```
**********************************************************
   90869 Wed May 20 12:11:52 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")
#ifdef __i386
#define LD_SUPPORT_MAKE_ARCH    NOCATGETS("i386")
#elif __sparc
#define LD_SUPPORT_MAKE_ARCH    NOCATGETS("sparc")
#else
#error "Unsupported architecture"
#endif
#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
```

```
126 static  Boolean         getname_stat = false;
127 #endif

129         static  time_t          start_time;
130         static  int             g_argc;
131         static  char            **g_argv;

133 /*
134  * File table of contents
135  */
136         extern "C" void         cleanup_after_exit(void);

138 extern "C" {
139         extern  void            dmake_exit_callback(void);
140         extern  void            dmake_message_callback(char *);
141 }

143 extern  Name            normalize_name(register wchar_t *name_string, register i

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

147 static  void            append_makeflags_string(Name, String);
148 static  void            doalarm(int);
149 static  void            enter_argv_values(int , char **, ASCII_Dyn_Array *);
150 static  void            make_targets(int, char **, Boolean);
151 static  int             parse_command_option(char);
152 static  void            read_command_options(int, char **);
153 static  void            read_environment(Boolean);
154 static  void            read_files_and_state(int, char **);
155 static  Boolean         read_makefile(Name, Boolean, Boolean, Boolean);
156 static  void            report_recursion(Name);
157 static  void            set_sgs_support(void);
158 static  void            setup_for_projectdir(void);
159 static  void            setup_makeflags_argv(void);
160 static  void            report_dir_enter_leave(Boolean entering);

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

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

166 jmp_buf jmpbuffer;
167 extern nl_catd catd;

169 /*
170  *      main(argc, argv)
171  *
172  *      Parameters:
173  *              argc                    You know what this is
174  *              argv                    You know what this is
175  *
176  *      Static variables used:
177  *              list_all_targets        make -T seen
178  *              trace_status            make -p seen
179  *
180  *      Global variables used:
181  *              debug_level             Should we trace make actions?
182  *              keep_state              Set if .KEEP_STATE seen
183  *              makeflags               The Name "MAKEFLAGS", used to get macro
184  *              remote_command_name     Name of remote invocation cmd ("on")
185  *              running_list            List of parallel running processes
186  *              stdout_stderr_same      true if stdout and stderr are the same
187  *              auto_dependencies       The Name "SUNPRO_DEPENDENCIES"
188  *              temp_file_directory     Set to the dir where we create tmp file
189  *              trace_reader            Set to reflect tracing status
190  *              working_on_targets      Set when building user targets
191  */
```



```
192 int
193 main(int argc, char *argv[])
194 {
195         /*
196          * cp is a -> to the value of the MAKEFLAGS env var,
197          * which has to be regular chars.
198          */
199         register char           *cp;
200         char                    make_state_dir[MAXPATHLEN];
201         Boolean                 parallel_flag = false;
202         char                    *prognameptr;
203         char                    *slash_ptr;
204         mode_t                  um;
205         int                     i;
206         struct itimerval        value;
207         char                    def_dmakerc_path[MAXPATHLEN];
208         Name                    dmake_name, dmake_name2;
209         Name                    dmake_value, dmake_value2;
210         Property                prop, prop2;
211         struct stat             statbuf;
212         int                     statval;

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

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


221 #ifdef DMAKE_STATISTICS
222         if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
223                 getname_stat = true;
224         }
225 #endif

227         catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);

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


232 /*
233  * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
234  * from avo_util library.
235  */
236         libmksdmsi18n_init();


239         textdomain(NOCATGETS("SUNW_SPRO_MAKE"));

241         g_argc = argc;
242         g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
243         for (i = 0; i < argc; i++) {
244                 g_argv[i] = argv[i];
245         }
246         g_argv[i] = NULL;

248         /*
249          * Set argv_zero_string to some form of argv[0] for
250          * recursive MAKE builds.
251          */

253         if (*argv[0] == (int) slash_char) {
254                 /* argv[0] starts with a slash */
255                 argv_zero_string = strdup(argv[0]);
256         } else if (strchr(argv[0], (int) slash_char) == NULL) {
257                 /* argv[0] contains no slashes */
```

```
258                 argv_zero_string = strdup(argv[0]);
259         } else {
260                 /*
261                  * argv[0] contains at least one slash,
262                  * but doesn't start with a slash
263                  */
264                 char    *tmp_current_path;
265                 char    *tmp_string;

267                 tmp_current_path = get_current_path();
268                 tmp_string = getmem(strlen(tmp_current_path) + 1 +
269                                     strlen(argv[0]) + 1);
270                 (void) sprintf(tmp_string,
271                                "%s/%s",
272                                tmp_current_path,
273                                argv[0]);
274                 argv_zero_string = strdup(tmp_string);
275                 retmem_mb(tmp_string);
276         }

278         /*
279          * The following flags are reset if we don't have the
280          * (.nse_depinfo or .make.state) files locked and only set
281          * AFTER the file has been locked. This ensures that if the user
282          * interrupts the program while file_lock() is waiting to lock
283          * the file, the interrupt handler doesn't remove a lock
284          * that doesn't belong to us.
285          */
286         make_state_lockfile = NULL;
287         make_state_locked = false;


290         /*
291          * look for last slash char in the path to look at the binary
292          * name. This is to resolve the hard link and invoke make
293          * in svr4 mode.
294          */

296         /* Sun OS make standart */
297         svr4 = false;
298         posix = false;
299         if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
300                 svr4 = false;
301                 posix = true;
302         } else {
303                 prognameptr = strrchr(argv[0], '/');
304                 if(prognameptr) {
305                         prognameptr++;
306                 } else {
307                         prognameptr = argv[0];
308                 }
309                 if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
310                         svr4 = true;
311                         posix = false;
312                 }
313         }
314         if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
315            svr4 = true;
316            posix = false;
317         }

319         /*
320          * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
321          */
322         char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
323         if (dmake_compat_mode_var != NULL) {
```



```
324                 if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
325                         gnu_style = true;
326                 }
327                 //svr4 = false;
328                 //posix = false;
329         }

331         /*
332          * Temporary directory set up.
333          */
334         char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
335         if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
336                 strcpy(mbs_buffer, tmpdir_var);
337                 for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
338                         *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
339                         *tmpdir_var = '\0');
340                 if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
341                         sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
342                                 mbs_buffer, getpid());
343                         int fd = mkstemp(mbs_buffer2);
344                         if (fd >= 0) {
345                                 close(fd);
346                                 unlink(mbs_buffer2);
347                                 tmpdir = strdup(mbs_buffer);
348                         }
349                 }
350         }

352         /* find out if stdout and stderr point to the same place */
353         if (fstat(1, &out_stat) < 0) {
354                 fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
355         }
356         if (fstat(2, &err_stat) < 0) {
357                 fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
358         }
359         if ((out_stat.st_dev == err_stat.st_dev) &&
360             (out_stat.st_ino == err_stat.st_ino)) {
361                 stdout_stderr_same = true;
362         } else {
363                 stdout_stderr_same = false;
364         }
365         /* Make the vroot package scan the path using shell semantics */
366         set_path_style(0);

368         setup_char_semantics();

370         setup_for_projectdir();

372         /*
373          * If running with .KEEP_STATE, curdir will be set with
374          * the connected directory.
375          */
376         (void) atexit(cleanup_after_exit);

378         load_cached_names();

380 /*
381  *      Set command line flags
382  */
383         setup_makeflags_argv();
384         read_command_options(mf_argc, mf_argv);
385         read_command_options(argc, argv);
386         if (debug_level > 0) {
387                 cp = getenv(makeflags->string_mb);
388                 (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
389         }
```

```
391         setup_interrupt(handle_interrupt);

393         read_files_and_state(argc, argv);

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

441                 if (strcmp(basename(s), NOCATGETS("dmake")) == 0) {
374         if (prop2 == NULL) {
375                 /* DMAKE_MODE not defined, default to parallel mode */
442                         dmake_mode_type = parallel_mode;
443                         no_parallel = false;
444                 }
445 #endif /* ! codereview */
446         } else {
447                 dmake_value2 = prop2->body.macro.value;
448                 if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel"))) {
449                         dmake_mode_type = parallel_mode;
450                         no_parallel = false;
451                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
452                         dmake_mode_type = serial_mode;
453                         no_parallel = true;
```



```
454                 } else {
455                         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
456                 }
457         }
458         free(s);
459     }

461         parallel_flag = true;
462         putenv(strdup(NOCATGETS("DMAKE_CHILD=TRUE")));

464 //
465 // If dmake is running with -t option, set dmake_mode_type to serial.
466 // This is done because doname() calls touch_command() that runs serially.
467 // If we do not do that, maketool will have problems.
468 //
469         if(touch) {
470                 dmake_mode_type = serial_mode;
471                 no_parallel = true;
472         }

474         /*
475          * Check whether stdout and stderr are physically same.
476          * This is in order to decide whether we need to redirect
477          * stderr separately from stdout.
478          * This check is performed only if __DMAKE_SEPARATE_STDERR
479          * is not set. This variable may be used in order to preserve
480          * the ’old’ behaviour.
481          */
482         out_err_same = true;
483         char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
484         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
485                 struct stat stdout_stat;
486                 struct stat stderr_stat;
487                 if( (fstat(1, &stdout_stat) == 0)
488                  && (fstat(2, &stderr_stat) == 0) )
489                 {
490                         if( (stdout_stat.st_dev != stderr_stat.st_dev)
491                          || (stdout_stat.st_ino != stderr_stat.st_ino) )
492                         {
493                                 out_err_same = false;
494                         }
495                 }
496         }

498
499 /*
500  *      Enable interrupt handler for alarms
501  */
502         (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

504 /*
505  *      Check if make should report
506  */
507         if (getenv(sunpro_dependencies->string_mb) != NULL) {
508                 FILE    *report_file;

510                 report_dependency("");
511                 report_file = get_report_file();
512                 if ((report_file != NULL) && (report_file != (FILE*)-1)) {
513                         (void) fprintf(report_file, "\n");
514                 }
515         }

517 /*
518  *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
```

```
 519  */
 520         if (keep_state) {
 521                 maybe_append_prop(sunpro_dependencies, macro_prop)->
 522                   body.macro.exported = true;
 523         } else {
 524                 maybe_append_prop(sunpro_dependencies, macro_prop)->
 525                   body.macro.exported = false;
 526         }

 528         working_on_targets = true;
 529         if (trace_status) {
 530                 dump_make_state();
 531                 fclose(stdout);
 532                 fclose(stderr);
 533                 exit_status = 0;
 534                 exit(0);
 535         }
 536         if (list_all_targets) {
 537                 dump_target_list();
 538                 fclose(stdout);
 539                 fclose(stderr);
 540                 exit_status = 0;
 541                 exit(0);
 542         }
 543         trace_reader = false;

 545         /*
 546          * Set temp_file_directory to the directory the .make.state
 547          * file is written to.
 548          */
 549         if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
 550                 temp_file_directory = strdup(get_current_path());
 551         } else {
 552                 *slash_ptr = (int) nul_char;
 553                 (void) strcpy(make_state_dir, make_state->string_mb);
 554                 *slash_ptr = (int) slash_char;
 555                    /* when there is only one slash and it's the first
 556                    ** character, make_state_dir should point to '/'.
 557                    */
 558                 if(make_state_dir[0] == '\0') {
 559                    make_state_dir[0] = '/';
 560                    make_state_dir[1] = '\0';
 561                 }
 562                 if (make_state_dir[0] == (int) slash_char) {
 563                         temp_file_directory = strdup(make_state_dir);
 564                 } else {
 565                         char    tmp_current_path2[MAXPATHLEN];
 566
 567                         (void) sprintf(tmp_current_path2,
 568                                        "%s/%s",
 569                                        get_current_path(),
 570                                        make_state_dir);
 571                         temp_file_directory = strdup(tmp_current_path2);
 572                 }
 573         }


 576         report_dir_enter_leave(true);

 578         make_targets(argc, argv, parallel_flag);

 580         report_dir_enter_leave(false);

 582         if (build_failed_ever_seen) {
 583                 if (posix) {
 584                         exit_status = 1;
```



```
 585                 }
 586                 exit(1);
 587         }
 588         exit_status = 0;
 589         exit(0);
 590         /* NOTREACHED */
 591 }
______unchanged_portion_omitted_

1285 /*
1286  *      parse_command_option(ch)
1287  *
1288  *      Parse make command line options.
1289  *
1290  *      Return value:
1291  *                              Indicates if any -f -c or -M were seen
1292  *
1293  *      Parameters:
1294  *              ch              The character to parse
1295  *
1296  *      Static variables used:
1297  *              dmake_group_specified   Set for make -g
1298  *              dmake_max_jobs_specified        Set for make -j
1299  *              dmake_mode_specified    Set for make -m
1300  *              dmake_add_mode_specified        Set for make -x
1301  *              dmake_compat_mode_specified     Set for make -x SUN_MAKE_COMPAT_
1302  *              dmake_output_mode_specified     Set for make -x DMAKE_OUTPUT_MOD
1303  *              dmake_odir_specified    Set for make -o
1304  *              dmake_rcfile_specified  Set for make -c
1305  *              env_wins                Set for make -e
1306  *              ignore_default_mk       Set for make -r
1307  *              trace_status            Set for make -p
1308  *
1309  *      Global variables used:
1310  *              .make.state path & name set for make -K
1311  *              continue_after_error    Set for make -k
1312  *              debug_level             Set for make -d
1313  *              do_not_exec_rule        Set for make -n
1314  *              filter_stderr           Set for make -X
1315  *              ignore_errors_all       Set for make -i
1316  *              no_parallel             Set for make -R
1317  *              quest                   Set for make -q
1318  *              read_trace_level        Set for make -D
1319  *              report_dependencies     Set for make -P
1320  *              send_mtool_msgs         Set for make -K
1321  *              silent_all              Set for make -s
1322  *              touch                   Set for make -t
1323  */
1324 static int
1325 parse_command_option(register char ch)
1326 {
1327         static int              invert_next = 0;
1328         int                     invert_this = invert_next;

1330         invert_next = 0;
1331         switch (ch) {
1332         case '-':                       /* Ignore "--" */
1333                 return 0;
1334         case '~':                       /* Invert next option */
1335                 invert_next = 1;
1336                 return 0;
1337         case 'B':                       /* Obsolete */
1338                 return 0;
1339         case 'b':                       /* Obsolete */
1340                 return 0;
1341         case 'c':                       /* Read alternative dmakerc file */
```

```
1342                 if (invert_this) {
1343                         dmake_rcfile_specified = false;
1344                 } else {
1345                         dmake_rcfile_specified = true;
1346                 }
1347                 return 2;
1348         case 'D':                       /* Show lines read */
1349                 if (invert_this) {
1350                         read_trace_level--;
1351                 } else {
1352                         read_trace_level++;
1353                 }
1354                 return 0;
1355         case 'd':                       /* Debug flag */
1356                 if (invert_this) {
1357                         debug_level--;
1358                 } else {
1359                         debug_level++;
1360                 }
1361                 return 0;
1362         case 'e':                       /* Environment override flag */
1363                 if (invert_this) {
1364                         env_wins = false;
1365                 } else {
1366                         env_wins = true;
1367                 }
1368                 return 0;
1369         case 'f':                       /* Read alternative makefile(s) */
1370                 return 1;
1371         case 'g':                       /* Use alternative DMake group */
1372                 if (invert_this) {
1373                         dmake_group_specified = false;
1374                 } else {
1375                         dmake_group_specified = true;
1376                 }
1377                 return 4;
1378         case 'i':                       /* Ignore errors */
1379                 if (invert_this) {
1380                         ignore_errors_all = false;
1381                 } else {
1382                         ignore_errors_all = true;
1383                 }
1384                 return 0;
1385         case 'j':                       /* Use alternative DMake max jobs */
1386                 if (invert_this) {
1387                         dmake_max_jobs_specified = false;
1388                 } else {
1389                         dmake_mode_type = parallel_mode;
1390                         no_parallel = false;
1391 #endif /* ! codereview */
1392                         dmake_max_jobs_specified = true;
1393                 }
1394                 return 8;
1395         case 'K':                       /* Read alternative .make.state */
1396                 return 256;
1397         case 'k':                       /* Keep making even after errors */
1398                 if (invert_this) {
1399                         continue_after_error = false;
1400                 } else {
1401                         continue_after_error = true;
1402                         continue_after_error_ever_seen = true;
1403                 }
1404                 return 0;
1405         case 'M':                       /* Read alternative make.machines file
1406                 if (invert_this) {
1407                         pmake_machinesfile_specified = false;
```



```
1408                 } else {
1409                         pmake_machinesfile_specified = true;
1410                         dmake_mode_type = parallel_mode;
1411                         no_parallel = false;
1412                 }
1413                 return 16;
1414         case 'm':                       /* Use alternative DMake build mode */
1415                 if (invert_this) {
1416                         dmake_mode_specified = false;
1417                 } else {
1418                         dmake_mode_specified = true;
1419                 }
1420                 return 32;
1421         case 'x':                       /* Use alternative DMake mode */
1422                 if (invert_this) {
1423                         dmake_add_mode_specified = false;
1424                 } else {
1425                         dmake_add_mode_specified = true;
1426                 }
1427                 return 1024;
1428         case 'N':                       /* Reverse -n */
1429                 if (invert_this) {
1430                         do_not_exec_rule = true;
1431                 } else {
1432                         do_not_exec_rule = false;
1433                 }
1434                 return 0;
1435         case 'n':                       /* Print, not exec commands */
1436                 if (invert_this) {
1437                         do_not_exec_rule = false;
1438                 } else {
1439                         do_not_exec_rule = true;
1440                 }
1441                 return 0;
1442         case 'O':                       /* Send job start & result msgs */
1443                 if (invert_this) {
1444                         send_mtool_msgs = false;
1445                 } else {
1446                 }
1447                 return 128;
1448         case 'o':                       /* Use alternative dmake output dir */
1449                 if (invert_this) {
1450                         dmake_odir_specified = false;
1451                 } else {
1452                         dmake_odir_specified = true;
1453                 }
1454                 return 512;
1455         case 'P':                       /* Print for selected targets */
1456                 if (invert_this) {
1457                         report_dependencies_level--;
1458                 } else {
1459                         report_dependencies_level++;
1460                 }
1461                 return 0;
1462         case 'p':                       /* Print description */
1463                 if (invert_this) {
1464                         trace_status = false;
1465                         do_not_exec_rule = false;
1466                 } else {
1467                         trace_status = true;
1468                         do_not_exec_rule = true;
1469                 }
1470                 return 0;
1471         case 'q':                       /* Question flag */
1472                 if (invert_this) {
1473                         quest = false;
```

```
1474                } else {
1475                        quest = true;
1476                }
1477                return 0;
1478        case 'R':                       /* Don't run in parallel */
1479                if (invert_this) {
1480                        pmake_cap_r_specified = false;
1481                        no_parallel = false;
1482                } else {
1483                        pmake_cap_r_specified = true;
1484                        dmake_mode_type = serial_mode;
1485                        no_parallel = true;
1486                }
1487                return 0;
1488        case 'r':                       /* Turn off internal rules */
1489                if (invert_this) {
1490                        ignore_default_mk = false;
1491                } else {
1492                        ignore_default_mk = true;
1493                }
1494                return 0;
1495        case 'S':                       /* Reverse -k */
1496                if (invert_this) {
1497                        continue_after_error = true;
1498                } else {
1499                        continue_after_error = false;
1500                        stop_after_error_ever_seen = true;
1501                }
1502                return 0;
1503        case 's':                       /* Silent flag */
1504                if (invert_this) {
1505                        silent_all = false;
1506                } else {
1507                        silent_all = true;
1508                }
1509                return 0;
1510        case 'T':                       /* Print target list */
1511                if (invert_this) {
1512                        list_all_targets = false;
1513                        do_not_exec_rule = false;
1514                } else {
1515                        list_all_targets = true;
1516                        do_not_exec_rule = true;
1517                }
1518                return 0;
1519        case 't':                       /* Touch flag */
1520                if (invert_this) {
1521                        touch = false;
1522                } else {
1523                        touch = true;
1524                }
1525                return 0;
1526        case 'u':                       /* Unconditional flag */
1527                if (invert_this) {
1528                        build_unconditional = false;
1529                } else {
1530                        build_unconditional = true;
1531                }
1532                return 0;
1533        case 'V':                      /* SVR4 mode */
1534                svr4 = true;
1535                return 0;
1536        case 'v':                      /* Version flag */
1537                if (invert_this) {
1538                } else {
1539                        fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
```



```
1540                        exit_status = 0;
1541                        exit(0);
1542                }
1543                return 0;
1544        case 'w':                       /* Unconditional flag */
1545                if (invert_this) {
1546                        report_cwd = false;
1547                } else {
1548                        report_cwd = true;
1549                }
1550                return 0;
1551 #if 0
1552        case 'X':                       /* Filter stdout */
1553                if (invert_this) {
1554                        filter_stderr = false;
1555                } else {
1556                        filter_stderr = true;
1557                }
1558                return 0;
1559 #endif
1560        default:
1561                break;
1562        }
1563        return 0;
1564 }

1566 /*
1567  *      setup_for_projectdir()
1568  *
1569  *      Read the PROJECTDIR variable, if defined, and set the sccs path
1570  *
1571  *      Parameters:
1572  *
1573  *      Global variables used:
1574  *              sccs_dir_path   Set to point to SCCS dir to use
1575  */
1576 static void
1577 setup_for_projectdir(void)
1578 {
1579 static char     path[MAXPATHLEN];
1580 char           cwdpath[MAXPATHLEN];
1581 uid_t uid;
1582 int   done=0;

1584        /* Check if we should use PROJECTDIR when reading the SCCS dir. */
1585        sccs_dir_path = getenv(NOCATGETS("PROJECTDIR"));
1586        if ((sccs_dir_path != NULL) &&
1587            (sccs_dir_path[0] != (int) slash_char)) {
1588                struct passwd *pwent;

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

```
1606                         if (access(path, F_OK) == 0) {
1607                                 sccs_dir_path = path;
1608                                 done = 1;
1609                         }
1610                     }
1611                 }
1612                 if (!done) {
1613                     if (getcwd(cwdpath, MAXPATHLEN - 1 )) {

1615                        (void) sprintf(path, NOCATGETS("%s/%s"), cwdpath,sccs_dir
1616                        if (access(path, F_OK) == 0) {
1617                                 sccs_dir_path = path;
1618                                 done = 1;
1619                         } else {
1620                                 fatal(catgets(catd, 1, 189, "Bogus PROJECTDIR '%
1621                         }
1622                     }
1623                 }
1624            }
1625         }
1626 }

1628 char *
1629 make_install_prefix(void)
1630 {
1631         int ret;
1632         char origin[PATH_MAX];
1633         char *dir;

1635         if ((ret = readlink("/proc/self/path/a.out", origin,
1636             PATH_MAX - 1)) < 0)
1637                 fatal("failed to read origin from /proc\n");

1639
1640         origin[ret] = '\0';
1641         return strdup(dirname(origin));
1642 }

1644 static char *
1645 add_to_env(const char *var, const char *value, const char *fallback)
1646 {
1647         const char *oldpath;
1648         char *newpath;

1650         oldpath = getenv(var);
1651         if (oldpath == NULL) {
1652                 if (value != NULL) {
1653                         asprintf(&newpath, "%s=%s",
1654                             var, value);
1655                 } else {
1656                         asprintf(&newpath, "%s=%s",
1657                             var, fallback);
1658                 }
1659         } else {
1660                 if (value != NULL) {
1661                         asprintf(&newpath, "%s=%s:%s",
1662                             var, oldpath, value);
1663                 } else {
1664                         asprintf(&newpath, "%s=%s:%s",
1665                             var, oldpath, fallback);
1666                 }
1667         }

1669         return (newpath);
1670 }
```



```
1672 #endif /* ! codereview */
1673 /*
1674  *      set_sgs_support()
1675  *
1676  *      Add the libmakestate.so.1 lib to the env var SGS_SUPPORT
1677  *        if it's not already in there.
1678  *      The SGS_SUPPORT env var and libmakestate.so.1 is used by
1679  *        the linker ld to report .make.state info back to make.
1680  *
1681  * In the new world we always will set the 32-bit and 64-bit versions of this
1682  * variable explicitly so that we can take into account the correct isa and our
1683  * prefix. So say that the prefix was /opt/local. Then we would want to search
1684  * /opt/local/lib/libmakestate.so.1:libmakestate.so.1. We still want to search
1685  * the original location just as a safety measure.
1686  */
1687 static void
1688 set_sgs_support()
1689 {
1690         int             len;
1691         char            *newpath, *newpath64;
1692         char            *lib32, *lib64;
1309         char            *oldpath, *oldpath64;
1693         static char     *prev_path, *prev_path64;
1694         char            *origin = make_install_prefix();
1695         struct stat st;
1696 #endif /* ! codereview */

1698         asprintf(&lib32, "%s/%s/%s", origin, "../lib",
1311         oldpath = getenv(LD_SUPPORT_ENV_VAR_32);
1312         if (oldpath == NULL) {
1313                 len = snprintf(NULL, 0, "%s=%s/%s/%s:%s",
1314                     LD_SUPPORT_ENV_VAR_32,
1315                     MAKE_PREFIX,
1316                     LD_SUPPORT_MAKE_LIB_DIR,
1317                     LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB) + 1;
1318                 newpath = (char *) malloc(len);
1319                 sprintf(newpath, "%s=%s/%s/%s:%s",
1320                     LD_SUPPORT_ENV_VAR_32,
1321                     MAKE_PREFIX,
1322                     LD_SUPPORT_MAKE_LIB_DIR,
1323                     LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB);
1324         } else {
1325                 len = snprintf(NULL, 0, "%s=%s:%s/%s/%s:%s",
1326                     LD_SUPPORT_ENV_VAR_32, oldpath, MAKE_PREFIX,
1327                     LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB,
1328                     LD_SUPPORT_MAKE_LIB) + 1;
1329                 newpath = (char *) malloc(len);
1330                 sprintf(newpath, "%s=%s:%s/%s/%s:%s",
1331                     LD_SUPPORT_ENV_VAR_32, oldpath, MAKE_PREFIX,
1332                     LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB,
1699                 LD_SUPPORT_MAKE_LIB);

1701         if (stat(lib32, &st) != 0) {
1702                 free(lib32);
1703                 // Try the tools path
1704                 asprintf(&lib32, "%s/%s/%s/%s", origin, "../../lib/",
1705                     LD_SUPPORT_MAKE_ARCH, LD_SUPPORT_MAKE_LIB);

1707                 if (stat(lib32, &st) != 0) {
1708                         free(lib32);
1709                         lib32 = NULL;
1710                 }
1711 #endif /* ! codereview */
1712         }

1714         asprintf(&lib64, "%s/%s/64/%s", origin, "../lib",
```

```
1334         oldpath64 = getenv(LD_SUPPORT_ENV_VAR_64);
1335         if (oldpath64 == NULL) {
1336                 len = snprintf(NULL, 0, "%s=%s/%s/%s/%s:%s",
1337                     LD_SUPPORT_ENV_VAR_64, MAKE_PREFIX, LD_SUPPORT_MAKE_LIB_DIR,
1338                     LD_SUPPORT_MAKE_LIB_DIR_64, LD_SUPPORT_MAKE_LIB,
1339                     LD_SUPPORT_MAKE_LIB) + 1;
1340                 newpath64 = (char *) malloc(len);
1341                 sprintf(newpath64, "%s=%s/%s/%s/%s:%s",
1342                     LD_SUPPORT_ENV_VAR_64, MAKE_PREFIX, LD_SUPPORT_MAKE_LIB_DIR,
1343                     LD_SUPPORT_MAKE_LIB_DIR_64, LD_SUPPORT_MAKE_LIB,
1715             LD_SUPPORT_MAKE_LIB);

1717         if (stat(lib64, &st) != 0) {
1718                 free(lib64);
1719                 // Try the tools path
1720                 asprintf(&lib64, "%s/%s/%s/64/%s", origin, "../../lib/",
1721                     LD_SUPPORT_MAKE_ARCH, LD_SUPPORT_MAKE_LIB);

1723                 if (stat(lib64, &st) != 0) {
1724                         free(lib64);
1725                         lib64 = NULL;
1726                 }
1345         } else {
1346                 len = snprintf(NULL, 0, "%s=%s:%s/%s/%s/%s:%s",
1347                     LD_SUPPORT_ENV_VAR_64, oldpath64, MAKE_PREFIX,
1348                     LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB_DIR_64,
1349                     LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB) + 1;
1350                 newpath64 = (char *) malloc(len);
1351                 sprintf(newpath64, "%s=%s:%s/%s/%s/%s:%s",
1352                     LD_SUPPORT_ENV_VAR_64, oldpath64, MAKE_PREFIX,
1353                     LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB_DIR_64,
1354                     LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB);
1727         }

1729         newpath = add_to_env(LD_SUPPORT_ENV_VAR_32, lib32, LD_SUPPORT_MAKE_LIB);
1730         newpath64 = add_to_env(LD_SUPPORT_ENV_VAR_64, lib64, LD_SUPPORT_MAKE_LIB

1732 #endif /* ! codereview */
1733         putenv(newpath);
1734         if (prev_path) {
1735                 free(prev_path);
1736         }
1737         prev_path = newpath;

1739         putenv(newpath64);
1740         if (prev_path64) {
1741                 free(prev_path64);
1742         }
1743         prev_path64 = newpath64;
1744         free(lib32);
1745         free(lib64);
1746         free(origin);
1747 #endif /* ! codereview */
1748 }

1750 /*
1751  *      read_files_and_state(argc, argv)
1752  *
1753  *      Read the makefiles we care about and the environment
1754  *      Also read the = style command line options
1755  *
1756  *      Parameters:
1757  *              argc            You know what this is
1758  *              argv            You know what this is
1759  *
1760  *      Static variables used:
```



```
1761  *              env_wins        make -e, determines if env vars are RO
1762  *              ignore_default_mk make -r, determines if make.rules is read
1763  *              not_auto_depen  dwight
1764  *
1765  *      Global variables used:
1766  *              default_target_to_build Set to first proper target from file
1767  *              do_not_exec_rule Set to false when makfile is made
1768  *              dot             The Name ".", used to read current dir
1769  *              empty_name      The Name "", use as macro value
1770  *              keep_state      Set if KEEP_STATE is in environment
1771  *              make_state      The Name ".make.state", used to read file
1772  *              makefile_type   Set to type of file being read
1773  *              makeflags       The Name "MAKEFLAGS", used to set macro value
1774  *              not_auto        dwight
1775  *              read_trace_level Checked to se if the reader should trace
1776  *              report_dependencies If -P is on we do not read .make.state
1777  *              trace_reader    Set if reader should trace
1778  *              virtual_root    The Name "VIRTUAL_ROOT", used to check value
1779  */
1780 static void
1781 read_files_and_state(int argc, char **argv)
1782 {
1783         wchar_t                 buffer[1000];
1784         wchar_t                 buffer_posix[1000];
1785         register char           ch;
1786         register char           *cp;
1787         Property                def_make_macro = NULL;
1788         Name                    def_make_name;
1789         Name                    default_makefile;
1790         String_rec              dest;
1791         wchar_t                 destbuffer[STRING_BUFFER_LENGTH];
1792         register int            i;
1793         register int            j;
1794         Name                    keep_state_name;
1795         int                     length;
1796         Name                    Makefile;
1797         register Property       macro;
1798         struct stat             make_state_stat;
1799         Name                    makefile_name;
1800         register int            makefile_next = 0;
1801         register Boolean        makefile_read = false;
1802         String_rec              makeflags_string;
1803         String_rec              makeflags_string_posix;
1804         String_rec *            makeflags_string_current;
1805         Name                    makeflags_value_saved;
1806         register Name           name;
1807         Name                    new_make_value;
1808         Boolean                 save_do_not_exec_rule;
1809         Name                    sdotMakefile;
1810         Name                    sdotmakefile_name;
1811         static wchar_t          state_file_str;
1812         static char             state_file_str_mb[MAXPATHLEN];
1813         static struct _Name     state_filename;
1814         Boolean                 temp;
1815         char                    tmp_char;
1816         wchar_t                 *tmp_wcs_buffer;
1817         register Name           value;
1818         ASCII_Dyn_Array         makeflags_and_macro;
1819         Boolean                 is_xpg4;

1821 /*
1822  *      Remember current mode. It may be changed after reading makefile
1823  *      and we will have to correct MAKEFLAGS variable.
1824  */
1825         is_xpg4 = posix;
```

```
1827         MBSTOWCS(wcs_buffer, NOCATGETS("KEEP_STATE"));
1828         keep_state_name = GETNAME(wcs_buffer, FIND_LENGTH);
1829         MBSTOWCS(wcs_buffer, NOCATGETS("Makefile"));
1830         Makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1831         MBSTOWCS(wcs_buffer, NOCATGETS("makefile"));
1832         makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
1833         MBSTOWCS(wcs_buffer, NOCATGETS("s.makefile"));
1834         sdotmakefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
1835         MBSTOWCS(wcs_buffer, NOCATGETS("s.Makefile"));
1836         sdotMakefile = GETNAME(wcs_buffer, FIND_LENGTH);

1838 /*
1839  *      initialize global dependency entry for .NOT_AUTO
1840  */
1841         not_auto_depen->next = NULL;
1842         not_auto_depen->name = not_auto;
1843         not_auto_depen->automatic = not_auto_depen->stale = false;

1845 /*
1846  *      Read internal definitions and rules.
1847  */
1848         if (read_trace_level > 1) {
1849                 trace_reader = true;
1850         }
1851         if (!ignore_default_mk) {
1852                 if (svr4) {
1853                         MBSTOWCS(wcs_buffer, NOCATGETS("svr4.make.rules"));
1854                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1855                 } else {
1856                         MBSTOWCS(wcs_buffer, NOCATGETS("make.rules"));
1857                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1858                 }
1859                 default_makefile->stat.is_file = true;

1861                 (void) read_makefile(default_makefile,
1862                                      true,
1863                                      false,
1864                                      true);
1865         }

1867         /*
1868          * If the user did not redefine the MAKE macro in the
1869          * default makefile (make.rules), then we'd like to
1870          * change the macro value of MAKE to be some form
1871          * of argv[0] for recursive MAKE builds.
1872          */
1873         MBSTOWCS(wcs_buffer, NOCATGETS("MAKE"));
1874         def_make_name = GETNAME(wcs_buffer, wslen(wcs_buffer));
1875         def_make_macro = get_prop(def_make_name->prop, macro_prop);
1876         if ((def_make_macro != NULL) &&
1877             (IS_EQUAL(def_make_macro->body.macro.value->string_mb,
1878                       NOCATGETS("make")))) {
1879                 MBSTOWCS(wcs_buffer, argv_zero_string);
1880                 new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
1881                 (void) SETVAR(def_make_name,
1882                               new_make_value,
1883                               false);
1884         }

1886         default_target_to_build = NULL;
1887         trace_reader = false;

1889 /*
1890  *      Read environment args. Let file args which follow override unless
1891  *      -e option seen. If -e option is not mentioned.
1892  */
```



```
1893         read_environment(env_wins);
1894         if (getvar(virtual_root)->hash.length == 0) {
1895                 maybe_append_prop(virtual_root, macro_prop)
1896                   ->body.macro.exported = true;
1897                 MBSTOWCS(wcs_buffer, "/");
1898                 (void) SETVAR(virtual_root,
1899                               GETNAME(wcs_buffer, FIND_LENGTH),
1900                               false);
1901         }

1903 /*
1904  * We now scan mf_argv and argv to see if we need to set
1905  * any of the DMake-added options/variables in MAKEFLAGS.
1906  */

1908         makeflags_and_macro.start = 0;
1909         makeflags_and_macro.size = 0;
1910         enter_argv_values(mf_argc, mf_argv, &makeflags_and_macro);
1911         enter_argv_values(argc, argv, &makeflags_and_macro);

1913 /*
1914  *      Set MFLAGS and MAKEFLAGS
1915  *
1916  *      Before reading makefile we do not know exactly which mode
1917  *      (posix or not) is used. So prepare two MAKEFLAGS strings
1918  *      for both posix and solaris modes because they are different.
1919  */
1920         INIT_STRING_FROM_STACK(makeflags_string, buffer);
1921         INIT_STRING_FROM_STACK(makeflags_string_posix, buffer_posix);
1922         append_char((int) hyphen_char, &makeflags_string);
1923         append_char((int) hyphen_char, &makeflags_string_posix);

1925         switch (read_trace_level) {
1926         case 2:
1927                 append_char('D', &makeflags_string);
1928                 append_char('D', &makeflags_string_posix);
1929         case 1:
1930                 append_char('D', &makeflags_string);
1931                 append_char('D', &makeflags_string_posix);
1932         }
1933         switch (debug_level) {
1934         case 2:
1935                 append_char('d', &makeflags_string);
1936                 append_char('d', &makeflags_string_posix);
1937         case 1:
1938                 append_char('d', &makeflags_string);
1939                 append_char('d', &makeflags_string_posix);
1940         }
1941         if (env_wins) {
1942                 append_char('e', &makeflags_string);
1943                 append_char('e', &makeflags_string_posix);
1944         }
1945         if (ignore_errors_all) {
1946                 append_char('i', &makeflags_string);
1947                 append_char('i', &makeflags_string_posix);
1948         }
1949         if (continue_after_error) {
1950                 if (stop_after_error_ever_seen) {
1951                         append_char('S', &makeflags_string_posix);
1952                         append_char((int) space_char, &makeflags_string_posix);
1953                         append_char((int) hyphen_char, &makeflags_string_posix);
1954                 }
1955                 append_char('k', &makeflags_string);
1956                 append_char('k', &makeflags_string_posix);
1957         } else {
1958                 if (stop_after_error_ever_seen
```

```
1959                    && continue_after_error_ever_seen) {
1960                        append_char('k', &makeflags_string_posix);
1961                        append_char((int) space_char, &makeflags_string_posix);
1962                        append_char((int) hyphen_char, &makeflags_string_posix);
1963                        append_char('S', &makeflags_string_posix);
1964                }
1965        }
1966        if (do_not_exec_rule) {
1967                append_char('n', &makeflags_string);
1968                append_char('n', &makeflags_string_posix);
1969        }
1970        switch (report_dependencies_level) {
1971        case 4:
1972                append_char('P', &makeflags_string);
1973                append_char('P', &makeflags_string_posix);
1974        case 3:
1975                append_char('P', &makeflags_string);
1976                append_char('P', &makeflags_string_posix);
1977        case 2:
1978                append_char('P', &makeflags_string);
1979                append_char('P', &makeflags_string_posix);
1980        case 1:
1981                append_char('P', &makeflags_string);
1982                append_char('P', &makeflags_string_posix);
1983        }
1984        if (trace_status) {
1985                append_char('p', &makeflags_string);
1986                append_char('p', &makeflags_string_posix);
1987        }
1988        if (quest) {
1989                append_char('q', &makeflags_string);
1990                append_char('q', &makeflags_string_posix);
1991        }
1992        if (silent_all) {
1993                append_char('s', &makeflags_string);
1994                append_char('s', &makeflags_string_posix);
1995        }
1996        if (touch) {
1997                append_char('t', &makeflags_string);
1998                append_char('t', &makeflags_string_posix);
1999        }
2000        if (build_unconditional) {
2001                append_char('u', &makeflags_string);
2002                append_char('u', &makeflags_string_posix);
2003        }
2004        if (report_cwd) {
2005                append_char('w', &makeflags_string);
2006                append_char('w', &makeflags_string_posix);
2007        }
2008        /* -c dmake_rcfile */
2009        if (dmake_rcfile_specified) {
2010                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2011                dmake_rcfile = GETNAME(wcs_buffer, FIND_LENGTH);
2012                append_makeflags_string(dmake_rcfile, &makeflags_string);
2013                append_makeflags_string(dmake_rcfile, &makeflags_string_posix);
2014        }
2015        /* -g dmake_group */
2016        if (dmake_group_specified) {
2017                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
2018                dmake_group = GETNAME(wcs_buffer, FIND_LENGTH);
2019                append_makeflags_string(dmake_group, &makeflags_string);
2020                append_makeflags_string(dmake_group, &makeflags_string_posix);
2021        }
2022        /* -j dmake_max_jobs */
2023        if (dmake_max_jobs_specified) {
2024                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
```



```
2025                dmake_max_jobs = GETNAME(wcs_buffer, FIND_LENGTH);
2026                append_makeflags_string(dmake_max_jobs, &makeflags_string);
2027                append_makeflags_string(dmake_max_jobs, &makeflags_string_posix)
2028        }
2029        /* -m dmake_mode */
2030        if (dmake_mode_specified) {
2031                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2032                dmake_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2033                append_makeflags_string(dmake_mode, &makeflags_string);
2034                append_makeflags_string(dmake_mode, &makeflags_string_posix);
2035        }
2036        /* -x dmake_compat_mode */
2037 //     if (dmake_compat_mode_specified) {
2038 //             MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE_COMPAT_MODE"));
2039 //             dmake_compat_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2040 //             append_makeflags_string(dmake_compat_mode, &makeflags_string);
2041 //             append_makeflags_string(dmake_compat_mode, &makeflags_string_pos
2042 //     }
2043        /* -x dmake_output_mode */
2044        if (dmake_output_mode_specified) {
2045                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
2046                dmake_output_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2047                append_makeflags_string(dmake_output_mode, &makeflags_string);
2048                append_makeflags_string(dmake_output_mode, &makeflags_string_pos
2049        }
2050        /* -o dmake_odir */
2051        if (dmake_odir_specified) {
2052                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2053                dmake_odir = GETNAME(wcs_buffer, FIND_LENGTH);
2054                append_makeflags_string(dmake_odir, &makeflags_string);
2055                append_makeflags_string(dmake_odir, &makeflags_string_posix);
2056        }
2057        /* -M pmake_machinesfile */
2058        if (pmake_machinesfile_specified) {
2059                MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
2060                pmake_machinesfile = GETNAME(wcs_buffer, FIND_LENGTH);
2061                append_makeflags_string(pmake_machinesfile, &makeflags_string);
2062                append_makeflags_string(pmake_machinesfile, &makeflags_string_po
2063        }
2064        /* -R */
2065        if (pmake_cap_r_specified) {
2066                append_char((int) space_char, &makeflags_string);
2067                append_char((int) hyphen_char, &makeflags_string);
2068                append_char('R', &makeflags_string);
2069                append_char((int) space_char, &makeflags_string_posix);
2070                append_char((int) hyphen_char, &makeflags_string_posix);
2071                append_char('R', &makeflags_string_posix);
2072        }

2074 /*
2075  *      Make sure MAKEFLAGS is exported
2076  */
2077        maybe_append_prop(makeflags, macro_prop)->
2078          body.macro.exported = true;

2080        if (makeflags_string.buffer.start[1] != (int) nul_char) {
2081                if (makeflags_string.buffer.start[1] != (int) space_char) {
2082                        MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2083                        (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2084                                      GETNAME(makeflags_string.buffer.start,
2085                                              FIND_LENGTH),
2086                                      false);
2087                } else {
2088                        MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2089                        (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2090                                      GETNAME(makeflags_string.buffer.start + 2,
```

```
2091                                            FIND_LENGTH),
2092                                    false);
2093                }
2094        }

2096 /*
2097  *      Add command line macro to POSIX makeflags_string
2098  */
2099        if (makeflags_and_macro.start) {
2100                tmp_char = (char) space_char;
2101                cp = makeflags_and_macro.start;
2102                do {
2103                        append_char(tmp_char, &makeflags_string_posix);
2104                } while ( tmp_char = *cp++ );
2105                retmem_mb(makeflags_and_macro.start);
2106        }

2108 /*
2109  *      Now set the value of MAKEFLAGS macro in accordance
2110  *      with current mode.
2111  */
2112        macro = maybe_append_prop(makeflags, macro_prop);
2113        temp = (Boolean) macro->body.macro.read_only;
2114        macro->body.macro.read_only = false;
2115        if(posix || gnu_style) {
2116                makeflags_string_current = &makeflags_string_posix;
2117        } else {
2118                makeflags_string_current = &makeflags_string;
2119        }
2120        if (makeflags_string_current->buffer.start[1] == (int) nul_char) {
2121                makeflags_value_saved =
2122                        GETNAME( makeflags_string_current->buffer.start + 1
2123                               , FIND_LENGTH
2124                               );
2125        } else {
2126                if (makeflags_string_current->buffer.start[1] != (int) space_cha
2127                        makeflags_value_saved =
2128                                GETNAME( makeflags_string_current->buffer.start
2129                                       , FIND_LENGTH
2130                                       );
2131                } else {
2132                        makeflags_value_saved =
2133                                GETNAME( makeflags_string_current->buffer.start
2134                                       , FIND_LENGTH
2135                                       );
2136                }
2137        }
2138        (void) SETVAR( makeflags
2139                     , makeflags_value_saved
2140                     , false
2141                     );
2142        macro->body.macro.read_only = temp;

2144 /*
2145  *      Read command line "-f" arguments and ignore -c, g, j, K, M, m, O and o a
2146  */
2147        save_do_not_exec_rule = do_not_exec_rule;
2148        do_not_exec_rule = false;
2149        if (read_trace_level > 0) {
2150                trace_reader = true;
2151        }

2153        for (i = 1; i < argc; i++) {
2154                if (argv[i] &&
2155                    (argv[i][0] == (int) hyphen_char) &&
2156                    (argv[i][1] == 'f') &&
```



```
2157                    (argv[i][2] == (int) nul_char)) {
2158                        argv[i] = NULL;         /* Remove -f */
2159                        if (i >= argc - 1) {
2160                                fatal(catgets(catd, 1, 190, "No filename argumen
2161                        }
2162                        MBSTOWCS(wcs_buffer, argv[++i]);
2163                        primary_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2164                        (void) read_makefile(primary_makefile, true, true, true)
2165                        argv[i] = NULL;         /* Remove filename */
2166                        makefile_read = true;
2167                } else if (argv[i] &&
2168                           (argv[i][0] == (int) hyphen_char) &&
2169                           (argv[i][1] == 'c' ||
2170                            argv[i][1] == 'g' ||
2171                            argv[i][1] == 'j' ||
2172                            argv[i][1] == 'K' ||
2173                            argv[i][1] == 'M' ||
2174                            argv[i][1] == 'm' ||
2175                            argv[i][1] == 'O' ||
2176                            argv[i][1] == 'o') &&
2177                           (argv[i][2] == (int) nul_char)) {
2178                        argv[i] = NULL;
2179                        argv[++i] = NULL;
2180                }
2181        }

2183 /*
2184  *      If no command line "-f" args then look for "makefile", and then for
2185  *      "Makefile" if "makefile" isn't found.
2186  */
2187        if (!makefile_read) {
2188                (void) read_dir(dot,
2189                                (wchar_t *) NULL,
2190                                (Property) NULL,
2191                                (wchar_t *) NULL);
2192            if (!posix) {
2193                if (makefile_name->stat.is_file) {
2194                        if (Makefile->stat.is_file) {
2195                                warning(catgets(catd, 1, 310, "Both 'makefile' a
2196                        }
2197                        primary_makefile = makefile_name;
2198                        makefile_read = read_makefile(makefile_name,
2199                                                      false,
2200                                                      false,
2201                                                      true);
2202                }
2203                if (!makefile_read &&
2204                    Makefile->stat.is_file) {
2205                        primary_makefile = Makefile;
2206                        makefile_read = read_makefile(Makefile,
2207                                                      false,
2208                                                      false,
2209                                                      true);
2210                }
2211            } else {

2213                enum sccs_stat save_m_has_sccs = NO_SCCS;
2214                enum sccs_stat save_M_has_sccs = NO_SCCS;

2216                if (makefile_name->stat.is_file) {
2217                        if (Makefile->stat.is_file) {
2218                                warning(catgets(catd, 1, 191, "Both 'makefile' a
2219                        }
2220                }
2221                if (makefile_name->stat.is_file) {
2222                        if (makefile_name->stat.has_sccs == NO_SCCS) {
```

```
2223                            primary_makefile = makefile_name;
2224                            makefile_read = read_makefile(makefile_name,
2225                                                          false,
2226                                                          false,
2227                                                          true);
2228                          } else {
2229                            save_m_has_sccs = makefile_name->stat.has_sccs;
2230                            makefile_name->stat.has_sccs = NO_SCCS;
2231                            primary_makefile = makefile_name;
2232                            makefile_read = read_makefile(makefile_name,
2233                                                          false,
2234                                                          false,
2235                                                          true);
2236                          }
2237                      }
2238                      if (!makefile_read &&
2239                          Makefile->stat.is_file) {
2240                              if (Makefile->stat.has_sccs == NO_SCCS) {
2241                                 primary_makefile = Makefile;
2242                                 makefile_read = read_makefile(Makefile,
2243                                                               false,
2244                                                               false,
2245                                                               true);
2246                              } else {
2247                                save_M_has_sccs = Makefile->stat.has_sccs;
2248                                Makefile->stat.has_sccs = NO_SCCS;
2249                                primary_makefile = Makefile;
2250                                makefile_read = read_makefile(Makefile,
2251                                                              false,
2252                                                              false,
2253                                                              true);
2254                              }
2255                      }
2256                      if (!makefile_read &&
2257                              makefile_name->stat.is_file) {
2258                                 makefile_name->stat.has_sccs = save_m_has_sccs;
2259                                 primary_makefile = makefile_name;
2260                                 makefile_read = read_makefile(makefile_name,
2261                                                               false,
2262                                                               false,
2263                                                               true);
2264                      }
2265                      if (!makefile_read &&
2266                          Makefile->stat.is_file) {
2267                                 Makefile->stat.has_sccs = save_M_has_sccs;
2268                                 primary_makefile = Makefile;
2269                                 makefile_read = read_makefile(Makefile,
2270                                                               false,
2271                                                               false,
2272                                                               true);
2273                      }
2274                  }
2275              }
2276              do_not_exec_rule = save_do_not_exec_rule;
2277              allrules_read = makefile_read;
2278              trace_reader = false;

2280      /*
2281       *      Now get current value of MAKEFLAGS and compare it with
2282       *      the saved value we set before reading makefile.
2283       *      If they are different then MAKEFLAGS is subsequently set by
2284       *      makefile, just leave it there. Otherwise, if make mode
2285       *      is changed by using .POSIX target in makefile we need
2286       *      to correct MAKEFLAGS value.
2287       */
2288              Name mf_val = getvar(makeflags);
```



```
2289              if( (posix != is_xpg4)
2290               && (!strcmp(mf_val->string_mb, makeflags_value_saved->string_mb)))
2291              {
2292                      if (makeflags_string_posix.buffer.start[1] == (int) nul_char) {
2293                              (void) SETVAR(makeflags,
2294                                            GETNAME(makeflags_string_posix.buffer.star
2295                                                    FIND_LENGTH),
2296                                            false);
2297                      } else {
2298                              if (makeflags_string_posix.buffer.start[1] != (int) spac
2299                                      (void) SETVAR(makeflags,
2300                                                    GETNAME(makeflags_string_posix.buf
2301                                                            FIND_LENGTH),
2302                                                    false);
2303                              } else {
2304                                      (void) SETVAR(makeflags,
2305                                                    GETNAME(makeflags_string_posix.buf
2306                                                            FIND_LENGTH),
2307                                                    false);
2308                              }
2309                      }
2310              }

2312              if (makeflags_string.free_after_use) {
2313                      retmem(makeflags_string.buffer.start);
2314              }
2315              if (makeflags_string_posix.free_after_use) {
2316                      retmem(makeflags_string_posix.buffer.start);
2317              }
2318              makeflags_string.buffer.start = NULL;
2319              makeflags_string_posix.buffer.start = NULL;

2321              if (posix) {
2322                      /*
2323                       * If the user did not redefine the ARFLAGS macro in the
2324                       * default makefile (make.rules), then we'd like to
2325                       * change the macro value of ARFLAGS to be in accordance
2326                       * with "POSIX" requirements.
2327                       */
2328                      MBSTOWCS(wcs_buffer, NOCATGETS("ARFLAGS"));
2329                      name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2330                      macro = get_prop(name->prop, macro_prop);
2331                      if ((macro != NULL) && /* Maybe (macro == NULL) || ? */
2332                          (IS_EQUAL(macro->body.macro.value->string_mb,
2333                                    NOCATGETS("rv")))) {
2334                              MBSTOWCS(wcs_buffer, NOCATGETS("-rv"));
2335                              value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2336                              (void) SETVAR(name,
2337                                            value,
2338                                            false);
2339                      }
2340              }

2342              if (!posix && !svr4) {
2343                      set_sgs_support();
2344              }


2347      /*
2348       *      Make sure KEEP_STATE is in the environment if KEEP_STATE is on.
2349       */
2350              macro = get_prop(keep_state_name->prop, macro_prop);
2351              if ((macro != NULL) &&
2352                  macro->body.macro.exported) {
2353                      keep_state = true;
2354              }
```

```
2355         if (keep_state) {
2356                 if (macro == NULL) {
2357                         macro = maybe_append_prop(keep_state_name,
2358                                                   macro_prop);
2359                 }
2360                 macro->body.macro.exported = true;
2361                 (void) SETVAR(keep_state_name,
2362                               empty_name,
2363                               false);
2365                 /*
2366                  *      Read state file
2367                  */
2369                 /* Before we read state, let's make sure we have
2370                 ** right state file.
2371                 */
2372                 /* just in case macro references are used in make_state file
2373                 ** name, we better expand them at this stage using expand_value.
2374                 */
2375                 INIT_STRING_FROM_STACK(dest, destbuffer);
2376                 expand_value(make_state, &dest, false);
2378                 make_state = GETNAME(dest.buffer.start, FIND_LENGTH);
2380                 if(!stat(make_state->string_mb, &make_state_stat)) {
2381                    if(!(make_state_stat.st_mode & S_IFREG) ) {
2382                         /* copy the make_state structure to the other
2383                         ** and then let make_state point to the new
2384                         ** one.
2385                         */
2386                        memcpy(&state_filename, make_state,sizeof(state_filename))
2387                        state_filename.string_mb = state_file_str_mb;
2388                 /* Just a kludge to avoid two slashes back to back */
2389                        if((make_state->hash.length == 1)&&
2390                                (make_state->string_mb[0] == '/')) {
2391                           make_state->hash.length = 0;
2392                           make_state->string_mb[0] = '\0';
2393                        }
2394                        sprintf(state_file_str_mb,NOCATGETS("%s%s"),
2395                         make_state->string_mb,NOCATGETS("/.make.state"));
2396                        make_state = &state_filename;
2397                          /* adjust the length to reflect the appended string */
2398                        make_state->hash.length += 12;
2399                    }
2400                 } else { /* the file doesn't exist or no permission */
2401                    char tmp_path[MAXPATHLEN];
2402                    char *slashp;
2404                    if (slashp = strrchr(make_state->string_mb, '/')) {
2405                       strncpy(tmp_path, make_state->string_mb,
2406                                (slashp - make_state->string_mb));
2407                         tmp_path[slashp - make_state->string_mb]=0;
2408                       if(strlen(tmp_path)) {
2409                         if(stat(tmp_path, &make_state_stat)) {
2410                           warning(catgets(catd, 1, 192, "directory %s for .KEEP_
2411                         }
2412                         if (access(tmp_path, F_OK) != 0) {
2413                           warning(catgets(catd, 1, 193, "can't access dir %s"),t
2414                         }
2415                       }
2416                    }
2417                 }
2418                 if (report_dependencies_level != 1) {
2419                         Makefile_type   makefile_type_temp = makefile_type;
2420                         makefile_type = reading_statefile;
```



```
2421                         if (read_trace_level > 1) {
2422                                 trace_reader = true;
2423                         }
2424                         (void) read_simple_file(make_state,
2425                                                 false,
2426                                                 false,
2427                                                 false,
2428                                                 false,
2429                                                 false,
2430                                                 true);
2431                         trace_reader = false;
2432                         makefile_type = makefile_type_temp;
2433                 }
2434         }
2435 }
2437 /*
2438  * Scan the argv for options and "=" type args and make them readonly.
2439  */
2440 static void
2441 enter_argv_values(int argc, char *argv[], ASCII_Dyn_Array *makeflags_and_macro)
2442 {
2443         register char           *cp;
2444         register int            i;
2445         int                     length;
2446         register Name           name;
2447         int                     opt_separator = argc;
2448         char                    tmp_char;
2449         wchar_t                 *tmp_wcs_buffer;
2450         register Name           value;
2451         Boolean                 append = false;
2452         Property                macro;
2453         struct stat             statbuf;
2456         /* Read argv options and "=" type args and make them readonly. */
2457         makefile_type = reading_nothing;
2458         for (i = 1; i < argc; ++i) {
2459                 append = false;
2460                 if (argv[i] == NULL) {
2461                         continue;
2462                 } else if (((argv[i][0] == '-') && (argv[i][1] == '-')) ||
2463                            ((argv[i][0] == (int) ' ') &&
2464                             (argv[i][1] == (int) '-') &&
2465                             (argv[i][2] == (int) ' ') &&
2466                             (argv[i][3] == (int) '-'))) {
2467                         argv[i] = NULL;
2468                         opt_separator = i;
2469                         continue;
2470                 } else if ((i < opt_separator) && (argv[i][0] == (int) hyphen_ch
2471                         switch (parse_command_option(argv[i][1])) {
2472                         case 1: /* -f seen */
2473                                 ++i;
2474                                 continue;
2475                         case 2: /* -c seen */
2476                                 if (argv[i+1] == NULL) {
2477                                         fatal(catgets(catd, 1, 194, "No dmake rc
2478                                 }
2479                                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2480                                 name = GETNAME(wcs_buffer, FIND_LENGTH);
2481                                 break;
2482                         case 4: /* -g seen */
2483                                 if (argv[i+1] == NULL) {
2484                                         fatal(catgets(catd, 1, 195, "No dmake gr
2485                                 }
2486                                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
```

```
2487                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2488                                    break;
2489                            case 8: /* -j seen */
2490                                    if (argv[i+1] == NULL) {
2491                                            fatal(catgets(catd, 1, 196, "No dmake ma
2492                                    }
2493                                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS")
2494                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2495                                    break;
2496                            case 16: /* -M seen */
2497                                    if (argv[i+1] == NULL) {
2498                                            fatal(catgets(catd, 1, 323, "No pmake ma
2499                                    }
2500                                    MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFI
2501                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2502                                    break;
2503                            case 32: /* -m seen */
2504                                    if (argv[i+1] == NULL) {
2505                                            fatal(catgets(catd, 1, 197, "No dmake mo
2506                                    }
2507                                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2508                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2509                                    break;
2510                            case 128: /* -O seen */
2511                                    if (argv[i+1] == NULL) {
2512                                            fatal(catgets(catd, 1, 287, "No file des
2513                                    }
2514                                    mtool_msgs_fd = atoi(argv[i+1]);
2515                                    /* find out if mtool_msgs_fd is a valid file des
2516                                    if (fstat(mtool_msgs_fd, &statbuf) < 0) {
2517                                            fatal(catgets(catd, 1, 355, "Invalid fil
2518                                    }
2519                                    argv[i] = NULL;
2520                                    argv[i+1] = NULL;
2521                                    continue;
2522                            case 256: /* -K seen */
2523                                    if (argv[i+1] == NULL) {
2524                                            fatal(catgets(catd, 1, 288, "No makestat
2525                                    }
2526                                    MBSTOWCS(wcs_buffer, argv[i+1]);
2527                                    make_state = GETNAME(wcs_buffer, FIND_LENGTH);
2528                                    keep_state = true;
2529                                    argv[i] = NULL;
2530                                    argv[i+1] = NULL;
2531                                    continue;
2532                            case 512:       /* -o seen */
2533                                    if (argv[i+1] == NULL) {
2534                                            fatal(catgets(catd, 1, 312, "No dmake ou
2535                                    }
2536                                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2537                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2538                                    break;
2539                            case 1024: /* -x seen */
2540                                    if (argv[i+1] == NULL) {
2541                                            fatal(catgets(catd, 1, 351, "No argument
2542                                    }
2543                                    length = strlen( NOCATGETS("SUN_MAKE_COMPAT_MODE
2544                                    if (strncmp(argv[i+1], NOCATGETS("SUN_MAKE_COMPA
2545                                            argv[i+1] = &argv[i+1][length];
2546                                            MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE
2547                                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2548                                            dmake_compat_mode_specified = dmake_add_
2549                                            break;
2550                                    }
2551                                    length = strlen( NOCATGETS("DMAKE_OUTPUT_MODE=")
2552                                    if (strncmp(argv[i+1], NOCATGETS("DMAKE_OUTPUT_M
```



```
2553                                            argv[i+1] = &argv[i+1][length];
2554                                            MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OU
2555                                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2556                                            dmake_output_mode_specified = dmake_add_
2557                                    } else {
2558                                            warning(catgets(catd, 1, 354, "Unknown a
2559                                                  argv[i+1]);
2560                                            argv[i] = argv[i + 1] = NULL;
2561                                            continue;
2562                                    }
2563                                    break;
2564                            default: /* Shouldn't reach here */
2565                                    argv[i] = NULL;
2566                                    continue;
2567                            }
2568                            argv[i] = NULL;
2569                            if (i == (argc - 1)) {
2570                                    break;
2571                            }
2572                            if ((length = strlen(argv[i+1])) >= MAXPATHLEN) {
2573                                    tmp_wcs_buffer = ALLOC_WC(length + 1);
2574                                    (void) mbstowcs(tmp_wcs_buffer, argv[i+1], lengt
2575                                    value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2576                                    retmem(tmp_wcs_buffer);
2577                            } else {
2578                                    MBSTOWCS(wcs_buffer, argv[i+1]);
2579                                    value = GETNAME(wcs_buffer, FIND_LENGTH);
2580                            }
2581                            argv[i+1] = NULL;
2582                    } else if ((cp = strchr(argv[i], (int) equal_char)) != NULL) {
2583 /*
2584  * Combine all macro in dynamic array
2585  */
2586                            if(*(cp-1) == (int) plus_char)
2587                            {
2588                                    if(isspace(*(cp-2))) {
2589                                            append = true;
2590                                            cp--;
2591                                    }
2592                            }
2593                            if(!append)
2594                                    append_or_replace_macro_in_dyn_array(makeflags_a
2596                            while (isspace(*(cp-1))) {
2597                                    cp--;
2598                            }
2599                            tmp_char = *cp;
2600                            *cp = (int) nul_char;
2601                            MBSTOWCS(wcs_buffer, argv[i]);
2602                            *cp = tmp_char;
2603                            name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2604                            while (*cp != (int) equal_char) {
2605                                    cp++;
2606                            }
2607                            cp++;
2608                            while (isspace(*cp) && (*cp != (int) nul_char)) {
2609                                    cp++;
2610                            }
2611                            if ((length = strlen(cp)) >= MAXPATHLEN) {
2612                                    tmp_wcs_buffer = ALLOC_WC(length + 1);
2613                                    (void) mbstowcs(tmp_wcs_buffer, cp, length + 1);
2614                                    value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2615                                    retmem(tmp_wcs_buffer);
2616                            } else {
2617                                    MBSTOWCS(wcs_buffer, cp);
2618                                    value = GETNAME(wcs_buffer, FIND_LENGTH);
```

```
2619                        }
2620                        argv[i] = NULL;
2621                } else {
2622                        /* Illegal MAKEFLAGS argument */
2623                        continue;
2624                }
2625                if(append) {
2626                        setvar_append(name, value);
2627                        append = false;
2628                } else {
2629                        macro = maybe_append_prop(name, macro_prop);
2630                        macro->body.macro.exported = true;
2631                        SETVAR(name, value, false)->body.macro.read_only = true;
2632                }
2633        }
2634 }

2636 /*
2637  * Append the DMake option and value to the MAKEFLAGS string.
2638  */
2639 static void
2640 append_makeflags_string(Name name, register String makeflags_string)
2641 {
2642        const char      *option;

2644        if (strcmp(name->string_mb, NOCATGETS("DMAKE_GROUP")) == 0) {
2645                option = NOCATGETS(" -g ");
2646        } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_MAX_JOBS")) == 0) {
2647                option = NOCATGETS(" -j ");
2648        } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_MODE")) == 0) {
2649                option = NOCATGETS(" -m ");
2650        } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_ODIR")) == 0) {
2651                option = NOCATGETS(" -o ");
2652        } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_RCFILE")) == 0) {
2653                option = NOCATGETS(" -c ");
2654        } else if (strcmp(name->string_mb, NOCATGETS("PMAKE_MACHINESFILE")) == 0
2655                option = NOCATGETS(" -M ");
2656        } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_OUTPUT_MODE")) == 0)
2657                option = NOCATGETS(" -x DMAKE_OUTPUT_MODE=");
2658        } else if (strcmp(name->string_mb, NOCATGETS("SUN_MAKE_COMPAT_MODE")) ==
2659                option = NOCATGETS(" -x SUN_MAKE_COMPAT_MODE=");
2660        } else {
2661                fatal(catgets(catd, 1, 289, "Internal error: name not recognized
2662        }
2663        Property prop = maybe_append_prop(name, macro_prop);
2664        if( prop == 0 || prop->body.macro.value == 0 ||
2665            prop->body.macro.value->string_mb == 0 ) {
2666                return;
2667        }
2668        char mbs_value[MAXPATHLEN + 100];
2669        strcpy(mbs_value, option);
2670        strcat(mbs_value, prop->body.macro.value->string_mb);
2671        MBSTOWCS(wcs_buffer, mbs_value);
2672        append_string(wcs_buffer, makeflags_string, FIND_LENGTH);
2673 }

2675 /*
2676  *      read_environment(read_only)
2677  *
2678  *      This routine reads the process environment when make starts and enters
2679  *      it as make macros. The environment variable SHELL is ignored.
2680  *
2681  *      Parameters:
2682  *              read_only       Should we make env vars read only?
2683  *
2684  *      Global variables used:
```



```
2685  *              report_pwd      Set if this make was started by other make
2686  */
2687 static void
2688 read_environment(Boolean read_only)
2689 {
2690        register char           **environment;
2691        int                     length;
2692        wchar_t                 *tmp_wcs_buffer;
2693        Boolean                 alloced_tmp_wcs_buffer = false;
2694        register wchar_t        *name;
2695        register wchar_t        *value;
2696        register Name           macro;
2697        Property                val;
2698        Boolean                 read_only_saved;

2700        reading_environment = true;
2701        environment = environ;
2702        for (; *environment; environment++) {
2703                read_only_saved = read_only;
2704                if ((length = strlen(*environment)) >= MAXPATHLEN) {
2705                        tmp_wcs_buffer = ALLOC_WC(length + 1);
2706                        alloced_tmp_wcs_buffer = true;
2707                        (void) mbstowcs(tmp_wcs_buffer, *environment, length + 1
2708                        name = tmp_wcs_buffer;
2709                } else {
2710                        MBSTOWCS(wcs_buffer, *environment);
2711                        name = wcs_buffer;
2712                }
2713                value = (wchar_t *) wschr(name, (int) equal_char);

2715                /*
2716                 * Looks like there's a bug in the system, but sometimes
2717                 * you can get blank lines in *environment.
2718                 */
2719                if (!value) {
2720                        continue;
2721                }
2722                MBSTOWCS(wcs_buffer2, NOCATGETS("SHELL="));
2723                if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2724                        continue;
2725                }
2726                MBSTOWCS(wcs_buffer2, NOCATGETS("MAKEFLAGS="));
2727                if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2728                        report_pwd = true;
2729                        /*
2730                         * In POSIX mode we do not want MAKEFLAGS to be readonly
2731                         * If the MAKEFLAGS macro is subsequently set by the mak
2732                         * it replaces the MAKEFLAGS variable currently found in
2733                         * environment.
2734                         * See Assertion 50 in section 6.2.5.3 of standard P1003
2735                         */
2736                        if(posix) {
2737                                read_only_saved = false;
2738                        }
2739                }

2741                /*
2742                 * We ignore SUNPRO_DEPENDENCIES. This environment variable is
2743                 * set by make and read by cpp which then writes info to
2744                 * .make.dependency.xxx.  When make is invoked by another make
2745                 * (recursive make), we don't want to read this because then
2746                 * the child make will end up writing to the parent
2747                 * directory's .make.state and clobbering them.
2748                 */
2749                MBSTOWCS(wcs_buffer2, NOCATGETS("SUNPRO_DEPENDENCIES"));
2750                if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
```

```
2751                        continue;
2752                }

2754                macro = GETNAME(name, value - name);
2755                maybe_append_prop(macro, macro_prop)->body.macro.exported =
2756                  true;
2757                if ((value == NULL) || ((value + 1)[0] == (int) nul_char)) {
2758                        val = setvar_daemon(macro,
2759                                            (Name) NULL,
2760                                            false, no_daemon, false, debug_level
2761                } else {
2762                        val = setvar_daemon(macro,
2763                                            GETNAME(value + 1, FIND_LENGTH),
2764                                            false, no_daemon, false, debug_level
2765                }
2766                val->body.macro.read_only = read_only_saved;
2767                if (alloced_tmp_wcs_buffer) {
2768                        retmem(tmp_wcs_buffer);
2769                        alloced_tmp_wcs_buffer = false;
2770                }
2771        }
2772        reading_environment = false;
2773 }

2775 /*
2776  *      read_makefile(makefile, complain, must_exist, report_file)
2777  *
2778  *      Read one makefile and check the result
2779  *
2780  *      Return value:
2781  *                              false is the read failed
2782  *
2783  *      Parameters:
2784  *              makefile        The file to read
2785  *              complain        Passed thru to read_simple_file()
2786  *              must_exist      Passed thru to read_simple_file()
2787  *              report_file     Passed thru to read_simple_file()
2788  *
2789  *      Global variables used:
2790  *              makefile_type   Set to indicate we are reading main file
2791  *              recursion_level Initialized
2792  */
2793 static Boolean
2794 read_makefile(register Name makefile, Boolean complain, Boolean must_exist, Bool
2795 {
2796        Boolean                 b;
2797
2798        makefile_type = reading_makefile;
2799        recursion_level = 0;
2800        reading_dependencies = true;
2801        b = read_simple_file(makefile, true, true, complain,
2802                             must_exist, report_file, false);
2803        reading_dependencies = false;
2804        return b;
2805 }

2807 /*
2808  *      make_targets(argc, argv, parallel_flag)
2809  *
2810  *      Call doname on the specified targets
2811  *
2812  *      Parameters:
2813  *              argc            You know what this is
2814  *              argv            You know what this is
2815  *              parallel_flag   True if building in parallel
2816  *
```



```
2817  *      Global variables used:
2818  *              build_failed_seen Used to generated message after failed -k
2819  *              commands_done   Used to generate message "Up to date"
2820  *              default_target_to_build First proper target in makefile
2821  *              init            The Name ".INIT", use to run command
2822  *              parallel        Global parallel building flag
2823  *              quest           make -q, suppresses messages
2824  *              recursion_level Initialized, used for tracing
2825  *              report_dependencies make -P, regroves whole process
2826  */
2827 static void
2828 make_targets(int argc, char **argv, Boolean parallel_flag)
2829 {
2830        int                     i;
2831        char                    *cp;
2832        Doname                  result;
2833        register Boolean        target_to_make_found = false;

2835        (void) doname(init, true, true);
2836        recursion_level = 1;
2837        parallel = parallel_flag;
2838 /*
2839  *      make remaining args
2840  */
2841 /*
2842        if ((report_dependencies_level == 0) && parallel) {
2843  */
2844        if (parallel) {
2845                /*
2846                 * If building targets in parallel, start all of the
2847                 * remaining args to build in parallel.
2848                 */
2849                for (i = 1; i < argc; i++) {
2850                        if ((cp = argv[i]) != NULL) {
2851                                commands_done = false;
2852                                if ((cp[0] == (int) period_char) &&
2853                                    (cp[1] == (int) slash_char)) {
2854                                        cp += 2;
2855                                }
2856                                 if((cp[0] == (int) ' ') &&
2857                                    (cp[1] == (int) '-') &&
2858                                    (cp[2] == (int) ' ') &&
2859                                    (cp[3] == (int) '-')) {
2860                                    argv[i] = NULL;
2861                                        continue;
2862                                }
2863                                MBSTOWCS(wcs_buffer, cp);
2864                                //default_target_to_build = GETNAME(wcs_buffer,
2865                                //                                  FIND_LENGTH);
2866                                default_target_to_build = normalize_name(wcs_buf
2867                                                                  wslen(wcs_buff
2868                                if (default_target_to_build == wait_name) {
2869                                        if (parallel_process_cnt > 0) {
2870                                                finish_running();
2871                                        }
2872                                        continue;
2873                                }
2874                                top_level_target = get_wstring(default_target_to
2875                                /*
2876                                 * If we can't execute the current target in
2877                                 * parallel, hold off the target processing
2878                                 * to preserve the order of the targets as they
2879                                 * in command line.
2880                                 */
2881                                if (!parallel_ok(default_target_to_build, false)
2882                                                        && parallel_process_cnt > 0) {
```

```
2883                                    finish_running();
2884                            }
2885                            result = doname_check(default_target_to_build,
2886                                                  true,
2887                                                  false,
2888                                                  false);
2889                            gather_recursive_deps();
2890                            if (/* !commands_done && */
2891                                (result == build_ok) &&
2892                                !quest &&
2893                                (report_dependencies_level == 0) /*  &&
2894                                (exists(default_target_to_build) > file_does
2895                                    if (posix) {
2896                                            if (!commands_done) {
2897                                                    (void) printf(catgets(ca
2898                                                                          default_ta
2899                                            } else {
2900                                                    if (no_action_was_taken)
2901                                                            (void) printf(ca
2902                                                                          de
2903                                                    }
2904                                            }
2905                                    } else {
2906                                            default_target_to_build->stat.ti
2907                                            if (!commands_done &&
2908                                                (exists(default_target_to_bu
2909                                                    (void) printf(catgets(ca
2910                                                                          default_ta
2911                                            }
2912                                    }
2913                            }
2914                    }
2915            }
2916            /* Now wait for all of the targets to finish running */
2917            finish_running();
2918            //              setjmp(jmpbuffer);
2919
2920        }
2921        for (i = 1; i < argc; i++) {
2922                if ((cp = argv[i]) != NULL) {
2923                        target_to_make_found = true;
2924                        if ((cp[0] == (int) period_char) &&
2925                            (cp[1] == (int) slash_char)) {
2926                                cp += 2;
2927                        }
2928                                 if((cp[0] == (int) ' ') &&
2929                                    (cp[1] == (int) '-') &&
2930                                    (cp[2] == (int) ' ') &&
2931                                    (cp[3] == (int) '-')) {
2932                                    argv[i] = NULL;
2933                                        continue;
2934                                }
2935                        MBSTOWCS(wcs_buffer, cp);
2936                        default_target_to_build = normalize_name(wcs_buffer, wsl
2937                        top_level_target = get_wstring(default_target_to_build->
2938                        report_recursion(default_target_to_build);
2939                        commands_done = false;
2940                        if (parallel) {
2941                                result = (Doname) default_target_to_build->state
2942                        } else {
2943                                result = doname_check(default_target_to_build,
2944                                                      true,
2945                                                      false,
2946                                                      false);
2947                        }
2948                        gather_recursive_deps();
```



```
2949                        if (build_failed_seen) {
2950                                build_failed_ever_seen = true;
2951                                warning(catgets(catd, 1, 200, "Target '%s' not r
2952                                        default_target_to_build->string_mb);
2953                        }
2954                        build_failed_seen = false;
2955                        if (report_dependencies_level > 0) {
2956                                print_dependencies(default_target_to_build,
2957                                                   get_prop(default_target_to_bu
2958                                                            line_prop));
2959                        }
2960                        default_target_to_build->stat.time =
2961                          file_no_time;
2962                        if (default_target_to_build->colon_splits > 0) {
2963                                default_target_to_build->state =
2964                                  build_dont_know;
2965                        }
2966                        if (!parallel &&
2967                            /* !commands_done && */
2968                            (result == build_ok) &&
2969                            !quest &&
2970                            (report_dependencies_level == 0) /*  &&
2971                            (exists(default_target_to_build) > file_doesnt_exist
2972                                if (posix) {
2973                                        if (!commands_done) {
2974                                                (void) printf(catgets(catd, 1, 2
2975                                                                      default_target_to_
2976                                        } else {
2977                                                if (no_action_was_taken) {
2978                                                        (void) printf(catgets(ca
2979                                                                              default_ta
2980                                                }
2981                                        }
2982                                } else {
2983                                        if (!commands_done &&
2984                                            (exists(default_target_to_build) > f
2985                                                (void) printf(catgets(catd, 1, 2
2986                                                                      default_target_to_
2987                                        }
2988                                }
2989                        }
2990                }
2991        }

2993 /*
2994  *      If no file arguments have been encountered,
2995  *      make the first name encountered that doesnt start with a dot
2996  */
2997        if (!target_to_make_found) {
2998                if (default_target_to_build == NULL) {
2999                        fatal(catgets(catd, 1, 202, "No arguments to build"));
3000                }
3001                commands_done = false;
3002                top_level_target = get_wstring(default_target_to_build->string_m
3003                report_recursion(default_target_to_build);


3006                if (getenv(NOCATGETS("SPRO_EXPAND_ERRORS"))){
3007                        (void) printf(NOCATGETS("::(%s)\n"),
3008                                      default_target_to_build->string_mb);
3009                }


3012                result = doname_parallel(default_target_to_build, true, false);
3013                gather_recursive_deps();
3014                if (build_failed_seen) {
```

```
3015                        build_failed_ever_seen = true;
3016                        warning(catgets(catd, 1, 203, "Target `%s' not remade be
3017                                default_target_to_build->string_mb);
3018                }
3019                build_failed_seen = false;
3020                if (report_dependencies_level > 0) {
3021                        print_dependencies(default_target_to_build,
3022                                           get_prop(default_target_to_build->
3023                                                    prop,
3024                                                    line_prop));
3025                }
3026                default_target_to_build->stat.time = file_no_time;
3027                if (default_target_to_build->colon_splits > 0) {
3028                        default_target_to_build->state = build_dont_know;
3029                }
3030                if (/* !commands_done && */
3031                    (result == build_ok) &&
3032                    !quest &&
3033                    (report_dependencies_level == 0) /*  &&
3034                    (exists(default_target_to_build) > file_doesnt_exist)  */) {
3035                        if (posix) {
3036                                if (!commands_done) {
3037                                        (void) printf(catgets(catd, 1, 299, "`%s
3038                                                      default_target_to_build->s
3039                                } else {
3040                                        if (no_action_was_taken) {
3041                                                (void) printf(catgets(catd, 1, 3
3042                                                              default_target_to_
3043                                        }
3044                                }
3045                        } else {
3046                                if (!commands_done &&
3047                                    (exists(default_target_to_build) > file_does
3048                                        (void) printf(catgets(catd, 1, 301, "`%s
3049                                                      default_target_to_build->s
3050                                }
3051                        }
3052                }
3053        }
3054 }

3056 /*
3057  *      report_recursion(target)
3058  *
3059  *      If this is a recursive make and the parent make has KEEP_STATE on
3060  *      this routine reports the dependency to the parent make
3061  *
3062  *      Parameters:
3063  *              target          Target to report
3064  *
3065  *      Global variables used:
3066  *              makefiles_used          List of makefiles read
3067  *              recursive_name          The Name ".RECURSIVE", printed
3068  *              report_dependency       dwight
3069  */
3070 static void
3071 report_recursion(register Name target)
3072 {
3073        register FILE           *report_file = get_report_file();

3075        if ((report_file == NULL) || (report_file == (FILE*)-1)) {
3076                return;
3077        }
3078        if (primary_makefile == NULL) {
3079                /*
3080                 * This can happen when there is no makefile and
```



```
3081                 * only implicit rules are being used.
3082                 */
3083                return;
3084        }
3085        (void) fprintf(report_file,
3086                       "%s: %s ",
3087                       get_target_being_reported_for(),
3088                       recursive_name->string_mb);
3089        report_dependency(get_current_path());
3090        report_dependency(target->string_mb);
3091        report_dependency(primary_makefile->string_mb);
3092        (void) fprintf(report_file, "\n");
3093 }

3095 /* Next function "append_or_replace_macro_in_dyn_array" must be in "misc.cc". */
3096 /* NIKMOL */
3097 extern void
3098 append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar, char *macro)
3099 {
3100        register char   *cp0;   /* work pointer in macro */
3101        register char   *cp1;   /* work pointer in array */
3102        register char   *cp2;   /* work pointer in array */
3103        register char   *cp3;   /* work pointer in array */
3104        register char   *name;  /* macro name */
3105        register char   *value; /* macro value */
3106        register int    len_array;
3107        register int    len_macro;

3109        char * esc_value = NULL;
3110        int esc_len;

3112        if (!(len_macro = strlen(macro))) return;
3113        name = macro;
3114        while (isspace(*(name))) {
3115                name++;
3116        }
3117        if (!(value = strchr(name, (int) equal_char))) {
3118                /* no `=' in macro */
3119                goto ERROR_MACRO;
3120        }
3121        cp0 = value;
3122        value++;
3123        while (isspace(*(value))) {
3124                value++;
3125        }
3126        while (isspace(*(cp0-1))) {
3127                cp0--;
3128        }
3129        if (cp0 <= name) goto ERROR_MACRO; /* no name */
3130        if (!(Ar->size)) goto ALLOC_ARRAY;
3131        cp1 = Ar->start;

3133 LOOK_FOR_NAME:
3134        if (!(cp1 = strchr(cp1, name[0]))) goto APPEND_MACRO;
3135        if (!(cp2 = strchr(cp1, (int) equal_char))) goto APPEND_MACRO;
3136        if (strncmp(cp1, name, (size_t)(cp0-name))) {
3137                /* another name */
3138                cp1++;
3139                goto LOOK_FOR_NAME;
3140        }
3141        if (cp1 != Ar->start) {
3142                if (!isspace(*(cp1-1))) {
3143                        /* another name */
3144                        cp1++;
3145                        goto LOOK_FOR_NAME;
3146                }
```

```
3147         }
3148         for (cp3 = cp1 + (cp0-name); cp3 < cp2; cp3++) {
3149                 if (isspace(*cp3)) continue;
3150                 /* else: another name */
3151                 cp1++;
3152                 goto LOOK_FOR_NAME;
3153         }
3154         /* Look for the next macro name in array */
3155         cp3 = cp2+1;
3156         if (*cp3 != (int) doublequote_char) {
3157                 /* internal error */
3158                 goto ERROR_MACRO;
3159         }
3160         if (!(cp3 = strchr(cp3+1, (int) doublequote_char))) {
3161                 /* internal error */
3162                 goto ERROR_MACRO;
3163         }
3164         cp3++;
3165         while (isspace(*cp3)) {
3166                 cp3++;
3167         }
3168
3169         cp2 = cp1;  /* remove old macro */
3170         if ((*cp3) && (cp3 < Ar->start + Ar->size)) {
3171                 for (; cp3 < Ar->start + Ar->size; cp3++) {
3172                         *cp2++ = *cp3;
3173                 }
3174         }
3175         for (; cp2 < Ar->start + Ar->size; cp2++) {
3176                 *cp2 = 0;
3177         }
3178         if (*cp1) {
3179                 /* check next name */
3180                 goto LOOK_FOR_NAME;
3181         }
3182         goto APPEND_MACRO;

3184 ALLOC_ARRAY:
3185         if (Ar->size) {
3186                 cp1 = Ar->start;
3187         } else {
3188                 cp1 = 0;
3189         }
3190         Ar->size += 128;
3191         Ar->start = getmem(Ar->size);
3192         for (len_array=0; len_array < Ar->size; len_array++) {
3193                 Ar->start[len_array] = 0;
3194         }
3195         if (cp1) {
3196                 strcpy(Ar->start, cp1);
3197                 retmem((wchar_t *) cp1);
3198         }

3200 APPEND_MACRO:
3201         len_array = strlen(Ar->start);
3202         esc_value = (char*)malloc(strlen(value)*2 + 1);
3203         quote_str(value, esc_value);
3204         esc_len = strlen(esc_value) - strlen(value);
3205         if (len_array + len_macro + esc_len + 5 >= Ar->size) goto  ALLOC_ARRAY;
3206         strcat(Ar->start, " ");
3207         strncat(Ar->start, name, cp0-name);
3208         strcat(Ar->start, "=");
3209         strncat(Ar->start, esc_value, strlen(esc_value));
3210         free(esc_value);
3211         return;
3212 ERROR_MACRO:
```



```
3213         /* Macro without '=' or with invalid left/right part */
3214         return;
3215 }

3217 #ifdef TEAMWARE_MAKE_CMN
3218 /*
3219  * This function, if registered w/ avo_cli_get_license(), will be called
3220  * if the application is about to exit because:
3221  *   1) there has been certain unrecoverable error(s) that cause the
3222  *      application to exit immediately.
3223  *   2) the user has lost a license while the application is running.
3224  */
3225 extern "C" void
3226 dmake_exit_callback(void)
3227 {
3228         fatal(catgets(catd, 1, 306, "can not get a license, exiting..."));
3229         exit(1);
3230 }

3232 /*
3233  * This function, if registered w/ avo_cli_get_license(), will be called
3234  * if the application can not get a license.
3235  */
3236 extern "C" void
3237 dmake_message_callback(char *err_msg)
3238 {
3239         static Boolean  first = true;

3241         if (!first) {
3242                 return;
3243         }
3244         first = false;
3245         if ((!list_all_targets) &&
3246             (report_dependencies_level == 0) &&
3247             (dmake_mode_type != serial_mode)) {
3248                 warning(catgets(catd, 1, 313, "can not get a TeamWare license, d
3249         }
3250 }
3251 #endif


3254 static void
3255 report_dir_enter_leave(Boolean entering)
3256 {
3257         char    rcwd[MAXPATHLEN];
3258 static  char *  mlev = NULL;
3259         char *  make_level_str = NULL;
3260         int     make_level_val = 0;

3262         make_level_str = getenv(NOCATGETS("MAKELEVEL"));
3263         if(make_level_str) {
3264                 make_level_val = atoi(make_level_str);
3265         }
3266         if(mlev == NULL) {
3267                 mlev = (char*) malloc(MAXPATHLEN);
3268         }
3269         if(entering) {
3270                 sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val + 1);
3271         } else {
3272                 make_level_val--;
3273                 sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val);
3274         }
3275         putenv(mlev);

3277         if(report_cwd) {
3278                 if(make_level_val <= 0) {
```

```
3279                        if(entering) {
3280                                sprintf( rcwd
3281                                       , catgets(catd, 1, 329, "dmake: Entering
3282                                       , get_current_path());
3283                        } else {
3284                                sprintf( rcwd
3285                                       , catgets(catd, 1, 331, "dmake: Leaving d
3286                                       , get_current_path());
3287                        }
3288                } else {
3289                        if(entering) {
3290                                sprintf( rcwd
3291                                       , catgets(catd, 1, 333, "dmake[%d]: Enter
3292                                       , make_level_val, get_current_path());
3293                        } else {
3294                                sprintf( rcwd
3295                                       , catgets(catd, 1, 335, "dmake[%d]: Leavi
3296                                       , make_level_val, get_current_path());
3297                        }
3298                }
3299                printf(NOCATGETS("%s"), rcwd);
3300        }
3301 }
```