *      before the command is executed and not while the file is
247  *      being parsed.
248  *
249  *      Parameters:
250  *              source          The source block that references the string
251  *                              to expand
252  *              destination     Where to put the result
253  *              current_string  The string we are expanding, for error msg
254  *              cmd             If we are evaluating a command line we
255  *                              turn \ quoting off
256  *
257  *      Global variables used:
258  *              funny           Vector of semantic tags for characters
259  *              is_conditional  Set if a conditional macro is refd
```

```
 *              make_word_mentioned Set if the word "MAKE" is mentioned
 *              makefile_type   We deliver extra msg when reading makefiles
 *              query           The Name "?", compared against
 *              query_mentioned Set if the word "?" is mentioned
 */
void
expand_macro(register Source source, register String destination, wchar_t *curre
{
        static Name             make = (Name)NULL;
        static wchar_t          colon_sh[4];
        static wchar_t          colon_shell[7];
        String_rec              string;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        register wchar_t        *source_p = source->string.text.p;
        register wchar_t        *source_end = source->string.text.end;
        register int            closer = 0;
        wchar_t                 *block_start = (wchar_t *)NULL;
        int                     quote_seen = 0;
        register int            closer_level = 1;
        Name                    name = (Name)NULL;
        wchar_t                 *colon = (wchar_t *)NULL;
        wchar_t                 *percent = (wchar_t *)NULL;
        wchar_t                 *eq = (wchar_t *) NULL;
        Property                macro = NULL;
        wchar_t                 *p = (wchar_t*)NULL;
        String_rec              extracted;
        wchar_t                 extracted_string[MAXPATHLEN];
        wchar_t                 *left_head = NULL;
        wchar_t                 *left_tail = NULL;
        wchar_t                 *right_tail = NULL;
        int                     left_head_len = 0;
        int                     left_tail_len = 0;
        int                     tmp_len = 0;
        wchar_t                 *right_hand[128];
        int                     i = 0;
        enum {
                no_extract,
                dir_extract,
                file_extract
        }                       extraction = no_extract;
        enum {
                no_replace,
                suffix_replace,
                pattern_replace,
                sh_replace
        }                       replacement = no_replace;

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

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

        right_hand[0] = NULL;

        /* First copy the (macro-expanded) macro name into string. */
        INIT_STRING_FROM_STACK(string, buffer);
recheck_first_char:
        /* Check the first char of the macro name to figure out what to do. */
        switch (GET_CHAR()) {
        case nul_char:
                GET_NEXT_BLOCK_NOCHK(source);
                if (source == NULL) {
                        WCSTOMBS(mbs_buffer, current_string);
```



```
                        fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 114, "'
                                        mbs_buffer);
                }
                if (source->error_converting) {
                        fatal_reader_mksh(NOCATGETS("Internal error: Invalid byt
                }
                goto recheck_first_char;
        case parenleft_char:
                /* Multi char name. */
                closer = (int) parenright_char;
                break;
        case braceleft_char:
                /* Multi char name. */
                closer = (int) braceright_char;
                break;
        case newline_char:
                fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 115, "'$' at en
        default:
                /* Single char macro name. Just suck it up */
                append_char(*source_p, &string);
                source->string.text.p = source_p + 1;
                goto get_macro_value;
        }

        /* Handle multi-char macro names */
        block_start = ++source_p;
        quote_seen = 0;
        for (; 1; source_p++) {
                switch (GET_CHAR()) {
                case nul_char:
                        append_string(block_start,
                                      &string,
                                      source_p - block_start);
                        GET_NEXT_BLOCK_NOCHK(source);
                        if (source == NULL) {
                                if (current_string != NULL) {
                                        WCSTOMBS(mbs_buffer, current_string);
                                        fatal_reader_mksh(catgets(libmksdmsi18n_
                                                closer ==
                                                (int) braceright_char ?
                                                (int) braceleft_char :
                                                (int) parenleft_char,
                                                mbs_buffer);
                                } else {
                                        fatal_reader_mksh(catgets(libmksdmsi18n_
                                }
                        }
                        if (source->error_converting) {
                                fatal_reader_mksh(NOCATGETS("Internal error: Inv
                        }
                        block_start = source_p;
                        source_p--;
                        continue;
                case newline_char:
                        fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 118, "U
                                closer == (int) braceright_char ?
                                (int) braceleft_char :
                                (int) parenleft_char);
                case backslash_char:
                        /* Quote dollar in macro value. */
                        if (!cmd) {
                                quote_seen = ~quote_seen;
                        }
                        continue;
                case dollar_char:
                        /*
```

```
392                          * Macro names may reference macros.
393                          * This expands the value of such macros into the
394                          * macro name string.
395                          */
396                         if (quote_seen) {
397                                 append_string(block_start,
398                                               &string,
399                                               source_p - block_start - 1);
400                                 block_start = source_p;
401                                 break;
402                         }
403                         append_string(block_start,
404                                       &string,
405                                       source_p - block_start);
406                         source->string.text.p = ++source_p;
407                         UNCACHE_SOURCE();
408                         expand_macro(source, &string, current_string, cmd);
409                         CACHE_SOURCE(0);
410                         block_start = source_p;
411                         source_p--;
412                         break;
413                 case parenleft_char:
414                         /* Allow nested pairs of () in the macro name. */
415                         if (closer == (int) parenright_char) {
416                                 closer_level++;
417                         }
418                         break;
419                 case braceleft_char:
420                         /* Allow nested pairs of {} in the macro name. */
421                         if (closer == (int) braceright_char) {
422                                 closer_level++;
423                         }
424                         break;
425                 case parenright_char:
426                 case braceright_char:
427                         /*
428                          * End of the name. Save the string in the macro
429                          * name string.
430                          */
431                         if ((*source_p == closer) && (--closer_level <= 0)) {
432                                 source->string.text.p = source_p + 1;
433                                 append_string(block_start,
434                                               &string,
435                                               source_p - block_start);
436                                 goto get_macro_value;
437                         }
438                         break;
439                 }
440                 quote_seen = 0;
441         }
442         /*
443          * We got the macro name. We now inspect it to see if it
444          * specifies any translations of the value.
445          */
446 get_macro_value:
447         name = NULL;
448         /* First check if we have a $(@D) type translation. */
449         if ((get_char_semantics_value(string.buffer.start[0]) &
450              (int) special_macro_sem) &&
451             (string.text.p - string.buffer.start >= 2) &&
452             ((string.buffer.start[1] == 'D') ||
453              (string.buffer.start[1] == 'F'))) {
454                 switch (string.buffer.start[1]) {
455                 case 'D':
456                         extraction = dir_extract;
457                         break;
```



```
458                 case 'F':
459                         extraction = file_extract;
460                         break;
461                 default:
462                         WCSTOMBS(mbs_buffer, string.buffer.start);
463                         fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 119, "I
464                                 mbs_buffer);
465                 }
466                 /* Internalize the macro name using the first char only. */
467                 name = GETNAME(string.buffer.start, 1);
468                 (void) wscpy(string.buffer.start, string.buffer.start + 2);
469         }
470         /* Check for other kinds of translations. */
471         if ((colon = (wchar_t *) wschr(string.buffer.start,
472                                        (int) colon_char)) != NULL) {
473                 /*
474                  * We have a $(FOO:.c=.o) type translation.
475                  * Get the name of the macro proper.
476                  */
477                 if (name == NULL) {
478                         name = GETNAME(string.buffer.start,
479                                        colon - string.buffer.start);
480                 }
481                 /* Pickup all the translations. */
482                 if (IS_WEQUAL(colon, colon_sh) || IS_WEQUAL(colon, colon_shell))
483                         replacement = sh_replace;
484                 } else if ((svr4) ||
485                            ((percent = (wchar_t *) wschr(colon + 1,
486                                                          (int) percent_char)) ==
487                         while (colon != NULL) {
488                                 if ((eq = (wchar_t *) wschr(colon + 1,
489                                                             (int) equal_char)) =
490                                         fatal_reader_mksh(catgets(libmksdmsi18n_
491                                 }
492                                 left_tail_len = eq - colon - 1;
493                                 if(left_tail) {
494                                         retmem(left_tail);
495                                 }
496                                 left_tail = ALLOC_WC(left_tail_len + 1);
497                                 (void) wsncpy(left_tail,
498                                               colon + 1,
499                                               eq - colon - 1);
500                                 left_tail[eq - colon - 1] = (int) nul_char;
501                                 replacement = suffix_replace;
502                                 if ((colon = (wchar_t *) wschr(eq + 1,
503                                                                (int) colon_char)
504                                         tmp_len = colon - eq;
505                                         if(right_tail) {
506                                                 retmem(right_tail);
507                                         }
508                                         right_tail = ALLOC_WC(tmp_len);
509                                         (void) wsncpy(right_tail,
510                                                       eq + 1,
511                                                       colon - eq - 1);
512                                         right_tail[colon - eq - 1] =
513                                           (int) nul_char;
514                                 } else {
515                                         if(right_tail) {
516                                                 retmem(right_tail);
517                                         }
518                                         right_tail = ALLOC_WC(wslen(eq) + 1);
519                                         (void) wscpy(right_tail, eq + 1);
520                                 }
521                         }
522                 } else {
523                         if ((eq = (wchar_t *) wschr(colon + 1,
```

```
524                                             (int) equal_char)) == NULL)
525                         fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1,
526                 }
527                 if ((percent = (wchar_t *) wschr(colon + 1,
528                                                  (int) percent_char)) ==
529                         fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1,
530                 }
531                 if (eq < percent) {
532                         fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1,
533                 }

535                 if (percent > (colon + 1)) {
536                         tmp_len = percent - colon;
537                         if(left_head) {
538                                 retmem(left_head);
539                         }
540                         left_head = ALLOC_WC(tmp_len);
541                         (void) wsncpy(left_head,
542                                       colon + 1,
543                                       percent - colon - 1);
544                         left_head[percent-colon-1] = (int) nul_char;
545                         left_head_len = percent-colon-1;
546                 } else {
547                         left_head = NULL;
548                         left_head_len = 0;
549                 }

551                 if (eq > percent+1) {
552                         tmp_len = eq - percent;
553                         if(left_tail) {
554                                 retmem(left_tail);
555                         }
556                         left_tail = ALLOC_WC(tmp_len);
557                         (void) wsncpy(left_tail,
558                                       percent + 1,
559                                       eq - percent - 1);
560                         left_tail[eq-percent-1] = (int) nul_char;
561                         left_tail_len = eq-percent-1;
562                 } else {
563                         left_tail = NULL;
564                         left_tail_len = 0;
565                 }

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

570                         right_hand[0] = ALLOC_WC(wslen(eq) + 1);
571                         right_hand[1] = NULL;
572                         (void) wscpy(right_hand[0], eq);
573                 } else {
574                         i = 0;
575                         do {
576                                 right_hand[i] = ALLOC_WC(percent-eq+1);
577                                 (void) wsncpy(right_hand[i],
578                                               eq,
579                                               percent - eq);
580                                 right_hand[i][percent-eq] =
581                                   (int) nul_char;
582                                 if (i++ >= VSIZEOF(right_hand)) {
583                                         fatal_mksh(catgets(libmksdmsi18n
584                                 }
585                                 eq = percent + 1;
586                                 if (eq[0] == (int) nul_char) {
587                                         MBSTOWCS(wcs_buffer, "");
588                                         right_hand[i] = (wchar_t *) wsdu
589                                         i++;
```



```
590                                                 break;
591                                         }
592                                 } while ((percent = (wchar_t *) wschr(eq, (int)
593                                 if (eq[0] != (int) nul_char) {
594                                         right_hand[i] = ALLOC_WC(wslen(eq) + 1);
595                                         (void) wscpy(right_hand[i], eq);
596                                         i++;
597                                 }
598                                 right_hand[i] = NULL;
599                         }
600                         replacement = pattern_replace;
601                 }
602         }
603         if (name == NULL) {
604                 /*
605                  * No translations found.
606                  * Use the whole string as the macro name.
607                  */
608                 name = GETNAME(string.buffer.start,
609                                string.text.p - string.buffer.start);
610         }
611         if (string.free_after_use) {
612                 retmem(string.buffer.start);
613         }
614         if (name == make) {
615                 make_word_mentioned = true;
616         }
617         if (name == query) {
618                 query_mentioned = true;
619         }
620         if ((name == host_arch) || (name == target_arch)) {
621                 if (!init_arch_done) {
622                         init_arch_done = true;
623                         init_arch_macros();
624                 }
625         }
626         if ((name == host_mach) || (name == target_mach)) {
627                 if (!init_mach_done) {
628                         init_mach_done = true;
629                         init_mach_macros();
630                 }
631         }
632         /* Get the macro value. */
633         macro = get_prop(name->prop, macro_prop);
634 #ifdef NSE
635         if (nse_watch_vars && nse && macro != NULL) {
636                 if (macro->body.macro.imported) {
637                         nse_shell_var_used= name;
638                 }
639                 if (macro->body.macro.value != NULL){
640                       if (nse_backquotes(macro->body.macro.value->string)) {
641                               nse_backquote_seen= name;
642                       }
643                 }
644         }
645 #endif
646         if ((macro != NULL) && macro->body.macro.is_conditional) {
647                 conditional_macro_used = true;
648                 /*
649                  * Add this conditional macro to the beginning of the
650                  * global list.
651                  */
652                 add_macro_to_global_list(name);
653                 if (makefile_type == reading_makefile) {
654                         warning_mksh(catgets(libmksdmsi18n_catd, 1, 164, "Condit
655                                              name->string_mb, file_being_read, line_n
```

```
656                 }
657         }
658         /* Macro name read and parsed. Expand the value. */
659         if ((macro == NULL) || (macro->body.macro.value == NULL)) {
660                 /* If the value is empty, we just get out of here. */
661                 goto exit;
662         }
663         if (replacement == sh_replace) {
664                 /* If we should do a :sh transform, we expand the command
665                  * and process it.
666                  */
667                 INIT_STRING_FROM_STACK(string, buffer);
668                 /* Expand the value into a local string buffer and run cmd. */
669                 expand_value_with_daemon(name, macro, &string, cmd);
670                 sh_command2string(&string, destination);
671         } else if ((replacement != no_replace) || (extraction != no_extract)) {
672                 /*
673                  * If there were any transforms specified in the macro
674                  * name, we deal with them here.
675                  */
676                 INIT_STRING_FROM_STACK(string, buffer);
677                 /* Expand the value into a local string buffer. */
678                 expand_value_with_daemon(name, macro, &string, cmd);
679                 /* Scan the expanded string. */
680                 p = string.buffer.start;
681                 while (*p != (int) nul_char) {
682                         wchar_t         chr;

684                         /*
685                          * First skip over any white space and append
686                          * that to the destination string.
687                          */
688                         block_start = p;
689                         while ((*p != (int) nul_char) && iswspace(*p)) {
690                                 p++;
691                         }
692                         append_string(block_start,
693                                       destination,
694                                       p - block_start);
695                         /* Then find the end of the next word. */
696                         block_start = p;
697                         while ((*p != (int) nul_char) && !iswspace(*p)) {
698                                 p++;
699                         }
700                         /* If we cant find another word we are done */
701                         if (block_start == p) {
702                                 break;
703                         }
704                         /* Then apply the transforms to the word */
705                         INIT_STRING_FROM_STACK(extracted, extracted_string);
706                         switch (extraction) {
707                         case dir_extract:
708                                 /*
709                                  * $(@D) type transform. Extract the
710                                  * path from the word. Deliver "." if
711                                  * none is found.
712                                  */
713                                 if (p != NULL) {
714                                         chr = *p;
715                                         *p = (int) nul_char;
716                                 }
717                                 eq = (wchar_t *) wsrchr(block_start, (int) slash
718                                 if (p != NULL) {
719                                         *p = chr;
720                                 }
721                                 if ((eq == NULL) || (eq > p)) {
```



```
722                                         MBSTOWCS(wcs_buffer, ".");
723                                         append_string(wcs_buffer, &extracted, 1)
724                                 } else {
725                                         append_string(block_start,
726                                                       &extracted,
727                                                       eq - block_start);
728                                 }
729                                 break;
730                         case file_extract:
731                                 /*
732                                  * $(@F) type transform. Remove the path
733                                  * from the word if any.
734                                  */
735                                 if (p != NULL) {
736                                         chr = *p;
737                                         *p = (int) nul_char;
738                                 }
739                                 eq = (wchar_t *) wsrchr(block_start, (int) slash
740                                 if (p != NULL) {
741                                         *p = chr;
742                                 }
743                                 if ((eq == NULL) || (eq > p)) {
744                                         append_string(block_start,
745                                                       &extracted,
746                                                       p - block_start);
747                                 } else {
748                                         append_string(eq + 1,
749                                                       &extracted,
750                                                       p - eq - 1);
751                                 }
752                                 break;
753                         case no_extract:
754                                 append_string(block_start,
755                                               &extracted,
756                                               p - block_start);
757                                 break;
758                         }
759                         switch (replacement) {
760                         case suffix_replace:
761                                 /*
762                                  * $(FOO:.o=.c) type transform.
763                                  * Maybe replace the tail of the word.
764                                  */
765                                 if (((extracted.text.p -
766                                       extracted.buffer.start) >=
767                                      left_tail_len) &&
768                                     IS_WEQUALN(extracted.text.p - left_tail_len,
769                                                left_tail,
770                                                left_tail_len)) {
771                                         append_string(extracted.buffer.start,
772                                                       destination,
773                                                       (extracted.text.p -
774                                                        extracted.buffer.start)
775                                                       - left_tail_len);
776                                         append_string(right_tail,
777                                                       destination,
778                                                       FIND_LENGTH);
779                                 } else {
780                                         append_string(extracted.buffer.start,
781                                                       destination,
782                                                       FIND_LENGTH);
783                                 }
784                                 break;
785                         case pattern_replace:
786                                 /* $(X:a%b=c%d) type transform. */
787                                 if (((extracted.text.p -
```

```
788                                     extracted.buffer.start) >=
789                                    left_head_len+left_tail_len) &&
790                                   IS_WEQUALN(left_head,
791                                           extracted.buffer.start,
792                                           left_head_len) &&
793                                   IS_WEQUALN(left_tail,
794                                           extracted.text.p - left_tail_len,
795                                           left_tail_len)) {
796                                       i = 0;
797                                       while (right_hand[i] != NULL) {
798                                           append_string(right_hand[i],
799                                                         destination,
800                                                         FIND_LENGTH);
801                                           i++;
802                                           if (right_hand[i] != NULL) {
803                                                   append_string(extracted.
804                                                                 start +
805                                                                 left_head_
806                                                                 destinatio
807                                                                 (extracted
808                                           }
809                                       }
810                                   } else {
811                                           append_string(extracted.buffer.start,
812                                                         destination,
813                                                         FIND_LENGTH);
814                                   }
815                                   break;
816                           case no_replace:
817                                   append_string(extracted.buffer.start,
818                                                 destination,
819                                                 FIND_LENGTH);
820                                   break;
821                           case sh_replace:
822                                   break;
823                               }
824                   }
825                   if (string.free_after_use) {
826                           retmem(string.buffer.start);
827                   }
828           } else {
829                   /*
830                    * This is for the case when the macro name did not
831                    * specify transforms.
832                    */
833                   if (!strncmp(name->string_mb, NOCATGETS("GET"), 3)) {
834                           dollarget_seen = true;
835                   }
836                   dollarless_flag = false;
837                   if (!strncmp(name->string_mb, "<", 1) &&
838                       dollarget_seen) {
839                           dollarless_flag = true;
840                           dollarget_seen = false;
841                   }
842                   expand_value_with_daemon(name, macro, destination, cmd);
843           }
844   exit:
845           if(left_tail) {
846                   retmem(left_tail);
847           }
848           if(right_tail) {
849                   retmem(right_tail);
850           }
851           if(left_head) {
852                   retmem(left_head);
853           }
```



```
854           i = 0;
855           while (right_hand[i] != NULL) {
856                   retmem(right_hand[i]);
857                   i++;
858           }
859           *destination->text.p = (int) nul_char;
860           destination->text.end = destination->text.p;
861   }

863   static void
864   add_macro_to_global_list(Name macro_to_add)
865   {
866           Macro_list      new_macro;
867           Macro_list      macro_on_list;
868           char            *name_on_list = (char*)NULL;
869           char            *name_to_add = macro_to_add->string_mb;
870           char            *value_on_list = (char*)NULL;
871           char            *value_to_add = (char*)NULL;

873           if (macro_to_add->prop->body.macro.value != NULL) {
874                   value_to_add = macro_to_add->prop->body.macro.value->string_mb;
875           } else {
876                   value_to_add = "";
877           }

879           /*
880            * Check if this macro is already on list, if so, do nothing
881            */
882           for (macro_on_list = cond_macro_list;
883                macro_on_list != NULL;
884                macro_on_list = macro_on_list->next) {

886                   name_on_list = macro_on_list->macro_name;
887                   value_on_list = macro_on_list->value;

889                   if (IS_EQUAL(name_on_list, name_to_add)) {
890                           if (IS_EQUAL(value_on_list, value_to_add)) {
891                                   return;
892                           }
893                   }
894           }
895           new_macro = (Macro_list) malloc(sizeof(Macro_list_rec));
896           new_macro->macro_name = strdup(name_to_add);
897           new_macro->value = strdup(value_to_add);
898           new_macro->next = cond_macro_list;
899           cond_macro_list = new_macro;
900   }

902   /*
903    *      init_arch_macros(void)
904    *
905    *      Set the magic macros TARGET_ARCH, HOST_ARCH,
906    *
907    *      Parameters:
908    *
909    *      Global variables used:
910    *                              host_arch   Property for magic macro HOST_ARCH
911    *                              target_arch Property for magic macro TARGET_ARCH
912    *
913    *      Return value:
914    *                              The function does not return a value, but can
915    *                              call fatal() in case of error.
916    */
917   static void
918   init_arch_macros(void)
919   {
```

```
920         String_rec      result_string;
921         wchar_t         wc_buf[STRING_BUFFER_LENGTH];
922         char            mb_buf[STRING_BUFFER_LENGTH];
923         FILE            *pipe;
924         Name            value;
925         int             set_host, set_target;
926 #ifdef NSE
927         Property        macro;
928 #endif
929 #if defined(linux)
930         const char      *mach_command = NOCATGETS("/bin/uname -p");
931 #else
932         const char      *mach_command = NOCATGETS("/bin/mach");
933 #endif

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

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

942                 if ((pipe = popen(mach_command, "r")) == NULL) {
943                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 185, "Execute
944                 }
945                 while (fgets(mb_buf, sizeof(mb_buf), pipe) != NULL) {
946                         MBSTOWCS(wcs_buffer, mb_buf);
947                         append_string(wcs_buffer, &result_string, wslen(wcs_buff
948                 }
949                 if (pclose(pipe) != 0) {
950                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 186, "Execute
951                 }

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

955 #ifdef NSE
956                 macro = setvar_daemon(host_arch, value, false, no_daemon, true,
957                 macro->body.macro.imported= true;
958                 macro = setvar_daemon(target_arch, value, false, no_daemon, true
959                 macro->body.macro.imported= true;
960 #else
961                 if (set_host) {
962                         (void) setvar_daemon(host_arch, value, false, no_daemon,
963                 }
964                 if (set_target) {
965                         (void) setvar_daemon(target_arch, value, false, no_daemo
966                 }
967 #endif
968         }
969 }

971 /*
972  *      init_mach_macros(void)
973  *
974  *      Set the magic macros TARGET_MACH, HOST_MACH,
975  *
976  *      Parameters:
977  *
978  *      Global variables used:
979  *                              host_mach   Property for magic macro HOST_MACH
980  *                              target_mach Property for magic macro TARGET_MACH
981  *
982  *      Return value:
983  *                              The function does not return a value, but can
984  *                              call fatal() in case of error.
985  */
```



```
986 static void
987 init_mach_macros(void)
988 {
989         String_rec      result_string;
990         wchar_t         wc_buf[STRING_BUFFER_LENGTH];
991         char            mb_buf[STRING_BUFFER_LENGTH];
992         FILE            *pipe;
993         Name            value;
994         int             set_host, set_target;
995         const char      *arch_command = NOCATGETS("/bin/arch");

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

1000        if (set_host || set_target) {
1001                INIT_STRING_FROM_STACK(result_string, wc_buf);
1002                append_char((int) hyphen_char, &result_string);

1004                if ((pipe = popen(arch_command, "r")) == NULL) {
1005                        fatal_mksh(catgets(libmksdmsi18n_catd, 1, 183, "Execute
1006                }
1007                while (fgets(mb_buf, sizeof(mb_buf), pipe) != NULL) {
1008                        MBSTOWCS(wcs_buffer, mb_buf);
1009                        append_string(wcs_buffer, &result_string, wslen(wcs_buff
1010                }
1011                if (pclose(pipe) != 0) {
1012                        fatal_mksh(catgets(libmksdmsi18n_catd, 1, 184, "Execute
1013                }

1015                value = GETNAME(result_string.buffer.start, wslen(result_string.

1017                if (set_host) {
1018                        (void) setvar_daemon(host_mach, value, false, no_daemon,
1019                }
1020                if (set_target) {
1021                        (void) setvar_daemon(target_mach, value, false, no_daemo
1022                }
1023        }
1024 }

1026 /*
1027  *      expand_value_with_daemon(name, macro, destination, cmd)
1028  *
1029  *      Checks for daemons and then maybe calls expand_value().
1030  *
1031  *      Parameters:
1032  *              name            Name of the macro  (Added by the NSE)
1033  *              macro           The property block with the value to expand
1034  *              destination     Where the result should be deposited
1035  *              cmd             If we are evaluating a command line we
1036  *                              turn \ quoting off
1037  *
1038  *      Global variables used:
1039  */
1040 static void
1041 #ifdef NSE
1042 expand_value_with_daemon(Name name, register Property macro, register String des
1043 #else
1044 expand_value_with_daemon(Name, register Property macro, register String destinat
1045 #endif
1046 {
1047         register Chain          chain;

1049 #ifdef NSE
1050         if (reading_dependencies) {
1051                 /*
```

```
1052                  * Processing the dependencies themselves
1053                  */
1054                 depvar_dep_macro_used(name);
1055         } else {
1056                 /*
1057                  * Processing the rules for the targets
1058                  * the nse_watch_vars flags chokes off most
1059                  * checks.  it is true only when processing
1060                  * the output from a recursive make run
1061                  * which is all we are interested in here.
1062                  */
1063                 if (nse_watch_vars) {
1064                         depvar_rule_macro_used(name);
1065                 }
1066          }
1067 #endif

1069         switch (macro->body.macro.daemon) {
1070         case no_daemon:
1071                 if (!svr4 && !posix) {
1072                         expand_value(macro->body.macro.value, destination, cmd);
1073                 } else {
1074                         if (dollarless_flag && tilde_rule) {
1075                                 expand_value(dollarless_value, destination, cmd)
1076                                 dollarless_flag = false;
1077                                 tilde_rule = false;
1078                         } else {
1079                                 expand_value(macro->body.macro.value, destinatio
1080                         }
1081                 }
1082                 return;
1083         case chain_daemon:
1084                 /* If this is a $? value we call the daemon to translate the */
1085                 /* list of names to a string */
1086                 for (chain = (Chain) macro->body.macro.value;
1087                      chain != NULL;
1088                      chain = chain->next) {
1089                         APPEND_NAME(chain->name,
1090                                     destination,
1091                                     (int) chain->name->hash.length);
1092                         if (chain->next != NULL) {
1093                                 append_char((int) space_char, destination);
1094                         }
1095                 }
1096                 return;
1097         }
1098 }

1100 /*
1101  * We use a permanent buffer to reset SUNPRO_DEPENDENCIES value.
1102  */
1103 char    *sunpro_dependencies_buf = NULL;
1104 char    *sunpro_dependencies_oldbuf = NULL;
1105 int     sunpro_dependencies_buf_size = 0;

1107 /*
1108  *      setvar_daemon(name, value, append, daemon, strip_trailing_spaces)
1109  *
1110  *      Set a macro value, possibly supplying a daemon to be used
1111  *      when referencing the value.
1112  *
1113  *      Return value:
1114  *                              The property block with the new value
1115  *
1116  *      Parameters:
1117  *              name            Name of the macro to set
```



```
1118  *              value           The value to set
1119  *              append          Should we reset or append to the current value?
1120  *              daemon          Special treatment when reading the value
1121  *              strip_trailing_spaces from the end of value->string
1122  *              debug_level     Indicates how much tracing we should do
1123  *
1124  *      Global variables used:
1125  *              makefile_type   Used to check if we should enforce read only
1126  *              path_name       The Name "PATH", compared against
1127  *              virtual_root    The Name "VIRTUAL_ROOT", compared against
1128  *              vpath_defined   Set if the macro VPATH is set
1129  *              vpath_name      The Name "VPATH", compared against
1130  *              envvar          A list of environment vars with $ in value
1131  */
1132 Property
1133 setvar_daemon(register Name name, register Name value, Boolean append, Daemon da
1134 {
1135         register Property       macro = maybe_append_prop(name, macro_prop);
1136         register Property       macro_apx = get_prop(name->prop, macro_append_pr
1137         int                     length = 0;
1138         String_rec              destination;
1139         wchar_t                 buffer[STRING_BUFFER_LENGTH];
1140         register Chain          chain;
1141         Name                    val;
1142         wchar_t                 *val_string = (wchar_t*)NULL;
1143         Wstring                 wcb;

1145 #ifdef NSE
1146         macro->body.macro.imported = false;
1147 #endif

1149         if ((makefile_type != reading_nothing) &&
1150             macro->body.macro.read_only) {
1151                 return macro;
1152         }
1153         /* Strip spaces from the end of the value */
1154         if (daemon == no_daemon) {
1155                 if(value != NULL) {
1156                         wcb.init(value);
1157                         length = wcb.length();
1158                         val_string = wcb.get_string();
1159                 }
1160                 if ((length > 0) && iswspace(val_string[length-1])) {
1161                         INIT_STRING_FROM_STACK(destination, buffer);
1162                         buffer[0] = 0;
1163                         append_string(val_string, &destination, length);
1164                         if (strip_trailing_spaces) {
1165                                 while ((length > 0) &&
1166                                        iswspace(destination.buffer.start[length-
1167                                         destination.buffer.start[--length] = 0;
1168                                 }
1169                         }
1170                         value = GETNAME(destination.buffer.start, FIND_LENGTH);
1171                 }
1172         }
1173
1174         if(macro_apx != NULL) {
1175                 val = macro_apx->body.macro_appendix.value;
1176         } else {
1177                 val = macro->body.macro.value;
1178         }

1180         if (append) {
1181                 /*
1182                  * If we are appending, we just tack the new value after
1183                  * the old one with a space in between.
```

```
1184                 */
1185                INIT_STRING_FROM_STACK(destination, buffer);
1186                buffer[0] = 0;
1187                if ((macro != NULL) && (val != NULL)) {
1188                        APPEND_NAME(val,
1189                                    &destination,
1190                                    (int) val->hash.length);
1191                        if (value != NULL) {
1192                                wcb.init(value);
1193                                if(wcb.length() > 0) {
1194                                        MBTOWC(wcs_buffer, " ");
1195                                        append_char(wcs_buffer[0], &destination)
1196                                }
1197                        }
1198                }
1199                if (value != NULL) {
1200                        APPEND_NAME(value,
1201                                    &destination,
1202                                    (int) value->hash.length);
1203                }
1204                value = GETNAME(destination.buffer.start, FIND_LENGTH);
1205                wcb.init(value);
1206                if (destination.free_after_use) {
1207                        retmem(destination.buffer.start);
1208                }
1209        }

1211        /* Debugging trace */
1212        if (debug_level > 1) {
1213                if (value != NULL) {
1214                        switch (daemon) {
1215                        case chain_daemon:
1216                                (void) printf("%s =", name->string_mb);
1217                                for (chain = (Chain) value;
1218                                     chain != NULL;
1219                                     chain = chain->next) {
1220                                        (void) printf(" %s", chain->name->string
1221                                }
1222                                (void) printf("\n");
1223                                break;
1224                        case no_daemon:
1225                                (void) printf("%s= %s\n",
1226                                              name->string_mb,
1227                                              value->string_mb);
1228                                break;
1229                        }
1230                } else {
1231                        (void) printf("%s =\n", name->string_mb);
1232                }
1233        }
1234        /* Set the new values in the macro property block */
1235 /**/
1236        if(macro_apx != NULL) {
1237                macro_apx->body.macro_appendix.value = value;
1238                INIT_STRING_FROM_STACK(destination, buffer);
1239                buffer[0] = 0;
1240                if (value != NULL) {
1241                        APPEND_NAME(value,
1242                                    &destination,
1243                                    (int) value->hash.length);
1244                        if (macro_apx->body.macro_appendix.value_to_append != NU
1245                                MBTOWC(wcs_buffer, " ");
1246                                append_char(wcs_buffer[0], &destination);
1247                        }
1248                }
1249                if (macro_apx->body.macro_appendix.value_to_append != NULL) {
```



```
1250                        APPEND_NAME(macro_apx->body.macro_appendix.value_to_appe
1251                                    &destination,
1252                                    (int) macro_apx->body.macro_appendix.value
1253                }
1254                value = GETNAME(destination.buffer.start, FIND_LENGTH);
1255                if (destination.free_after_use) {
1256                        retmem(destination.buffer.start);
1257                }
1258        }
1259 /**/
1260        macro->body.macro.value = value;
1261        macro->body.macro.daemon = daemon;
1262        /*
1263         * If the user changes the VIRTUAL_ROOT, we need to flush
1264         * the vroot package cache.
1265         */
1266        if (name == path_name) {
1267                flush_path_cache();
1268        }
1269        if (name == virtual_root) {
1270                flush_vroot_cache();
1271        }
1272        /* If this sets the VPATH we remember that */
1273        if ((name == vpath_name) &&
1274            (value != NULL) &&
1275            (value->hash.length > 0)) {
1276                vpath_defined = true;
1277        }
1278        /*
1279         * For environment variables we also set the
1280         * environment value each time.
1281         */
1282        if (macro->body.macro.exported) {
1283                static char     *env;

1285 #ifdef DISTRIBUTED
1286                if (!reading_environment && (value != NULL)) {
1287 #else
1288                if (!reading_environment && (value != NULL) && value->dollar) {
1289 #endif
1290                        Envvar  p;

1292                        for (p = envvar; p != NULL; p = p->next) {
1293                                if (p->name == name) {
1294                                        p->value = value;
1295                                        p->already_put = false;
1296                                        goto found_it;
1297                                }
1298                        }
1299                        p = ALLOC(Envvar);
1300                        p->name = name;
1301                        p->value = value;
1302                        p->next = envvar;
1303                        p->env_string = NULL;
1304                        p->already_put = false;
1305                        envvar = p;
1306 found_it:;
1307 #ifdef DISTRIBUTED
1308                }
1309                if (reading_environment || (value == NULL) || !value->dollar) {
1310 #else
1311                } else {
1312 #endif
1313                        length = 2 + strlen(name->string_mb);
1314                        if (value != NULL) {
1315                                length += strlen(value->string_mb);
```

```
1316                        }
1317                        Property env_prop = maybe_append_prop(name, env_mem_prop
1318                        /*
1319                         * We use a permanent buffer to reset SUNPRO_DEPENDENCIE
1320                         */
1321                        if (!strncmp(name->string_mb, NOCATGETS("SUNPRO_DEPENDEN
1322                                if (length >= sunpro_dependencies_buf_size) {
1323                                        sunpro_dependencies_buf_size=length*2;
1324                                        if (sunpro_dependencies_buf_size < 4096)
1325                                                sunpro_dependencies_buf_size = 4
1326                                        if (sunpro_dependencies_buf)
1327                                                sunpro_dependencies_oldbuf = sun
1328                                        sunpro_dependencies_buf=getmem(sunpro_de
1329                                }
1330                                env = sunpro_dependencies_buf;
1331                        } else {
1332                                env = getmem(length);
1333                        }
1334                        env_alloc_num++;
1335                        env_alloc_bytes += length;
1336                        (void) sprintf(env,
1337                                        "%s=%s",
1338                                        name->string_mb,
1339                                        value == NULL ?
1340                                          "" : value->string_mb);
1341                        (void) putenv(env);
1342                        env_prop->body.env_mem.value = env;
1343                        if (sunpro_dependencies_oldbuf) {
1344                                /* Return old buffer */
1345                                retmem_mb(sunpro_dependencies_oldbuf);
1346                                sunpro_dependencies_oldbuf = NULL;
1347                        }
1348                }
1349        }
1350        if (name == target_arch) {
1351                Name            ha = getvar(host_arch);
1352                Name            ta = getvar(target_arch);
1353                Name            vr = getvar(virtual_root);
1354                int             length;
1355                wchar_t         *new_value;
1356                wchar_t         *old_vr;
1357                Boolean         new_value_allocated = false;

1359                Wstring         ha_str(ha);
1360                Wstring         ta_str(ta);
1361                Wstring         vr_str(vr);

1363                wchar_t * wcb_ha = ha_str.get_string();
1364                wchar_t * wcb_ta = ta_str.get_string();
1365                wchar_t * wcb_vr = vr_str.get_string();

1367                length = 32 +
1368                  wslen(wcb_ha) +
1369                    wslen(wcb_ta) +
1370                      wslen(wcb_vr);
1371                old_vr = wcb_vr;
1372                MBSTOWCS(wcs_buffer, NOCATGETS("/usr/arch/"));
1373                if (IS_WEQUALN(old_vr,
1374                               wcs_buffer,
1375                               wslen(wcs_buffer))) {
1376                        old_vr = (wchar_t *) wschr(old_vr, (int) colon_char) + 1
1377                }
1378                if ( (ha == ta) || (wslen(wcb_ta) == 0) ) {
1379                        new_value = old_vr;
1380                } else {
1381                        new_value = ALLOC_WC(length);
```



```
1382                        new_value_allocated = true;
1383                        WCSTOMBS(mbs_buffer, old_vr);
1384 #if !defined(linux)
1385                        (void) wsprintf(new_value,
1386                                        NOCATGETS("/usr/arch/%s/%s:%s"),
1387                                        ha->string_mb + 1,
1388                                        ta->string_mb + 1,
1389                                        mbs_buffer);
1390 #else
1391                        char * mbs_new_value = (char *)getmem(length);
1392                        (void) sprintf(mbs_new_value,
1393                                        NOCATGETS("/usr/arch/%s/%s:%s"),
1394                                        ha->string_mb + 1,
1395                                        ta->string_mb + 1,
1396                                        mbs_buffer);
1397                        MBSTOWCS(new_value, mbs_new_value);
1398                        retmem_mb(mbs_new_value);
1399 #endif
1400                }
1401                if (new_value[0] != 0) {
1402                        (void) setvar_daemon(virtual_root,
1403                                             GETNAME(new_value, FIND_LENGTH),
1404                                             false,
1405                                             no_daemon,
1406                                             true,
1407                                             debug_level);
1408                }
1409                if (new_value_allocated) {
1410                        retmem(new_value);
1411                }
1412        }
1413        return macro;
1414 }
1415 #endif /* ! codereview */
```

```
**********************************************************
   25971 Wed May 20 11:04:22 2015
new/usr/src/cmd/make/lib/mksh/src/misc.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)misc.cc 1.31 06/12/12
 */

#pragma ident   "@(#)misc.cc    1.31    06/12/12"

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

/*
 * Included files
 */
#include <bsd/bsd.h>            /* bsd_signal() */
#include <mksh/i18n.h>          /* get_char_semantics_value() */
#include <mksh/misc.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <stdarg.h>             /* va_list, va_start(), va_end() */
#include <stdlib.h>             /* mbstowcs() */
#include <sys/signal.h>         /* SIG_DFL */
#include <sys/wait.h>           /* wait() */

#ifdef SUN5_0
#include <string.h>             /* strerror() */
#endif

#if defined (HP_UX) || defined (linux)
#include <unistd.h>
```



```
#endif

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
#ifdef SUN5_0
extern "C" void         (*sigivalue)(int) = SIG_DFL;
extern "C" void         (*sigqvalue)(int) = SIG_DFL;
extern "C" void         (*sigtvalue)(int) = SIG_DFL;
extern "C" void         (*sighvalue)(int) = SIG_DFL;
#else
static  void            (*sigivalue)(int) = (void (*) (int)) SIG_DFL;
static  void            (*sigqvalue)(int) = (void (*) (int)) SIG_DFL;
static  void            (*sigtvalue)(int) = (void (*) (int)) SIG_DFL;
static  void            (*sighvalue)(int) = (void (*) (int)) SIG_DFL;
#endif

long    getname_bytes_count = 0;
long    getname_names_count = 0;
long    getname_struct_count = 0;

long    freename_bytes_count = 0;
long    freename_names_count = 0;
long    freename_struct_count = 0;

long    expandstring_count = 0;
long    getwstring_count = 0;

/*
 * File table of contents
 */
static void     expand_string(register String string, register int length);

#define FATAL_ERROR_MSG_SIZE 200

/*
 *      getmem(size)
 *
 *      malloc() version that checks the returned value.
 *
 *      Return value:
 *                              The memory chunk we allocated
 *
 *      Parameters:
 *              size            The size of the chunk we need
 *
 *      Global variables used:
 */
char *
getmem(register int size)
{
        register char           *result = (char *) malloc((unsigned) size);
        if (result == NULL) {
                char buf[FATAL_ERROR_MSG_SIZE];
                sprintf(buf, NOCATGETS("*** Error: malloc(%d) failed: %s\n"), si
                strcat(buf, catgets(libmksdmsi18n_catd, 1, 126, "mksh: Fatal err
                fputs(buf, stderr);
#ifdef SUN5_0
```

```
128                 exit_status = 1;
129 #endif
130                 exit(1);
131         }
132         return result;
133 }

135 /*
136  *      retmem(p)
137  *
138  *      Cover funtion for free() to make it possible to insert advises.
139  *
140  *      Parameters:
141  *              p               The memory block to free
142  *
143  *      Global variables used:
144  */
145 void
146 retmem(wchar_t *p)
147 {
148         (void) free((char *) p);
149 }

151 void
152 retmem_mb(caddr_t p)
153 {
154         (void) free(p);
155 }

157 /*
158  *      getname_fn(name, len, dont_enter)
159  *
160  *      Hash a name string to the corresponding nameblock.
161  *
162  *      Return value:
163  *                              The Name block for the string
164  *
165  *      Parameters:
166  *              name            The string we want to internalize
167  *              len             The length of that string
168  *              dont_enter      Don't enter the name if it does not exist
169  *
170  *      Global variables used:
171  *              funny           The vector of semantic tags for characters
172  *              hashtab         The hashtable used for the nametable
173  */
174 Name
175 getname_fn(wchar_t *name, register int len, register Boolean dont_enter, registe
176 {
177         register int            length;
178         register wchar_t        *cap = name;
179         register Name           np;
180         static Name_rec         empty_Name;
181         char                    *tmp_mbs_buffer = NULL;
182         char                    *mbs_name = mbs_buffer;

184         /*
185          * First figure out how long the string is.
186          * If the len argument is -1 we count the chars here.
187          */
188         if (len == FIND_LENGTH) {
189                 length = wslen(name);
190         } else {
191                 length = len;
192         }
```



```
194         Wstring ws;
195         ws.init(name, length);
196         if (length >= MAXPATHLEN) {
197                 mbs_name = tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
198         }
199         (void) wcstombs(mbs_name, ws.get_string(), (length * MB_LEN_MAX) + 1);

201         /* Look for the string */
202         if (dont_enter || (foundp != 0)) {
203                 np = hashtab.lookup(mbs_name);
204                 if (foundp != 0) {
205                         *foundp = (np != 0) ? true : false;
206                 }
207                 if ((np != 0) || dont_enter) {
208                         if(tmp_mbs_buffer != NULL) {
209                                 retmem_mb(tmp_mbs_buffer);
210                         }
211                         return np;
212                 } else {
213                         np = ALLOC(Name);
214                 }
215         } else {
216                 Boolean found;
217                 np = hashtab.insert(mbs_name, found);
218                 if (found) {
219                         if(tmp_mbs_buffer != NULL) {
220                                 retmem_mb(tmp_mbs_buffer);
221                         }
222                         return np;
223                 }
224         }
225         getname_struct_count += sizeof(struct _Name);
226         *np = empty_Name;

228         np->string_mb = strdup(mbs_name);
229         if(tmp_mbs_buffer != NULL) {
230                 retmem_mb(tmp_mbs_buffer);
231                 mbs_name = tmp_mbs_buffer = NULL;
232         }
233         getname_bytes_count += strlen(np->string_mb) + 1;
234         /* Fill in the new Name */
235         np->stat.time = file_no_time;
236         np->hash.length = length;
237         /* Scan the namestring to classify it */
238         for (cap = name, len = 0; --length >= 0;) {
239                 len |= get_char_semantics_value(*cap++);
240         }
241         np->dollar = BOOLEAN((len & (int) dollar_sem) != 0);
242         np->meta = BOOLEAN((len & (int) meta_sem) != 0);
243         np->percent = BOOLEAN((len & (int) percent_sem) != 0);
244         np->wildcard = BOOLEAN((len & (int) wildcard_sem) != 0);
245         np->colon = BOOLEAN((len & (int) colon_sem) != 0);
246         np->parenleft = BOOLEAN((len & (int) parenleft_sem) != 0);
247         getname_names_count++;
248         return np;
249 }

251 void
252 store_name(Name name)
253 {
254         hashtab.insert(name);
255 }

257 void
258 free_name(Name name)
259 {
```

```
260         freename_names_count++;
261         freename_struct_count += sizeof(struct _Name);
262         freename_bytes_count += strlen(name->string_mb) + 1;
263         retmem_mb(name->string_mb);
264         for (Property next, p = name->prop; p != NULL; p = next) {
265                 next = p->next;
266                 free(p);
267         }
268         free(name);
269 }

271 /*
272  *      enable_interrupt(handler)
273  *
274  *      This routine sets a new interrupt handler for the signals make
275  *      wants to deal with.
276  *
277  *      Parameters:
278  *              handler         The function installed as interrupt handler
279  *
280  *      Static variables used:
281  *              sigivalue       The original signal handler
282  *              sigqvalue       The original signal handler
283  *              sigtvalue       The original signal handler
284  *              sighvalue       The original signal handler
285  */
286 void
287 enable_interrupt(register void (*handler) (int))
288 {
289 #ifdef SUN5_0
290         if (sigivalue != SIG_IGN) {
291 #else
292         if (sigivalue != (void (*) (int)) SIG_IGN) {
293 #endif
294                 (void) bsd_signal(SIGINT, (SIG_PF) handler);
295         }
296 #ifdef SUN5_0
297         if (sigqvalue != SIG_IGN) {
298 #else
299         if (sigqvalue != (void (*) (int)) SIG_IGN) {
300 #endif
301                 (void) bsd_signal(SIGQUIT, (SIG_PF) handler);
302         }
303 #ifdef SUN5_0
304         if (sigtvalue != SIG_IGN) {
305 #else
306         if (sigtvalue != (void (*) (int)) SIG_IGN) {
307 #endif
308                 (void) bsd_signal(SIGTERM, (SIG_PF) handler);
309         }
310 #ifdef SUN5_0
311         if (sighvalue != SIG_IGN) {
312 #else
313         if (sighvalue != (void (*) (int)) SIG_IGN) {
314 #endif
315                 (void) bsd_signal(SIGHUP, (SIG_PF) handler);
316         }
317 }

319 /*
320  *      setup_char_semantics()
321  *
322  *      Load the vector char_semantics[] with lexical markers
323  *
324  *      Parameters:
325  *
```



```
326  *      Global variables used:
327  *              char_semantics  The vector of character semantics that we set
328  */
329 void
330 setup_char_semantics(void)
331 {
332         char            *s;
333         wchar_t         wc_buffer[1];
334         int             entry;

336         if (svr4) {
337                 s = "@-";
338         } else {
339                 s = "=@-?!+";
340         }
341         for (s; MBTOWC(wc_buffer, s); s++) {
342                 entry = get_char_semantics_entry(*wc_buffer);
343                 char_semantics[entry] |= (int) command_prefix_sem;
344         }
345         char_semantics[dollar_char_entry] |= (int) dollar_sem;
346         for (s = "#|=^();&<>*?[]:$`'\"\\\n"; MBTOWC(wc_buffer, s); s++) {
347                 entry = get_char_semantics_entry(*wc_buffer);
348                 char_semantics[entry] |= (int) meta_sem;
349         }
350         char_semantics[percent_char_entry] |= (int) percent_sem;
351         for (s = "@*<%?^"; MBTOWC(wc_buffer, s); s++) {
352                 entry = get_char_semantics_entry(*wc_buffer);
353                 char_semantics[entry] |= (int) special_macro_sem;
354         }
355         for (s = "?[*"; MBTOWC(wc_buffer, s); s++) {
356                 entry = get_char_semantics_entry(*wc_buffer);
357                 char_semantics[entry] |= (int) wildcard_sem;
358         }
359         char_semantics[colon_char_entry] |= (int) colon_sem;
360         char_semantics[parenleft_char_entry] |= (int) parenleft_sem;
361 }

363 /*
364  *      errmsg(errnum)
365  *
366  *      Return the error message for a system call error
367  *
368  *      Return value:
369  *                              An error message string
370  *
371  *      Parameters:
372  *              errnum          The number of the error we want to describe
373  *
374  *      Global variables used:
375  *              sys_errlist     A vector of error messages
376  *              sys_nerr        The size of sys_errlist
377  */
378 char *
379 errmsg(int errnum)
380 {
381 #ifdef linux
382         return strerror(errnum);
383 #else // linux

385         extern int              sys_nerr;
386 #ifdef SUN4_x
387         extern char             *sys_errlist[];
388 #endif
389         char                    *errbuf;

391         if ((errnum < 0) || (errnum > sys_nerr)) {
```

```
392                 errbuf = getmem(6+1+11+1);
393                 (void) sprintf(errbuf, catgets(libmksdmsi18n_catd, 1, 127, "Erro
394                 return errbuf;
395         } else {
396 #ifdef SUN4_x
397                 return(sys_errlist[errnum]);
398 #endif
399 #ifdef SUN5_0
400                 return strerror(errnum);
401 #endif

403         }
404 #endif // linux
405 }

407 static char static_buf[MAXPATHLEN*3];

409 /*
410  *      fatal_mksh(format, args...)
411  *
412  *      Print a message and die
413  *
414  *      Parameters:
415  *              format          printf type format string
416  *              args            Arguments to match the format
417  */
418 /*VARARGS*/
419 void
420 fatal_mksh(char * message, ...)
421 {
422         va_list args;
423         char    *buf = static_buf;
424         char    *mksh_fat_err = catgets(libmksdmsi18n_catd, 1, 128, "mksh: Fatal
425         char    *cur_wrk_dir = catgets(libmksdmsi18n_catd, 1, 129, "Current work
426         int     mksh_fat_err_len = strlen(mksh_fat_err);

428         va_start(args, message);
429         (void) fflush(stdout);
430         (void) strcpy(buf, mksh_fat_err);
431         size_t buf_len = vsnprintf(static_buf + mksh_fat_err_len,
432                                    sizeof(static_buf) - mksh_fat_err_len,
433                                    message, args)
434                         + mksh_fat_err_len
435                         + strlen(cur_wrk_dir)
436                         + strlen(get_current_path_mksh())
437                         + 3; // "\n\n"
438         va_end(args);
439         if (buf_len >= sizeof(static_buf)) {
440                 buf = getmem(buf_len);
441                 (void) strcpy(buf, mksh_fat_err);
442                 va_start(args, message);
443                 (void) vsprintf(buf + mksh_fat_err_len, message, args);
444                 va_end(args);
445         }
446         (void) strcat(buf, "\n");
447 /*
448         if (report_pwd) {
449  */
450         if (1) {
451                 (void) strcat(buf, cur_wrk_dir);
452                 (void) strcat(buf, get_current_path_mksh());
453                 (void) strcat(buf, "\n");
454         }
455         (void) fputs(buf, stderr);
456         (void) fflush(stderr);
457         if (buf != static_buf) {
```



```
458                 retmem_mb(buf);
459         }
460 #ifdef SUN5_0
461         exit_status = 1;
462 #endif
463         exit(1);
464 }

466 /*
467  *      fatal_reader_mksh(format, args...)
468  *
469  *      Parameters:
470  *              format          printf style format string
471  *              args            arguments to match the format
472  */
473 /*VARARGS*/
474 void
475 fatal_reader_mksh(char * pattern, ...)
476 {
477         va_list args;
478         char    message[1000];

480         va_start(args, pattern);
481 /*
482         if (file_being_read != NULL) {
483                 WCSTOMBS(mbs_buffer, file_being_read);
484                 if (line_number != 0) {
485                         (void) sprintf(message,
486                                        catgets(libmksdmsi18n_catd, 1, 130, "%s,
487                                        mbs_buffer,
488                                        line_number,
489                                        pattern);
490                 } else {
491                         (void) sprintf(message,
492                                        "%s: %s",
493                                        mbs_buffer,
494                                        pattern);
495                 }
496                 pattern = message;
497         }
498  */

500         (void) fflush(stdout);
501         (void) fprintf(stderr, catgets(libmksdmsi18n_catd, 1, 131, "mksh: Fatal
502         (void) vfprintf(stderr, pattern, args);
503         (void) fprintf(stderr, "\n");
504         va_end(args);

506 /*
507         if (temp_file_name != NULL) {
508                 (void) fprintf(stderr,
509                                catgets(libmksdmsi18n_catd, 1, 132, "mksh: Temp-f
510                                temp_file_name->string_mb);
511                 temp_file_name = NULL;
512         }
513  */

515 /*
516         if (report_pwd) {
517  */
518         if (1) {
519                 (void) fprintf(stderr,
520                                catgets(libmksdmsi18n_catd, 1, 133, "Current work
521                                get_current_path_mksh());
522         }
523         (void) fflush(stderr);
```

```
524 #ifdef SUN5_0
525         exit_status = 1;
526 #endif
527         exit(1);
528 }

530 /*
531  *      warning_mksh(format, args...)
532  *
533  *      Print a message and continue.
534  *
535  *      Parameters:
536  *              format          printf type format string
537  *              args            Arguments to match the format
538  */
539 /*VARARGS*/
540 void
541 warning_mksh(char * message, ...)
542 {
543         va_list args;

545         va_start(args, message);
546         (void) fflush(stdout);
547         (void) fprintf(stderr, catgets(libmksdmsi18n_catd, 1, 134, "mksh: Warnin
548         (void) vfprintf(stderr, message, args);
549         (void) fprintf(stderr, "\n");
550         va_end(args);
551 /*
552         if (report_pwd) {
553  */
554         if (1) {
555                 (void) fprintf(stderr,
556                                 catgets(libmksdmsi18n_catd, 1, 135, "Current work
557                                 get_current_path_mksh());
558         }
559         (void) fflush(stderr);
560 }

562 /*
563  *      get_current_path_mksh()
564  *
565  *      Stuff current_path with the current path if it isnt there already.
566  *
567  *      Parameters:
568  *
569  *      Global variables used:
570  */
571 char *
572 get_current_path_mksh(void)
573 {
574         char                    pwd[(MAXPATHLEN * MB_LEN_MAX)];
575         static char             *current_path;

577         if (current_path == NULL) {
578 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
579                 getcwd(pwd, sizeof(pwd));
580 #else
581                 (void) getwd(pwd);
582 #endif
583                 if (pwd[0] == (int) nul_char) {
584                         pwd[0] = (int) slash_char;
585                         pwd[1] = (int) nul_char;
586                 }
587                 current_path = strdup(pwd);
588         }
589         return current_path;
```



```
590 }

592 /*
593  *      append_prop(target, type)
594  *
595  *      Create a new property and append it to the property list of a Name.
596  *
597  *      Return value:
598  *                              A new property block for the target
599  *
600  *      Parameters:
601  *              target          The target that wants a new property
602  *              type            The type of property being requested
603  *
604  *      Global variables used:
605  */
606 Property
607 append_prop(register Name target, register Property_id type)
608 {
609         register Property       *insert = &target->prop;
610         register Property       prop = *insert;
611         register int            size;

613         switch (type) {
614         case conditional_prop:
615                 size = sizeof (struct Conditional);
616                 break;
617         case line_prop:
618                 size = sizeof (struct Line);
619                 break;
620         case macro_prop:
621                 size = sizeof (struct _Macro);
622                 break;
623         case makefile_prop:
624                 size = sizeof (struct Makefile);
625                 break;
626         case member_prop:
627                 size = sizeof (struct Member);
628                 break;
629         case recursive_prop:
630                 size = sizeof (struct Recursive);
631                 break;
632         case sccs_prop:
633                 size = sizeof (struct Sccs);
634                 break;
635         case suffix_prop:
636                 size = sizeof (struct Suffix);
637                 break;
638         case target_prop:
639                 size = sizeof (struct Target);
640                 break;
641         case time_prop:
642                 size = sizeof (struct STime);
643                 break;
644         case vpath_alias_prop:
645                 size = sizeof (struct Vpath_alias);
646                 break;
647         case long_member_name_prop:
648                 size = sizeof (struct Long_member_name);
649                 break;
650         case macro_append_prop:
651                 size = sizeof (struct _Macro_appendix);
652                 break;
653         case env_mem_prop:
654                 size = sizeof (struct _Env_mem);
655                 break;
```

```
656         default:
657                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 136, "Internal error.
658         }
659         for (; prop != NULL; insert = &prop->next, prop = *insert);
660         size += PROPERTY_HEAD_SIZE;
661         *insert = prop = (Property) getmem(size);
662         memset((char *) prop, 0, size);
663         prop->type = type;
664         prop->next = NULL;
665         return prop;
666 }

668 /*
669  *      maybe_append_prop(target, type)
670  *
671  *      Append a property to the Name if none of this type exists
672  *      else return the one already there
673  *
674  *      Return value:
675  *                              A property of the requested type for the target
676  *
677  *      Parameters:
678  *              target          The target that wants a new property
679  *              type            The type of property being requested
680  *
681  *      Global variables used:
682  */
683 Property
684 maybe_append_prop(register Name target, register Property_id type)
685 {
686         register Property       prop;

688         if ((prop = get_prop(target->prop, type)) != NULL) {
689                 return prop;
690         }
691         return append_prop(target, type);
692 }

694 /*
695  *      get_prop(start, type)
696  *
697  *      Scan the property list of a Name to find the next property
698  *      of a given type.
699  *
700  *      Return value:
701  *                              The first property of the type, if any left
702  *
703  *      Parameters:
704  *              start           The first property block to check for type
705  *              type            The type of property block we need
706  *
707  *      Global variables used:
708  */
709 Property
710 get_prop(register Property start, register Property_id type)
711 {
712         for (; start != NULL; start = start->next) {
713                 if (start->type == type) {
714                         return start;
715                 }
716         }
717         return NULL;
718 }

720 /*
721  *      append_string(from, to, length)
```



```
722  *
723  *      Append a C string to a make string expanding it if nessecary
724  *
725  *      Parameters:
726  *              from            The source (C style) string
727  *              to              The destination (make style) string
728  *              length          The length of the from string
729  *
730  *      Global variables used:
731  */
732 void
733 append_string(register wchar_t *from, register String to, register int length)
734 {
735         if (length == FIND_LENGTH) {
736                 length = wslen(from);
737         }
738         if (to->buffer.start == NULL) {
739                 expand_string(to, 32 + length);
740         }
741         if (to->buffer.end - to->text.p <= length) {
742                 expand_string(to,
743                                 (to->buffer.end - to->buffer.start) * 2 +
744                                 length);
745         }
746         if (length > 0) {
747                 (void) wsncpy(to->text.p, from, length);
748                 to->text.p += length;
749         }
750         *(to->text.p) = (int) nul_char;
751 }

753 wchar_t * get_wstring(char *from) {
754         if(from == NULL) {
755                 return NULL;
756         }
757         getwstring_count++;
758         wchar_t * wcbuf = ALLOC_WC(strlen(from) + 1);
759         mbstowcs(wcbuf, from, strlen(from)+1);
760         return wcbuf;
761 }

763 void
764 append_string(register char *from, register String to, register int length)
765 {
766         if (length == FIND_LENGTH) {
767                 length = strlen(from);
768         }
769         if (to->buffer.start == NULL) {
770                 expand_string(to, 32 + length);
771         }
772         if (to->buffer.end - to->text.p <= length) {
773                 expand_string(to,
774                                 (to->buffer.end - to->buffer.start) * 2 +
775                                 length);
776         }
777         if (length > 0) {
778                 (void) mbstowcs(to->text.p, from, length);
779                 to->text.p += length;
780         }
781         *(to->text.p) = (int) nul_char;
782 }

784 /*
785  *      expand_string(string, length)
786  *
787  *      Allocate more memory for strings that run out of space.
```

```
 *
 *      Parameters:
 *              string          The make style string we want to expand
 *              length          The new length we need
 *
 *      Global variables used:
 */
static void
expand_string(register String string, register int length)
{
        register wchar_t        *p;

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

/*
 *      append_char(from, to)
 *
 *      Append one char to a make string expanding it if nessecary
 *
 *      Parameters:
 *              from            Single character to append to string
 *              to              The destination (make style) string
 *
 *      Global variables used:
 */
void
append_char(wchar_t from, register String to)
{
        if (to->buffer.start == NULL) {
                expand_string(to, 32);
        }
        if (to->buffer.end - to->text.p <= 2) {
                expand_string(to, to->buffer.end - to->buffer.start + 32);
        }
        *(to->text.p)++ = from;
```



```
        *(to->text.p) = (int) nul_char;
}

/*
 *      handle_interrupt_mksh()
 *
 *      This is where C-C traps are caught.
 */
void
handle_interrupt_mksh(int)
{
        (void) fflush(stdout);
        /* Make sure the processes running under us terminate first. */
        if (childPid > 0) {
                kill(childPid, SIGTERM);
                childPid = -1;
        }
#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
        while (wait((int *) NULL) != -1);
#if defined(SUN5_0)
        exit_status = 2;
#endif
#else
        while (wait((union wait *) NULL) != -1);
#endif
        exit(2);
}

/*
 *      setup_interrupt()
 *
 *      This routine saves the original interrupt handler pointers
 *
 *      Parameters:
 *
 *      Static variables used:
 *              sigivalue       The original signal handler
 *              sigqvalue       The original signal handler
 *              sigtvalue       The original signal handler
 *              sighvalue       The original signal handler
 */
void
setup_interrupt(register void (*handler) (int))
{
#ifdef SUN5_0
        sigivalue = bsd_signal(SIGINT, SIG_IGN);
        sigqvalue = bsd_signal(SIGQUIT, SIG_IGN);
        sigtvalue = bsd_signal(SIGTERM, SIG_IGN);
        sighvalue = bsd_signal(SIGHUP, SIG_IGN);
#else
        sigivalue = (void (*) (int)) bsd_signal(SIGINT, SIG_IGN);
        sigqvalue = (void (*) (int)) bsd_signal(SIGQUIT, SIG_IGN);
        sigtvalue = (void (*) (int)) bsd_signal(SIGTERM, SIG_IGN);
        sighvalue = (void (*) (int)) bsd_signal(SIGHUP, SIG_IGN);
#endif
        enable_interrupt(handler);
}


void
mbstowcs_with_check(wchar_t *pwcs, const char *s, size_t n)
{
        if(mbstowcs(pwcs, s, n) == -1) {
                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 143, "The string `%s'
        }
}
```

```
Wstring::Wstring()
{
        INIT_STRING_FROM_STACK(string, string_buf);
}

Wstring::Wstring(struct _Name * name)
{
        INIT_STRING_FROM_STACK(string, string_buf);
        append_string(name->string_mb, &string, name->hash.length);
}

Wstring::~Wstring()
{
        if(string.free_after_use) {
                retmem(string.buffer.start);
        }
}

void
Wstring::init(struct _Name * name)
{
        if(string.free_after_use) {
                retmem(string.buffer.start);
        }
        INIT_STRING_FROM_STACK(string, string_buf);
        append_string(name->string_mb, &string, name->hash.length);
}

void
Wstring::init(wchar_t * name, unsigned length)
{
        INIT_STRING_FROM_STACK(string, string_buf);
        append_string(name, &string, length);
        string.buffer.start[length] = 0;
}

Boolean
Wstring::equaln(wchar_t * str, unsigned length)
{
        return (Boolean)IS_WEQUALN(string.buffer.start, str, length);
}

Boolean
Wstring::equaln(Wstring * str, unsigned length)
{
        return (Boolean)IS_WEQUALN(string.buffer.start, str->string.buffer.start
}

Boolean
Wstring::equal(wchar_t * str, unsigned off, unsigned length)
{
        return (Boolean)IS_WEQUALN(string.buffer.start + off, str, length);
}

Boolean
Wstring::equal(wchar_t * str, unsigned off)
{
        return (Boolean)IS_WEQUAL(string.buffer.start + off, str);
}

Boolean
Wstring::equal(wchar_t * str)
{
```



```
        return equal(str, 0);
}

Boolean
Wstring::equal(Wstring * str, unsigned off, unsigned length)
{
        return (Boolean)IS_WEQUALN(string.buffer.start + off, str->string.buffer
}

Boolean
Wstring::equal(Wstring * str)
{
        return equal(str, 0);
}

Boolean
Wstring::equal(Wstring * str, unsigned off)
{
        return (Boolean)IS_WEQUAL(string.buffer.start + off, str->string.buffer.
}

void
Wstring::append_to_str(struct _String * str, unsigned off, unsigned length)
{
        append_string(string.buffer.start + off, str, length);
}

Name
Name_set::lookup(const char *key)
{
        for (entry *node = root; node != 0;) {
                int res = strcmp(key, node->name->string_mb);
                if (res < 0) {
                        node = node->left;
                } else if (res > 0) {
                        node = node->right;
                } else {
                        return node->name;
                }
        }
        return 0;
}

Name
Name_set::insert(const char *key, Boolean &found)
{
        Name    name = 0;

        if (root != 0) {
                for (entry *node = root; name == 0;) {
                        int res = strcmp(key, node->name->string_mb);
                        if (res < 0) {
                                if (node->left != 0) {
                                        node = node->left;
                                } else {
                                        found = false;
                                        name = ALLOC(Name);

                                        node->left = new entry(name, node);
                                        rebalance(node);
                                }
                        } else if (res > 0) {
                                if (node->right != 0) {
                                        node = node->right;
                                } else {
                                        found = false;
```

```
1052                                    name = ALLOC(Name);

1054                                    node->right = new entry(name, node);
1055                                    rebalance(node);
1056                            }
1057                    } else {
1058                            found = true;
1059                            name = node->name;
1060                    }
1061            }
1062    } else {
1063            found = false;
1064            name = ALLOC(Name);

1066            root = new entry(name, 0);
1067    }
1068    return name;
1069 }

1071 void
1072 Name_set::insert(Name name) {
1073    if (root != 0) {
1074            for (entry *node = root;;) {
1075                    int res = strcmp(name->string_mb, node->name->string_mb)
1076                    if (res < 0) {
1077                            if (node->left != 0) {
1078                                    node = node->left;
1079                            } else {
1080                                    node->left = new entry(name, node);
1081                                    rebalance(node);
1082                                    break;
1083                            }
1084                    } else if (res > 0) {
1085                            if (node->right != 0) {
1086                                    node = node->right;
1087                            } else {
1088                                    node->right = new entry(name, node);
1089                                    rebalance(node);
1090                                    break;
1091                            }
1092                    } else {
1093                            // should be an error: inserting already existin
1094                            break;
1095                    }
1096            }
1097    } else {
1098            root = new entry(name, 0);
1099    }
1100 }

1102 void
1103 Name_set::rebalance(Name_set::entry *node) {
1104    for (; node != 0; node = node->parent) {
1105            entry *right = node->right;
1106            entry *left = node->left;

1108            unsigned rdepth = (right != 0) ? right->depth : 0;
1109            unsigned ldepth = (left != 0) ? left->depth : 0;

1111            if (ldepth > rdepth + 1) {
1112                    if ((node->left = left->right) != 0) {
1113                            left->right->parent = node;
1114                    }
1115                    if ((left->parent = node->parent) != 0) {
1116                            if (node == node->parent->right) {
1117                                    node->parent->right = left;
```



```
1118                            } else {
1119                                    node->parent->left = left;
1120                            }
1121                    } else {
1122                            root = left;
1123                    }
1124                    left->right = node;
1125                    node->parent = left;

1127                    node->setup_depth();
1128                    node = left;
1129            } else if (rdepth > ldepth + 1) {
1130                    if ((node->right = right->left) != 0) {
1131                            right->left->parent = node;
1132                    }
1133                    if ((right->parent = node->parent) != 0) {
1134                            if (node == node->parent->right) {
1135                                    node->parent->right = right;
1136                            } else {
1137                                    node->parent->left = right;
1138                            }
1139                    } else {
1140                            root = right;
1141                    }
1142                    right->left = node;
1143                    node->parent = right;

1145                    node->setup_depth();
1146                    node = right;
1147            }
1148            node->setup_depth();
1149    }
1150 }

1152 Name_set::iterator
1153 Name_set::begin() const {
1154    for (entry *node = root; node != 0; node = node->left) {
1155            if (node->left == 0) {
1156                    return iterator(node);
1157            }
1158    }
1159    return iterator();
1160 }

1162 Name_set::iterator&
1163 Name_set::iterator::operator++() {
1164    if (node != 0) {
1165            if (node->right != 0) {
1166                    node = node->right;
1167                    while (node->left != 0) {
1168                            node = node->left;
1169                    }
1170            } else {
1171                    while ((node->parent != 0) && (node->parent->right == no
1172                            node = node->parent;
1173                    }
1174                    node = node->parent;
1175            }
1176    }
1177    return *this;
1178 }
1179 #endif /* ! codereview */
```

```
**********************************************************
    7676 Wed May 20 11:04:22 2015
new/usr/src/cmd/make/lib/mksh/src/mksh.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)mksh.cc 1.22 06/12/12
 */

#pragma ident   "@(#)mksh.cc    1.22    06/12/12"

/*
 *      mksh.cc
 *
 *      Execute the command(s) of one Make or DMake rule
 */

/*
 * Included files
 */
#if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL) /* tolik */
#       include <avo/util.h>
#endif

#include <mksh/dosys.h>         /* redirect_io() */
#include <mksh/misc.h>          /* retmem() */
#include <mksh/mksh.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <errno.h>
#include <signal.h>

#ifdef HP_UX
        extern void (*sigset(int, void (*)(__harg)))(__harg);
#endif

/*
 * Workaround for NFS bug. Sometimes, when running 'chdir' on a remote
 * dmake server, it fails with "Stale NFS file handle" error.
 * The second attempt seems to work.
 */
int
my_chdir(char * dir) {
```



```
        int res = chdir(dir);
        if (res != 0 && (errno == ESTALE || errno == EAGAIN)) {
                /* Stale NFS file handle. Try again */
                res = chdir(dir);
        }
        return res;
}


/*
 * File table of contents
 */
static void     change_sunpro_dependencies_value(char *oldpath, char *newpath);
static void     init_mksh_globals(char *shell);
static void     set_env_vars(char *env_list[]);

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
/*
 *      Execute the command(s) of one Make or DMake rule
 */
int
do_job(Avo_DmakeCommand *cmd_list[], char *env_list[], char *stdout_file, char *
{
        Boolean                 always_exec_flag;
        char                    *cmd;
        Avo_DmakeCommand        **cmd_list_p;
        Name                    command;
        Boolean                 do_not_exec_flag;
        Boolean                 ignore_flag;
        int                     length;
        Boolean                 make_refd_flag;
        Boolean                 meta_flag;
        char                    pathname[MAXPATHLEN];
        Doname                  result;
        Boolean                 silent_flag;
        wchar_t                 *tmp_wcs_buffer;

        if ((childPid = fork()) < 0) {  /* error */
                ;
        } else if (childPid > 0) {              /* parent */
                ;
        } else {                                /* child, mksh */
                (void) sigset(SIGCHLD, SIG_DFL);
                enable_interrupt(handle_interrupt_mksh);
                /* set environment variables */
                set_env_vars(env_list);
                /* redirect stdout and stderr to temp files */
                dup2(1, 2);     // Because fatal_mksh() prints error messages in
                                // stderr but dmake uses stderr for XDR communic
                                // and stdout for errors messages.
                redirect_io(stdout_file, stderr_file);
                /* try cd'ing to cwd */
                if (my_chdir(cwd) != 0) {
                        /* try the netpath machine:pathname */
                        if (!avo_netpath_to_path(cnwd, pathname)) {
                                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 137, "
                        } else if (my_chdir(pathname) != 0) {
                                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 138, "
                        }
                        /*
                         * change the value of SUNPRO_DEPENDENCIES
                         * to the new path.
                         */
                        change_sunpro_dependencies_value(cwd, pathname);
                }
                init_mksh_globals(shell);
```

```
128                 for (cmd_list_p = cmd_list;
129                      *cmd_list_p != (Avo_DmakeCommand *) NULL;
130                      cmd_list_p++) {
131                         if ((*cmd_list_p)->ignore()) {
132                                 ignore_flag = true;
133                         } else {
134                                 ignore_flag = false;
135                         }
136                         if ((*cmd_list_p)->silent()) {
137                                 silent_flag = true;
138                         } else {
139                                 silent_flag = false;
140                         }
141 /*
142                         if ((*cmd_list_p)->always_exec()) {
143                                 always_exec_flag = true;
144                         } else {
145                                 always_exec_flag = false;
146                         }
147  */
148                         always_exec_flag = false;
149                         if ((*cmd_list_p)->meta()) {
150                                 meta_flag = true;
151                         } else {
152                                 meta_flag = false;
153                         }
154                         if ((*cmd_list_p)->make_refd()) {
155                                 make_refd_flag = true;
156                         } else {
157                                 make_refd_flag = false;
158                         }
159                         if ((*cmd_list_p)->do_not_exec()) {
160                                 do_not_exec_flag = true;
161                         } else {
162                                 do_not_exec_flag = false;
163                         }
164                         do_not_exec_rule = do_not_exec_flag;
165                         cmd = (*cmd_list_p)->getCmd();
166                         if ((length = strlen(cmd)) >= MAXPATHLEN) {
167                                 tmp_wcs_buffer = ALLOC_WC(length + 1);
168                                 (void) mbstowcs(tmp_wcs_buffer, cmd, length + 1)
169                                 command = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
170                                 retmem(tmp_wcs_buffer);
171                         } else {
172                                 MBSTOWCS(wcs_buffer, cmd);
173                                 command = GETNAME(wcs_buffer, FIND_LENGTH);
174                         }
175                         if ((command->hash.length > 0) &&
176                             (!silent_flag || do_not_exec_flag)) {
177                                 (void) printf("%s\n", command->string_mb);
178                         }
179                         result = dosys_mksh(command,
180                                             ignore_flag,
181                                             make_refd_flag,
182                                             false, /* bugs #4085164 & #4990057 *
183                                             /* BOOLEAN(silent_flag && ignore_fla
184                                             always_exec_flag,
185                                             (Name) NULL,
186                                             false,
187                                             NULL,
188                                             NULL,
189                                             vroot_path,
190                                             nice_prio);
191                         if (result == build_failed) {

193 #ifdef PRINT_EXIT_STATUS
```



```
194                                 warning_mksh(NOCATGETS("I'm in do_job(), and dos
195 #endif

197                                 if (silent_flag) {
198                                         (void) printf(catgets(libmksdmsi18n_catd
199                                                               command->string_mb);
200                                 }
201                                 if (!ignore_flag && !ignore) {

203 #ifdef PRINT_EXIT_STATUS
204                                         warning_mksh(NOCATGETS("I'm in do_job(),
205 #endif

207                                         exit(1);
208                                 }
209                         }
210                 }

212 #ifdef PRINT_EXIT_STATUS
213                 warning_mksh(NOCATGETS("I'm in do_job(), exiting 0."));
214 #endif

216                 exit(0);
217         }
218         return childPid;
219 }
220 #endif /* TEAMWARE_MAKE_CMN */

222 static void
223 set_env_vars(char *env_list[])
224 {
225         char                    **env_list_p;

227         for (env_list_p = env_list;
228              *env_list_p != (char *) NULL;
229              env_list_p++) {
230                 putenv(*env_list_p);
231         }
232 }

234 static void
235 init_mksh_globals(char *shell)
236 {
237 /*
238         MBSTOWCS(wcs_buffer, NOCATGETS("SHELL"));
239         shell_name = GETNAME(wcs_buffer, FIND_LENGTH);
240         MBSTOWCS(wcs_buffer, shell);
241         (void) SETVAR(shell_name, GETNAME(wcs_buffer, FIND_LENGTH), false);
242  */
243         char * dmake_shell;
244         if ((dmake_shell = getenv(NOCATGETS("DMAKE_SHELL"))) == NULL) {
245                 dmake_shell = shell;
246         }
247         MBSTOWCS(wcs_buffer, dmake_shell);
248         shell_name = GETNAME(wcs_buffer, FIND_LENGTH);
249 }

251 /*
252  * Change the pathname in the value of the SUNPRO_DEPENDENCIES env variable
253  * from oldpath to newpath.
254  */
255 static void
256 change_sunpro_dependencies_value(char *oldpath, char *newpath)
257 {
258         char            buf[MAXPATHLEN];
259         static char     *env;
```

```
260         int             length;
261         int             oldpathlen;
262         char            *sp_dep_value;

264         /* check if SUNPRO_DEPENDENCIES is set in the environment */
265         if ((sp_dep_value = getenv(NOCATGETS("SUNPRO_DEPENDENCIES"))) != NULL) {
266                 oldpathlen = strlen(oldpath);
267                 /* check if oldpath is indeed in the value of SUNPRO_DEPENDENCIE
268                 if (strncmp(oldpath, sp_dep_value, oldpathlen) == 0) {
269                         (void) sprintf(buf,
270                                        "%s%s",
271                                        newpath,
272                                        sp_dep_value + oldpathlen);
273                         length = 2 +
274                                 strlen(NOCATGETS("SUNPRO_DEPENDENCIES")) +
275                                 strlen(buf);
276                         env = getmem(length);
277                         (void) sprintf(env,
278                                        "%s=%s",
279                                        NOCATGETS("SUNPRO_DEPENDENCIES"),
280                                        buf);
281                         (void) putenv(env);
282                 }
283         }
284 }


287 #endif /* ! codereview */
```

```
**********************************************************
    4731 Wed May 20 11:04:23 2015
new/usr/src/cmd/make/lib/mksh/src/read.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)read.cc 1.11 06/12/12
 */

#pragma ident   "@(#)read.cc    1.11    06/12/12"

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

/*
 * Included files
 */
#include <mksh/misc.h>          /* retmem() */
#include <mksh/read.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <sys/uio.h>            /* read() */
#include <unistd.h>             /* close(), unlink(), read() */

#define STRING_LEN_TO_CONVERT   (8*1024)

/*
 *      get_next_block_fn(source)
 *
 *      Will get the next block of text to read either
 *      by popping one source bVSIZEOFlock of the stack of Sources
 *      or by reading some more from the makefile.
 *
 *      Return value:
 *                              The new source block to read from
 *
 *      Parameters:
 *              source          The old source block
 *
 *      Global variables used:
```



```
 *              file_being_read The name of the current file, error msg
 */
Boolean         make_state_locked;
Source
get_next_block_fn(register Source source)
{
        register off_t          to_read;
        register int            length;
        register size_t         num_wc_chars;
        char                    ch_save;
        char                    *ptr;

        if (source == NULL) {
                return NULL;
        }
        if ((source->fd < 0) ||
                ((source->bytes_left_in_file <= 0) &&
                        (source->inp_buf_ptr >= source->inp_buf_end))) {
                /* We can't read from the makefile, so pop the source block */
                if (source->fd > 2) {
                        (void) close(source->fd);
                        if (make_state_lockfile != NULL) {
                                (void) unlink(make_state_lockfile);
                                retmem_mb(make_state_lockfile);
                                make_state_lockfile = NULL;
                                make_state_locked = false;
                        }
                }
                if (source->string.free_after_use &&
                    (source->string.buffer.start != NULL)) {
                        retmem(source->string.buffer.start);
                        source->string.buffer.start = NULL;
                }
                if (source->inp_buf != NULL) {
                        retmem_mb(source->inp_buf);
                        source->inp_buf = NULL;
                }
                source = source->previous;
                if (source != NULL) {
                        source->error_converting = false;
                }
                return source;
        }
        if (source->bytes_left_in_file > 0) {
        /*
         * Read the whole makefile.
         * Hopefully the kernel managed to prefetch the stuff.
         */
                to_read = source->bytes_left_in_file;
                source->inp_buf_ptr = source->inp_buf = getmem(to_read + 1);
                source->inp_buf_end = source->inp_buf + to_read;
                length = read(source->fd, source->inp_buf, (unsigned int) to_rea
                if (length != to_read) {
                        WCSTOMBS(mbs_buffer, file_being_read);
                        if (length == 0) {
                                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 140, "
                                        mbs_buffer);
                        } else {
                                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 141, "
                                        mbs_buffer,
                                        errmsg(errno));
                        }
                }
                *source->inp_buf_end = nul_char;
                source->bytes_left_in_file = 0;
        }
```

```
128         /*
129          * Try to convert the next piece.
130          */
131         ptr = source->inp_buf_ptr + STRING_LEN_TO_CONVERT;
132         if (ptr > source->inp_buf_end) {
133                 ptr = source->inp_buf_end;
134         }
135         for (num_wc_chars = 0; ptr > source->inp_buf_ptr; ptr--) {
136                 ch_save = *ptr;
137                 *ptr = nul_char;
138                 num_wc_chars = mbstowcs(source->string.text.end,
139                                         source->inp_buf_ptr,
140                                         STRING_LEN_TO_CONVERT);
141                 *ptr = ch_save;
142                 if (num_wc_chars != (size_t)-1) {
143                         break;
144                 }
145         }
146
147         if ((int) num_wc_chars == (size_t)-1) {
148                 source->error_converting = true;
149                 return source;
150         }

152         source->error_converting = false;
153         source->inp_buf_ptr = ptr;
154         source->string.text.end += num_wc_chars;
155         *source->string.text.end = 0;

157         if (source->inp_buf_ptr >= source->inp_buf_end) {
158                 if (*(source->string.text.end - 1) != (int) newline_char) {
159                         WCSTOMBS(mbs_buffer, file_being_read);
160                         warning_mksh(catgets(libmksdmsi18n_catd, 1, 142, "newlin
161                                              mbs_buffer);
162                         *source->string.text.end++ = (int) newline_char;
163                         *source->string.text.end = (int) nul_char;
164                         *source->string.buffer.end++;
165                 }
166                 if (source->inp_buf != NULL) {
167                         retmem_mb(source->inp_buf);
168                         source->inp_buf = NULL;
169                 }
170         }
171         return source;
172 }


175 #endif /* ! codereview */
```

```
**********************************************************
    1008 Wed May 20 11:04:23 2015
new/usr/src/cmd/make/lib/vroot/Makefile
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.5 06/12/12
#

TOP =           ../../..
include $(TOP)/rules/variant.mk
include $(TOP)/rules/derived.mk
#endif /* ! codereview */
```

****
1564 Wed May 20 11:04:23 2015
new/usr/src/cmd/make/lib/vroot/src/Makefile
make: initial Sun make source, disconnected from the build
****

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Makefile 1.3 06/12/12
#

# Generic makefile for use in src directories.  Knows how to make common things
# in the right $(VARIANT) directory.

include $(TOP)/rules/variant.mk

all :=          TARG = all
install :=              TARG = install
clean :=                TARG = clean
test :=         TARG = test
l10n_install := TARG = l10n_install
i18n_install := TARG = i18n_install

SRC =           ../src
MFLAGS +=       SRC=$(SRC)

# See $(TOP)/rules/master.mk for how these are built.
%.h %.cc %.C %.E %.o all install clean test l10n_install i18n_install: FRC
        @ if [ ! -d ../$(VARIANT) ]; then \
                mkdir ../$(VARIANT) ; \
        fi
        cd ../$(VARIANT); $(MAKE) $(MFLAGS) -f $(SRC)/Variant.mk DESTDIR=$(DESTD

FRC:
#endif /* ! codereview */
```

```
**********************************************************
    1633 Wed May 20 11:04:23 2015
new/usr/src/cmd/make/lib/vroot/src/Variant.mk
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
#
# 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 1996 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# @(#)Variant.mk 1.28 06/12/12
#

TOP =           ../../../..
include $(TOP)/rules/master.mk
include $(TOP)/rules/dmake.mk

PKG_TOP =       $(TOP)/Make
CPPFLAGS +=     -I$(PKG_TOP)/include

CCSRCS =        \
                access.cc \
                args.cc \
                chdir.cc \
                chmod.cc \
                chown.cc \
                chroot.cc \
                creat.cc \
                execve.cc \
                lock.cc \
                lstat.cc \
                mkdir.cc \
                mount.cc \
                open.cc \
                readlink.cc \
                report.cc \
                rmdir.cc \
                stat.cc \
                statfs.cc \
                truncate.cc \
                unlink.cc \
                unmount.cc \
                utimes.cc \
                vroot.cc \
                setenv.cc


HDRS_DIR =      $(PKG_TOP)/include/vroot
```



```
.INIT: $(HDRS_DIR)/args.h $(HDRS_DIR)/report.h $(HDRS_DIR)/vroot.h

LIBNAME =       libvroot.a
MSG_FILE = libvroot.msg
I18N_DIRS = $(SRC)

include $(TOP)/Make/lib/Lib.mk
include $(TOP)/rules/lib.mk

#endif /* ! codereview */
```

****
1435 Wed May 20 11:04:23 2015
new/usr/src/cmd/make/lib/vroot/src/access.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)access.cc 1.4 06/12/12
 */

#pragma ident   "@(#)access.cc  1.4     06/12/12"

#include <unistd.h>
#include <vroot/vroot.h>
#include <vroot/args.h>

static int      access_thunk(char *path)
{
        vroot_result= access(path, vroot_args.access.mode);
        return((vroot_result == 0) || (errno != ENOENT));
}

int     access_vroot(char *path, int mode, pathpt vroot_path, pathpt vroot_vroot
{
        vroot_args.access.mode= mode;
        translate_with_thunk(path, access_thunk, vroot_path, vroot_vroot, rw_rea
        return(vroot_result);
}
#endif /* ! codereview */
```

```
**********************************************************
    1095 Wed May 20 11:04:24 2015
new/usr/src/cmd/make/lib/vroot/src/args.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)args.cc 1.3 06/12/12
 */

#pragma ident   "@(#)args.cc    1.3     06/12/12"

#include <vroot/vroot.h>
#include <vroot/args.h>

union Args      vroot_args;
int             vroot_result;
#endif /* ! codereview */
```

****
1341 Wed May 20 11:04:24 2015
new/usr/src/cmd/make/lib/vroot/src/chdir.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)chdir.cc 1.4 06/12/12
 */

#pragma ident   "@(#)chdir.cc   1.4     06/12/12"

#include <unistd.h>
#include <vroot/vroot.h>
#include <vroot/args.h>

static int      chdir_thunk(char *path)
{
        vroot_result= chdir(path);
        return(vroot_result == 0);
}

int     chdir_vroot(char *path, pathpt vroot_path, pathpt vroot_vroot)
{
        translate_with_thunk(path, chdir_thunk, vroot_path, vroot_vroot, rw_read
        return(vroot_result);
}
#endif /* ! codereview */
```

****
1480 Wed May 20 11:04:24 2015
new/usr/src/cmd/make/lib/vroot/src/chmod.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)chmod.cc 1.4 06/12/12
 */

#pragma ident   "@(#)chmod.cc   1.4     06/12/12"

#include <sys/types.h>
#include <sys/stat.h>

extern int chmod(const char *path, mode_t mode);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      chmod_thunk(char *path)
{
        vroot_result= chmod(path, vroot_args.chmod.mode);
        return(vroot_result == 0);
}

int     chmod_vroot(char *path, int mode, pathpt vroot_path, pathpt vroot_vroot)
{
        vroot_args.chmod.mode= mode;
        translate_with_thunk(path, chmod_thunk, vroot_path, vroot_vroot, rw_read
        return(vroot_result);
}
#endif /* ! codereview */
```

****
1558 Wed May 20 11:04:24 2015
new/usr/src/cmd/make/lib/vroot/src/chown.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)chown.cc 1.4 06/12/12
 */

#pragma ident   "@(#)chown.cc   1.4     06/12/12"

#include <unistd.h>
#include <sys/types.h>

extern int chown(const char *path, uid_t owner, gid_t group);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      chown_thunk(char *path)
{
        vroot_result= chown(path, vroot_args.chown.user, vroot_args.chown.group)
        return(vroot_result == 0);
}

int     chown_vroot(char *path, int user, int group, pathpt vroot_path, pathpt v
{
        vroot_args.chown.user= user;
        vroot_args.chown.group= group;
        translate_with_thunk(path, chown_thunk, vroot_path, vroot_vroot, rw_read
        return(vroot_result);
}
#endif /* ! codereview */
```

****
1386 Wed May 20 11:04:24 2015
new/usr/src/cmd/make/lib/vroot/src/chroot.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)chroot.cc 1.4 06/12/12
 */

#pragma ident   "@(#)chroot.cc  1.4     06/12/12"

#include <unistd.h>

extern int chroot(const char *path);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      chroot_thunk(char *path)
{
        vroot_result= chroot(path);
        return(vroot_result == 0);
}

int     chroot_vroot(char *path, pathpt vroot_path, pathpt vroot_vroot)
{
        translate_with_thunk(path, chroot_thunk, vroot_path, vroot_vroot, rw_rea
        return(vroot_result);
}
#endif /* ! codereview */
```

****
1500 Wed May 20 11:04:24 2015
new/usr/src/cmd/make/lib/vroot/src/creat.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)creat.cc 1.4 06/12/12
 */

#pragma ident   "@(#)creat.cc   1.4     06/12/12"

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

extern int creat(const char *path, mode_t mode);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      creat_thunk(char *path)
{
        vroot_result= creat(path, vroot_args.creat.mode);
        return(vroot_result >= 0);
}

int     creat_vroot(char *path, int mode, pathpt vroot_path, pathpt vroot_vroot)
{
        vroot_args.creat.mode= mode;
        translate_with_thunk(path, creat_thunk, vroot_path, vroot_vroot, rw_writ
        return(vroot_result);
}
#endif /* ! codereview */
```

****
1607 Wed May 20 11:04:25 2015
new/usr/src/cmd/make/lib/vroot/src/execve.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)execve.cc 1.4 06/12/12
 */

#pragma ident   "@(#)execve.cc  1.4     06/12/12"

#include <unistd.h>

extern int execve (const char *path, char *const argv[], char *const envp[]);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      execve_thunk(char *path)
{
        execve(path, vroot_args.execve.argv, vroot_args.execve.environ);
        switch (errno) {
                case ETXTBSY:
                case ENOEXEC: return 1;
                default: return 0;
        }
}

int     execve_vroot(char *path, char **argv, char **environ, pathpt vroot_path,
{
        vroot_args.execve.argv= argv;
        vroot_args.execve.environ= environ;
        translate_with_thunk(path, execve_thunk, vroot_path, vroot_vroot, rw_rea
        return(-1);
}
#endif /* ! codereview */
```

****
1258 Wed May 20 11:04:25 2015
new/usr/src/cmd/make/lib/vroot/src/libvroot.msg
make: initial Sun make source, disconnected from the build
****

```
$quote "


$set    1
$
$ 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 1996 Sun Microsystems, Inc. All rights reserved.
$ Use is subject to license terms.
$
$ @(#)libvroot.msg 1.2 06/12/12
$
142     "file_lock: file %s is already locked.\n"
143     "file_lock: will periodically check the lockfile %s for two minutes.\n"
144     "Current working directory %s\n"
145     "Could not lock file ‘%s’; "
146     " failed - "
147     "Couldn’t write to %s"
148     "Cannot open ‘%s’ for writing\n"
149     "Cannot open %s for writing\n"
#endif /* ! codereview */
```

```
**********************************************************
    5590 Wed May 20 11:04:25 2015
new/usr/src/cmd/make/lib/vroot/src/lock.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)lock.cc 1.17 06/12/12
 */

#pragma ident   "@(#)lock.cc    1.17    06/12/12"

#include <avo/intl.h>   /* for NOCATGETS */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/errno.h>
#include <sys/param.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <vroot/vroot.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <signal.h>
#include <errno.h>                      /* errno */

#if !defined(linux)
extern  char            *sys_errlist[];
extern  int             sys_nerr;
#endif

static  void            file_lock_error(char *msg, char *file, char *str, int ar

#define BLOCK_INTERUPTS sigfillset(&newset) ; \
        sigprocmask(SIG_SETMASK, &newset, &oldset)

#define UNBLOCK_INTERUPTS \
        sigprocmask(SIG_SETMASK, &oldset, &newset)

/*
 * This code stolen from the NSE library and changed to not depend
 * upon any NSE routines or header files.
 *
```



```
 * Simple file locking.
 * Create a symlink to a file.  The "test and set" will be
 * atomic as creating the symlink provides both functions.
 *
 * The timeout value specifies how long to wait for stale locks
 * to disappear.  If the lock is more than 'timeout' seconds old
 * then it is ok to blow it away.  This part has a small window
 * of vunerability as the operations of testing the time,
 * removing the lock and creating a new one are not atomic.
 * It would be possible for two processes to both decide to blow
 * away the lock and then have process A remove the lock and establish
 * its own, and then then have process B remove the lock which accidentily
 * removes A's lock rather than the stale one.
 *
 * A further complication is with the NFS.  If the file in question is
 * being served by an NFS server, then its time is set by that server.
 * We can not use the time on the client machine to check for a stale
 * lock.  Therefore, a temp file on the server is created to get
 * the servers current time.
 *
 * Returns an error message.  NULL return means the lock was obtained.
 *
 * 12/6/91 Added the parameter "file_locked".  Before this parameter
 * was added, the calling procedure would have to wait for file_lock()
 * to return before it sets the flag. If the user interrupted "make"
 * between the time the lock was acquired and the time file_lock()
 * returns, make wouldn't know that the file has been locked, and therefore
 * it wouldn' remove the lock. Setting the flag right after locking the file
 * makes this window much smaller.
 */

int
file_lock(char *name, char *lockname, int *file_locked, int timeout)
{
        int             counter = 0;
        static char     msg[MAXPATHLEN+1];
        int             printed_warning = 0;
        int             r;
        struct stat     statb;
        sigset_t newset;
        sigset_t oldset;

        *file_locked = 0;
        if (timeout <= 0) {
                timeout = 120;
        }
        for (;;) {
                BLOCK_INTERUPTS;
                r = symlink(name, lockname);
                if (r == 0) {
                        *file_locked = 1;
                        UNBLOCK_INTERUPTS;
                        return 0; /* success */
                }
                UNBLOCK_INTERUPTS;

                if (errno != EEXIST) {
                        file_lock_error(msg, name, NOCATGETS("symlink(%s, %s)"),
                            (int) name, (int) lockname);
                        fprintf(stderr, "%s", msg);
                        return errno;
                }

                counter = 0;
                for (;;) {
                        sleep(1);
```

```
128                         r = lstat(lockname, &statb);
129                         if (r == -1) {
130                                 /*
131                                  * The lock must have just gone away - try
132                                  * again.
133                                  */
134                                 break;
135                         }

137                         if ((counter > 5) && (!printed_warning)) {
138                                 /* Print waiting message after 5 secs */
139 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
140                                 (void) getcwd(msg, MAXPATHLEN);
141 #else
142                                 (void) getwd(msg);
143 #endif
144                                 fprintf(stderr,
145                                         catgets(libmksdmsi18n_catd, 1, 162, "fil
146                                         name);
147                                 fprintf(stderr,
148                                         catgets(libmksdmsi18n_catd, 1, 163, "fil
149                                         lockname);
150                                 fprintf(stderr,
151                                         catgets(libmksdmsi18n_catd, 1, 144, "Cur
152                                         msg);

154                                 printed_warning = 1;
155                         }

157                         if (++counter > timeout ) {
158                                 /*
159                                  * Waited enough - return an error..
160                                  */
161                                 return EEXIST;
162                         }
163                 }
164         }
165         /* NOTREACHED */
166 }

168 /*
169  * Format a message telling why the lock could not be created.
170  */
171 static  void
172 file_lock_error(char *msg, char *file, char *str, int arg1, int arg2)
173 {
174         int             len;

176         sprintf(msg, catgets(libmksdmsi18n_catd, 1, 145, "Could not lock file ‘%
177         len = strlen(msg);
178         sprintf(&msg[len], str, arg1, arg2);
179         strcat(msg, catgets(libmksdmsi18n_catd, 1, 146, " failed - "));
180 #if !defined(linux)
181         if (errno < sys_nerr) {
182 #ifdef SUN4_x
183                 strcat(msg, sys_errlist[errno]);
184 #endif
185 #ifdef SUN5_0
186                 strcat(msg, strerror(errno));
187 #endif
188         } else {
189                 len = strlen(msg);
190                 sprintf(&msg[len], NOCATGETS("errno %d"), errno);
191         }
192 #else
193         strcat(msg, strerror(errno));
```



```
194 #endif
195 }

197 #endif /* ! codereview */
```

****
1502 Wed May 20 11:04:25 2015
new/usr/src/cmd/make/lib/vroot/src/lstat.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1998 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)lstat.cc 1.6 06/12/12
 */

#pragma ident   "@(#)lstat.cc   1.6     06/12/12"

#include <sys/types.h>
#include <sys/stat.h>

extern int lstat(const char *path, struct stat *buf);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      lstat_thunk(char *path)
{
        vroot_result= lstat(path, vroot_args.lstat.buffer);
        return(vroot_result == 0);
}

int     lstat_vroot(char *path, struct stat *buffer, pathpt vroot_path, pathpt v
{
        vroot_args.lstat.buffer= buffer;
        translate_with_thunk(path, lstat_thunk, vroot_path, vroot_vroot, rw_read
        return(vroot_result);
}
#endif /* ! codereview */
```

****
1481 Wed May 20 11:04:25 2015
new/usr/src/cmd/make/lib/vroot/src/mkdir.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)mkdir.cc 1.4 06/12/12
 */

#pragma ident   "@(#)mkdir.cc   1.4     06/12/12"

#include <sys/types.h>
#include <sys/stat.h>

extern int mkdir(const char *path, mode_t mode);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      mkdir_thunk(char *path)
{
        vroot_result= mkdir(path, vroot_args.mkdir.mode);
        return(vroot_result == 0);
}

int     mkdir_vroot(char *path, int mode, pathpt vroot_path, pathpt vroot_vroot)
{
        vroot_args.mkdir.mode= mode;
        translate_with_thunk(path, mkdir_thunk, vroot_path, vroot_vroot, rw_writ
        return(vroot_result);
}
#endif /* ! codereview */
```

****
1591 Wed May 20 11:04:26 2015
new/usr/src/cmd/make/lib/vroot/src/mount.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1995 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)mount.cc 1.5 06/12/12
 */

#pragma ident   "@(#)mount.cc   1.5     06/12/12"

#include <sys/types.h>
#include <sys/mount.h>

#ifndef HP_UX
extern int mount(const char *spec, const char *dir, int mflag, ...);
#endif

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      mount_thunk(char *path)
{
        vroot_result= mount(path, vroot_args.mount.name, vroot_args.mount.mode);
        return(vroot_result == 0);
}

int     mount_vroot(char *target, char *name, int mode, pathpt vroot_path, pathp
{
        vroot_args.mount.name= name;
        vroot_args.mount.mode= mode;
        translate_with_thunk(target, mount_thunk, vroot_path, vroot_vroot, rw_re
        return(vroot_result);
}
#endif /* ! codereview */
```

```
**********************************************************
    1611 Wed May 20 11:04:26 2015
new/usr/src/cmd/make/lib/vroot/src/open.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)open.cc 1.4 06/12/12
 */

#pragma ident   "@(#)open.cc    1.4     06/12/12"

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

extern int open(const char *path, int oflag, ...);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      open_thunk(char *path)
{
        vroot_result= open(path, vroot_args.open.flags, vroot_args.open.mode);
        return(vroot_result >= 0);
}

int     open_vroot(char *path, int flags, int mode, pathpt vroot_path, pathpt vr
{
        vroot_args.open.flags= flags;
        vroot_args.open.mode= mode;
        translate_with_thunk(path, open_thunk, vroot_path, vroot_vroot,
                             ((flags & (O_CREAT|O_APPEND)) != 0) ? rw_write :
        return(vroot_result);
}
#endif /* ! codereview */
```

```
**********************************************************
    1602 Wed May 20 11:04:26 2015
new/usr/src/cmd/make/lib/vroot/src/readlink.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)readlink.cc 1.4 06/12/12
 */

#pragma ident   "@(#)readlink.cc        1.4     06/12/12"

#include <unistd.h>

extern int readlink(const char *path, void *buf, size_t bufsiz);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      readlink_thunk(char *path)
{
        vroot_result= readlink(path, vroot_args.readlink.buffer, vroot_args.read
        return(vroot_result >= 0);
}

int     readlink_vroot(char *path, char *buffer, int buffer_size, pathpt vroot_p
{
        vroot_args.readlink.buffer= buffer;
        vroot_args.readlink.buffer_size= buffer_size;
        translate_with_thunk(path, readlink_thunk, vroot_path, vroot_vroot, rw_r
        return(vroot_result);
}
#endif /* ! codereview */
```

```
**********************************************************
    9611 Wed May 20 11:04:26 2015
new/usr/src/cmd/make/lib/vroot/src/report.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)report.cc 1.17 06/12/12
 */

#pragma ident   "@(#)report.cc  1.17    06/12/12"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/param.h>
#include <sys/wait.h>
#include <unistd.h>

#include <vroot/report.h>
#include <vroot/vroot.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <avo/intl.h>   /* for NOCATGETS */
#include <mk/defs.h>    /* for tmpdir */

static  FILE    *report_file;
static  FILE    *command_output_fp;
static  char    *target_being_reported_for;
static  char    *search_dir;
static  char    command_output_tmpfile[30];
static  int     is_path = 0;
static  char    sfile[MAXPATHLEN];
extern "C" {
static  void    (*warning_ptr) (char *, ...) = (void (*) (char *, ...)) NULL;
}

FILE *
get_report_file(void)
{
        return(report_file);
}

char *
```



```
get_target_being_reported_for(void)
{
        return(target_being_reported_for);
}

#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
extern "C" {
static void
close_report_file(void)
{
        (void)fputs("\n", report_file);
        (void)fclose(report_file);
}
} // extern "C"
#else
static void
close_report_file(int, ...)
{
        (void)fputs("\n", report_file);
        (void)fclose(report_file);
}
#endif

static void
clean_up(FILE *nse_depinfo_fp, FILE *merge_fp, char *nse_depinfo_file, char *mer
{
        fclose(nse_depinfo_fp);
        fclose(merge_fp);
        fclose(command_output_fp);
        unlink(command_output_tmpfile);
        if (unlinkf)
                unlink(merge_file);
        else
                rename(merge_file, nse_depinfo_file);
}


/*
 *  Update the file, if necessary.  We don't want to rewrite
 *  the file if we don't have to because we don't want the time of the file
 *  to change in that case.
 */

#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
extern "C" {
static void
close_file(void)
#else
static void
close_file(int, ...)
#endif
{
        char            line[MAXPATHLEN+2];
        char            buf[MAXPATHLEN+2];
        FILE            *nse_depinfo_fp;
        FILE            *merge_fp;
        char            nse_depinfo_file[MAXPATHLEN];
        char            merge_file[MAXPATHLEN];
        char            lock_file[MAXPATHLEN];
        int             err;
        int             len;
        int             changed = 0;
        int             file_locked;

        fprintf(command_output_fp, "\n");
        fclose(command_output_fp);
```

```
128         if ((command_output_fp = fopen(command_output_tmpfile, "r")) == NULL) {
129                 return;
130         }
131         sprintf(nse_depinfo_file, "%s/%s", search_dir, NSE_DEPINFO);
132         sprintf(merge_file, NOCATGETS("%s/.tmp%s.%d"), search_dir, NSE_DEPINFO,
133         sprintf(lock_file, "%s/%s", search_dir, NSE_DEPINFO_LOCK);
134         err = file_lock(nse_depinfo_file, lock_file, &file_locked, 0);
135         if (err) {
136                 if (warning_ptr != (void (*) (char *, ...)) NULL) {
137                         (*warning_ptr)(catgets(libmksdmsi18n_catd, 1, 147, "Coul
138                     }
139                 unlink(command_output_tmpfile);
140                 return;
141         }
142         /* If .nse_depinfo file doesn't exist */
143         if ((nse_depinfo_fp = fopen(nse_depinfo_file, "r+")) == NULL) {
144                 if (is_path) {
145                         if ((nse_depinfo_fp =
146                              fopen(nse_depinfo_file, "w")) == NULL) {
147                                 fprintf(stderr, catgets(libmksdmsi18n_catd, 1, 1
148                                     nse_depinfo_file);
149                                 unlink(command_output_tmpfile);
150
151                                 unlink(lock_file);
152                                 return;
153                         }
154                         while (fgets(line, MAXPATHLEN+2, command_output_fp)
155                                != NULL) {
156                                 fprintf(nse_depinfo_fp, "%s", line);
157                         }
158                         fclose(command_output_fp);
159                 }
160                 fclose(nse_depinfo_fp);
161                 if (file_locked) {
162                         unlink(lock_file);
163                 }
164                 unlink(command_output_tmpfile);
165                 return;
166         }
167         if ((merge_fp = fopen(merge_file, "w")) == NULL) {
168                 fprintf(stderr, catgets(libmksdmsi18n_catd, 1, 149, "Cannot open
169                 if (file_locked) {
170                         unlink(lock_file);
171                 }
172                 unlink(command_output_tmpfile);
173                 return;
174         }
175         len = strlen(sfile);
176         while (fgets(line, MAXPATHLEN+2, nse_depinfo_fp) != NULL) {
177                 if (strncmp(line, sfile, len) == 0 && line[len] == ':') {
178                         while (fgets(buf, MAXPATHLEN+2, command_output_fp)
179                                != NULL) {
180                                 if (is_path) {
181                                         fprintf(merge_fp, "%s", buf);
182                                         if (strcmp(line, buf)) {
183                                                 /* changed */
184                                                 changed = 1;
185                                         }
186                                 }
187                                 if (buf[strlen(buf)-1] == '\n') {
188                                         break;
189                                 }
190                         }
191                         if (changed || !is_path) {
192                                 while (fgets(line, MAXPATHLEN, nse_depinfo_fp)
193                                        != NULL) {
```



```
194                                         fputs(line, merge_fp);
195                                 }
196                                 clean_up(nse_depinfo_fp, merge_fp,
197                                          nse_depinfo_file, merge_file, 0);
198                         } else {
199                                 clean_up(nse_depinfo_fp, merge_fp,
200                                          nse_depinfo_file, merge_file, 1);
201                         }
202                         if (file_locked) {
203                                 unlink(lock_file);
204                         }
205                         unlink(command_output_tmpfile);
206                         return;
207                 } /* entry found */
208                 fputs(line, merge_fp);
209         }
210         /* Entry never found.  Add it if there is a search path */
211         if (is_path) {
212                 while (fgets(line, MAXPATHLEN+2, command_output_fp) != NULL) {
213                         fprintf(nse_depinfo_fp, "%s", line);
214                 }
215         }
216         clean_up(nse_depinfo_fp, merge_fp, nse_depinfo_file, merge_file, 1);
217         if (file_locked) {
218                 unlink(lock_file);
219         }
220 }

222 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
223 } // extern "C"
224 #endif

226 static void
227 report_dep(char *iflag, char *filename)
228 {

230         if (command_output_fp == NULL) {
231                 sprintf(command_output_tmpfile,
232                         NOCATGETS("%s/%s.%d.XXXXXX"), tmpdir, NSE_DEPINFO, getpi
233                 int fd = mkstemp(command_output_tmpfile);
234                 if ((fd < 0) || (command_output_fp = fdopen(fd, "w")) == NULL) {
235                         return;
236                 }
237                 if ((search_dir = getenv(NOCATGETS("NSE_DEP"))) == NULL) {
238                         return;
239                 }
240 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
241                 atexit(close_file);
242 #else
243                 on_exit(close_file, 0);
244 #endif
245                 strcpy(sfile, filename);
246                 if (iflag == NULL || *iflag == '\0') {
247                         return;
248                 }
249                 fprintf(command_output_fp, "%s:", sfile);
250         }
251         fprintf(command_output_fp, " ");
252         fprintf(command_output_fp, iflag);
253         if (iflag != NULL) {
254                 is_path = 1;
255         }
256 }

258 void
259 report_libdep(char *lib, char *flag)
```

```
{
        char            *ptr;
        char            filename[MAXPATHLEN];
        char            *p;

        if ((p= getenv(SUNPRO_DEPENDENCIES)) == NULL) {
                return;
        }
        ptr = strchr(p, ' ');
        if(ptr) {
                sprintf(filename, "%s-%s", ptr+1, flag);
                is_path = 1;
                report_dep(lib, filename);
        }
}

void
report_search_path(char *iflag)
{
        char            curdir[MAXPATHLEN];
        char            *sdir;
        char            *newiflag;
        char            filename[MAXPATHLEN];
        char            *p, *ptr;

        if ((sdir = getenv(NOCATGETS("NSE_DEP"))) == NULL) {
                return;
        }
        if ((p= getenv(SUNPRO_DEPENDENCIES)) == NULL) {
                return;
        }
        ptr = strchr(p, ' ');
        if( ! ptr ) {
                return;
        }
        sprintf(filename, NOCATGETS("%s-CPP"), ptr+1);
#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
        getcwd(curdir, sizeof(curdir));
#else
        getwd(curdir);
#endif
        if (strcmp(curdir, sdir) != 0 && strlen(iflag) > 2 &&
            iflag[2] != '/') {
                /* Makefile must have had an "cd xx; cc ..." */
                /* Modify the -I path to be relative to the cd */
                newiflag = (char *)malloc(strlen(iflag) + strlen(curdir) + 2);
                sprintf(newiflag, "-%c%s/%s", iflag[1], curdir, &iflag[2]);
                report_dep(newiflag, filename);
        } else {
                report_dep(iflag, filename);
        }
}

void
report_dependency(register char *name)
{
        register char   *filename;
        char            buffer[MAXPATHLEN+1];
        register char   *p;
        register char   *p2;
        char            nse_depinfo_file[MAXPATHLEN];

        if (report_file == NULL) {
                if ((filename= getenv(SUNPRO_DEPENDENCIES)) == NULL) {
                        report_file = (FILE *)-1;
                        return;
```



```
                }
                if (strlen(filename) == 0) {
                        report_file = (FILE *)-1;
                        return;
                }
                (void)strcpy(buffer, name);
                name = buffer;
                p = strchr(filename, ' ');
                if(p) {
                        *p= 0;
                } else {
                        report_file = (FILE *)-1;
                        return;
                }
                if ((report_file= fopen(filename, "a")) == NULL) {
                        if ((report_file= fopen(filename, "w")) == NULL) {
                                report_file= (FILE *)-1;
                                return;
                        }
                }
#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
                atexit(close_report_file);
#else
                (void)on_exit(close_report_file, (char *)report_file);
#endif
                if ((p2= strchr(p+1, ' ')) != NULL)
                        *p2= 0;
                target_being_reported_for= (char *)malloc((unsigned)(strlen(p+1)
                (void)strcpy(target_being_reported_for, p+1);
                (void)fputs(p+1, report_file);
                (void)fputs(":", report_file);
                *p= ' ';
                if (p2 != NULL)
                        *p2= ' ';
        }
        if (report_file == (FILE *)-1)
                return;
        (void)fputs(name, report_file);
        (void)fputs(" ", report_file);
}

#ifdef MAKE_IT
void
make_it(filename)
        register char   *filename;
{
        register char   *command;
        register char   *argv[6];
        register int    pid;
        union wait      foo;

        if (getenv(SUNPRO_DEPENDENCIES) == NULL) return;
        command= alloca(strlen(filename)+32);
        (void)sprintf(command, NOCATGETS("make %s\n"), filename);
        switch (pid= fork()) {
                case 0: /* child */
                        argv[0]= NOCATGETS("csh");
                        argv[1]= NOCATGETS("-c");
                        argv[2]= command;
                        argv[3]= 0;
                        (void)dup2(2, 1);
                        execve(NOCATGETS("/bin/sh"), argv, environ);
                        perror(NOCATGETS("execve error"));
                        exit(1);
                case -1: /* error */
                        perror(NOCATGETS("fork error"));
```

```
392                 default: /* parent */
393                         while (wait(&foo) != pid);};
394 }
395 #endif

397 #endif /* ! codereview */
```

****
1379 Wed May 20 11:04:26 2015
new/usr/src/cmd/make/lib/vroot/src/rmdir.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)rmdir.cc 1.4 06/12/12
 */

#pragma ident   "@(#)rmdir.cc   1.4     06/12/12"

#include <unistd.h>

extern int rmdir(const char *path);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      rmdir_thunk(char *path)
{
        vroot_result= rmdir(path);
        return(vroot_result == 0);
}

int     rmdir_vroot(char *path, pathpt vroot_path, pathpt vroot_vroot)
{
        translate_with_thunk(path, rmdir_thunk, vroot_path, vroot_vroot, rw_read
        return(vroot_result);
}
#endif /* ! codereview */
```

```
**********************************************************
    1926 Wed May 20 11:04:27 2015
new/usr/src/cmd/make/lib/vroot/src/setenv.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 1994 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)setenv.cc 1.6 06/12/12
 */

#pragma ident   "@(#)setenv.cc  1.6     06/12/12"

#include <stdlib.h>
#include <string.h>
#include <unistd.h>

extern  char    **environ;

static  short   setenv_made_new_vector= 0;

char *setenv(char *name, char *value)
{       char *p= NULL, **q;
        int length= 0, vl;

        if ((p= getenv(name)) == NULL) {        /* Allocate new vector */
                for (q= environ; *q != NULL; q++, length++);
                q= (char **)malloc((unsigned)(sizeof(char *)*(length+2)));
                memcpy(((char *)q)+sizeof(char *), (char *)environ, sizeof(char
                if (setenv_made_new_vector++)
                        free((char *)environ);
                length= strlen(name);
                environ= q;}
        else { /* Find old slot */
                length= strlen(name);
                for (q= environ; *q != NULL; q++)
                        if (!strncmp(*q, name, length))
                                break;};
        vl= strlen(value);
        if (!p || (length+vl+1 > strlen(p)))
                *q= p= (char *) malloc((unsigned)(length+vl+2));
        else
                p= *q;
        (void)strcpy(p, name); p+= length;
```



```
        *p++= '=';
        (void)strcpy(p, value);
        return(value);
}
#endif /* ! codereview */
```

****
1493 Wed May 20 11:04:27 2015
new/usr/src/cmd/make/lib/vroot/src/stat.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1998 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)stat.cc 1.6 06/12/12
 */

#pragma ident   "@(#)stat.cc    1.6     06/12/12"

#include <sys/types.h>
#include <sys/stat.h>

extern int stat(const char *path, struct stat *buf);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      stat_thunk(char *path)
{
        vroot_result= stat(path, vroot_args.stat.buffer);
        return(vroot_result == 0);
}

int     stat_vroot(char *path, struct stat *buffer, pathpt vroot_path, pathpt vr
{
        vroot_args.stat.buffer= buffer;
        translate_with_thunk(path, stat_thunk, vroot_path, vroot_vroot, rw_read)
        return(vroot_result);
}
#endif /* ! codereview */
```

****
1455 Wed May 20 11:04:27 2015
new/usr/src/cmd/make/lib/vroot/src/statfs.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)statfs.cc 1.5 06/12/12
 */

#pragma ident   "@(#)statfs.cc  1.5     06/12/12"

#ifndef SUN5_0

#include <vroot/vroot.h>
#include <vroot/args.h>
#include <sys/vfs.h>

static int      statfs_thunk(char *path)
{
        vroot_result= statfs(path, vroot_args.statfs.buffer);
        return(vroot_result == 0);
}

int     statfs_vroot(char *path, struct statfs *buffer, pathpt vroot_path, pathp
{
        vroot_args.statfs.buffer= buffer;
        translate_with_thunk(path, statfs_thunk, vroot_path, vroot_vroot, rw_rea
        return(vroot_result);
}
#endif
#endif /* ! codereview */
```

****
1491 Wed May 20 11:04:28 2015
new/usr/src/cmd/make/lib/vroot/src/truncate.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)truncate.cc 1.4 06/12/12
 */

#pragma ident   "@(#)truncate.cc        1.4     06/12/12"

#include <unistd.h>

extern int truncate(const char *path, off_t length);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      truncate_thunk(char *path)
{
        vroot_result= truncate(path, vroot_args.truncate.length);
        return(vroot_result == 0);
}

int     truncate_vroot(char *path, int length, pathpt vroot_path, pathpt vroot_v
{
        vroot_args.truncate.length= length;
        translate_with_thunk(path, truncate_thunk, vroot_path, vroot_vroot, rw_r
        return(vroot_result);
}
#endif /* ! codereview */
```

```
**********************************************************
    1386 Wed May 20 11:04:29 2015
new/usr/src/cmd/make/lib/vroot/src/unlink.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)unlink.cc 1.4 06/12/12
 */

#pragma ident   "@(#)unlink.cc  1.4     06/12/12"

#include <unistd.h>

extern int unlink(const char *path);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      unlink_thunk(char *path)
{
        vroot_result= unlink(path);
        return(vroot_result == 0);
}

int     unlink_vroot(char *path, pathpt vroot_path, pathpt vroot_vroot)
{
        translate_with_thunk(path, unlink_thunk, vroot_path, vroot_vroot, rw_rea
        return(vroot_result);
}
#endif /* ! codereview */
```

****
1388 Wed May 20 11:04:29 2015
new/usr/src/cmd/make/lib/vroot/src/unmount.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)unmount.cc 1.5 06/12/12
 */

#pragma ident   "@(#)unmount.cc 1.5     06/12/12"

#ifndef SUN5_0
#include <vroot/vroot.h>
#include <vroot/args.h>

extern int      unmount(char *name);

static int      unmount_thunk(char *path)
{
        vroot_result= unmount(path);
        return(vroot_result == 0);
}

int     unmount_vroot(char *path, pathpt vroot_path, pathpt vroot_vroot)
{
        translate_with_thunk(path, unmount_thunk, vroot_path, vroot_vroot, rw_re
        return(vroot_result);
}
#endif
#endif /* ! codereview */
```

****
1503 Wed May 20 11:04:29 2015
new/usr/src/cmd/make/lib/vroot/src/utimes.cc
make: initial Sun make source, disconnected from the build
****

```
/*
 * 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 1993 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)utimes.cc 1.4 06/12/12
 */

#pragma ident   "@(#)utimes.cc  1.4     06/12/12"

#include <sys/types.h>
#include <sys/time.h>

extern int utimes(char *file, struct timeval *tvp);

#include <vroot/vroot.h>
#include <vroot/args.h>

static int      utimes_thunk(char *path)
{
        vroot_result= utimes(path, vroot_args.utimes.time);
        return(vroot_result == 0);
}

int     utimes_vroot(char *path, struct timeval *time, pathpt vroot_path, pathpt
{
        vroot_args.utimes.time= time;
        translate_with_thunk(path, utimes_thunk, vroot_path, vroot_vroot, rw_rea
        return(vroot_result);
}
#endif /* ! codereview */
```

```
**********************************************************
    9243 Wed May 20 11:04:29 2015
new/usr/src/cmd/make/lib/vroot/src/vroot.cc
make: initial Sun make source, disconnected from the build
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */
/*
 * @(#)vroot.cc 1.11 06/12/12
 */

#pragma ident   "@(#)vroot.cc   1.11    06/12/12"

#include <stdlib.h>
#include <string.h>

#include <vroot/vroot.h>
#include <vroot/args.h>

#include <string.h>
#include <sys/param.h>
#include <sys/file.h>

#include <avo/intl.h>   /* for NOCATGETS */

typedef struct {
        short           init;
        pathpt          vector;
        char            *env_var;
} vroot_patht;

typedef struct {
        vroot_patht     vroot;
        vroot_patht     path;
        char            full_path[MAXPATHLEN+1];
        char            *vroot_start;
        char            *path_start;
        char            *filename_start;
        int             scan_vroot_first;
        int             cpp_style_path;
} vroot_datat, *vroot_datapt;

static vroot_datat      vroot_data= {
        { 0, NULL, NOCATGETS("VIRTUAL_ROOT")},
```



```
        { 0, NULL, NOCATGETS("PATH")},
        "", NULL, NULL, NULL, 0, 1};

void
add_dir_to_path(register char *path, register pathpt *pointer, register int posi
{
        register int            size= 0;
        register int            length;
        register char           *name;
        register pathcellpt     p;
        pathpt                  new_path;

        if (*pointer != NULL) {
                for (p= &((*pointer)[0]); p->path != NULL; p++, size++);
                if (position < 0)
                        position= size;}
        else
                if (position < 0)
                        position= 0;
        if (position >= size) {
                new_path= (pathpt)calloc((unsigned)(position+2), sizeof(pathcell
                if (*pointer != NULL) {
                        memcpy((char *)new_path,(char *)(*pointer),  size*sizeof
                        free((char *)(*pointer));};
                *pointer= new_path;};
        length= strlen(path);
        name= (char *)malloc((unsigned)(length+1));
        (void)strcpy(name, path);
        if ((*pointer)[position].path != NULL)
                free((*pointer)[position].path);
        (*pointer)[position].path= name;
        (*pointer)[position].length= length;
}

pathpt
parse_path_string(register char *string, register int remove_slash)
{
        register char           *p;
        pathpt                  result= NULL;

        if (string != NULL)
                for (; 1; string= p+1) {
                        if (p= strchr(string, ':')) *p= 0;
                        if ((remove_slash == 1) && !strcmp(string, "/"))
                                add_dir_to_path("", &result, -1);
                        else
                                add_dir_to_path(string, &result, -1);
                        if (p) *p= ':';
                        else return(result);};
        return((pathpt)NULL);
}

char *
get_vroot_name(void)
{
        return(vroot_data.vroot.env_var);
}

char *
get_path_name(void)
{
        return(vroot_data.path.env_var);
}

void
flush_path_cache(void)
```

```
128 {
129         vroot_data.path.init= 0;
130 }

132 void
133 flush_vroot_cache(void)
134 {
135         vroot_data.vroot.init= 0;
136 }

138 void
139 scan_path_first(void)
140 {
141         vroot_data.scan_vroot_first= 0;
142 }

144 void
145 scan_vroot_first(void)
146 {
147         vroot_data.scan_vroot_first= 1;
148 }

150 void
151 set_path_style(int style)
152 {
153         vroot_data.cpp_style_path= style;
154 }

156 char *
157 get_vroot_path(register char **vroot, register char **path, register char **file
158 {
159         if (vroot != NULL) {
160                 if ((*vroot= vroot_data.vroot_start) == NULL)
161                 if ((*vroot= vroot_data.path_start) == NULL)
162                 *vroot= vroot_data.filename_start;};
163         if (path != NULL) {
164                 if ((*path= vroot_data.path_start) == NULL)
165                 *path= vroot_data.filename_start;};
166         if (filename != NULL)
167                 *filename= vroot_data.filename_start;
168         return(vroot_data.full_path);
169 }

171 void
172 translate_with_thunk(register char *filename, int (*thunk) (char *), pathpt path
173 {
174         register pathcellt      *vp;
175         pathcellt               *pp;
176         register pathcellt      *pp1;
177         register char           *p;
178         int                     flags[256];

180 /* Setup path to use */
181         if (rw == rw_write)
182                 pp1= NULL;              /* Do not use path when writing */
183         else {
184                 if (path_vector == VROOT_DEFAULT) {
185                         if (!vroot_data.path.init) {
186                                 vroot_data.path.init= 1;
187                                 vroot_data.path.vector= parse_path_string(getenv
188                         path_vector= vroot_data.path.vector;};
189                 pp1= path_vector == NULL ? NULL : &(path_vector)[0];};

191 /* Setup vroot to use */
192         if (vroot_vector == VROOT_DEFAULT) {
193                 if (!vroot_data.vroot.init) {
```



```
194                         vroot_data.vroot.init= 1;
195                         vroot_data.vroot.vector= parse_path_string(getenv(vroot_
196                 vroot_vector= vroot_data.vroot.vector;};
197         vp= vroot_vector == NULL ? NULL : &(vroot_vector)[0];

199 /* Setup to remember pieces */
200         vroot_data.vroot_start= NULL;
201         vroot_data.path_start= NULL;
202         vroot_data.filename_start= NULL;

204         int flen = strlen(filename);
205         if(flen >= MAXPATHLEN) {
206                 errno = ENAMETOOLONG;
207                 return;
208         }

210         switch ((vp ?1:0) + (pp1 ? 2:0)) {
211             case 0:     /* No path. No vroot. */
212             use_name:
213                 (void)strcpy(vroot_data.full_path, filename);
214                 vroot_data.filename_start= vroot_data.full_path;
215                 (void)(*thunk)(vroot_data.full_path);
216                 return;
217             case 1:     /* No path. Vroot */
218                 if (filename[0] != '/') goto use_name;
219                 for (; vp->path != NULL; vp++) {
220                         if((1 + flen + vp->length) >= MAXPATHLEN) {
221                                 errno = ENAMETOOLONG;
222                                 continue;
223                         }
224                         p= vroot_data.full_path;
225                         (void)strcpy(vroot_data.vroot_start= p, vp->path);
226                         p+= vp->length;
227                         (void)strcpy(vroot_data.filename_start= p, filename);
228                         if ((*thunk)(vroot_data.full_path)) return;};
229                 (void)strcpy(vroot_data.full_path, filename);
230                 return;
231             case 2:     /* Path. No vroot. */
232                 if (vroot_data.cpp_style_path) {
233                         if (filename[0] == '/') goto use_name;
234                 } else {
235                         if (strchr(filename, '/') != NULL) goto use_name;
236                 };
237                 for (; pp1->path != NULL; pp1++) {
238                         p= vroot_data.full_path;
239                         if((1 + flen + pp1->length) >= MAXPATHLEN) {
240                                 errno = ENAMETOOLONG;
241                                 continue;
242                         }
243                         if (vroot_data.cpp_style_path) {
244                                 (void)strcpy(vroot_data.path_start= p, pp1->path
245                                 p+= pp1->length;
246                                 *p++= '/';
247                         } else {
248                                 if (pp1->length != 0) {
249                                         (void)strcpy(vroot_data.path_start= p,
250                                             pp1->path);
251                                         p+= pp1->length;
252                                         *p++= '/';
253                                 };
254                         };
255                         (void)strcpy(vroot_data.filename_start= p, filename);
256                         if ((*thunk)(vroot_data.full_path)) return;};
257                 (void)strcpy(vroot_data.full_path, filename);
258                 return;
259             case 3: {   /* Path. Vroot. */
```

```
260                     int *rel_path, path_len= 1;
261             if (vroot_data.scan_vroot_first == 0) {
262                     for (pp= pp1; pp->path != NULL; pp++) path_len++;
263                     rel_path= flags;
264                     for (path_len-= 2; path_len >= 0; path_len--) rel_path[p
265                     for (; vp->path != NULL; vp++)
266                             for (pp= pp1, path_len= 0; pp->path != NULL; pp+
267                                     int len = 0;
268                                     if (rel_path[path_len] == 1) continue;
269                                     if (pp->path[0] != '/') rel_path[path_le
270                                     p= vroot_data.full_path;
271                                     if ((filename[0] == '/') || (pp->path[0]
272                                             if(vp->length >= MAXPATHLEN) {
273                                                     errno = ENAMETOOLONG;
274                                                     continue;
275                                             }
276                                             (void)strcpy(vroot_data.vroot_st
277                                             len += vp->length;
278                                     };
279                                     if (vroot_data.cpp_style_path) {
280                                             if (filename[0] != '/') {
281                                                     if(1 + len + pp->length
282                                                             errno = ENAMETOO
283                                                             continue;
284                                                     }
285                                                     (void)strcpy(vroot_data.
286                                                     *p++= '/';
287                                                     len += 1 + pp->length;
288                                             };
289                                     } else {
290                                             if (strchr(filename, '/') == NUL
291                                                     if (pp->length != 0) {
292                                                             if(1 + len + pp-
293                                                                     errno =
294                                                                     continue
295                                                             }
296                                                             (void)strcpy(vro
297                                                                 pp->path);
298                                                             p+= pp->length;
299                                                             *p++= '/';
300                                                             len += 1 + pp->l
301                                                     }
302                                             }
303                                     };
304                                     (void)strcpy(vroot_data.filename_start=
305                                     if ((*thunk)(vroot_data.full_path)) retu
306             else { pathcellt *vp1= vp;
307                     for (pp= pp1, path_len= 0; pp->path != NULL; pp++, path_
308                             for (vp= vp1; vp->path != NULL; vp++) {
309                                     int len = 0;
310                                     p= vroot_data.full_path;
311                                     if ((filename[0] == '/') || (pp->path[0]
312                                             if(vp->length >= MAXPATHLEN) {
313                                                     errno = ENAMETOOLONG;
314                                                     continue;
315                                             }
316                                             (void)strcpy(vroot_data.vroot_st
317                                             len += vp->length;
318                                     }
319                                     if (vroot_data.cpp_style_path) {
320                                             if (filename[0] != '/') {
321                                                     if(1 + len + pp->length
322                                                             errno = ENAMETOO
323                                                             continue;
324                                                     }
325                                                     (void)strcpy(vroot_data.
```



```
326                                                     *p++= '/';
327                                                     len += 1 + pp->length;
328                                             }
329                                     } else {
330                                             if (strchr(filename, '/') == NUL
331                                                     if(1 + len + pp->length
332                                                             errno = ENAMETOO
333                                                             continue;
334                                                     }
335                                                     (void)strcpy(vroot_data.
336                                                     *p++= '/';
337                                                     len += 1 + pp->length;
338                                             }
339                                     }
340                                     (void)strcpy(vroot_data.filename_start=
341                                     if ((*thunk)(vroot_data.full_path)) retu
342                     (void)strcpy(vroot_data.full_path, filename);
343                     return;};};
344 }
345 #endif /* ! codereview */
```