```
**********************************************************
   90924 Wed May 20 12:08:17 2015
new/usr/src/cmd/make/bin/main.cc
make: be serial if 'make', parallel if 'dmake', and parallel if '-j' is specifie
**********************************************************
```

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

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

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

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


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

#if defined(TEAMWARE_MAKE_CMN)
#endif

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

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



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

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

// From parallel.cc
#define MAXJOBS_ADJUST_RFE4694000

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


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

/*
 * typedefs & structs
 */

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

#ifdef DMAKE_STATISTICS
static  Boolean         getname_stat = false;
#endif

        static  time_t          start_time;
        static  int             g_argc;
        static  char            **g_argv;
```

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

extern "C" {
        extern  void            dmake_exit_callback(void);
        extern  void            dmake_message_callback(char *);
}

extern  Name            normalize_name(register wchar_t *name_string, register i

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

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

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

static const char       verstring[] = "illumos make";

jmp_buf jmpbuffer;
extern nl_catd catd;

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



```
        register char           *cp;
        char                    make_state_dir[MAXPATHLEN];
        Boolean                 parallel_flag = false;
        char                    *prognameptr;
        char                    *slash_ptr;
        mode_t                  um;
        int                     i;
        struct itimerval        value;
        char                    def_dmakerc_path[MAXPATHLEN];
        Name                    dmake_name, dmake_name2;
        Name                    dmake_value, dmake_value2;
        Property                prop, prop2;
        struct stat             statbuf;
        int                     statval;

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

        (void) setlocale(LC_ALL, "");


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

        catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);

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


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


        textdomain(NOCATGETS("SUNW_SPRO_MAKE"));

        g_argc = argc;
        g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
        for (i = 0; i < argc; i++) {
                g_argv[i] = argv[i];
        }
        g_argv[i] = NULL;

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

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

```
260                 char    *tmp_string;

262                 tmp_current_path = get_current_path();
263                 tmp_string = getmem(strlen(tmp_current_path) + 1 +
264                                 strlen(argv[0]) + 1);
265                 (void) sprintf(tmp_string,
266                                 "%s/%s",
267                                 tmp_current_path,
268                                 argv[0]);
269                 argv_zero_string = strdup(tmp_string);
270                 retmem_mb(tmp_string);
271         }

273         /*
274          * The following flags are reset if we don't have the
275          * (.nse_depinfo or .make.state) files locked and only set
276          * AFTER the file has been locked. This ensures that if the user
277          * interrupts the program while file_lock() is waiting to lock
278          * the file, the interrupt handler doesn't remove a lock
279          * that doesn't belong to us.
280          */
281         make_state_lockfile = NULL;
282         make_state_locked = false;


285         /*
286          * look for last slash char in the path to look at the binary
287          * name. This is to resolve the hard link and invoke make
288          * in svr4 mode.
289          */

291         /* Sun OS make standart */
292         svr4 = false;
293         posix = false;
294         if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
295                 svr4 = false;
296                 posix = true;
297         } else {
298                 prognameptr = strrchr(argv[0], '/');
299                 if(prognameptr) {
300                         prognameptr++;
301                 } else {
302                         prognameptr = argv[0];
303                 }
304                 if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
305                         svr4 = true;
306                         posix = false;
307                 }
308         }
309         if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
310            svr4 = true;
311            posix = false;
312         }

314         /*
315          * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
316          */
317         char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
318         if (dmake_compat_mode_var != NULL) {
319                 if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
320                         gnu_style = true;
321                 }
322                 //svr4 = false;
323                 //posix = false;
324         }
```



```
326         /*
327          * Temporary directory set up.
328          */
329         char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
330         if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
331                 strcpy(mbs_buffer, tmpdir_var);
332                 for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
333                         *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
334                         *tmpdir_var = '\0');
335                 if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
336                         sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
337                                 mbs_buffer, getpid());
338                         int fd = mkstemp(mbs_buffer2);
339                         if (fd >= 0) {
340                                 close(fd);
341                                 unlink(mbs_buffer2);
342                                 tmpdir = strdup(mbs_buffer);
343                         }
344                 }
345         }

347         /* find out if stdout and stderr point to the same place */
348         if (fstat(1, &out_stat) < 0) {
349                 fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
350         }
351         if (fstat(2, &err_stat) < 0) {
352                 fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
353         }
354         if ((out_stat.st_dev == err_stat.st_dev) &&
355             (out_stat.st_ino == err_stat.st_ino)) {
356                 stdout_stderr_same = true;
357         } else {
358                 stdout_stderr_same = false;
359         }
360         /* Make the vroot package scan the path using shell semantics */
361         set_path_style(0);

363         setup_char_semantics();

365         setup_for_projectdir();

367         /*
368          * If running with .KEEP_STATE, curdir will be set with
369          * the connected directory.
370          */
371         (void) atexit(cleanup_after_exit);

373         load_cached_names();

375 /*
376  *      Set command line flags
377  */
378         setup_makeflags_argv();
379         read_command_options(mf_argc, mf_argv);
380         read_command_options(argc, argv);
381         if (debug_level > 0) {
382                 cp = getenv(makeflags->string_mb);
383                 (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
384         }

386         setup_interrupt(handle_interrupt);

388         read_files_and_state(argc, argv);

390         /*
391          * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
```

```
392          */
393         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
394         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
395         prop2 = get_prop(dmake_name2->prop, macro_prop);
396         if (prop2 == NULL) {
397                 /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
398                 output_mode = txt1_mode;
399         } else {
400                 dmake_value2 = prop2->body.macro.value;
401                 if ((dmake_value2 == NULL) ||
402                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
403                         output_mode = txt1_mode;
404                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
405                         output_mode = txt2_mode;
406                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))
407                         output_mode = html1_mode;
408                 } else {
409                         warning(catgets(catd, 1, 352, "Unsupported value '%s' fo
410                               dmake_value2->string_mb);
411                 }
412         }
413         /*
414          * Find the dmake_mode: parallel, or serial.
415          */
416     if ((!pmake_cap_r_specified) &&
417         (!pmake_machinesfile_specified)) {
418         char *s = strdup(argv[0]);
419
420 #endif /* ! codereview */
421         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
422         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
423         prop2 = get_prop(dmake_name2->prop, macro_prop);
424         // If we're invoked as 'make' run serially, regardless of DMAKE_MODE
425         // If we're invoked as 'make' but passed -j, run parallel
426         // If we're invoked as 'dmake', without DMAKE_MODE, default parallel
427         // If we're invoked as 'dmake' and DMAKE_MODE is set, honour it.
428         if ((strcmp(basename(s), NOCATGETS("make")) == 0) &&
429             !dmake_max_jobs_specified) {
430                 dmake_mode_type = serial_mode;
431                 no_parallel = true;
432         } else if (prop2 == NULL) {
433                 /* DMAKE_MODE not defined, default based on our name */
434                 char *s = strdup(argv[0]);

436                 if (strcmp(basename(s), NOCATGETS("dmake")) == 0) {
 43         if (prop2 == NULL) {
 44                 /* DMAKE_MODE not defined, default to parallel mode */
437                         dmake_mode_type = parallel_mode;
438                         no_parallel = false;
439                 }
440 #endif /* ! codereview */
441         } else {
442                 dmake_value2 = prop2->body.macro.value;
443                 if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel"))) {
444                         dmake_mode_type = parallel_mode;
445                         no_parallel = false;
446                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
447                         dmake_mode_type = serial_mode;
448                         no_parallel = true;
449                 } else {
450                         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
451                 }
452         }
453         free(s);

454     }
```



```
456         parallel_flag = true;
457         putenv(strdup(NOCATGETS("DMAKE_CHILD=TRUE")));

459 //
460 // If dmake is running with -t option, set dmake_mode_type to serial.
461 // This is done because doname() calls touch_command() that runs serially.
462 // If we do not do that, maketool will have problems.
463 //
464         if(touch) {
465                 dmake_mode_type = serial_mode;
466                 no_parallel = true;
467         }

469         /*
470          * Check whether stdout and stderr are physically same.
471          * This is in order to decide whether we need to redirect
472          * stderr separately from stdout.
473          * This check is performed only if __DMAKE_SEPARATE_STDERR
474          * is not set. This variable may be used in order to preserve
475          * the 'old' behaviour.
476          */
477         out_err_same = true;
478         char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
479         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
480                 struct stat stdout_stat;
481                 struct stat stderr_stat;
482                 if( (fstat(1, &stdout_stat) == 0)
483                  && (fstat(2, &stderr_stat) == 0) )
484                 {
485                         if( (stdout_stat.st_dev != stderr_stat.st_dev)
486                          || (stdout_stat.st_ino != stderr_stat.st_ino) )
487                         {
488                                 out_err_same = false;
489                         }
490                 }
491         }

493
494 /*
495  *      Enable interrupt handler for alarms
496  */
497         (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

499 /*
500  *      Check if make should report
501  */
502         if (getenv(sunpro_dependencies->string_mb) != NULL) {
503                 FILE    *report_file;

505                 report_dependency("");
506                 report_file = get_report_file();
507                 if ((report_file != NULL) && (report_file != (FILE*)-1)) {
508                         (void) fprintf(report_file, "\n");
509                 }
510         }

512 /*
513  *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
514  */
515         if (keep_state) {
516                 maybe_append_prop(sunpro_dependencies, macro_prop)->
517                   body.macro.exported = true;
518         } else {
519                 maybe_append_prop(sunpro_dependencies, macro_prop)->
520                   body.macro.exported = false;
```

```
521         }

523         working_on_targets = true;
524         if (trace_status) {
525                 dump_make_state();
526                 fclose(stdout);
527                 fclose(stderr);
528                 exit_status = 0;
529                 exit(0);
530         }
531         if (list_all_targets) {
532                 dump_target_list();
533                 fclose(stdout);
534                 fclose(stderr);
535                 exit_status = 0;
536                 exit(0);
537         }
538         trace_reader = false;

540         /*
541          * Set temp_file_directory to the directory the .make.state
542          * file is written to.
543          */
544         if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
545                 temp_file_directory = strdup(get_current_path());
546         } else {
547                 *slash_ptr = (int) nul_char;
548                 (void) strcpy(make_state_dir, make_state->string_mb);
549                 *slash_ptr = (int) slash_char;
550                    /* when there is only one slash and it's the first
551                    ** character, make_state_dir should point to '/'.
552                    */
553                 if(make_state_dir[0] == '\0') {
554                    make_state_dir[0] = '/';
555                    make_state_dir[1] = '\0';
556                 }
557                 if (make_state_dir[0] == (int) slash_char) {
558                         temp_file_directory = strdup(make_state_dir);
559                 } else {
560                         char    tmp_current_path2[MAXPATHLEN];
561
562                         (void) sprintf(tmp_current_path2,
563                                         "%s/%s",
564                                         get_current_path(),
565                                         make_state_dir);
566                         temp_file_directory = strdup(tmp_current_path2);
567                 }
568         }


571         report_dir_enter_leave(true);

573         make_targets(argc, argv, parallel_flag);

575         report_dir_enter_leave(false);

577         if (build_failed_ever_seen) {
578                 if (posix) {
579                         exit_status = 1;
580                 }
581                 exit(1);
582         }
583         exit_status = 0;
584         exit(0);
585         /* NOTREACHED */
586 }
```

***unchanged_portion_omitted_***



```
1280 /*
1281  *      parse_command_option(ch)
1282  *
1283  *      Parse make command line options.
1284  *
1285  *      Return value:
1286  *                              Indicates if any -f -c or -M were seen
1287  *
1288  *      Parameters:
1289  *              ch              The character to parse
1290  *
1291  *      Static variables used:
1292  *              dmake_group_specified   Set for make -g
1293  *              dmake_max_jobs_specified        Set for make -j
1294  *              dmake_mode_specified    Set for make -m
1295  *              dmake_add_mode_specified        Set for make -x
1296  *              dmake_compat_mode_specified     Set for make -x SUN_MAKE_COMPAT_
1297  *              dmake_output_mode_specified     Set for make -x DMAKE_OUTPUT_MOD
1298  *              dmake_odir_specified    Set for make -o
1299  *              dmake_rcfile_specified  Set for make -c
1300  *              env_wins                Set for make -e
1301  *              ignore_default_mk       Set for make -r
1302  *              trace_status            Set for make -p
1303  *
1304  *      Global variables used:
1305  *              .make.state path & name set for make -K
1306  *              continue_after_error    Set for make -k
1307  *              debug_level             Set for make -d
1308  *              do_not_exec_rule        Set for make -n
1309  *              filter_stderr           Set for make -X
1310  *              ignore_errors_all       Set for make -i
1311  *              no_parallel             Set for make -R
1312  *              quest                   Set for make -q
1313  *              read_trace_level        Set for make -D
1314  *              report_dependencies     Set for make -P
1315  *              send_mtool_msgs         Set for make -K
1316  *              silent_all              Set for make -s
1317  *              touch                   Set for make -t
1318  */
1319 static int
1320 parse_command_option(register char ch)
1321 {
1322         static int              invert_next = 0;
1323         int                     invert_this = invert_next;

1325         invert_next = 0;
1326         switch (ch) {
1327         case '-':                       /* Ignore "--" */
1328                 return 0;
1329         case '~':                       /* Invert next option */
1330                 invert_next = 1;
1331                 return 0;
1332         case 'B':                       /* Obsolete */
1333                 return 0;
1334         case 'b':                       /* Obsolete */
1335                 return 0;
1336         case 'c':                       /* Read alternative dmakerc file */
1337                 if (invert_this) {
1338                         dmake_rcfile_specified = false;
1339                 } else {
1340                         dmake_rcfile_specified = true;
1341                 }
1342                 return 2;
1343         case 'D':                       /* Show lines read */
1344                 if (invert_this) {
```

```
1345                        read_trace_level--;
1346                } else {
1347                        read_trace_level++;
1348                }
1349                return 0;
1350        case 'd':                       /* Debug flag */
1351                if (invert_this) {
1352                        debug_level--;
1353                } else {
1354                        debug_level++;
1355                }
1356                return 0;
1357        case 'e':                       /* Environment override flag */
1358                if (invert_this) {
1359                        env_wins = false;
1360                } else {
1361                        env_wins = true;
1362                }
1363                return 0;
1364        case 'f':                       /* Read alternative makefile(s) */
1365                return 1;
1366        case 'g':                       /* Use alternative DMake group */
1367                if (invert_this) {
1368                        dmake_group_specified = false;
1369                } else {
1370                        dmake_group_specified = true;
1371                }
1372                return 4;
1373        case 'i':                       /* Ignore errors */
1374                if (invert_this) {
1375                        ignore_errors_all = false;
1376                } else {
1377                        ignore_errors_all = true;
1378                }
1379                return 0;
1380        case 'j':                       /* Use alternative DMake max jobs */
1381                if (invert_this) {
1382                        dmake_max_jobs_specified = false;
1383                } else {
1384                        dmake_mode_type = parallel_mode;
1385                        no_parallel = false;
1386 #endif /* ! codereview */
1387                        dmake_max_jobs_specified = true;
1388                }
1389                return 8;
1390        case 'K':                       /* Read alternative .make.state */
1391                return 256;
1392        case 'k':                       /* Keep making even after errors */
1393                if (invert_this) {
1394                        continue_after_error = false;
1395                } else {
1396                        continue_after_error = true;
1397                        continue_after_error_ever_seen = true;
1398                }
1399                return 0;
1400        case 'M':                       /* Read alternative make.machines file
1401                if (invert_this) {
1402                        pmake_machinesfile_specified = false;
1403                } else {
1404                        pmake_machinesfile_specified = true;
1405                        dmake_mode_type = parallel_mode;
1406                        no_parallel = false;
1407                }
1408                return 16;
1409        case 'm':                       /* Use alternative DMake build mode */
1410                if (invert_this) {
```



```
1411                        dmake_mode_specified = false;
1412                } else {
1413                        dmake_mode_specified = true;
1414                }
1415                return 32;
1416        case 'x':                       /* Use alternative DMake mode */
1417                if (invert_this) {
1418                        dmake_add_mode_specified = false;
1419                } else {
1420                        dmake_add_mode_specified = true;
1421                }
1422                return 1024;
1423        case 'N':                       /* Reverse -n */
1424                if (invert_this) {
1425                        do_not_exec_rule = true;
1426                } else {
1427                        do_not_exec_rule = false;
1428                }
1429                return 0;
1430        case 'n':                       /* Print, not exec commands */
1431                if (invert_this) {
1432                        do_not_exec_rule = false;
1433                } else {
1434                        do_not_exec_rule = true;
1435                }
1436                return 0;
1437        case 'O':                       /* Send job start & result msgs */
1438                if (invert_this) {
1439                        send_mtool_msgs = false;
1440                } else {
1441                }
1442                return 128;
1443        case 'o':                       /* Use alternative dmake output dir */
1444                if (invert_this) {
1445                        dmake_odir_specified = false;
1446                } else {
1447                        dmake_odir_specified = true;
1448                }
1449                return 512;
1450        case 'P':                       /* Print for selected targets */
1451                if (invert_this) {
1452                        report_dependencies_level--;
1453                } else {
1454                        report_dependencies_level++;
1455                }
1456                return 0;
1457        case 'p':                       /* Print description */
1458                if (invert_this) {
1459                        trace_status = false;
1460                        do_not_exec_rule = false;
1461                } else {
1462                        trace_status = true;
1463                        do_not_exec_rule = true;
1464                }
1465                return 0;
1466        case 'q':                       /* Question flag */
1467                if (invert_this) {
1468                        quest = false;
1469                } else {
1470                        quest = true;
1471                }
1472                return 0;
1473        case 'R':                       /* Don't run in parallel */
1474                if (invert_this) {
1475                        pmake_cap_r_specified = false;
1476                        no_parallel = false;
```

```
1477                } else {
1478                        pmake_cap_r_specified = true;
1479                        dmake_mode_type = serial_mode;
1480                        no_parallel = true;
1481                }
1482                return 0;
1483        case 'r':                       /* Turn off internal rules */
1484                if (invert_this) {
1485                        ignore_default_mk = false;
1486                } else {
1487                        ignore_default_mk = true;
1488                }
1489                return 0;
1490        case 'S':                       /* Reverse -k */
1491                if (invert_this) {
1492                        continue_after_error = true;
1493                } else {
1494                        continue_after_error = false;
1495                        stop_after_error_ever_seen = true;
1496                }
1497                return 0;
1498        case 's':                       /* Silent flag */
1499                if (invert_this) {
1500                        silent_all = false;
1501                } else {
1502                        silent_all = true;
1503                }
1504                return 0;
1505        case 'T':                       /* Print target list */
1506                if (invert_this) {
1507                        list_all_targets = false;
1508                        do_not_exec_rule = false;
1509                } else {
1510                        list_all_targets = true;
1511                        do_not_exec_rule = true;
1512                }
1513                return 0;
1514        case 't':                       /* Touch flag */
1515                if (invert_this) {
1516                        touch = false;
1517                } else {
1518                        touch = true;
1519                }
1520                return 0;
1521        case 'u':                       /* Unconditional flag */
1522                if (invert_this) {
1523                        build_unconditional = false;
1524                } else {
1525                        build_unconditional = true;
1526                }
1527                return 0;
1528        case 'V':                      /* SVR4 mode */
1529                svr4 = true;
1530                return 0;
1531        case 'v':                      /* Version flag */
1532                if (invert_this) {
1533                } else {
1534                        fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
1535                        exit_status = 0;
1536                        exit(0);
1537                }
1538                return 0;
1539        case 'w':                       /* Unconditional flag */
1540                if (invert_this) {
1541                        report_cwd = false;
1542                } else {
```



```
1543                        report_cwd = true;
1544                }
1545                return 0;
1546 #if 0
1547        case 'X':                       /* Filter stdout */
1548                if (invert_this) {
1549                        filter_stderr = false;
1550                } else {
1551                        filter_stderr = true;
1552                }
1553                return 0;
1554 #endif
1555        default:
1556                break;
1557        }
1558        return 0;
1559 }

1561 /*
1562  *      setup_for_projectdir()
1563  *
1564  *      Read the PROJECTDIR variable, if defined, and set the sccs path
1565  *
1566  *      Parameters:
1567  *
1568  *      Global variables used:
1569  *              sccs_dir_path   Set to point to SCCS dir to use
1570  */
1571 static void
1572 setup_for_projectdir(void)
1573 {
1574 static char     path[MAXPATHLEN];
1575 char            cwdpath[MAXPATHLEN];
1576 uid_t uid;
1577 int   done=0;

1579        /* Check if we should use PROJECTDIR when reading the SCCS dir. */
1580        sccs_dir_path = getenv(NOCATGETS("PROJECTDIR"));
1581        if ((sccs_dir_path != NULL) &&
1582            (sccs_dir_path[0] != (int) slash_char)) {
1583                struct passwd *pwent;

1585             {
1586                uid = getuid();
1587                pwent = getpwuid(uid);
1588                if (pwent == NULL) {
1589                   fatal(catgets(catd, 1, 188, "Bogus USERID "));
1590                }
1591                if ((pwent = getpwnam(sccs_dir_path)) == NULL) {
1592                        /*empty block : it'll go & check cwd  */
1593                }
1594                else {
1595                  (void) sprintf(path, NOCATGETS("%s/src"), pwent->pw_dir);
1596                  if (access(path, F_OK) == 0) {
1597                        sccs_dir_path = path;
1598                        done = 1;
1599                  } else {
1600                        (void) sprintf(path, NOCATGETS("%s/source"), pwent->pw_d
1601                        if (access(path, F_OK) == 0) {
1602                                sccs_dir_path = path;
1603                                done = 1;
1604                        }
1605                   }
1606                }
1607                if (!done) {
1608                    if (getcwd(cwdpath, MAXPATHLEN - 1 )) {
```

```
1610                        (void) sprintf(path, NOCATGETS("%s/%s"), cwdpath,sccs_dir
1611                        if (access(path, F_OK) == 0) {
1612                                sccs_dir_path = path;
1613                                done = 1;
1614                         } else {
1615                                fatal(catgets(catd, 1, 189, "Bogus PROJECTDIR '%
1616                         }
1617                    }
1618                }
1619           }
1620        }
1621 }

1623 /*
1624  *      set_sgs_support()
1625  *
1626  *      Add the libmakestate.so.1 lib to the env var SGS_SUPPORT
1627  *        if it's not already in there.
1628  *      The SGS_SUPPORT env var and libmakestate.so.1 is used by
1629  *        the linker ld to report .make.state info back to make.
1630  *
1631  * In the new world we always will set the 32-bit and 64-bit versions of this
1632  * variable explicitly so that we can take into account the correct isa and our
1633  * prefix. So say that the prefix was /opt/local. Then we would want to search
1634  * /opt/local/lib/libmakestate.so.1:libmakestate.so.1. We still want to search
1635  * the original location just as a safety measure.
1636  */
1637 static void
1638 set_sgs_support()
1639 {
1640        int             len;
1641        char            *newpath, *newpath64;
1642        char            *oldpath, *oldpath64;
1643        static char     *prev_path, *prev_path64;

1645        oldpath = getenv(LD_SUPPORT_ENV_VAR_32);
1646        if (oldpath == NULL) {
1647                len = snprintf(NULL, 0, "%s=%s/%s/%s:%s",
1648                    LD_SUPPORT_ENV_VAR_32,
1649                    MAKE_PREFIX,
1650                    LD_SUPPORT_MAKE_LIB_DIR,
1651                    LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB) + 1;
1652                newpath = (char *) malloc(len);
1653                sprintf(newpath, "%s=%s/%s/%s:%s",
1654                    LD_SUPPORT_ENV_VAR_32,
1655                    MAKE_PREFIX,
1656                    LD_SUPPORT_MAKE_LIB_DIR,
1657                    LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB);
1658        } else {
1659                len = snprintf(NULL, 0, "%s=%s:%s/%s/%s:%s",
1660                    LD_SUPPORT_ENV_VAR_32, oldpath, MAKE_PREFIX,
1661                    LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB,
1662                    LD_SUPPORT_MAKE_LIB) + 1;
1663                newpath = (char *) malloc(len);
1664                sprintf(newpath, "%s=%s:%s/%s/%s:%s",
1665                    LD_SUPPORT_ENV_VAR_32, oldpath, MAKE_PREFIX,
1666                    LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB,
1667                    LD_SUPPORT_MAKE_LIB);
1668        }

1670        oldpath64 = getenv(LD_SUPPORT_ENV_VAR_64);
1671        if (oldpath64 == NULL) {
1672                len = snprintf(NULL, 0, "%s=%s/%s/%s/%s:%s",
1673                    LD_SUPPORT_ENV_VAR_64, MAKE_PREFIX, LD_SUPPORT_MAKE_LIB_DIR,
1674                    LD_SUPPORT_MAKE_LIB_DIR_64, LD_SUPPORT_MAKE_LIB,
```



```
1675                        LD_SUPPORT_MAKE_LIB) + 1;
1676                newpath64 = (char *) malloc(len);
1677                sprintf(newpath64, "%s=%s/%s/%s/%s:%s",
1678                    LD_SUPPORT_ENV_VAR_64, MAKE_PREFIX, LD_SUPPORT_MAKE_LIB_DIR,
1679                    LD_SUPPORT_MAKE_LIB_DIR_64, LD_SUPPORT_MAKE_LIB,
1680                    LD_SUPPORT_MAKE_LIB);
1681        } else {
1682                len = snprintf(NULL, 0, "%s=%s:%s/%s/%s/%s:%s",
1683                    LD_SUPPORT_ENV_VAR_64, oldpath64, MAKE_PREFIX,
1684                    LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB_DIR_64,
1685                    LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB) + 1;
1686                newpath64 = (char *) malloc(len);
1687                sprintf(newpath64, "%s=%s:%s/%s/%s/%s:%s",
1688                    LD_SUPPORT_ENV_VAR_64, oldpath64, MAKE_PREFIX,
1689                    LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB_DIR_64,
1690                    LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB);
1691        }

1693        putenv(newpath);
1694        if (prev_path) {
1695                free(prev_path);
1696        }
1697        prev_path = newpath;

1699        putenv(newpath64);
1700        if (prev_path64) {
1701                free(prev_path64);
1702        }
1703        prev_path64 = newpath64;
1704 }

1706 /*
1707  *      read_files_and_state(argc, argv)
1708  *
1709  *      Read the makefiles we care about and the environment
1710  *      Also read the = style command line options
1711  *
1712  *      Parameters:
1713  *              argc            You know what this is
1714  *              argv            You know what this is
1715  *
1716  *      Static variables used:
1717  *              env_wins        make -e, determines if env vars are RO
1718  *              ignore_default_mk make -r, determines if make.rules is read
1719  *              not_auto_depen  dwight
1720  *
1721  *      Global variables used:
1722  *              default_target_to_build Set to first proper target from file
1723  *              do_not_exec_rule Set to false when makfile is made
1724  *              dot             The Name ".", used to read current dir
1725  *              empty_name      The Name "", use as macro value
1726  *              keep_state      Set if KEEP_STATE is in environment
1727  *              make_state      The Name ".make.state", used to read file
1728  *              makefile_type   Set to type of file being read
1729  *              makeflags       The Name "MAKEFLAGS", used to set macro value
1730  *              not_auto        dwight
1731  *              read_trace_level Checked to se if the reader should trace
1732  *              report_dependencies If -P is on we do not read .make.state
1733  *              trace_reader    Set if reader should trace
1734  *              virtual_root    The Name "VIRTUAL_ROOT", used to check value
1735  */
1736 static void
1737 read_files_and_state(int argc, char **argv)
1738 {
1739        wchar_t                 buffer[1000];
1740        wchar_t                 buffer_posix[1000];
```

```
1741         register char           ch;
1742         register char           *cp;
1743         Property                def_make_macro = NULL;
1744         Name                    def_make_name;
1745         Name                    default_makefile;
1746         String_rec              dest;
1747         wchar_t                 destbuffer[STRING_BUFFER_LENGTH];
1748         register int            i;
1749         register int            j;
1750         Name                    keep_state_name;
1751         int                     length;
1752         Name                    Makefile;
1753         register Property       macro;
1754         struct stat             make_state_stat;
1755         Name                    makefile_name;
1756         register int            makefile_next = 0;
1757         register Boolean        makefile_read = false;
1758         String_rec              makeflags_string;
1759         String_rec              makeflags_string_posix;
1760         String_rec *            makeflags_string_current;
1761         Name                    makeflags_value_saved;
1762         register Name           name;
1763         Name                    new_make_value;
1764         Boolean                 save_do_not_exec_rule;
1765         Name                    sdotMakefile;
1766         Name                    sdotmakefile_name;
1767         static wchar_t          state_file_str;
1768         static char             state_file_str_mb[MAXPATHLEN];
1769         static struct _Name     state_filename;
1770         Boolean                 temp;
1771         char                    tmp_char;
1772         wchar_t                 *tmp_wcs_buffer;
1773         register Name           value;
1774         ASCII_Dyn_Array         makeflags_and_macro;
1775         Boolean                 is_xpg4;

1777 /*
1778  *      Remember current mode. It may be changed after reading makefile
1779  *      and we will have to correct MAKEFLAGS variable.
1780  */
1781         is_xpg4 = posix;

1783         MBSTOWCS(wcs_buffer, NOCATGETS("KEEP_STATE"));
1784         keep_state_name = GETNAME(wcs_buffer, FIND_LENGTH);
1785         MBSTOWCS(wcs_buffer, NOCATGETS("Makefile"));
1786         Makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1787         MBSTOWCS(wcs_buffer, NOCATGETS("makefile"));
1788         makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
1789         MBSTOWCS(wcs_buffer, NOCATGETS("s.makefile"));
1790         sdotmakefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
1791         MBSTOWCS(wcs_buffer, NOCATGETS("s.Makefile"));
1792         sdotMakefile = GETNAME(wcs_buffer, FIND_LENGTH);

1794 /*
1795  *      initialize global dependency entry for .NOT_AUTO
1796  */
1797         not_auto_depen->next = NULL;
1798         not_auto_depen->name = not_auto;
1799         not_auto_depen->automatic = not_auto_depen->stale = false;

1801 /*
1802  *      Read internal definitions and rules.
1803  */
1804         if (read_trace_level > 1) {
1805                 trace_reader = true;
1806         }
```



```
1807         if (!ignore_default_mk) {
1808                 if (svr4) {
1809                         MBSTOWCS(wcs_buffer, NOCATGETS("svr4.make.rules"));
1810                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1811                 } else {
1812                         MBSTOWCS(wcs_buffer, NOCATGETS("make.rules"));
1813                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1814                 }
1815                 default_makefile->stat.is_file = true;

1817                 (void) read_makefile(default_makefile,
1818                                      true,
1819                                      false,
1820                                      true);
1821         }

1823         /*
1824          * If the user did not redefine the MAKE macro in the
1825          * default makefile (make.rules), then we'd like to
1826          * change the macro value of MAKE to be some form
1827          * of argv[0] for recursive MAKE builds.
1828          */
1829         MBSTOWCS(wcs_buffer, NOCATGETS("MAKE"));
1830         def_make_name = GETNAME(wcs_buffer, wslen(wcs_buffer));
1831         def_make_macro = get_prop(def_make_name->prop, macro_prop);
1832         if ((def_make_macro != NULL) &&
1833             (IS_EQUAL(def_make_macro->body.macro.value->string_mb,
1834                       NOCATGETS("make")))) {
1835                 MBSTOWCS(wcs_buffer, argv_zero_string);
1836                 new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
1837                 (void) SETVAR(def_make_name,
1838                               new_make_value,
1839                               false);
1840         }

1842         default_target_to_build = NULL;
1843         trace_reader = false;

1845 /*
1846  *      Read environment args. Let file args which follow override unless
1847  *      -e option seen. If -e option is not mentioned.
1848  */
1849         read_environment(env_wins);
1850         if (getvar(virtual_root)->hash.length == 0) {
1851                 maybe_append_prop(virtual_root, macro_prop)
1852                   ->body.macro.exported = true;
1853                 MBSTOWCS(wcs_buffer, "/");
1854                 (void) SETVAR(virtual_root,
1855                               GETNAME(wcs_buffer, FIND_LENGTH),
1856                               false);
1857         }

1859 /*
1860  * We now scan mf_argv and argv to see if we need to set
1861  * any of the DMake-added options/variables in MAKEFLAGS.
1862  */

1864         makeflags_and_macro.start = 0;
1865         makeflags_and_macro.size = 0;
1866         enter_argv_values(mf_argc, mf_argv, &makeflags_and_macro);
1867         enter_argv_values(argc, argv, &makeflags_and_macro);

1869 /*
1870  *      Set MFLAGS and MAKEFLAGS
1871  *
1872  *      Before reading makefile we do not know exactly which mode
```

```
1873  *      (posix or not) is used. So prepare two MAKEFLAGS strings
1874  *      for both posix and solaris modes because they are different.
1875  */
1876         INIT_STRING_FROM_STACK(makeflags_string, buffer);
1877         INIT_STRING_FROM_STACK(makeflags_string_posix, buffer_posix);
1878         append_char((int) hyphen_char, &makeflags_string);
1879         append_char((int) hyphen_char, &makeflags_string_posix);

1881         switch (read_trace_level) {
1882         case 2:
1883                 append_char('D', &makeflags_string);
1884                 append_char('D', &makeflags_string_posix);
1885         case 1:
1886                 append_char('D', &makeflags_string);
1887                 append_char('D', &makeflags_string_posix);
1888         }
1889         switch (debug_level) {
1890         case 2:
1891                 append_char('d', &makeflags_string);
1892                 append_char('d', &makeflags_string_posix);
1893         case 1:
1894                 append_char('d', &makeflags_string);
1895                 append_char('d', &makeflags_string_posix);
1896         }
1897         if (env_wins) {
1898                 append_char('e', &makeflags_string);
1899                 append_char('e', &makeflags_string_posix);
1900         }
1901         if (ignore_errors_all) {
1902                 append_char('i', &makeflags_string);
1903                 append_char('i', &makeflags_string_posix);
1904         }
1905         if (continue_after_error) {
1906                 if (stop_after_error_ever_seen) {
1907                         append_char('S', &makeflags_string_posix);
1908                         append_char((int) space_char, &makeflags_string_posix);
1909                         append_char((int) hyphen_char, &makeflags_string_posix);
1910                 }
1911                 append_char('k', &makeflags_string);
1912                 append_char('k', &makeflags_string_posix);
1913         } else {
1914                 if (stop_after_error_ever_seen
1915                     && continue_after_error_ever_seen) {
1916                         append_char('k', &makeflags_string_posix);
1917                         append_char((int) space_char, &makeflags_string_posix);
1918                         append_char((int) hyphen_char, &makeflags_string_posix);
1919                         append_char('S', &makeflags_string_posix);
1920                 }
1921         }
1922         if (do_not_exec_rule) {
1923                 append_char('n', &makeflags_string);
1924                 append_char('n', &makeflags_string_posix);
1925         }
1926         switch (report_dependencies_level) {
1927         case 4:
1928                 append_char('P', &makeflags_string);
1929                 append_char('P', &makeflags_string_posix);
1930         case 3:
1931                 append_char('P', &makeflags_string);
1932                 append_char('P', &makeflags_string_posix);
1933         case 2:
1934                 append_char('P', &makeflags_string);
1935                 append_char('P', &makeflags_string_posix);
1936         case 1:
1937                 append_char('P', &makeflags_string);
1938                 append_char('P', &makeflags_string_posix);
```



```
1939         }
1940         if (trace_status) {
1941                 append_char('p', &makeflags_string);
1942                 append_char('p', &makeflags_string_posix);
1943         }
1944         if (quest) {
1945                 append_char('q', &makeflags_string);
1946                 append_char('q', &makeflags_string_posix);
1947         }
1948         if (silent_all) {
1949                 append_char('s', &makeflags_string);
1950                 append_char('s', &makeflags_string_posix);
1951         }
1952         if (touch) {
1953                 append_char('t', &makeflags_string);
1954                 append_char('t', &makeflags_string_posix);
1955         }
1956         if (build_unconditional) {
1957                 append_char('u', &makeflags_string);
1958                 append_char('u', &makeflags_string_posix);
1959         }
1960         if (report_cwd) {
1961                 append_char('w', &makeflags_string);
1962                 append_char('w', &makeflags_string_posix);
1963         }
1964         /* -c dmake_rcfile */
1965         if (dmake_rcfile_specified) {
1966                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
1967                 dmake_rcfile = GETNAME(wcs_buffer, FIND_LENGTH);
1968                 append_makeflags_string(dmake_rcfile, &makeflags_string);
1969                 append_makeflags_string(dmake_rcfile, &makeflags_string_posix);
1970         }
1971         /* -g dmake_group */
1972         if (dmake_group_specified) {
1973                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
1974                 dmake_group = GETNAME(wcs_buffer, FIND_LENGTH);
1975                 append_makeflags_string(dmake_group, &makeflags_string);
1976                 append_makeflags_string(dmake_group, &makeflags_string_posix);
1977         }
1978         /* -j dmake_max_jobs */
1979         if (dmake_max_jobs_specified) {
1980                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
1981                 dmake_max_jobs = GETNAME(wcs_buffer, FIND_LENGTH);
1982                 append_makeflags_string(dmake_max_jobs, &makeflags_string);
1983                 append_makeflags_string(dmake_max_jobs, &makeflags_string_posix)
1984         }
1985         /* -m dmake_mode */
1986         if (dmake_mode_specified) {
1987                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
1988                 dmake_mode = GETNAME(wcs_buffer, FIND_LENGTH);
1989                 append_makeflags_string(dmake_mode, &makeflags_string);
1990                 append_makeflags_string(dmake_mode, &makeflags_string_posix);
1991         }
1992         /* -x dmake_compat_mode */
1993 //      if (dmake_compat_mode_specified) {
1994 //              MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE_COMPAT_MODE"));
1995 //              dmake_compat_mode = GETNAME(wcs_buffer, FIND_LENGTH);
1996 //              append_makeflags_string(dmake_compat_mode, &makeflags_string);
1997 //              append_makeflags_string(dmake_compat_mode, &makeflags_string_pos
1998 //      }
1999         /* -x dmake_output_mode */
2000         if (dmake_output_mode_specified) {
2001                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
2002                 dmake_output_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2003                 append_makeflags_string(dmake_output_mode, &makeflags_string);
2004                 append_makeflags_string(dmake_output_mode, &makeflags_string_pos
```

```
2005         }
2006         /* -o dmake_odir */
2007         if (dmake_odir_specified) {
2008                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2009                 dmake_odir = GETNAME(wcs_buffer, FIND_LENGTH);
2010                 append_makeflags_string(dmake_odir, &makeflags_string);
2011                 append_makeflags_string(dmake_odir, &makeflags_string_posix);
2012         }
2013         /* -M pmake_machinesfile */
2014         if (pmake_machinesfile_specified) {
2015                 MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
2016                 pmake_machinesfile = GETNAME(wcs_buffer, FIND_LENGTH);
2017                 append_makeflags_string(pmake_machinesfile, &makeflags_string);
2018                 append_makeflags_string(pmake_machinesfile, &makeflags_string_po
2019         }
2020         /* -R */
2021         if (pmake_cap_r_specified) {
2022                 append_char((int) space_char, &makeflags_string);
2023                 append_char((int) hyphen_char, &makeflags_string);
2024                 append_char('R', &makeflags_string);
2025                 append_char((int) space_char, &makeflags_string_posix);
2026                 append_char((int) hyphen_char, &makeflags_string_posix);
2027                 append_char('R', &makeflags_string_posix);
2028         }

2030 /*
2031  *      Make sure MAKEFLAGS is exported
2032  */
2033         maybe_append_prop(makeflags, macro_prop)->
2034           body.macro.exported = true;

2036         if (makeflags_string.buffer.start[1] != (int) nul_char) {
2037                 if (makeflags_string.buffer.start[1] != (int) space_char) {
2038                         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2039                         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2040                                       GETNAME(makeflags_string.buffer.start,
2041                                               FIND_LENGTH),
2042                                       false);
2043                 } else {
2044                         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2045                         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2046                                       GETNAME(makeflags_string.buffer.start + 2,
2047                                               FIND_LENGTH),
2048                                       false);
2049                 }
2050         }

2052 /*
2053  *      Add command line macro to POSIX makeflags_string
2054  */
2055         if (makeflags_and_macro.start) {
2056                 tmp_char = (char) space_char;
2057                 cp = makeflags_and_macro.start;
2058                 do {
2059                         append_char(tmp_char, &makeflags_string_posix);
2060                 } while ( tmp_char = *cp++ );
2061                 retmem_mb(makeflags_and_macro.start);
2062         }

2064 /*
2065  *      Now set the value of MAKEFLAGS macro in accordance
2066  *      with current mode.
2067  */
2068         macro = maybe_append_prop(makeflags, macro_prop);
2069         temp = (Boolean) macro->body.macro.read_only;
2070         macro->body.macro.read_only = false;
```



```
2071         if(posix || gnu_style) {
2072                 makeflags_string_current = &makeflags_string_posix;
2073         } else {
2074                 makeflags_string_current = &makeflags_string;
2075         }
2076         if (makeflags_string_current->buffer.start[1] == (int) nul_char) {
2077                 makeflags_value_saved =
2078                         GETNAME( makeflags_string_current->buffer.start + 1
2079                                , FIND_LENGTH
2080                                );
2081         } else {
2082                 if (makeflags_string_current->buffer.start[1] != (int) space_cha
2083                         makeflags_value_saved =
2084                                 GETNAME( makeflags_string_current->buffer.start
2085                                        , FIND_LENGTH
2086                                        );
2087                 } else {
2088                         makeflags_value_saved =
2089                                 GETNAME( makeflags_string_current->buffer.start
2090                                        , FIND_LENGTH
2091                                        );
2092                 }
2093         }
2094         (void) SETVAR( makeflags
2095                      , makeflags_value_saved
2096                      , false
2097                      );
2098         macro->body.macro.read_only = temp;

2100 /*
2101  *      Read command line "-f" arguments and ignore -c, g, j, K, M, m, O and o a
2102  */
2103         save_do_not_exec_rule = do_not_exec_rule;
2104         do_not_exec_rule = false;
2105         if (read_trace_level > 0) {
2106                 trace_reader = true;
2107         }

2109         for (i = 1; i < argc; i++) {
2110                 if (argv[i] &&
2111                     (argv[i][0] == (int) hyphen_char) &&
2112                     (argv[i][1] == 'f') &&
2113                     (argv[i][2] == (int) nul_char)) {
2114                         argv[i] = NULL;         /* Remove -f */
2115                         if (i >= argc - 1) {
2116                                 fatal(catgets(catd, 1, 190, "No filename argumen
2117                         }
2118                         MBSTOWCS(wcs_buffer, argv[++i]);
2119                         primary_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2120                         (void) read_makefile(primary_makefile, true, true, true)
2121                         argv[i] = NULL;         /* Remove filename */
2122                         makefile_read = true;
2123                 } else if (argv[i] &&
2124                            (argv[i][0] == (int) hyphen_char) &&
2125                            (argv[i][1] == 'c' ||
2126                             argv[i][1] == 'g' ||
2127                             argv[i][1] == 'j' ||
2128                             argv[i][1] == 'K' ||
2129                             argv[i][1] == 'M' ||
2130                             argv[i][1] == 'm' ||
2131                             argv[i][1] == 'O' ||
2132                             argv[i][1] == 'o') &&
2133                           (argv[i][2] == (int) nul_char)) {
2134                         argv[i] = NULL;
2135                         argv[++i] = NULL;
2136                 }
```

```
2137         }

2139 /*
2140  *      If no command line "-f" args then look for "makefile", and then for
2141  *      "Makefile" if "makefile" isn't found.
2142  */
2143         if (!makefile_read) {
2144                 (void) read_dir(dot,
2145                                 (wchar_t *) NULL,
2146                                 (Property) NULL,
2147                                 (wchar_t *) NULL);
2148             if (!posix) {
2149                 if (makefile_name->stat.is_file) {
2150                         if (Makefile->stat.is_file) {
2151                                 warning(catgets(catd, 1, 310, "Both 'makefile' a
2152                         }
2153                         primary_makefile = makefile_name;
2154                         makefile_read = read_makefile(makefile_name,
2155                                                       false,
2156                                                       false,
2157                                                       true);
2158                 }
2159                 if (!makefile_read &&
2160                     Makefile->stat.is_file) {
2161                         primary_makefile = Makefile;
2162                         makefile_read = read_makefile(Makefile,
2163                                                       false,
2164                                                       false,
2165                                                       true);
2166                 }
2167             } else {

2169                 enum sccs_stat save_m_has_sccs = NO_SCCS;
2170                 enum sccs_stat save_M_has_sccs = NO_SCCS;

2172                 if (makefile_name->stat.is_file) {
2173                         if (Makefile->stat.is_file) {
2174                                 warning(catgets(catd, 1, 191, "Both 'makefile' a
2175                         }
2176                 }
2177                 if (makefile_name->stat.is_file) {
2178                         if (makefile_name->stat.has_sccs == NO_SCCS) {
2179                            primary_makefile = makefile_name;
2180                            makefile_read = read_makefile(makefile_name,
2181                                                          false,
2182                                                          false,
2183                                                          true);
2184                         } else {
2185                           save_m_has_sccs = makefile_name->stat.has_sccs;
2186                           makefile_name->stat.has_sccs = NO_SCCS;
2187                           primary_makefile = makefile_name;
2188                           makefile_read = read_makefile(makefile_name,
2189                                                         false,
2190                                                         false,
2191                                                         true);
2192                         }
2193                 }
2194                 if (!makefile_read &&
2195                     Makefile->stat.is_file) {
2196                         if (Makefile->stat.has_sccs == NO_SCCS) {
2197                            primary_makefile = Makefile;
2198                            makefile_read = read_makefile(Makefile,
2199                                                          false,
2200                                                          false,
2201                                                          true);
2202                         } else {
```



```
2203                           save_M_has_sccs = Makefile->stat.has_sccs;
2204                           Makefile->stat.has_sccs = NO_SCCS;
2205                           primary_makefile = Makefile;
2206                           makefile_read = read_makefile(Makefile,
2207                                                         false,
2208                                                         false,
2209                                                         true);
2210                         }
2211                 }
2212                 if (!makefile_read &&
2213                         makefile_name->stat.is_file) {
2214                            makefile_name->stat.has_sccs = save_m_has_sccs;
2215                            primary_makefile = makefile_name;
2216                            makefile_read = read_makefile(makefile_name,
2217                                                          false,
2218                                                          false,
2219                                                          true);
2220                 }
2221                 if (!makefile_read &&
2222                     Makefile->stat.is_file) {
2223                            Makefile->stat.has_sccs = save_M_has_sccs;
2224                            primary_makefile = Makefile;
2225                            makefile_read = read_makefile(Makefile,
2226                                                          false,
2227                                                          false,
2228                                                          true);
2229                 }
2230             }
2231         }
2232         do_not_exec_rule = save_do_not_exec_rule;
2233         allrules_read = makefile_read;
2234         trace_reader = false;

2236 /*
2237  *      Now get current value of MAKEFLAGS and compare it with
2238  *      the saved value we set before reading makefile.
2239  *      If they are different then MAKEFLAGS is subsequently set by
2240  *      makefile, just leave it there. Otherwise, if make mode
2241  *      is changed by using .POSIX target in makefile we need
2242  *      to correct MAKEFLAGS value.
2243  */
2244         Name mf_val = getvar(makeflags);
2245         if( (posix != is_xpg4)
2246          && (!strcmp(mf_val->string_mb, makeflags_value_saved->string_mb)))
2247         {
2248                 if (makeflags_string_posix.buffer.start[1] == (int) nul_char) {
2249                         (void) SETVAR(makeflags,
2250                                       GETNAME(makeflags_string_posix.buffer.star
2251                                               FIND_LENGTH),
2252                                       false);
2253                 } else {
2254                         if (makeflags_string_posix.buffer.start[1] != (int) spac
2255                                 (void) SETVAR(makeflags,
2256                                               GETNAME(makeflags_string_posix.buf
2257                                                       FIND_LENGTH),
2258                                               false);
2259                         } else {
2260                                 (void) SETVAR(makeflags,
2261                                               GETNAME(makeflags_string_posix.buf
2262                                                       FIND_LENGTH),
2263                                               false);
2264                         }
2265                 }
2266         }

2268         if (makeflags_string.free_after_use) {
```

```
2269                retmem(makeflags_string.buffer.start);
2270        }
2271        if (makeflags_string_posix.free_after_use) {
2272                retmem(makeflags_string_posix.buffer.start);
2273        }
2274        makeflags_string.buffer.start = NULL;
2275        makeflags_string_posix.buffer.start = NULL;

2277        if (posix) {
2278                /*
2279                 * If the user did not redefine the ARFLAGS macro in the
2280                 * default makefile (make.rules), then we'd like to
2281                 * change the macro value of ARFLAGS to be in accordance
2282                 * with "POSIX" requirements.
2283                 */
2284                MBSTOWCS(wcs_buffer, NOCATGETS("ARFLAGS"));
2285                name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2286                macro = get_prop(name->prop, macro_prop);
2287                if ((macro != NULL) && /* Maybe (macro == NULL) || ? */
2288                    (IS_EQUAL(macro->body.macro.value->string_mb,
2289                              NOCATGETS("rv")))) {
2290                        MBSTOWCS(wcs_buffer, NOCATGETS("-rv"));
2291                        value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2292                        (void) SETVAR(name,
2293                                      value,
2294                                      false);
2295                }
2296        }

2298        if (!posix && !svr4) {
2299                set_sgs_support();
2300        }



2303 /*
2304  *      Make sure KEEP_STATE is in the environment if KEEP_STATE is on.
2305  */
2306        macro = get_prop(keep_state_name->prop, macro_prop);
2307        if ((macro != NULL) &&
2308            macro->body.macro.exported) {
2309                keep_state = true;
2310        }
2311        if (keep_state) {
2312                if (macro == NULL) {
2313                        macro = maybe_append_prop(keep_state_name,
2314                                                  macro_prop);
2315                }
2316                macro->body.macro.exported = true;
2317                (void) SETVAR(keep_state_name,
2318                              empty_name,
2319                              false);

2321                /*
2322                 *      Read state file
2323                 */

2325                /* Before we read state, let's make sure we have
2326                ** right state file.
2327                */
2328                /* just in case macro references are used in make_state file
2329                ** name, we better expand them at this stage using expand_value.
2330                */
2331                INIT_STRING_FROM_STACK(dest, destbuffer);
2332                expand_value(make_state, &dest, false);

2334                make_state = GETNAME(dest.buffer.start, FIND_LENGTH);
```



```
2336                if(!stat(make_state->string_mb, &make_state_stat)) {
2337                   if(!(make_state_stat.st_mode & S_IFREG) ) {
2338                        /* copy the make_state structure to the other
2339                        ** and then let make_state point to the new
2340                        ** one.
2341                        */
2342                      memcpy(&state_filename, make_state,sizeof(state_filename))
2343                      state_filename.string_mb = state_file_str_mb;
2344                /* Just a kludge to avoid two slashes back to back */
2345                      if((make_state->hash.length == 1)&&
2346                                (make_state->string_mb[0] == '/')) {
2347                         make_state->hash.length = 0;
2348                         make_state->string_mb[0] = '\0';
2349                      }
2350                      sprintf(state_file_str_mb,NOCATGETS("%s%s"),
2351                       make_state->string_mb,NOCATGETS("/.make.state"));
2352                      make_state = &state_filename;
2353                        /* adjust the length to reflect the appended string */
2354                      make_state->hash.length += 12;
2355                   }
2356                } else { /* the file doesn't exist or no permission */
2357                   char tmp_path[MAXPATHLEN];
2358                   char *slashp;

2360                   if (slashp = strrchr(make_state->string_mb, '/')) {
2361                      strncpy(tmp_path, make_state->string_mb,
2362                                (slashp - make_state->string_mb));
2363                        tmp_path[slashp - make_state->string_mb]=0;
2364                      if(strlen(tmp_path)) {
2365                        if(stat(tmp_path, &make_state_stat)) {
2366                          warning(catgets(catd, 1, 192, "directory %s for .KEEP_
2367                        }
2368                        if (access(tmp_path, F_OK) != 0) {
2369                          warning(catgets(catd, 1, 193, "can't access dir %s"),t
2370                        }
2371                      }
2372                   }
2373                }
2374                if (report_dependencies_level != 1) {
2375                        Makefile_type   makefile_type_temp = makefile_type;
2376                        makefile_type = reading_statefile;
2377                        if (read_trace_level > 1) {
2378                                trace_reader = true;
2379                        }
2380                        (void) read_simple_file(make_state,
2381                                                false,
2382                                                false,
2383                                                false,
2384                                                false,
2385                                                false,
2386                                                true);
2387                        trace_reader = false;
2388                        makefile_type = makefile_type_temp;
2389                }
2390        }
2391 }

2393 /*
2394  * Scan the argv for options and "=" type args and make them readonly.
2395  */
2396 static void
2397 enter_argv_values(int argc, char *argv[], ASCII_Dyn_Array *makeflags_and_macro)
2398 {
2399        register char           *cp;
2400        register int            i;
```

```
2401        int                     length;
2402        register Name           name;
2403        int                     opt_separator = argc;
2404        char                    tmp_char;
2405        wchar_t                 *tmp_wcs_buffer;
2406        register Name           value;
2407        Boolean                 append = false;
2408        Property                macro;
2409        struct stat             statbuf;


2412        /* Read argv options and "=" type args and make them readonly. */
2413        makefile_type = reading_nothing;
2414        for (i = 1; i < argc; ++i) {
2415                append = false;
2416                if (argv[i] == NULL) {
2417                        continue;
2418                } else if (((argv[i][0] == '-') && (argv[i][1] == '-')) ||
2419                           ((argv[i][0] == (int) ' ') &&
2420                            (argv[i][1] == (int) '-') &&
2421                            (argv[i][2] == (int) ' ') &&
2422                            (argv[i][3] == (int) '-'))) {
2423                        argv[i] = NULL;
2424                        opt_separator = i;
2425                        continue;
2426                } else if ((i < opt_separator) && (argv[i][0] == (int) hyphen_ch
2427                        switch (parse_command_option(argv[i][1])) {
2428                        case 1: /* -f seen */
2429                                ++i;
2430                                continue;
2431                        case 2: /* -c seen */
2432                                if (argv[i+1] == NULL) {
2433                                        fatal(catgets(catd, 1, 194, "No dmake rc
2434                                }
2435                                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2436                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2437                                break;
2438                        case 4: /* -g seen */
2439                                if (argv[i+1] == NULL) {
2440                                        fatal(catgets(catd, 1, 195, "No dmake gr
2441                                }
2442                                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
2443                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2444                                break;
2445                        case 8: /* -j seen */
2446                                if (argv[i+1] == NULL) {
2447                                        fatal(catgets(catd, 1, 196, "No dmake ma
2448                                }
2449                                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS")
2450                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2451                                break;
2452                        case 16: /* -M seen */
2453                                if (argv[i+1] == NULL) {
2454                                        fatal(catgets(catd, 1, 323, "No pmake ma
2455                                }
2456                                MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFI
2457                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2458                                break;
2459                        case 32: /* -m seen */
2460                                if (argv[i+1] == NULL) {
2461                                        fatal(catgets(catd, 1, 197, "No dmake mo
2462                                }
2463                                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2464                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2465                                break;
2466                        case 128: /* -O seen */
```



```
2467                                if (argv[i+1] == NULL) {
2468                                        fatal(catgets(catd, 1, 287, "No file des
2469                                }
2470                                mtool_msgs_fd = atoi(argv[i+1]);
2471                                /* find out if mtool_msgs_fd is a valid file des
2472                                if (fstat(mtool_msgs_fd, &statbuf) < 0) {
2473                                        fatal(catgets(catd, 1, 355, "Invalid fil
2474                                }
2475                                argv[i] = NULL;
2476                                argv[i+1] = NULL;
2477                                continue;
2478                        case 256: /* -K seen */
2479                                if (argv[i+1] == NULL) {
2480                                        fatal(catgets(catd, 1, 288, "No makestat
2481                                }
2482                                MBSTOWCS(wcs_buffer, argv[i+1]);
2483                                make_state = GETNAME(wcs_buffer, FIND_LENGTH);
2484                                keep_state = true;
2485                                argv[i] = NULL;
2486                                argv[i+1] = NULL;
2487                                continue;
2488                        case 512:       /* -o seen */
2489                                if (argv[i+1] == NULL) {
2490                                        fatal(catgets(catd, 1, 312, "No dmake ou
2491                                }
2492                                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2493                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2494                                break;
2495                        case 1024: /* -x seen */
2496                                if (argv[i+1] == NULL) {
2497                                        fatal(catgets(catd, 1, 351, "No argument
2498                                }
2499                                length = strlen( NOCATGETS("SUN_MAKE_COMPAT_MODE
2500                                if (strncmp(argv[i+1], NOCATGETS("SUN_MAKE_COMPA
2501                                        argv[i+1] = &argv[i+1][length];
2502                                        MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE
2503                                        name = GETNAME(wcs_buffer, FIND_LENGTH);
2504                                        dmake_compat_mode_specified = dmake_add_
2505                                        break;
2506                                }
2507                                length = strlen( NOCATGETS("DMAKE_OUTPUT_MODE=")
2508                                if (strncmp(argv[i+1], NOCATGETS("DMAKE_OUTPUT_M
2509                                        argv[i+1] = &argv[i+1][length];
2510                                        MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OU
2511                                        name = GETNAME(wcs_buffer, FIND_LENGTH);
2512                                        dmake_output_mode_specified = dmake_add_
2513                                } else {
2514                                        warning(catgets(catd, 1, 354, "Unknown a
2515                                              argv[i+1]);
2516                                        argv[i] = argv[i + 1] = NULL;
2517                                        continue;
2518                                }
2519                                break;
2520                        default: /* Shouldn't reach here */
2521                                argv[i] = NULL;
2522                                continue;
2523                        }
2524                        argv[i] = NULL;
2525                        if (i == (argc - 1)) {
2526                                break;
2527                        }
2528                        if ((length = strlen(argv[i+1])) >= MAXPATHLEN) {
2529                                tmp_wcs_buffer = ALLOC_WC(length + 1);
2530                                (void) mbstowcs(tmp_wcs_buffer, argv[i+1], lengt
2531                                value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2532                                retmem(tmp_wcs_buffer);
```

```
2533                        } else {
2534                                MBSTOWCS(wcs_buffer, argv[i+1]);
2535                                value = GETNAME(wcs_buffer, FIND_LENGTH);
2536                        }
2537                        argv[i+1] = NULL;
2538                } else if ((cp = strchr(argv[i], (int) equal_char)) != NULL) {
2539 /*
2540  * Combine all macro in dynamic array
2541  */
2542                        if(*(cp-1) == (int) plus_char)
2543                        {
2544                                if(isspace(*(cp-2))) {
2545                                        append = true;
2546                                        cp--;
2547                                }
2548                        }
2549                        if(!append)
2550                                append_or_replace_macro_in_dyn_array(makeflags_a

2552                        while (isspace(*(cp-1))) {
2553                                cp--;
2554                        }
2555                        tmp_char = *cp;
2556                        *cp = (int) nul_char;
2557                        MBSTOWCS(wcs_buffer, argv[i]);
2558                        *cp = tmp_char;
2559                        name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2560                        while (*cp != (int) equal_char) {
2561                                cp++;
2562                        }
2563                        cp++;
2564                        while (isspace(*cp) && (*cp != (int) nul_char)) {
2565                                cp++;
2566                        }
2567                        if ((length = strlen(cp)) >= MAXPATHLEN) {
2568                                tmp_wcs_buffer = ALLOC_WC(length + 1);
2569                                (void) mbstowcs(tmp_wcs_buffer, cp, length + 1);
2570                                value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2571                                retmem(tmp_wcs_buffer);
2572                        } else {
2573                                MBSTOWCS(wcs_buffer, cp);
2574                                value = GETNAME(wcs_buffer, FIND_LENGTH);
2575                        }
2576                        argv[i] = NULL;
2577                } else {
2578                        /* Illegal MAKEFLAGS argument */
2579                        continue;
2580                }
2581                if(append) {
2582                        setvar_append(name, value);
2583                        append = false;
2584                } else {
2585                        macro = maybe_append_prop(name, macro_prop);
2586                        macro->body.macro.exported = true;
2587                        SETVAR(name, value, false)->body.macro.read_only = true;
2588                }
2589        }
2590 }

2592 /*
2593  * Append the DMake option and value to the MAKEFLAGS string.
2594  */
2595 static void
2596 append_makeflags_string(Name name, register String makeflags_string)
2597 {
2598         const char      *option;
```



```
2600         if (strcmp(name->string_mb, NOCATGETS("DMAKE_GROUP")) == 0) {
2601                 option = NOCATGETS(" -g ");
2602         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_MAX_JOBS")) == 0) {
2603                 option = NOCATGETS(" -j ");
2604         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_MODE")) == 0) {
2605                 option = NOCATGETS(" -m ");
2606         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_ODIR")) == 0) {
2607                 option = NOCATGETS(" -o ");
2608         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_RCFILE")) == 0) {
2609                 option = NOCATGETS(" -c ");
2610         } else if (strcmp(name->string_mb, NOCATGETS("PMAKE_MACHINESFILE")) == 0
2611                 option = NOCATGETS(" -M ");
2612         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_OUTPUT_MODE")) == 0)
2613                 option = NOCATGETS(" -x DMAKE_OUTPUT_MODE=");
2614         } else if (strcmp(name->string_mb, NOCATGETS("SUN_MAKE_COMPAT_MODE")) ==
2615                 option = NOCATGETS(" -x SUN_MAKE_COMPAT_MODE=");
2616         } else {
2617                 fatal(catgets(catd, 1, 289, "Internal error: name not recognized
2618         }
2619         Property prop = maybe_append_prop(name, macro_prop);
2620         if( prop == 0 || prop->body.macro.value == 0 ||
2621             prop->body.macro.value->string_mb == 0 ) {
2622                 return;
2623         }
2624         char mbs_value[MAXPATHLEN + 100];
2625         strcpy(mbs_value, option);
2626         strcat(mbs_value, prop->body.macro.value->string_mb);
2627         MBSTOWCS(wcs_buffer, mbs_value);
2628         append_string(wcs_buffer, makeflags_string, FIND_LENGTH);
2629 }

2631 /*
2632  *      read_environment(read_only)
2633  *
2634  *      This routine reads the process environment when make starts and enters
2635  *      it as make macros. The environment variable SHELL is ignored.
2636  *
2637  *      Parameters:
2638  *              read_only       Should we make env vars read only?
2639  *
2640  *      Global variables used:
2641  *              report_pwd      Set if this make was started by other make
2642  */
2643 static void
2644 read_environment(Boolean read_only)
2645 {
2646         register char           **environment;
2647         int                     length;
2648         wchar_t                 *tmp_wcs_buffer;
2649         Boolean                 alloced_tmp_wcs_buffer = false;
2650         register wchar_t        *name;
2651         register wchar_t        *value;
2652         register Name           macro;
2653         Property                val;
2654         Boolean                 read_only_saved;

2656         reading_environment = true;
2657         environment = environ;
2658         for (; *environment; environment++) {
2659                 read_only_saved = read_only;
2660                 if ((length = strlen(*environment)) >= MAXPATHLEN) {
2661                         tmp_wcs_buffer = ALLOC_WC(length + 1);
2662                         alloced_tmp_wcs_buffer = true;
2663                         (void) mbstowcs(tmp_wcs_buffer, *environment, length + 1
2664                         name = tmp_wcs_buffer;
```

```
2665                } else {
2666                        MBSTOWCS(wcs_buffer, *environment);
2667                        name = wcs_buffer;
2668                }
2669                value = (wchar_t *) wschr(name, (int) equal_char);

2671                /*
2672                 * Looks like there's a bug in the system, but sometimes
2673                 * you can get blank lines in *environment.
2674                 */
2675                if (!value) {
2676                        continue;
2677                }
2678                MBSTOWCS(wcs_buffer2, NOCATGETS("SHELL="));
2679                if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2680                        continue;
2681                }
2682                MBSTOWCS(wcs_buffer2, NOCATGETS("MAKEFLAGS="));
2683                if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2684                        report_pwd = true;
2685                        /*
2686                         * In POSIX mode we do not want MAKEFLAGS to be readonly
2687                         * If the MAKEFLAGS macro is subsequently set by the mak
2688                         * it replaces the MAKEFLAGS variable currently found in
2689                         * environment.
2690                         * See Assertion 50 in section 6.2.5.3 of standard P1003
2691                         */
2692                        if(posix) {
2693                                read_only_saved = false;
2694                        }
2695                }

2697                /*
2698                 * We ignore SUNPRO_DEPENDENCIES. This environment variable is
2699                 * set by make and read by cpp which then writes info to
2700                 * .make.dependency.xxx.  When make is invoked by another make
2701                 * (recursive make), we don't want to read this because then
2702                 * the child make will end up writing to the parent
2703                 * directory's .make.state and clobbering them.
2704                 */
2705                MBSTOWCS(wcs_buffer2, NOCATGETS("SUNPRO_DEPENDENCIES"));
2706                if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2707                        continue;
2708                }

2710                macro = GETNAME(name, value - name);
2711                maybe_append_prop(macro, macro_prop)->body.macro.exported =
2712                  true;
2713                if ((value == NULL) || ((value + 1)[0] == (int) nul_char)) {
2714                        val = setvar_daemon(macro,
2715                                        (Name) NULL,
2716                                        false, no_daemon, false, debug_level
2717                } else {
2718                        val = setvar_daemon(macro,
2719                                        GETNAME(value + 1, FIND_LENGTH),
2720                                        false, no_daemon, false, debug_level
2721                }
2722                val->body.macro.read_only = read_only_saved;
2723                if (alloced_tmp_wcs_buffer) {
2724                        retmem(tmp_wcs_buffer);
2725                        alloced_tmp_wcs_buffer = false;
2726                }
2727        }
2728        reading_environment = false;
2729 }
```



```
2731 /*
2732  *      read_makefile(makefile, complain, must_exist, report_file)
2733  *
2734  *      Read one makefile and check the result
2735  *
2736  *      Return value:
2737  *                              false is the read failed
2738  *
2739  *      Parameters:
2740  *              makefile        The file to read
2741  *              complain        Passed thru to read_simple_file()
2742  *              must_exist      Passed thru to read_simple_file()
2743  *              report_file     Passed thru to read_simple_file()
2744  *
2745  *      Global variables used:
2746  *              makefile_type   Set to indicate we are reading main file
2747  *              recursion_level Initialized
2748  */
2749 static Boolean
2750 read_makefile(register Name makefile, Boolean complain, Boolean must_exist, Bool
2751 {
2752         Boolean                 b;
2753
2754         makefile_type = reading_makefile;
2755         recursion_level = 0;
2756         reading_dependencies = true;
2757         b = read_simple_file(makefile, true, true, complain,
2758                              must_exist, report_file, false);
2759         reading_dependencies = false;
2760         return b;
2761 }

2763 /*
2764  *      make_targets(argc, argv, parallel_flag)
2765  *
2766  *      Call doname on the specified targets
2767  *
2768  *      Parameters:
2769  *              argc            You know what this is
2770  *              argv            You know what this is
2771  *              parallel_flag   True if building in parallel
2772  *
2773  *      Global variables used:
2774  *              build_failed_seen Used to generated message after failed -k
2775  *              commands_done   Used to generate message "Up to date"
2776  *              default_target_to_build First proper target in makefile
2777  *              init            The Name ".INIT", use to run command
2778  *              parallel        Global parallel building flag
2779  *              quest           make -q, suppresses messages
2780  *              recursion_level Initialized, used for tracing
2781  *              report_dependencies make -P, regroves whole process
2782  */
2783 static void
2784 make_targets(int argc, char **argv, Boolean parallel_flag)
2785 {
2786         int                     i;
2787         char                    *cp;
2788         Doname                  result;
2789         register Boolean        target_to_make_found = false;

2791         (void) doname(init, true, true);
2792         recursion_level = 1;
2793         parallel = parallel_flag;
2794 /*
2795  *      make remaining args
2796  */
```

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



```
                                                if (!commands_done &&
                                                    (exists(default_target_to_bu
                                                        (void) printf(catgets(ca
                                                                      default_ta
                                                }
                                        }
                                }
                        }
                }
                /* Now wait for all of the targets to finish running */
                finish_running();
                //              setjmp(jmpbuffer);

        }
        for (i = 1; i < argc; i++) {
                if ((cp = argv[i]) != NULL) {
                        target_to_make_found = true;
                        if ((cp[0] == (int) period_char) &&
                            (cp[1] == (int) slash_char)) {
                                cp += 2;
                        }
                                 if((cp[0] == (int) ' ') &&
                                    (cp[1] == (int) '-') &&
                                    (cp[2] == (int) ' ') &&
                                    (cp[3] == (int) '-')) {
                                    argv[i] = NULL;
                                        continue;
                                }
                        MBSTOWCS(wcs_buffer, cp);
                        default_target_to_build = normalize_name(wcs_buffer, wsl
                        top_level_target = get_wstring(default_target_to_build->
                        report_recursion(default_target_to_build);
                        commands_done = false;
                        if (parallel) {
                                result = (Doname) default_target_to_build->state
                        } else {
                                result = doname_check(default_target_to_build,
                                                      true,
                                                      false,
                                                      false);
                        }
                        gather_recursive_deps();
                        if (build_failed_seen) {
                                build_failed_ever_seen = true;
                                warning(catgets(catd, 1, 200, "Target '%s' not r
                                        default_target_to_build->string_mb);
                        }
                        build_failed_seen = false;
                        if (report_dependencies_level > 0) {
                                print_dependencies(default_target_to_build,
                                                   get_prop(default_target_to_bu
                                                            line_prop));
                        }
                        default_target_to_build->stat.time =
                          file_no_time;
                        if (default_target_to_build->colon_splits > 0) {
                                default_target_to_build->state =
                                  build_dont_know;
                        }
                        if (!parallel &&
                            /* !commands_done && */
                            (result == build_ok) &&
                            !quest &&
                            (report_dependencies_level == 0) /*  &&
                            (exists(default_target_to_build) > file_doesnt_exist
                                if (posix) {
```

```
2929                                    if (!commands_done) {
2930                                            (void) printf(catgets(catd, 1, 2
2931                                                                  default_target_to_
2932                                    } else {
2933                                            if (no_action_was_taken) {
2934                                                    (void) printf(catgets(ca
2935                                                                          default_ta
2936                                            }
2937                                    }
2938                            } else {
2939                                    if (!commands_done &&
2940                                        (exists(default_target_to_build) > f
2941                                            (void) printf(catgets(catd, 1, 2
2942                                                                  default_target_to_
2943                                    }
2944                            }
2945                    }
2946            }
2947    }

2949 /*
2950  *      If no file arguments have been encountered,
2951  *      make the first name encountered that doesnt start with a dot
2952  */
2953    if (!target_to_make_found) {
2954            if (default_target_to_build == NULL) {
2955                    fatal(catgets(catd, 1, 202, "No arguments to build"));
2956            }
2957            commands_done = false;
2958            top_level_target = get_wstring(default_target_to_build->string_m
2959            report_recursion(default_target_to_build);


2962            if (getenv(NOCATGETS("SPRO_EXPAND_ERRORS"))){
2963                    (void) printf(NOCATGETS("::(%s)\n"),
2964                                  default_target_to_build->string_mb);
2965            }


2968            result = doname_parallel(default_target_to_build, true, false);
2969            gather_recursive_deps();
2970            if (build_failed_seen) {
2971                    build_failed_ever_seen = true;
2972                    warning(catgets(catd, 1, 203, "Target `%s' not remade be
2973                            default_target_to_build->string_mb);
2974            }
2975            build_failed_seen = false;
2976            if (report_dependencies_level > 0) {
2977                    print_dependencies(default_target_to_build,
2978                                       get_prop(default_target_to_build->
2979                                                prop,
2980                                                line_prop));
2981            }
2982            default_target_to_build->stat.time = file_no_time;
2983            if (default_target_to_build->colon_splits > 0) {
2984                    default_target_to_build->state = build_dont_know;
2985            }
2986            if (/* !commands_done && */
2987                (result == build_ok) &&
2988                !quest &&
2989                (report_dependencies_level == 0) /*  &&
2990                (exists(default_target_to_build) > file_doesnt_exist)  */) {
2991                    if (posix) {
2992                            if (!commands_done) {
2993                                    (void) printf(catgets(catd, 1, 299, "`%s
2994                                                          default_target_to_build->s
```



```
2995                                    } else {
2996                                            if (no_action_was_taken) {
2997                                                    (void) printf(catgets(catd, 1, 3
2998                                                                          default_target_to_
2999                                            }
3000                                    }
3001                            } else {
3002                                    if (!commands_done &&
3003                                        (exists(default_target_to_build) > file_does
3004                                            (void) printf(catgets(catd, 1, 301, "`%s
3005                                                          default_target_to_build->s
3006                                    }
3007                            }
3008                    }
3009            }
3010    }

3012 /*
3013  *      report_recursion(target)
3014  *
3015  *      If this is a recursive make and the parent make has KEEP_STATE on
3016  *      this routine reports the dependency to the parent make
3017  *
3018  *      Parameters:
3019  *              target          Target to report
3020  *
3021  *      Global variables used:
3022  *              makefiles_used          List of makefiles read
3023  *              recursive_name          The Name ".RECURSIVE", printed
3024  *              report_dependency       dwight
3025  */
3026 static void
3027 report_recursion(register Name target)
3028 {
3029            register FILE           *report_file = get_report_file();

3031            if ((report_file == NULL) || (report_file == (FILE*)-1)) {
3032                    return;
3033            }
3034            if (primary_makefile == NULL) {
3035                    /*
3036                     * This can happen when there is no makefile and
3037                     * only implicit rules are being used.
3038                     */
3039                    return;
3040            }
3041            (void) fprintf(report_file,
3042                           "%s: %s ",
3043                           get_target_being_reported_for(),
3044                           recursive_name->string_mb);
3045            report_dependency(get_current_path());
3046            report_dependency(target->string_mb);
3047            report_dependency(primary_makefile->string_mb);
3048            (void) fprintf(report_file, "\n");
3049 }

3051 /* Next function "append_or_replace_macro_in_dyn_array" must be in "misc.cc". */
3052 /* NIKMOL */
3053 extern void
3054 append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar, char *macro)
3055 {
3056            register char   *cp0;   /* work pointer in macro */
3057            register char   *cp1;   /* work pointer in array */
3058            register char   *cp2;   /* work pointer in array */
3059            register char   *cp3;   /* work pointer in array */
3060            register char   *name;  /* macro name */
```

```
3061         register char   *value; /* macro value */
3062         register int    len_array;
3063         register int    len_macro;

3065         char * esc_value = NULL;
3066         int esc_len;

3068         if (!(len_macro = strlen(macro))) return;
3069         name = macro;
3070         while (isspace(*(name))) {
3071                 name++;
3072         }
3073         if (!(value = strchr(name, (int) equal_char))) {
3074                 /* no '=' in macro */
3075                 goto ERROR_MACRO;
3076         }
3077         cp0 = value;
3078         value++;
3079         while (isspace(*(value))) {
3080                 value++;
3081         }
3082         while (isspace(*(cp0-1))) {
3083                 cp0--;
3084         }
3085         if (cp0 <= name) goto ERROR_MACRO; /* no name */
3086         if (!(Ar->size)) goto ALLOC_ARRAY;
3087         cp1 = Ar->start;

3089 LOOK_FOR_NAME:
3090         if (!(cp1 = strchr(cp1, name[0]))) goto APPEND_MACRO;
3091         if (!(cp2 = strchr(cp1, (int) equal_char))) goto APPEND_MACRO;
3092         if (strncmp(cp1, name, (size_t)(cp0-name))) {
3093                 /* another name */
3094                 cp1++;
3095                 goto LOOK_FOR_NAME;
3096         }
3097         if (cp1 != Ar->start) {
3098                 if (!isspace(*(cp1-1))) {
3099                         /* another name */
3100                         cp1++;
3101                         goto LOOK_FOR_NAME;
3102                 }
3103         }
3104         for (cp3 = cp1 + (cp0-name); cp3 < cp2; cp3++) {
3105                 if (isspace(*cp3)) continue;
3106                 /* else: another name */
3107                 cp1++;
3108                 goto LOOK_FOR_NAME;
3109         }
3110         /* Look for the next macro name in array */
3111         cp3 = cp2+1;
3112         if (*cp3 != (int) doublequote_char) {
3113                 /* internal error */
3114                 goto ERROR_MACRO;
3115         }
3116         if (!(cp3 = strchr(cp3+1, (int) doublequote_char))) {
3117                 /* internal error */
3118                 goto ERROR_MACRO;
3119         }
3120         cp3++;
3121         while (isspace(*cp3)) {
3122                 cp3++;
3123         }
3124
3125         cp2 = cp1;  /* remove old macro */
3126         if ((*cp3) && (cp3 < Ar->start + Ar->size)) {
```



```
3127                 for (; cp3 < Ar->start + Ar->size; cp3++) {
3128                         *cp2++ = *cp3;
3129                 }
3130         }
3131         for (; cp2 < Ar->start + Ar->size; cp2++) {
3132                 *cp2 = 0;
3133         }
3134         if (*cp1) {
3135                 /* check next name */
3136                 goto LOOK_FOR_NAME;
3137         }
3138         goto APPEND_MACRO;

3140 ALLOC_ARRAY:
3141         if (Ar->size) {
3142                 cp1 = Ar->start;
3143         } else {
3144                 cp1 = 0;
3145         }
3146         Ar->size += 128;
3147         Ar->start = getmem(Ar->size);
3148         for (len_array=0; len_array < Ar->size; len_array++) {
3149                 Ar->start[len_array] = 0;
3150         }
3151         if (cp1) {
3152                 strcpy(Ar->start, cp1);
3153                 retmem((wchar_t *) cp1);
3154         }

3156 APPEND_MACRO:
3157         len_array = strlen(Ar->start);
3158         esc_value = (char*)malloc(strlen(value)*2 + 1);
3159         quote_str(value, esc_value);
3160         esc_len = strlen(esc_value) - strlen(value);
3161         if (len_array + len_macro + esc_len + 5 >= Ar->size) goto  ALLOC_ARRAY;
3162         strcat(Ar->start, " ");
3163         strncat(Ar->start, name, cp0-name);
3164         strcat(Ar->start, "=");
3165         strncat(Ar->start, esc_value, strlen(esc_value));
3166         free(esc_value);
3167         return;
3168 ERROR_MACRO:
3169         /* Macro without '=' or with invalid left/right part */
3170         return;
3171 }

3173 #ifdef TEAMWARE_MAKE_CMN
3174 /*
3175  * This function, if registered w/ avo_cli_get_license(), will be called
3176  * if the application is about to exit because:
3177  *   1) there has been certain unrecoverable error(s) that cause the
3178  *      application to exit immediately.
3179  *   2) the user has lost a license while the application is running.
3180  */
3181 extern "C" void
3182 dmake_exit_callback(void)
3183 {
3184         fatal(catgets(catd, 1, 306, "can not get a license, exiting..."));
3185         exit(1);
3186 }

3188 /*
3189  * This function, if registered w/ avo_cli_get_license(), will be called
3190  * if the application can not get a license.
3191  */
3192 extern "C" void
```

```
3193 dmake_message_callback(char *err_msg)
3194 {
3195         static Boolean  first = true;

3197         if (!first) {
3198                 return;
3199         }
3200         first = false;
3201         if ((!list_all_targets) &&
3202             (report_dependencies_level == 0) &&
3203             (dmake_mode_type != serial_mode)) {
3204                 warning(catgets(catd, 1, 313, "can not get a TeamWare license, d
3205         }
3206 }
3207 #endif


3210 static void
3211 report_dir_enter_leave(Boolean entering)
3212 {
3213         char    rcwd[MAXPATHLEN];
3214 static  char *  mlev = NULL;
3215         char *  make_level_str = NULL;
3216         int     make_level_val = 0;

3218         make_level_str = getenv(NOCATGETS("MAKELEVEL"));
3219         if(make_level_str) {
3220                 make_level_val = atoi(make_level_str);
3221         }
3222         if(mlev == NULL) {
3223                 mlev = (char*) malloc(MAXPATHLEN);
3224         }
3225         if(entering) {
3226                 sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val + 1);
3227         } else {
3228                 make_level_val--;
3229                 sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val);
3230         }
3231         putenv(mlev);

3233         if(report_cwd) {
3234                 if(make_level_val <= 0) {
3235                         if(entering) {
3236                                 sprintf( rcwd
3237                                         , catgets(catd, 1, 329, "dmake: Entering
3238                                         , get_current_path());
3239                         } else {
3240                                 sprintf( rcwd
3241                                         , catgets(catd, 1, 331, "dmake: Leaving d
3242                                         , get_current_path());
3243                         }
3244                 } else {
3245                         if(entering) {
3246                                 sprintf( rcwd
3247                                         , catgets(catd, 1, 333, "dmake[%d]: Enter
3248                                         , make_level_val, get_current_path());
3249                         } else {
3250                                 sprintf( rcwd
3251                                         , catgets(catd, 1, 335, "dmake[%d]: Leavi
3252                                         , make_level_val, get_current_path());
3253                         }
3254                 }
3255                 printf(NOCATGETS("%s"), rcwd);
3256         }
3257 }
```