

new/usr/src/cmd/make/bin/doname.cc

```
*****
104107 Wed May 20 11:42:01 2015
new/usr/src/cmd/make/bin/doname.cc
make: unifdef for PARALLEL (undefined, this relates to the old pmake)
*****
1 /*
2 * CDDL HEADER START
3 *
4 * The contents of this file are subject to the terms of the
5 * Common Development and Distribution License (the "License").
6 * You may not use this file except in compliance with the License.
7 *
8 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

25 /*
26 *      donne.c
27 *
28 *      Figure out which targets are out of date and rebuild them
29 */
30 */

31 /*
32 * Included files
33 */
34 /*
35 #include <avo/avo_alloca.h>           /* alloca() */
36 #if defined(TEAMWARE_MAKE_CMN)
37 #include <avo/util.h>                  /* avo_get_user(), avo_hostname() */
38 #endif

39 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
40     include <avo/strings.h> /* AVO_STRDUP() */
41     include <dm/Avo_MToolJobResultMsg.h>
42     include <dm/Avo_MToolJobStartMsg.h>
43     include <dm/Avo_MToolRsrcInfoMsg.h>
44     include <dm/Avo_macro_defs.h> /* AVO_BLOCK_INTERRUPTS & AVO_UNBLOCK_INTER
45     include <dmthread/Avo_ServerState.h>
46     include <rw/pstream.h>
47     include <rw/xdrstrea.h>
48 */

49 #endif

50 #include <fcntl.h>
51 #include <mk/defs.h>
52 #include <mksh/i18n.h>                  /* get_char_semantics_value() */
53 #include <mksh/macro.h>                 /* getvar(), expand_value() */
54 #include <mksh/misc.h>                  /* getmem() */
55 #include <poll.h>

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

1

new/usr/src/cmd/make/bin/doname.cc

```
59 #include <signal.h>
60 #
61 #      include <stropts.h>
62 #
63 #include <sys/errno.h>
64 #include <sys/stat.h>
65 #include <sys/types.h>
66 #include <sys/utsname.h>          /* uname() */
67 #include <sys/wait.h>
68 #include <unistd.h>              /* close() */

69 /*
70 * Defined macros
71 */
72 #
73 #ifndef PARALLEL
74 #define LOCALHOST "localhost"
75#endif

76#define MAXRULES 100

77#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
78#define SEND_MTOOL_MSG(cmds) \
79    if (send_mtool_msgs) { \
80        cmds \
81    }
82#else
83#define SEND_MTOOL_MSG(cmds)
84#endif

85// Sleep for .1 seconds between stat()'s
86const int      STAT_RETRY_SLEEP_TIME = 100000;

87/*
88 * typedefs & structs
89 */
90

91/*
92 * Static variables
93 */
94

95static char    hostName[MAXNAMELEN] = "";
96static char    userName[MAXNAMELEN] = "";

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

102static int    second_pass = 0;

103/*
104 * File table of contents
105 */
106extern Doname    donne_check(register Name target, register Boolean do_g
107                                donne(register Name target, register Boolean do_get, re
108                                check_dependencies(Doname *result, Property line, Boolea
109                                dynamic_dependencies(Name target);
110static Doname    run_command(register Property line, Boolean print_machin
111extern Doname    execute_serial(Property line);
112static Boolean   vpath_translation(register Name cmd);
113extern void      check_state(Name temp_file_name);
114static void      read_dependency_file(register Name filename);
115extern void      check_read_state_file(void);
116static void      do_assign(register Name line, register Name target);
117static void      build_command_strings(Name target, register Property lin
118static void      touch_command(register Property line, register Name targ
```

2

```

123 extern void update_target(Property line, Doname result);
124 static Doname sccs_get(register Name target, register Property *command
125 extern void read_directory_of_file(register Name file);
126 static void add_pattern_conditionals(register Name target);
127 extern void set_locals(register Name target, register Property old_l
128 extern void reset_locals(register Name target, register Property old
129 extern Boolean check_auto_dependencies(Name target, int auto_count, Nam
130 static void delete_query_chain(Chain ch);

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

136 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
137     static void append_job_result_msg(Avo_MToolJobResultMsg *job
138     static int pollResults(char *outFn, char *errFn, char *host
139     static void pollResultsAction(char *outFn, char *errFn);
140     static void rxmGetNextResultsBlock(int fd);
141     static int us_sleep(unsigned int nusecs);
142     extern "C" void Avo_PollResultsAction_Sigusr1Handler(int foo);
143 #endiff

145 /*
146 * DONE.
147 *
148 * doname_check(target, do_get, implicit, automatic)
149 *
150 * Will call doname() and then inspect the return value
151 *
152 * Return value:
153 *               Indication if the build failed or not
154 *
155 * Parameters:
156 *   target      The target to build
157 *   do_get      Passed thru to doname()
158 *   implicit    Passed thru to doname()
159 *   automatic   Are we building a hidden dependency?
160 *
161 * Global variables used:
162 *   build_failed_seen Set if -k is on and error occurs
163 *   continue_after_error Indicates that -k is on
164 *   report_dependencies No error msg if -P is on
165 */
166 Doname
167 doname_check(register Name target, register Boolean do_get, register Boolean imp
168 {
169     int first_time = 1;
170     (void) fflush(stdout);
171 try_again:
172     switch (doname(target, do_get, implicit, automatic)) {
173     case build_ok:
174         second_pass = 0;
175         return build_ok;
176     case build_running:
177         second_pass = 0;
178         return build_running;
179     case build_failed:
180         if (!continue_after_error) {
181             fatal(catgets(catd, 1, 13, "Target '%s' not remade because
182                             target->string_mb));
183         }
184         build_failed_seen = true;
185         second_pass = 0;
186         return build_failed;
187     case build_dont_know:
188         /*

```

```

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

unchanged_portion_omitted

293 /*
294 * DONE.
295 *
296 *   doname(target, do_get, implicit)
297 *
298 * Chases all files the target depends on and builds any that
299 * are out of date. If the target is out of date it is then rebuilt.
300 *
301 * Return value:
302 *               Indiates if build failed or nt
303 *
304 * Parameters:
305 *   target      Target to build
306 *   do_get      Run sccs get is nessecary
307 *   implicit    doname is trying to find an implicit rule
308 *
309 * Global variables used:
310 *   assign_done  True if command line assgnment has happened
311 *   commands_done Preserved for the case that we need local value
312 *   debug_level Should we trace make's actions?
313 *   default_rule The rule for ".DEFAULT", used as last resort
314 *   empty_name   The Name "", used when looking for single sfxf
315 *   keep_state   Indicates that .KEEP_STATE is on
316 *   parallel    True if building in parallel
317 *   recursion_level Used for tracing
318 *   report_dependencies make -P is on
319 */
320 Doname
321 doname(register Name target, register Boolean do_get, register Boolean implicit,
322 {
323     Doname result = build_dont_know;
324     Chain out_of_date_list = NULL;
325 #ifdef TEAMWARE_MAKE_CMN
326     Chain target_group;

```

```

327 #endiff
328     Property          old_locals = NULL;
329     register Property line;
330     Property          command = NULL;
331     register Dependency dependency;
332     Name              less = NULL;
333     Name              true_target = target;
334     Name              *automatics = NULL;
335     register int      auto_count;
336     Boolean           rechecking_target = false;
337     Boolean           saved_commands_done;
338     Boolean           restart = false;
339     Boolean           save_parallel = parallel;
340     Boolean           doing_subtree = false;
341
342     Boolean           recheck_conditionals = false;
343
344     if (target->state == build_running) {
345         return build_running;
346     }
347     line = get_prop(target->prop, line_prop);
348 #ifdef TEAMWARE_MAKE_CMN
349     if (line != NULL) {
350         /*
351          * If this target is a member of target group and one of the
352          * other members of the group is running, mark this target
353          * as running.
354         */
355         for (target_group = line->body.line.target_group;
356              target_group != NULL;
357              target_group = target_group->next) {
358             if (is_running(target_group->name)) {
359                 target->state = build_running;
360                 add_pending(target,
361                             recursion_level,
362                             do_get,
363                             implicit,
364                             false);
365             }
366         }
367     }
368 }
369 #endiff
370 /*
371  * If the target is a constructed one for a ":::" target,
372  * we need to consider that.
373 */
374 if (target->has_target_prop) {
375     true_target = get_prop(target->prop,
376                           target_prop)->body.target.target;
377     if (true_target->colon_splits > 0) {
378         /* Make sure we have a valid time for :: targets */
379         Property          time;
380
381         time = get_prop(true_target->prop, time_prop);
382         if (time != NULL) {
383             true_target->stat.time = time->body.time.time;
384         }
385     }
386 }
387 (void) exists(true_target);
388 /*
389  * If the target has been processed, we don't need to do it again,
390  * unless it depends on conditional macros or a delayed assignment,
391  * or it has been done when KEEP_STATE is on.
392 */

```

```

393     if (target->state == build_ok) {
394         if ((!keep_state) || (!target->depends_on_conditional && !assign_d
395             )
396             return build_ok;
397         } else {
398             recheck_conditionals = true;
399         }
400     }
401     if (target->state == build_subtree) {
402         /* A dynamic macro subtree is being built */
403         target->state = build_dont_know;
404         doing_subtree = true;
405         if (!target->checking_subtree) {
406             /*
407              * This target has been started before and therefore
408              * not all dependencies have to be built.
409             */
410             restart = true;
411         }
412     } else if (target->state == build_pending) {
413         target->state = build_dont_know;
414         restart = true;
415     } /*ifdef TEAMWARE_MAKE_CMN
416     */ else if (parallel &&
417                 keep_state &&
418                 (target->conditional_cnt > 0)) {
419         if (!parallel_ok(target, false)) {
420             add_subtree(target, recursion_level, do_get, implicit);
421             target->state = build_running;
422             return build_running;
423         }
424     } /*endif
425     */
426     /*
427      * If KEEP_STATE is on, we have to rebuild the target if the
428      * building of it caused new automatic dependencies to be reported.
429      * This is where we restart the build.
430      */
431     if (line != NULL) {
432         line->body.line.percent = NULL;
433     }
434     recheck_target:
435     /* Init all local variables */
436     result = build_dont_know;
437     out_of_date_list = NULL;
438     command = NULL;
439     less = NULL;
440     auto_count = 0;
441     auto_count != 0;
442     if (!restart && line != NULL) {
443         /*
444          * If this target has never been built before, mark all
445          * of the dependencies as never built.
446          */
447         for (dependency = line->body.line.dependencies;
448              dependency != NULL;
449              dependency = dependency->next) {
450                 dependency->built = false;
451             }
452         }
453     /* Save the set of automatic depes defined for this target */
454     if (keep_state &&
455         (line != NULL) &&
456         (line->body.line.dependencies != NULL)) {
457         Name *p;

```

```

459      /*
460       * First run thru the dependency list to see how many
461       * autos there are.
462       */
463     for (dependency = line->body.line.dependencies;
464         dependency != NULL;
465         dependency = dependency->next) {
466           if (dependency->automatic && !dependency->stale) {
467             auto_count++;
468           }
469         }
470     /* Create vector to hold the current autos */
471     automatics =
472       (Name *) alloca((int) (auto_count * sizeof (Name)));
473     /* Copy them */
474     for (p = automatics, dependency = line->body.line.dependencies;
475         dependency != NULL;
476         dependency = dependency->next) {
477           if (dependency->automatic && !dependency->stale) {
478             *p++ = dependency->name;
479           }
480         }
481     if (debug_level > 1) {
482       (void) printf(NOCATGETS("%*sdoname(%s)\n"),
483                     recursion_level,
484                     "",
485                     target->string_mb);
486     }
487     recursion_level++;
488     /* Avoid infinite loops */
489     if (target->state == build_in_progress) {
490       warning(catgets(catd, 1, 16, "Infinite loop: Target '%s' depends
491                     target->string_mb);
492       return build_ok;
493     }
494     target->state = build_in_progress;
495
496     /* Activate conditional macros for the target */
497     if (!target->added_pattern_conditional) {
498       add_pattern_conditional(target);
499       target->added_pattern_conditional = true;
500     }
501     if (target->conditional_cnt > 0) {
502       old_locals = (Property) alloca(target->conditional_cnt *
503                               sizeof (Property_rec));
504       set_locals(target, old_locals);
505     }
506   }
507
508 /**
509  * after making the call to dynamic_dependencies unconditional we can handle
510  * target names that are same as file name. In this case $$@ in the
511  * dependencies did not mean anything. With this change it expands it
512  * as expected.
513 */
514   if (!target->has_depe_list_expanded)
515   {
516     dynamic_dependencies(target);
517   }
518
519 /**
520  * FIRST SECTION -- GO THROUGH DEPENDENCIES AND COLLECT EXPLICIT
521  * COMMANDS TO RUN
522 */
523   if ((line = get_prop(target->prop, line_prop)) != NULL) {
524     if (check_dependencies(&result,

```

```

525   line,
526   do_get,
527   target,
528   true_target,
529   doing_subtree,
530   &out_of_date_list,
531   old_locals,
532   implicit,
533   &command,
534   less,
535   rechecking_target,
536   recheck_conditional);
537   return build_running;
538 }
539 if (line->body.line.query != NULL) {
540   delete_query_chain(line->body.line.query);
541 }
542 line->body.line.query = out_of_date_list;
543 }
544
545 #ifdef PARALLEL
546   if (doing_subtree) {
547     parallel = false;
548   }
549 #endif
550
551 /*
552  * If the target is a :: type, do not try to find the rule for the target,
553  * all actions will be taken by separate branches.
554  * Else, we try to find an implicit rule using various methods,
555  * we quit as soon as one is found.
556  *
557  * [tolik, 12 Sep 2002] Do not try to find implicit rule for the target
558  * being rechecked - the target is being rechecked means that it already
559  * has explicit dependencies derived from an implicit rule found
560  * in previous step.
561 */
562 if (target->colon_splits == 0 && !rechecking_target) {
563   /* Look for percent matched rule */
564   if ((result == build_dont_know) &&
565       (command == NULL)) {
566     switch (find_percent_rule(
567       target,
568       &command,
569       recheck_conditional)) {
570       case build_failed:
571         result = build_failed;
572         break;
573     }
574   #ifdef TEAMWARE_MAKE_CMN
575   case build_running:
576     target->state = build_running;
577     add_pending(target,
578                 --recursion_level,
579                 do_get,
580                 implicit,
581                 false);
582     if (target->conditional_cnt > 0) {
583       reset_locals(target,
584                   old_locals,
585                   get_prop(target->prop,
586                           conditional_prop),
587                   0);
588     }
589   return build_running;
590 }
591
592 case build_ok:

```

```

586             result = build_ok;
587             break;
588         }
589     } /* Look for double suffix rule */
590     if (result == build_dont_know) {
591         Property member;
592
593         if (target->is_member &&
594             ((member = get_prop(target->prop, member_prop)) != NULL)) {
595             switch (find_ar_suffix_rule(target,
596                                         member->body,
597                                         member.member,
598                                         &command,
599                                         recheck_conditionals)) {
600                 case build_failed:
601                     result = build_failed;
602                     break;
603
604                 case build_running:
605                     target->state = build_running;
606                     add_pending(target,
607                                 --recursion_level,
608                                 do_get,
609                                 implicit,
610                                 false);
611                     if (target->conditional_cnt > 0) {
612                         reset_locals(target,
613                                     old_locals,
614                                     get_prop(target->prop,
615                                               conditional_prop),
616                                     0);
617                     }
618                     return build_running;
619
620                 default:
621                     /* ALWAYS bind $% for old style */
622                     /* ar rules */
623                     if (line == NULL) {
624                         line =
625                             maybe_append_prop(target,
626                                             line_prop);
627
628                         line->body.line.percent =
629                             member->body.member.member;
630                         break;
631
632                     } else {
633                         switch (find_double_suffix_rule(target,
634                                         &command,
635                                         recheck_conditionals)) {
636                             case build_failed:
637                                 result = build_failed;
638                                 break;
639
640                             case build_running:
641                                 target->state = build_running;
642                                 add_pending(target,
643                                             --recursion_level,
644                                             do_get,
645                                             implicit,
646                                             false);
647                                 if (target->conditional_cnt > 0) {
648                                     reset_locals(target,
649                                                 old_locals,
650
651 #ifdef TEAMWARE_MAKE_CMN
652
653             result = build_ok;
654             break;
655         }
656     } /* Look for double suffix rule */
657
658     #endif
659
660     /* Look for single suffix rule */
661
662     /* /tolik/
663     * I commented !implicit to fix bug 1247448: Suffix Rules failed when combine wi
664     * This caused problem with SVR4 tilde rules (infinite recursion). So I made som
665     */
666     /* /tolik, 06.21.96/
667     * Regression! See BugId 1255360
668     * If more than one percent rules are defined for the same target then
669     * the behaviour of 'make' with my previous fix may be different from one
670     * of the 'old make'.
671     * The global variable second_pass (maybe it should be an argument to doname())
672     * is intended to avoid this regression. It is set in doname_check().
673     * First, 'make' will work as it worked before. Only when it is
674     * going to say "don't know how to make target" it sets second_pass to true and
675     * run 'doname' again but now trying to use Single Suffix Rules.
676     */
677
678     if ((result == build_dont_know) && !automatic && (!implicit || s
679         ((line == NULL) ||
680         ((line->body.line.target != NULL) &&
681           !line->body.line.target->has_regular_dependency))) {
682         switch (find_suffix_rule(target,
683                                 target,
684                                 empty_name,
685                                 &command,
686                                 recheck_conditionals)) {
687             case build_failed:
688                 result = build_failed;
689                 break;
690
691 #ifdef TEAMWARE_MAKE_CMN
692             case build_running:
693                 target->state = build_running;
694                 add_pending(target,
695                             --recursion_level,
696                             do_get,
697                             implicit,
698                             false);
699                 if (target->conditional_cnt > 0) {
700                     reset_locals(target,
701                                 old_locals,
702                                 get_prop(target->prop,
703                                           conditional_prop),
704                                 0);
705                 }
706                 return build_running;
707             #endif
708         }
709     }
710
711     /* Try to sccs get */
712     if ((command == NULL) &&
713         (result == build_dont_know) &&
714         do_get) {
715         result = sccs_get(target, &command);
716     }
717
718     /* Use .DEFAULT rule if it is defined. */
719
720     /* If we have a command, then we can't have an implicit rule */
721     if (command != NULL) {
722         if (implicit) {
723             result = build_failed;
724         }
725     }
726
727     /* If we have an implicit rule, then we can't have a command */
728     if (implicit) {
729         if (command != NULL) {
730             result = build_failed;
731         }
732     }
733
734     /* If we have a command, then we can't have an implicit rule */
735     if (command != NULL) {
736         if (implicit) {
737             result = build_failed;
738         }
739     }
740
741     /* If we have an implicit rule, then we can't have a command */
742     if (implicit) {
743         if (command != NULL) {
744             result = build_failed;
745         }
746     }
747
748     /* If we have a command, then we can't have an implicit rule */
749     if (command != NULL) {
750         if (implicit) {
751             result = build_failed;
752         }
753     }
754
755     /* If we have an implicit rule, then we can't have a command */
756     if (implicit) {
757         if (command != NULL) {
758             result = build_failed;
759         }
760     }
761
762     /* If we have a command, then we can't have an implicit rule */
763     if (command != NULL) {
764         if (implicit) {
765             result = build_failed;
766         }
767     }
768
769     /* If we have an implicit rule, then we can't have a command */
770     if (implicit) {
771         if (command != NULL) {
772             result = build_failed;
773         }
774     }
775
776     /* If we have a command, then we can't have an implicit rule */
777     if (command != NULL) {
778         if (implicit) {
779             result = build_failed;
780         }
781     }
782
783     /* If we have an implicit rule, then we can't have a command */
784     if (implicit) {
785         if (command != NULL) {
786             result = build_failed;
787         }
788     }
789
790     /* If we have a command, then we can't have an implicit rule */
791     if (command != NULL) {
792         if (implicit) {
793             result = build_failed;
794         }
795     }
796
797     /* If we have an implicit rule, then we can't have a command */
798     if (implicit) {
799         if (command != NULL) {
800             result = build_failed;
801         }
802     }
803
804     /* If we have a command, then we can't have an implicit rule */
805     if (command != NULL) {
806         if (implicit) {
807             result = build_failed;
808         }
809     }
810
811     /* If we have an implicit rule, then we can't have a command */
812     if (implicit) {
813         if (command != NULL) {
814             result = build_failed;
815         }
816     }
817
818     /* If we have a command, then we can't have an implicit rule */
819     if (command != NULL) {
820         if (implicit) {
821             result = build_failed;
822         }
823     }
824
825     /* If we have an implicit rule, then we can't have a command */
826     if (implicit) {
827         if (command != NULL) {
828             result = build_failed;
829         }
830     }
831
832     /* If we have a command, then we can't have an implicit rule */
833     if (command != NULL) {
834         if (implicit) {
835             result = build_failed;
836         }
837     }
838
839     /* If we have an implicit rule, then we can't have a command */
840     if (implicit) {
841         if (command != NULL) {
842             result = build_failed;
843         }
844     }
845
846     /* If we have a command, then we can't have an implicit rule */
847     if (command != NULL) {
848         if (implicit) {
849             result = build_failed;
850         }
851     }
852
853     /* If we have an implicit rule, then we can't have a command */
854     if (implicit) {
855         if (command != NULL) {
856             result = build_failed;
857         }
858     }
859
860     /* If we have a command, then we can't have an implicit rule */
861     if (command != NULL) {
862         if (implicit) {
863             result = build_failed;
864         }
865     }
866
867     /* If we have an implicit rule, then we can't have a command */
868     if (implicit) {
869         if (command != NULL) {
870             result = build_failed;
871         }
872     }
873
874     /* If we have a command, then we can't have an implicit rule */
875     if (command != NULL) {
876         if (implicit) {
877             result = build_failed;
878         }
879     }
880
881     /* If we have an implicit rule, then we can't have a command */
882     if (implicit) {
883         if (command != NULL) {
884             result = build_failed;
885         }
886     }
887
888     /* If we have a command, then we can't have an implicit rule */
889     if (command != NULL) {
890         if (implicit) {
891             result = build_failed;
892         }
893     }
894
895     /* If we have an implicit rule, then we can't have a command */
896     if (implicit) {
897         if (command != NULL) {
898             result = build_failed;
899         }
900     }
901
902     /* If we have a command, then we can't have an implicit rule */
903     if (command != NULL) {
904         if (implicit) {
905             result = build_failed;
906         }
907     }
908
909     /* If we have an implicit rule, then we can't have a command */
910     if (implicit) {
911         if (command != NULL) {
912             result = build_failed;
913         }
914     }
915
916     /* If we have a command, then we can't have an implicit rule */
917     if (command != NULL) {
918         if (implicit) {
919             result = build_failed;
920         }
921     }
922
923     /* If we have an implicit rule, then we can't have a command */
924     if (implicit) {
925         if (command != NULL) {
926             result = build_failed;
927         }
928     }
929
930     /* If we have a command, then we can't have an implicit rule */
931     if (command != NULL) {
932         if (implicit) {
933             result = build_failed;
934         }
935     }
936
937     /* If we have an implicit rule, then we can't have a command */
938     if (implicit) {
939         if (command != NULL) {
940             result = build_failed;
941         }
942     }
943
944     /* If we have a command, then we can't have an implicit rule */
945     if (command != NULL) {
946         if (implicit) {
947             result = build_failed;
948         }
949     }
950
951     /* If we have an implicit rule, then we can't have a command */
952     if (implicit) {
953         if (command != NULL) {
954             result = build_failed;
955         }
956     }
957
958     /* If we have a command, then we can't have an implicit rule */
959     if (command != NULL) {
960         if (implicit) {
961             result = build_failed;
962         }
963     }
964
965     /* If we have an implicit rule, then we can't have a command */
966     if (implicit) {
967         if (command != NULL) {
968             result = build_failed;
969         }
970     }
971
972     /* If we have a command, then we can't have an implicit rule */
973     if (command != NULL) {
974         if (implicit) {
975             result = build_failed;
976         }
977     }
978
979     /* If we have an implicit rule, then we can't have a command */
980     if (implicit) {
981         if (command != NULL) {
982             result = build_failed;
983         }
984     }
985
986     /* If we have a command, then we can't have an implicit rule */
987     if (command != NULL) {
988         if (implicit) {
989             result = build_failed;
990         }
991     }
992
993     /* If we have an implicit rule, then we can't have a command */
994     if (implicit) {
995         if (command != NULL) {
996             result = build_failed;
997         }
998     }
999
1000    /* If we have a command, then we can't have an implicit rule */
1001    if (command != NULL) {
1002        if (implicit) {
1003            result = build_failed;
1004        }
1005    }
1006
1007    /* If we have an implicit rule, then we can't have a command */
1008    if (implicit) {
1009        if (command != NULL) {
1010            result = build_failed;
1011        }
1012    }
1013
1014    /* If we have a command, then we can't have an implicit rule */
1015    if (command != NULL) {
1016        if (implicit) {
1017            result = build_failed;
1018        }
1019    }
1020
1021    /* If we have an implicit rule, then we can't have a command */
1022    if (implicit) {
1023        if (command != NULL) {
1024            result = build_failed;
1025        }
1026    }
1027
1028    /* If we have a command, then we can't have an implicit rule */
1029    if (command != NULL) {
1030        if (implicit) {
1031            result = build_failed;
1032        }
1033    }
1034
1035    /* If we have an implicit rule, then we can't have a command */
1036    if (implicit) {
1037        if (command != NULL) {
1038            result = build_failed;
1039        }
1040    }
1041
1042    /* If we have a command, then we can't have an implicit rule */
1043    if (command != NULL) {
1044        if (implicit) {
1045            result = build_failed;
1046        }
1047    }
1048
1049    /* If we have an implicit rule, then we can't have a command */
1050    if (implicit) {
1051        if (command != NULL) {
1052            result = build_failed;
1053        }
1054    }
1055
1056    /* If we have a command, then we can't have an implicit rule */
1057    if (command != NULL) {
1058        if (implicit) {
1059            result = build_failed;
1060        }
1061    }
1062
1063    /* If we have an implicit rule, then we can't have a command */
1064    if (implicit) {
1065        if (command != NULL) {
1066            result = build_failed;
1067        }
1068    }
1069
1070    /* If we have a command, then we can't have an implicit rule */
1071    if (command != NULL) {
1072        if (implicit) {
1073            result = build_failed;
1074        }
1075    }
1076
1077    /* If we have an implicit rule, then we can't have a command */
1078    if (implicit) {
1079        if (command != NULL) {
1080            result = build_failed;
1081        }
1082    }
1083
1084    /* If we have a command, then we can't have an implicit rule */
1085    if (command != NULL) {
1086        if (implicit) {
1087            result = build_failed;
1088        }
1089    }
1090
1091    /* If we have an implicit rule, then we can't have a command */
1092    if (implicit) {
1093        if (command != NULL) {
1094            result = build_failed;
1095        }
1096    }
1097
1098    /* If we have a command, then we can't have an implicit rule */
1099    if (command != NULL) {
1100        if (implicit) {
1101            result = build_failed;
1102        }
1103    }
1104
1105    /* If we have an implicit rule, then we can't have a command */
1106    if (implicit) {
1107        if (command != NULL) {
1108            result = build_failed;
1109        }
1110    }
1111
1112    /* If we have a command, then we can't have an implicit rule */
1113    if (command != NULL) {
1114        if (implicit) {
1115            result = build_failed;
1116        }
1117    }
1118
1119    /* If we have an implicit rule, then we can't have a command */
1120    if (implicit) {
1121        if (command != NULL) {
1122            result = build_failed;
1123        }
1124    }
1125
1126    /* If we have a command, then we can't have an implicit rule */
1127    if (command != NULL) {
1128        if (implicit) {
1129            result = build_failed;
1130        }
1131    }
1132
1133    /* If we have an implicit rule, then we can't have a command */
1134    if (implicit) {
1135        if (command != NULL) {
1136            result = build_failed;
1137        }
1138    }
1139
1140    /* If we have a command, then we can't have an implicit rule */
1141    if (command != NULL) {
1142        if (implicit) {
1143            result = build_failed;
1144        }
1145    }
1146
1147    /* If we have an implicit rule, then we can't have a command */
1148    if (implicit) {
1149        if (command != NULL) {
1150            result = build_failed;
1151        }
1152    }
1153
1154    /* If we have a command, then we can't have an implicit rule */
1155    if (command != NULL) {
1156        if (implicit) {
1157            result = build_failed;
1158        }
1159    }
1160
1161    /* If we have an implicit rule, then we can't have a command */
1162    if (implicit) {
1163        if (command != NULL) {
1164            result = build_failed;
1165        }
1166    }
1167
1168    /* If we have a command, then we can't have an implicit rule */
1169    if (command != NULL) {
1170        if (implicit) {
1171            result = build_failed;
1172        }
1173    }
1174
1175    /* If we have an implicit rule, then we can't have a command */
1176    if (implicit) {
1177        if (command != NULL) {
1178            result = build_failed;
1179        }
1180    }
1181
1182    /* If we have a command, then we can't have an implicit rule */
1183    if (command != NULL) {
1184        if (implicit) {
1185            result = build_failed;
1186        }
1187    }
1188
1189    /* If we have an implicit rule, then we can't have a command */
1190    if (implicit) {
1191        if (command != NULL) {
1192            result = build_failed;
1193        }
1194    }
1195
1196    /* If we have a command, then we can't have an implicit rule */
1197    if (command != NULL) {
1198        if (implicit) {
1199            result = build_failed;
1200        }
1201    }
1202
1203    /* If we have an implicit rule, then we can't have a command */
1204    if (implicit) {
1205        if (command != NULL) {
1206            result = build_failed;
1207        }
1208    }
1209
1210    /* If we have a command, then we can't have an implicit rule */
1211    if (command != NULL) {
1212        if (implicit) {
1213            result = build_failed;
1214        }
1215    }
1216
1217    /* If we have an implicit rule, then we can't have a command */
1218    if (implicit) {
1219        if (command != NULL) {
1220            result = build_failed;
1221        }
1222    }
1223
1224    /* If we have a command, then we can't have an implicit rule */
1225    if (command != NULL) {
1226        if (implicit) {
1227            result = build_failed;
1228        }
1229    }
1230
1231    /* If we have an implicit rule, then we can't have a command */
1232    if (implicit) {
1233        if (command != NULL) {
1234            result = build_failed;
1235        }
1236    }
1237
1238    /* If we have a command, then we can't have an implicit rule */
1239    if (command != NULL) {
1240        if (implicit) {
1241            result = build_failed;
1242        }
1243    }
1244
1245    /* If we have an implicit rule, then we can't have a command */
1246    if (implicit) {
1247        if (command != NULL) {
1248            result = build_failed;
1249        }
1250    }
1251
1252    /* If we have a command, then we can't have an implicit rule */
1253    if (command != NULL) {
1254        if (implicit) {
1255            result = build_failed;
1256        }
1257    }
1258
1259    /* If we have an implicit rule, then we can't have a command */
1260    if (implicit) {
1261        if (command != NULL) {
1262            result = build_failed;
1263        }
1264    }
1265
1266    /* If we have a command, then we can't have an implicit rule */
1267    if (command != NULL) {
1268        if (implicit) {
1269            result = build_failed;
1270        }
1271    }
1272
1273    /* If we have an implicit rule, then we can't have a command */
1274    if (implicit) {
1275        if (command != NULL) {
1276            result = build_failed;
1277        }
1278    }
1279
1280    /* If we have a command, then we can't have an implicit rule */
1281    if (command != NULL) {
1282        if (implicit) {
1283            result = build_failed;
1284        }
1285    }
1286
1287    /* If we have an implicit rule, then we can't have a command */
1288    if (implicit) {
1289        if (command != NULL) {
1290            result = build_failed;
1291        }
1292    }
1293
1294    /* If we have a command, then we can't have an implicit rule */
1295    if (command != NULL) {
1296        if (implicit) {
1297            result = build_failed;
1298        }
1299    }
1300
1301    /* If we have an implicit rule, then we can't have a command */
1302    if (implicit) {
1303        if (command != NULL) {
1304            result = build_failed;
1305        }
1306    }
1307
1308    /* If we have a command, then we can't have an implicit rule */
1309    if (command != NULL) {
1310        if (implicit) {
1311            result = build_failed;
1312        }
1313    }
1314
1315    /* If we have an implicit rule, then we can't have a command */
1316    if (implicit) {
1317        if (command != NULL) {
1318            result = build_failed;
1319        }
1320    }
1321
1322    /* If we have a command, then we can't have an implicit rule */
1323    if (command != NULL) {
1324        if (implicit) {
1325            result = build_failed;
1326        }
1327    }
1328
1329    /* If we have an implicit rule, then we can't have a command */
1330    if (implicit) {
1331        if (command != NULL) {
1332            result = build_failed;
1333        }
1334    }
1335
1336    /* If we have a command, then we can't have an implicit rule */
1337    if (command != NULL) {
1338        if (implicit) {
1339            result = build_failed;
1340        }
1341    }
1342
1343    /* If we have an implicit rule, then we can't have a command */
1344    if (implicit) {
1345        if (command != NULL) {
1346            result = build_failed;
1347        }
1348    }
1349
1350    /* If we have a command, then we can't have an implicit rule */
1351    if (command != NULL) {
1352        if (implicit) {
1353            result = build_failed;
1354        }
1355    }
1356
1357    /* If we have an implicit rule, then we can't have a command */
1358    if (implicit) {
1359        if (command != NULL) {
1360            result = build_failed;
1361        }
1362    }
1363
1364    /* If we have a command, then we can't have an implicit rule */
1365    if (command != NULL) {
1366        if (implicit) {
1367            result = build_failed;
1368        }
1369    }
1370
1371    /* If we have an implicit rule, then we can't have a command */
1372    if (implicit) {
1373        if (command != NULL) {
1374            result = build_failed;
1375        }
1376    }
1377
1378    /* If we have a command, then we can't have an implicit rule */
1379    if (command != NULL) {
1380        if (implicit) {
1381            result = build_failed;
1382        }
1383    }
1384
1385    /* If we have an implicit rule, then we can't have a command */
1386    if (implicit) {
1387        if (command != NULL) {
1388            result = build_failed;
1389        }
1390    }
1391
1392    /* If we have a command, then we can't have an implicit rule */
1393    if (command != NULL) {
1394        if (implicit) {
1395            result = build_failed;
1396        }
1397    }
1398
1399    /* If we have an implicit rule, then we can't have a command */
1400    if (implicit) {
1401        if (command != NULL) {
1402            result = build_failed;
1403        }
1404    }
1405
1406    /* If we have a command, then we can't have an implicit rule */
1407    if (command != NULL) {
1408        if (implicit) {
1409            result = build_failed;
1410        }
1411    }
1412
1413    /* If we have an implicit rule, then we can't have a command */
1414    if (implicit) {
1415        if (command != NULL) {
1416            result = build_failed;
1417        }
1418    }
1419
1420    /* If we have a command, then we can't have an implicit rule */
1421    if (command != NULL) {
1422        if (implicit) {
1423            result = build_failed;
1424        }
1425    }
1426
1427    /* If we have an implicit rule, then we can't have a command */
1428    if (implicit) {
1429        if (command != NULL) {
1430            result = build_failed;
1431        }
1432    }
1433
1434    /* If we have a command, then we can't have an implicit rule */
1435    if (command != NULL) {
1436        if (implicit) {
1437            result = build_failed;
1438        }
1439    }
1440
1441    /* If we have an implicit rule, then we can't have a command */
1442    if (implicit) {
1443        if (command != NULL) {
1444            result = build_failed;
1445        }
1446    }
1447
1448    /* If we have a command, then we can't have an implicit rule */
1449    if (command != NULL) {
1450        if (implicit) {
1451            result = build_failed;
1452        }
1453    }
1454
1455    /* If we have an implicit rule, then we can't have a command */
1456    if (implicit) {
1457        if (command != NULL) {
1458            result = build_failed;
1459        }
1460    }
1461
1462    /* If we have a command, then we can't have an implicit rule */
1463    if (command != NULL) {
1464        if (implicit) {
1465            result = build_failed;
1466        }
1467    }
1468
1469    /* If we have an implicit rule, then we can't have a command */
1470    if (implicit) {
1471        if (command != NULL) {
1472            result = build_failed;
1473        }
1474    }
1475
1476    /* If we have a command, then we can't have an implicit rule */
1477    if (command != NULL) {
1478        if (implicit) {
1479            result = build_failed;
1480        }
1481    }
1482
1483    /* If we have an implicit rule, then we can't have a command */
1484    if (implicit) {
1485        if (command != NULL) {
1486            result = build_failed;
1487        }
1488    }
1489
1490    /* If we have a command, then we can't have an implicit rule */
1491    if (command != NULL) {
1492        if (implicit) {
1493            result = build_failed;
1494        }
1495    }
1496
1497    /* If we have an implicit rule, then we can't have a command */
1498    if (implicit) {
1499        if (command != NULL) {
1500            result = build_failed;
1501        }
1502    }
1503
1504    /* If we have a command, then we can't have an implicit rule */
1505    if (command != NULL) {
1506        if (implicit) {
1507            result = build_failed;
1508        }
1509    }
1510
1511    /* If we have an implicit rule, then we can't have a command */
1512    if (implicit) {
1513        if (command != NULL) {
1514            result = build_failed;
1515        }
1516    }
1517
1518    /* If we have a command, then we can't have an implicit rule */
1519    if (command != NULL) {
1520        if (implicit) {
1521            result = build_failed;
1522        }
1523    }
1524
1525    /* If we have an implicit rule, then we can't have a command */
1526    if (implicit) {
1527        if (command != NULL) {
1528            result = build_failed;
1529        }
1530    }
1531
1532    /* If we have a command, then we can't have an implicit rule */
1533    if (command != NULL) {
1534        if (implicit) {
1535            result = build_failed;
1536        }
1537    }
1538
1539    /* If we have an implicit rule, then we can't have a command */
1540    if (implicit) {
1541        if (command != NULL) {
1542            result = build_failed;
1543        }
1544    }
1545
1546    /* If we have a command, then we can't have an implicit rule */
1547    if (command != NULL) {
1548        if (implicit) {
1549            result = build_failed;
1550        }
1551    }
1552
1553    /* If we have an implicit rule, then we can't have a command */
1554    if (implicit) {
1555        if (command != NULL) {
1556            result = build_failed;
1557        }
1558    }
1559
1560    /* If we have a command, then we can't have an implicit rule */
1561    if (command != NULL) {
1562        if (implicit) {
1563            result = build_failed;
1564        }
1565    }
1566
1567    /* If we have an implicit rule, then we can't have a command */
1568    if (implicit) {
1569        if (command != NULL) {
1570            result = build_failed;
1571        }
1572    }
1573
1574    /* If we have a command, then we can't have an implicit rule */
1575    if (command != NULL) {
1576        if (implicit) {
1577            result = build_failed;
1578        }
1579    }
1580
1581    /* If we have an implicit rule, then we can't have a command */
1582    if (implicit) {
1583        if (command != NULL) {
1584            result = build_failed;
1585        }
1586    }
1587
1588    /* If we have a command, then we can't have an implicit rule */
1589    if (command != NULL) {
1590        if (implicit) {
1591            result = build_failed;
1592        }
1593    }
1594
1595    /* If we have an implicit rule, then we can't have a command */
1596    if (implicit) {
1597        if (command != NULL) {
1598            result = build_failed;
1599        }
1600    }
1601
1602    /* If we have a command, then we can't have an implicit rule */
1603    if (command != NULL) {
1604        if (implicit) {
1605            result = build_failed;
1606        }
1607    }
1608
1609    /* If we have an implicit rule, then we can't have a command */
1610    if (implicit) {
1611        if (command != NULL) {
1612            result = build_failed;
1613        }
1614    }
1615
1616    /* If we have a command, then we can't have an implicit rule */
1617    if (command != NULL) {
1618        if (implicit) {
1619            result = build_failed;
1620        }
1621    }
1622
1623    /* If we have an implicit rule, then we can't have a command */
1624    if (implicit) {
1625        if (command != NULL) {
1626            result = build_failed;
1627        }
1628    }
1629
1630    /* If we have a command, then we can't have an implicit rule */
1631    if (command != NULL) {
1632        if (implicit) {
1633            result = build_failed;
1634        }
1635    }
1636
1637    /* If we have an implicit rule, then we can't have a command */
1638    if (implicit) {
1639        if (command != NULL) {
1640            result = build_failed;
1641        }
1642    }
1643
1644    /* If we have a command, then we can't have an implicit rule */
1645    if (command != NULL) {
1646        if (implicit) {
1647            result = build_failed;
1648        }
1649    }
1650
1651    /* If we have an implicit rule, then we can't have a command */
1652    if (implicit) {
1653        if (command != NULL) {
1654            result = build_failed;
1655        }
1656    }
1657
1658    /* If we have a command, then we can't have an implicit rule */
1659    if (command != NULL) {
1660        if (implicit) {
1661            result = build_failed;
1662        }
1663    }
1664
1665    /* If we have an implicit rule, then we can't have a command */
1666    if (implicit) {
1667        if (command != NULL) {
1668            result = build_failed;
1669        }
1670    }
1671
1672    /* If we have a command, then we can't have an implicit rule */
1673    if (command != NULL) {
1674        if (implicit) {
1675            result = build_failed;
1676        }
1677    }
1678
1679    /* If we have an implicit rule, then we can't have a command */
1680    if (implicit) {
1681        if (command != NULL) {
1682            result = build_failed;
1683        }
1684    }
1685
1686    /* If we have a command, then we can't have an implicit rule */
1687    if (command != NULL) {
1688        if (implicit) {
1689            result = build_failed;
1690        }
1691    }
1692
1693    /* If we have an implicit rule, then we can't have a command */
1694    if (implicit) {
1695        if (command != NULL) {
1696            result = build_failed;
1697        }
1698    }
1699
1700    /* If we have a command, then we can't have an implicit rule */
1701    if (command != NULL) {
1702        if (implicit) {
1703            result = build_failed;
1704        }
1705    }
1706
1707    /* If we have an implicit rule, then we can't have a command */
1708    if (implicit) {
1709        if (command != NULL) {
1710            result = build_failed;
1711        }
1712    }
1713
1714    /* If we have a command, then we can't have an implicit rule */
1715    if (command != NULL) {
1716        if (implicit) {
1717            result = build_failed;
1718        }
1719    }
1720
1721    /* If we have an implicit rule, then we can't have a command */
1722    if (implicit) {
1723        if (command != NULL) {
1724            result = build_failed;
1725        }
1726    }
1727
1728    /* If we have a command, then we can't have an implicit rule */
1729    if (command != NULL) {
1730        if (implicit) {
1731            result = build_failed;
1732        }
1733    }
1734
1735    /* If we have an implicit rule, then we can't have a command */
1736    if (implicit) {
1737        if (command != NULL) {
1738            result = build_failed;
1739        }
1740    }
1741
1742    /* If we have a command, then we can't have an implicit rule */
1743
```

```

719     if ((command == NULL) &&
720         (result == build_dont_know) &&
721         (true_target->colons == no_colon) &&
722         default_rule &&
723         !implicit) {
724         /* Make sure we have a line prop */
725         line = maybe_append_prop(target, line_prop);
726         command = line;
727         Boolean out_of_date;
728         if (true_target->is_member) {
729             out_of_date = (Boolean) OUT_OF_DATE_SEC(true_target-
730                                         line->body.l
731         } else {
732             out_of_date = (Boolean) OUT_OF_DATE(true_target-
733                                         line->body.l
734         }
735         if (build_unconditional || out_of_date) {
736             line->body.line.is_out_of_date = true;
737             if (debug_level > 0) {
738                 (void) printf(catgets(catd, 1, 17, "%*sB
739                                         recursion_level,
740                                         "",
741                                         true_target->string_mb));
742             }
743             line->body.line.sccs_command = false;
744             line->body.line.command_template = default_rule;
745             line->body.line.target = true_target;
746             line->body.line.star = NULL;
747             line->body.line.less = true_target;
748             line->body.line.percent = NULL;
749         }
750     }
751
752     /* We say "target up to date" if no cmd were executed for the target */
753     if (!target->is_double_colon_parent) {
754         commands_done = false;
755     }
756
757     silent = silent_all;
758     ignore_errors = ignore_errors_all;
759     if (posix)
760     {
761         if (!silent)
762         {
763             silent = (Boolean) target->silent_mode;
764         }
765         if (!ignore_errors)
766         {
767             ignore_errors = (Boolean) target->ignore_error_mode;
768         }
769     }
770
771     int doname_dyntarget = 0;
772 r_command:
773     /* Run commands if any. */
774     if ((command != NULL) &&
775         (command->body.line.command_template != NULL)) {
776         if (result != build_failed) {
777             result = run_command(command,
778                                 (Boolean) ((parallel || save_parallel
779
780                         switch (result) {
781 #ifdef TEAMWARE_MAKE_CMN
782                         case build_running:
783                             add_running(target,

```

```

784     true_target,
785     command,
786     --recursion_level,
787     auto_count,
788     automatics,
789     do_get,
790     implicit);
791     target->state = build_running;
792     if ((line = get_prop(target->prop,
793                           line_prop)) != NULL) {
794         if (line->body.line.query != NULL) {
795             delete_query_chain(line->body.line.query
796         }
797         line->body.line.query = NULL;
798     }
799     if (target->conditional_cnt > 0) {
800         reset_locals(target,
801                     old_locals,
802                     get_prop(target->prop,
803                               conditional_prop),
804                     0);
805     }
806     return build_running;
807     case build_serial:
808         add_serial(target,
809                     --recursion_level,
810                     do_get,
811                     implicit);
812         target->state = build_running;
813         line = get_prop(target->prop, line_prop);
814         if (line != NULL) {
815             if (line->body.line.query != NULL) {
816                 delete_query_chain(line->body.line.query
817             }
818             line->body.line.query = NULL;
819         }
820         if (target->conditional_cnt > 0) {
821             reset_locals(target,
822                         old_locals,
823                         get_prop(target->prop,
824                                   conditional_prop),
825                         0);
826         }
827     }
828 #endif
829
830     case build_ok:
831         /* If all went OK set a nice timestamp */
832         if (true_target->stat.time == file_doesnt_exist) {
833             true_target->stat.time = file_max_time;
834         }
835         break;
836     } else {
837         /*
838          * If no command was found for the target, and it doesn't
839          * exist, and it is mentioned as a target in the makefile,
840          * we say it is extremely new and that it is OK.
841          */
842         if (target->colons != no_colon) {
843             if (true_target->stat.time == file_doesnt_exist){
844                 true_target->stat.time = file_max_time;
845             }
846             result = build_ok;
847         }
848     }
849     /* Trying dynamic targets.
```

```

850
851         */
852     if (!doname_dyntarget) {
853         donne_dyntarget = 1;
854         Name dtarg = find_dyntarget(target);
855         if (dtarg!=NULL) {
856             if (!target->has_depe_list_expanded) {
857                 dynamic_dependencies(target);
858             }
859             if ((line = get_prop(target->prop, line_prop)) != NULL) {
860                 if (check_dependencies(&result,
861                     line,
862                     do_get,
863                     target,
864                     true_target,
865                     doing_subtree,
866                     &out_of_date_list,
867                     old_locals,
868                     implicit,
869                     &command,
870                     less,
871                     rechecking_target,
872                     recheck_condition
873                 ) {
874                     return build_running;
875                 }
876                 if (line->body.line.query != NULL) {
877                     delete_query_chain(line->body.line.query);
878                 }
879                 line->body.line.query = out_of_date_list
880             }
881             goto r_command;
882         }
883     }
884     /* If the file exists, it is OK that we couldnt figure
885      * out how to build it.
886     */
887     (void) exists(target);
888     if ((target->stat.time != file_doesnt_exist) &&
889         (result == build_dont_know)) {
890         result = build_ok;
891     }
892 }

893 /*
894  * Some of the following is duplicated in the function finish_doname.
895  * If anything is changed here, check to see if it needs to be
896  * changed there.
897 */
898 if ((line = get_prop(target->prop, line_prop)) != NULL) {
899     if (line->body.line.query != NULL) {
900         delete_query_chain(line->body.line.query);
901     }
902     line->body.line.query = NULL;
903 }
904 target->state = result;
905 parallel = save_parallel;
906 if (target->conditional_cnt > 0) {
907     reset_locals(target,
908         old_locals,
909         get_prop(target->prop, conditional_prop),
910         0);
911 }
912 recursion_level--;
913 if (target->is_member) {
914     Property member;
915 }
```

```

917     /* Propagate the timestamp from the member file to the member*/
918     if ((target->stat.time != file_max_time) &&
919         ((member = get_prop(target->prop, member_prop)) != NULL) &&
920         (exists(member->body.member.member) > file_doesnt_exist)) {
921         target->stat.time =
922             member->body.member.member->stat.time;
923     }
924     /*
925      * Check if we found any new auto dependencies when we
926      * built the target.
927      */
928     if ((result == build_ok) && check_auto_dependencies(target,
929             auto_count,
930             automatics)) {
931         if (debug_level > 0) {
932             (void) printf(catgets(catd, 1, 18, "%*sTarget '%s' acqui
933             recursion_level,
934             "",
935             true_target->string_mb);
936         }
937         rechecking_target = true;
938         saved_commands_done = commands_done;
939         goto recheck_target;
940     }
941     if (rechecking_target && !commands_done) {
942         commands_done = saved_commands_done;
943     }
944 }
945
946 return result;
947 }
948 }  



---


unchanged_portion_omitted
```

new/usr/src/cmd/make/bin/main.cc

1

```
*****
98375 Wed May 20 11:42:02 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef for PARALLEL (undefined, this relates to the old pmake)
*****
unchanged_portion_omitted_
156 #endif

158 extern Name normalize_name(register wchar_t *name_string, register i
160 extern int main(int, char * []);

162 static void append_makeflags_string(Name, String);
163 static void doalarm(int);
164 static void enter_argv_values(int , char **, ASCII_Dyn_Array *);
165 static void make_targets(int, char **, Boolean);
166 static int parse_command_option(char);
167 static void read_command_options(int, char **);
168 static void read_environment(Boolean);
169 static void read_files_and_state(int, char **);
170 static Boolean read_makefile(Name, Boolean, Boolean, Boolean);
171 static void report_recursion(Name);
172 static void set_sgs_support(void);
173 static void setup_for_projectdir(void);
174 static void setup_makeflags_argv(void);
175 static void report_dir_enter_leave(Boolean entering);

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

179 #ifdef DISTRIBUTED
180     extern int dmake_ofd;
181     extern FILE* dmake_ofp;
182     extern int rxmPid;
183     extern XDR xdrs_out;
184 #endif
185 #ifdef TEAMWARE_MAKE_CMN
186     extern char verstring[];
187 #endif

189 jmp_buf jmpbuffer;
190 extern nl_catd catd;

192 /*
193 *      main(argc, argv)
194 *
195 *      Parameters:
196 *          argc          You know what this is
197 *          argv          You know what this is
198 *
199 *      Static variables used:
200 *          list_all_targets    make -T seen
201 *          trace_status        make -p seen
202 *
203 *      Global variables used:
204 *          debug_level       Should we trace make actions?
205 *          keep_state         Set if .KEEP_STATE seen
206 *          makeflags          The Name "MAKEFLAGS", used to get macro
207 *          remote_command_name Name of remote invocation cmd ("on")
208 *          running_list       List of parallel running processes
209 *          stdout_stderr_same true if stdout and stderr are the same
210 *          auto_dependencies   The Name "SUNPRO_DEPENDENCIES"
211 *          temp_file_directory Set to the dir where we create tmp file
212 *          trace_reader       Set to reflect tracing status
213 *          working_on_targets Set when building user targets
214 */
215 int
```

new/usr/src/cmd/make/bin/main.cc

2

```
216 main(int argc, char *argv[])
217 {
218     /*
219      * cp is a -> to the value of the MAKEFLAGS env var,
220      * which has to be regular chars.
221      */
222     register char *cp;
223     char make_state_dir[MAXPATHLEN];
224     Boolean parallel_flag = false;
225     char *programeptr;
226     char *slash_ptr;
227     mode_t um;
228     int i;
229 #ifdef TEAMWARE_MAKE_CMN
230     struct itimerval value;
231     char def_dmakerc_path[MAXPATHLEN];
232     Name dmake_name, dmake_name2;
233     Name dmake_value, dmake_value2;
234     Property prop, prop2;
235     struct stat statbuf;
236     int statval;
237 #endif
238
239 #ifndef PARALLEL
240     struct stat out_stat, err_stat;
241 #endif
242     hostid = gethostid();
243 #ifdef TEAMWARE_MAKE_CMN
244     avo_get_user(NULL, NULL); // Initialize user name
245 #endif
246     bsd_signals();
247
248     (void) setlocale(LC_ALL, "");

249 #ifdef DMAKE_STATISTICS
250     if (getenv(NOCATGETS("DMAKE_STATISTICS")) ) {
251         getname_stat = true;
252     }
253 #endif
254
255     /*
256      * avo_init() sets the umask to 0. Save it here and restore
257      * it after the avo_init() call.
258      */
259
260 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
261     um = umask(0);
262     avo_init(argv[0]);
263     umask(um);
264
265     cleanup = new Avo_cleanup(NOCATGETS("dmake"), argc, argv);
266 #endif
267
268 #if defined(TEAMWARE_MAKE_CMN)
269     catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
270     libcli_init();
271
272 #ifdef _CHECK_UPDATE_H
273     /* This is for dmake only (not for Solaris make).
274      * Check (in background) if there is an update (dmake patch)
275      * and inform user
276      */
277     {
278         Avo_err *err;
279         char *dir;
```

```

280         err = avo_find_run_dir(&dir);
281         if (AVO_OK == err) {
282             AU_check_update_service(NOCATGETS("Dmake"), dir);
283         }
284     }
285 #endif /* _CHECK_UPDATE_H */
286 #endif

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

291 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
292 /* I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
293 * from avo_util library.
294 */
295 libmksdmsi18n_init();
297 #endif

300 #ifndef TEAMWARE_MAKE_CMN
301     textdomain(NOCATGETS("SUNW_SPRO_MAKE"));
302 #endif /* TEAMWARE_MAKE_CMN */

304 #ifdef TEAMWARE_MAKE_CMN
305     g_argc = argc;
306     g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
307     for (i = 0; i < argc; i++) {
308         g_argv[i] = argv[i];
309     }
310     g_argv[i] = NULL;
311 #endif /* TEAMWARE_MAKE_CMN */

313 /*
314 * Set argv_zero_string to some form of argv[0] for
315 * recursive MAKE builds.
316 */
317
318     if (*argv[0] == (int) slash_char) {
319         /* argv[0] starts with a slash */
320         argv_zero_string = strdup(argv[0]);
321     } else if (strchr(argv[0], (int) slash_char) == NULL) {
322         /* argv[0] contains no slashes */
323         argv_zero_string = strdup(argv[0]);
324     } else {
325         /*
326          * argv[0] contains at least one slash,
327          * but doesn't start with a slash
328         */
329         char *tmp_current_path;
330         char *tmp_string;

332         tmp_current_path = get_current_path();
333         tmp_string = getmem(strlen(tmp_current_path) + 1 +
334                             strlen(argv[0]) + 1);
335         (void) sprintf(tmp_string,
336                         "%s/%s",
337                         tmp_current_path,
338                         argv[0]);
339         argv_zero_string = strdup(tmp_string);
340         retmem_mb(tmp_string);
341     }

343 /*
344 * The following flags are reset if we don't have the
345 * (.nse_depinfo or .make.state) files locked and only set

```

```

346         * AFTER the file has been locked. This ensures that if the user
347         * interrupts the program while file_lock() is waiting to lock
348         * the file, the interrupt handler doesn't remove a lock
349         * that doesn't belong to us.
350     */
351     make_state_lockfile = NULL;
352     make_state_locked = false;

355 /*
356 * look for last slash char in the path to look at the binary
357 * name. This is to resolve the hard link and invoke make
358 * in svr4 mode.
359 */

361 /* Sun OS make standard */
362 svr4 = false;
363 posix = false;
364 if (!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
365     svr4 = false;
366     posix = true;
367 } else {
368     programeptr = strrchr(argv[0], '/');
369     if (programeptr) {
370         programeptr++;
371     } else {
372         programeptr = argv[0];
373     }
374     if (!strcmp(programeptr, NOCATGETS("svr4.make"))) {
375         svr4 = true;
376         posix = false;
377     }
378 }
379 if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
380     svr4 = true;
381     posix = false;
382 }

384 /*
385 * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
386 */
387 char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"));
388 if (dmake_compat_mode_var != NULL) {
389     if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU")))
390         gnu_style = true;
391     //svr4 = false;
392     //posix = false;
393 }
394

396 /*
397 * Temporary directory set up.
398 */
399 char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
400 if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
401     strcpy(mbs_buffer, tmpdir_var);
402     for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
403          *tmpdir_var == '/' && tmpdir_var > mbs_buffer;
404          *tmpdir_var = '\0');
405     if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
406         sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
407                     mbs_buffer, getpid());
408         int fd = mkstemp(mbs_buffer2);
409         if (fd >= 0) {
410             close(fd);
411             unlink(mbs_buffer2);

```

```

412
413         }
414     }
415 }
416
417 #ifndef PARALLEL
418     /* find out if stdout and stderr point to the same place */
419     if (fstat(1, &out_stat) < 0) {
420         fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
421             "");
422     if (fstat(2, &err_stat) < 0) {
423         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
424             );
425     if ((out_stat.st_dev == err_stat.st_dev) &&
426         (out_stat.st_ino == err_stat.st_ino)) {
427         stdout_stderr_same = true;
428     } else {
429         stdout_stderr_same = false;
430     }
431 #else
432     stdout_stderr_same = false;
433 #endif
434
435 /* Make the vroot package scan the path using shell semantics */
436 set_path_style(0);
437
438 setup_char_semantics();
439
440 setup_for_projectdir();
441
442 /*
443 * If running with .KEEP_STATE, curdir will be set with
444 * the connected directory.
445 */
446 (void) atexit(cleanup_after_exit);
447
448 load_cached_names();
449
450 /*
451 * Set command line flags
452 */
453 setup_makeflags_argv();
454 read_command_options(mf_argc, mf_argv);
455 read_command_options(argc, argv);
456 if (debug_level > 0) {
457     cp = getenv(makeflags->string_mb);
458     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
459 }
460
461 setup_interrupt(handle_interrupt);
462
463 read_files_and_state(argc, argv);
464
465 #ifdef TEAMWARE_MAKE_CMN
466     /*
467      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
468      */
469     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
470     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
471     prop2 = get_prop(dmake_name2->prop, macro_prop);
472     if (prop2 == NULL) {
473         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
474         output_mode = txt1_mode;
475     } else {
476         dmake_value2 = prop2->body.macro.value;
477         if ((dmake_value2 == NULL) ||
478             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
479             tmpdir = strdup(mbs_buffer);
480         }
481     }
482 }
483
484 #endif /* PARALLEL */
485
486 /* find out if stdout and stderr point to the same place */
487 if (fstat(1, &out_stat) < 0) {
488     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
489             "");
490     if (fstat(2, &err_stat) < 0) {
491         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
492             );
493     if ((out_stat.st_dev == err_stat.st_dev) &&
494         (out_stat.st_ino == err_stat.st_ino)) {
495         stdout_stderr_same = true;
496     } else {
497         stdout_stderr_same = false;
498     }
499 }
500
501 /* Make the vroot package scan the path using shell semantics */
502 set_path_style(0);
503
504 setup_char_semantics();
505
506 setup_for_projectdir();
507
508 /*
509 * If running with .KEEP_STATE, curdir will be set with
510 * the connected directory.
511 */
512 (void) atexit(cleanup_after_exit);
513
514 load_cached_names();
515
516 /*
517 * Set command line flags
518 */
519 setup_makeflags_argv();
520 read_command_options(mf_argc, mf_argv);
521 read_command_options(argc, argv);
522
523 if (debug_level > 0) {
524     cp = getenv(makeflags->string_mb);
525     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
526 }
527
528 setup_interrupt(handle_interrupt);
529
530 read_files_and_state(argc, argv);
531
532 #ifdef TEAMWARE_MAKE_CMN
533     /*
534      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
535      */
536     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
537     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
538     prop2 = get_prop(dmake_name2->prop, macro_prop);
539     if (prop2 == NULL) {
540         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
541         output_mode = txt1_mode;
542     } else {
543         dmake_value2 = prop2->body.macro.value;
544         if ((dmake_value2 == NULL) ||
545             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
546             tmpdir = strdup(mbs_buffer);
547         }
548     }
549 }
550
551 #endif /* PARALLEL */
552
553 /* find out if stdout and stderr point to the same place */
554 if (fstat(1, &out_stat) < 0) {
555     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
556             "");
557     if (fstat(2, &err_stat) < 0) {
558         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
559             );
560     if ((out_stat.st_dev == err_stat.st_dev) &&
561         (out_stat.st_ino == err_stat.st_ino)) {
562         stdout_stderr_same = true;
563     } else {
564         stdout_stderr_same = false;
565     }
566 }
567
568 /* Make the vroot package scan the path using shell semantics */
569 set_path_style(0);
570
571 setup_char_semantics();
572
573 setup_for_projectdir();
574
575 /*
576 * If running with .KEEP_STATE, curdir will be set with
577 * the connected directory.
578 */
579 (void) atexit(cleanup_after_exit);
580
581 load_cached_names();
582
583 /*
584 * Set command line flags
585 */
586 setup_makeflags_argv();
587 read_command_options(mf_argc, mf_argv);
588 read_command_options(argc, argv);
589
590 if (debug_level > 0) {
591     cp = getenv(makeflags->string_mb);
592     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
593 }
594
595 setup_interrupt(handle_interrupt);
596
597 read_files_and_state(argc, argv);
598
599 #ifdef TEAMWARE_MAKE_CMN
600     /*
601      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
602      */
603     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
604     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
605     prop2 = get_prop(dmake_name2->prop, macro_prop);
606     if (prop2 == NULL) {
607         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
608         output_mode = txt1_mode;
609     } else {
610         dmake_value2 = prop2->body.macro.value;
611         if ((dmake_value2 == NULL) ||
612             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
613             tmpdir = strdup(mbs_buffer);
614         }
615     }
616 }
617
618 #endif /* PARALLEL */
619
620 /* find out if stdout and stderr point to the same place */
621 if (fstat(1, &out_stat) < 0) {
622     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
623             "");
624     if (fstat(2, &err_stat) < 0) {
625         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
626             );
627     if ((out_stat.st_dev == err_stat.st_dev) &&
628         (out_stat.st_ino == err_stat.st_ino)) {
629         stdout_stderr_same = true;
630     } else {
631         stdout_stderr_same = false;
632     }
633 }
634
635 /* Make the vroot package scan the path using shell semantics */
636 set_path_style(0);
637
638 setup_char_semantics();
639
640 setup_for_projectdir();
641
642 /*
643 * If running with .KEEP_STATE, curdir will be set with
644 * the connected directory.
645 */
646 (void) atexit(cleanup_after_exit);
647
648 load_cached_names();
649
650 /*
651 * Set command line flags
652 */
653 setup_makeflags_argv();
654 read_command_options(mf_argc, mf_argv);
655 read_command_options(argc, argv);
656
657 if (debug_level > 0) {
658     cp = getenv(makeflags->string_mb);
659     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
660 }
661
662 setup_interrupt(handle_interrupt);
663
664 read_files_and_state(argc, argv);
665
666 #ifdef TEAMWARE_MAKE_CMN
667     /*
668      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
669      */
670     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
671     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
672     prop2 = get_prop(dmake_name2->prop, macro_prop);
673     if (prop2 == NULL) {
674         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
675         output_mode = txt1_mode;
676     } else {
677         dmake_value2 = prop2->body.macro.value;
678         if ((dmake_value2 == NULL) ||
679             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
680             tmpdir = strdup(mbs_buffer);
681         }
682     }
683 }
684
685 #endif /* PARALLEL */
686
687 /* find out if stdout and stderr point to the same place */
688 if (fstat(1, &out_stat) < 0) {
689     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
690             "");
691     if (fstat(2, &err_stat) < 0) {
692         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
693             );
694     if ((out_stat.st_dev == err_stat.st_dev) &&
695         (out_stat.st_ino == err_stat.st_ino)) {
696         stdout_stderr_same = true;
697     } else {
698         stdout_stderr_same = false;
699     }
700 }
701
702 /* Make the vroot package scan the path using shell semantics */
703 set_path_style(0);
704
705 setup_char_semantics();
706
707 setup_for_projectdir();
708
709 /*
710 * If running with .KEEP_STATE, curdir will be set with
711 * the connected directory.
712 */
713 (void) atexit(cleanup_after_exit);
714
715 load_cached_names();
716
717 /*
718 * Set command line flags
719 */
720 setup_makeflags_argv();
721 read_command_options(mf_argc, mf_argv);
722 read_command_options(argc, argv);
723
724 if (debug_level > 0) {
725     cp = getenv(makeflags->string_mb);
726     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
727 }
728
729 setup_interrupt(handle_interrupt);
730
731 read_files_and_state(argc, argv);
732
733 #ifdef TEAMWARE_MAKE_CMN
734     /*
735      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
736      */
737     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
738     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
739     prop2 = get_prop(dmake_name2->prop, macro_prop);
740     if (prop2 == NULL) {
741         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
742         output_mode = txt1_mode;
743     } else {
744         dmake_value2 = prop2->body.macro.value;
745         if ((dmake_value2 == NULL) ||
746             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
747             tmpdir = strdup(mbs_buffer);
748         }
749     }
750 }
751
752 #endif /* PARALLEL */
753
754 /* find out if stdout and stderr point to the same place */
755 if (fstat(1, &out_stat) < 0) {
756     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
757             "");
758     if (fstat(2, &err_stat) < 0) {
759         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
760             );
761     if ((out_stat.st_dev == err_stat.st_dev) &&
762         (out_stat.st_ino == err_stat.st_ino)) {
763         stdout_stderr_same = true;
764     } else {
765         stdout_stderr_same = false;
766     }
767 }
768
769 /* Make the vroot package scan the path using shell semantics */
770 set_path_style(0);
771
772 setup_char_semantics();
773
774 setup_for_projectdir();
775
776 /*
777 * If running with .KEEP_STATE, curdir will be set with
778 * the connected directory.
779 */
780 (void) atexit(cleanup_after_exit);
781
782 load_cached_names();
783
784 /*
785 * Set command line flags
786 */
787 setup_makeflags_argv();
788 read_command_options(mf_argc, mf_argv);
789 read_command_options(argc, argv);
790
791 if (debug_level > 0) {
792     cp = getenv(makeflags->string_mb);
793     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
794 }
795
796 setup_interrupt(handle_interrupt);
797
798 read_files_and_state(argc, argv);
799
800 #ifdef TEAMWARE_MAKE_CMN
801     /*
802      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
803      */
804     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
805     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
806     prop2 = get_prop(dmake_name2->prop, macro_prop);
807     if (prop2 == NULL) {
808         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
809         output_mode = txt1_mode;
810     } else {
811         dmake_value2 = prop2->body.macro.value;
812         if ((dmake_value2 == NULL) ||
813             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
814             tmpdir = strdup(mbs_buffer);
815         }
816     }
817 }
818
819 #endif /* PARALLEL */
820
821 /* find out if stdout and stderr point to the same place */
822 if (fstat(1, &out_stat) < 0) {
823     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
824             "");
825     if (fstat(2, &err_stat) < 0) {
826         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
827             );
828     if ((out_stat.st_dev == err_stat.st_dev) &&
829         (out_stat.st_ino == err_stat.st_ino)) {
830         stdout_stderr_same = true;
831     } else {
832         stdout_stderr_same = false;
833     }
834 }
835
836 /* Make the vroot package scan the path using shell semantics */
837 set_path_style(0);
838
839 setup_char_semantics();
840
841 setup_for_projectdir();
842
843 /*
844 * If running with .KEEP_STATE, curdir will be set with
845 * the connected directory.
846 */
847 (void) atexit(cleanup_after_exit);
848
849 load_cached_names();
850
851 /*
852 * Set command line flags
853 */
854 setup_makeflags_argv();
855 read_command_options(mf_argc, mf_argv);
856 read_command_options(argc, argv);
857
858 if (debug_level > 0) {
859     cp = getenv(makeflags->string_mb);
860     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
861 }
862
863 setup_interrupt(handle_interrupt);
864
865 read_files_and_state(argc, argv);
866
867 #ifdef TEAMWARE_MAKE_CMN
868     /*
869      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
870      */
871     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
872     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
873     prop2 = get_prop(dmake_name2->prop, macro_prop);
874     if (prop2 == NULL) {
875         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
876         output_mode = txt1_mode;
877     } else {
878         dmake_value2 = prop2->body.macro.value;
879         if ((dmake_value2 == NULL) ||
880             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
881             tmpdir = strdup(mbs_buffer);
882         }
883     }
884 }
885
886 #endif /* PARALLEL */
887
888 /* find out if stdout and stderr point to the same place */
889 if (fstat(1, &out_stat) < 0) {
890     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
891             "");
892     if (fstat(2, &err_stat) < 0) {
893         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
894             );
895     if ((out_stat.st_dev == err_stat.st_dev) &&
896         (out_stat.st_ino == err_stat.st_ino)) {
897         stdout_stderr_same = true;
898     } else {
899         stdout_stderr_same = false;
900     }
901 }
902
903 /* Make the vroot package scan the path using shell semantics */
904 set_path_style(0);
905
906 setup_char_semantics();
907
908 setup_for_projectdir();
909
910 /*
911 * If running with .KEEP_STATE, curdir will be set with
912 * the connected directory.
913 */
914 (void) atexit(cleanup_after_exit);
915
916 load_cached_names();
917
918 /*
919 * Set command line flags
920 */
921 setup_makeflags_argv();
922 read_command_options(mf_argc, mf_argv);
923 read_command_options(argc, argv);
924
925 if (debug_level > 0) {
926     cp = getenv(makeflags->string_mb);
927     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
928 }
929
930 setup_interrupt(handle_interrupt);
931
932 read_files_and_state(argc, argv);
933
934 #ifdef TEAMWARE_MAKE_CMN
935     /*
936      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
937      */
938     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
939     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
940     prop2 = get_prop(dmake_name2->prop, macro_prop);
941     if (prop2 == NULL) {
942         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
943         output_mode = txt1_mode;
944     } else {
945         dmake_value2 = prop2->body.macro.value;
946         if ((dmake_value2 == NULL) ||
947             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
948             tmpdir = strdup(mbs_buffer);
949         }
950     }
951 }
952
953 #endif /* PARALLEL */
954
955 /* find out if stdout and stderr point to the same place */
956 if (fstat(1, &out_stat) < 0) {
957     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
958             "");
959     if (fstat(2, &err_stat) < 0) {
960         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
961             );
962     if ((out_stat.st_dev == err_stat.st_dev) &&
963         (out_stat.st_ino == err_stat.st_ino)) {
964         stdout_stderr_same = true;
965     } else {
966         stdout_stderr_same = false;
967     }
968 }
969
970 /* Make the vroot package scan the path using shell semantics */
971 set_path_style(0);
972
973 setup_char_semantics();
974
975 setup_for_projectdir();
976
977 /*
978 * If running with .KEEP_STATE, curdir will be set with
979 * the connected directory.
980 */
981 (void) atexit(cleanup_after_exit);
982
983 load_cached_names();
984
985 /*
986 * Set command line flags
987 */
988 setup_makeflags_argv();
989 read_command_options(mf_argc, mf_argv);
990 read_command_options(argc, argv);
991
992 if (debug_level > 0) {
993     cp = getenv(makeflags->string_mb);
994     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
995 }
996
997 setup_interrupt(handle_interrupt);
998
999 read_files_and_state(argc, argv);
1000
1001 #ifdef TEAMWARE_MAKE_CMN
1002     /*
1003      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
1004      */
1005     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
1006     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
1007     prop2 = get_prop(dmake_name2->prop, macro_prop);
1008     if (prop2 == NULL) {
1009         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
1010         output_mode = txt1_mode;
1011     } else {
1012         dmake_value2 = prop2->body.macro.value;
1013         if ((dmake_value2 == NULL) ||
1014             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
1015             tmpdir = strdup(mbs_buffer);
1016         }
1017     }
1018 }
1019
1020 #endif /* PARALLEL */
1021
1022 /* find out if stdout and stderr point to the same place */
1023 if (fstat(1, &out_stat) < 0) {
1024     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
1025             "");
1026     if (fstat(2, &err_stat) < 0) {
1027         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
1028             );
1029     if ((out_stat.st_dev == err_stat.st_dev) &&
1030         (out_stat.st_ino == err_stat.st_ino)) {
1031         stdout_stderr_same = true;
1032     } else {
1033         stdout_stderr_same = false;
1034     }
1035 }
1036
1037 /* Make the vroot package scan the path using shell semantics */
1038 set_path_style(0);
1039
1040 setup_char_semantics();
1041
1042 setup_for_projectdir();
1043
1044 /*
1045 * If running with .KEEP_STATE, curdir will be set with
1046 * the connected directory.
1047 */
1048 (void) atexit(cleanup_after_exit);
1049
1050 load_cached_names();
1051
1052 /*
1053 * Set command line flags
1054 */
1055 setup_makeflags_argv();
1056 read_command_options(mf_argc, mf_argv);
1057 read_command_options(argc, argv);
1058
1059 if (debug_level > 0) {
1060     cp = getenv(makeflags->string_mb);
1061     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
1062 }
1063
1064 setup_interrupt(handle_interrupt);
1065
1066 read_files_and_state(argc, argv);
1067
1068 #ifdef TEAMWARE_MAKE_CMN
1069     /*
1070      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
1071      */
1072     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
1073     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
1074     prop2 = get_prop(dmake_name2->prop, macro_prop);
1075     if (prop2 == NULL) {
1076         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
1077         output_mode = txt1_mode;
1078     } else {
1079         dmake_value2 = prop2->body.macro.value;
1080         if ((dmake_value2 == NULL) ||
1081             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
1082             tmpdir = strdup(mbs_buffer);
1083         }
1084     }
1085 }
1086
1087 #endif /* PARALLEL */
1088
1089 /* find out if stdout and stderr point to the same place */
1090 if (fstat(1, &out_stat) < 0) {
1091     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
1092             "");
1093     if (fstat(2, &err_stat) < 0) {
1094         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
1095             );
1096     if ((out_stat.st_dev == err_stat.st_dev) &&
1097         (out_stat.st_ino == err_stat.st_ino)) {
1098         stdout_stderr_same = true;
1099     } else {
1100         stdout_stderr_same = false;
1101     }
1102 }
1103
1104 /* Make the vroot package scan the path using shell semantics */
1105 set_path_style(0);
1106
1107 setup_char_semantics();
1108
1109 setup_for_projectdir();
1110
1111 /*
1112 * If running with .KEEP_STATE, curdir will be set with
1113 * the connected directory.
1114 */
1115 (void) atexit(cleanup_after_exit);
1116
1117 load_cached_names();
1118
1119 /*
1120 * Set command line flags
1121 */
1122 setup_makeflags_argv();
1123 read_command_options(mf_argc, mf_argv);
1124 read_command_options(argc, argv);
1125
1126 if (debug_level > 0) {
1127     cp = getenv(makeflags->string_mb);
1128     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
1129 }
1130
1131 setup_interrupt(handle_interrupt);
1132
1133 read_files_and_state(argc, argv);
1134
1135 #ifdef TEAMWARE_MAKE_CMN
1136     /*
1137      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
1138      */
1139     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
1140     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
1141     prop2 = get_prop(dmake_name2->prop, macro_prop);
1142     if (prop2 == NULL) {
1143         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
1144         output_mode = txt1_mode;
1145     } else {
1146         dmake_value2 = prop2->body.macro.value;
1147         if ((dmake_value2 == NULL) ||
1148             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
1149             tmpdir = strdup(mbs_buffer);
1150         }
1151     }
1152 }
1153
1154 #endif /* PARALLEL */
1155
1156 /* find out if stdout and stderr point to the same place */
1157 if (fstat(1, &out_stat) < 0) {
1158     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
1159             "");
1160     if (fstat(2, &err_stat) < 0) {
1161         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
1162             );
1163     if ((out_stat.st_dev == err_stat.st_dev) &&
1164         (out_stat.st_ino == err_stat.st_ino)) {
1165         stdout_stderr_same = true;
1166     } else {
1167         stdout_stderr_same = false;
1168     }
1169 }
1170
1171 /* Make the vroot package scan the path using shell semantics */
1172 set_path_style(0);
1173
1174 setup_char_semantics();
1175
1176 setup_for_projectdir();
1177
1178 /*
1179 * If running with .KEEP_STATE, curdir will be set with
1180 * the connected directory.
1181 */
1182 (void) atexit(cleanup_after_exit);
1183
1184 load_cached_names();
1185
1186 /*
1187 * Set command line flags
1188 */
1189 setup_makeflags_argv();
1190 read_command_options(mf_argc, mf_argv);
1191 read_command_options(argc, argv);
1192
1193 if (debug_level > 0) {
1194     cp = getenv(makeflags->string_mb);
1195     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
1196 }
1197
1198 setup_interrupt(handle_interrupt);
1199
1200 read_files_and_state(argc, argv);
1201
1202 #ifdef TEAMWARE_MAKE_CMN
1203     /*
1204      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
1205      */
1206     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
1207     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
1208     prop2 = get_prop(dmake_name2->prop, macro_prop);
1209     if (prop2 == NULL) {
1210         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
1211         output_mode = txt1_mode;
1212     } else {
1213         dmake_value2 = prop2->body.macro.value;
1214         if ((dmake_value2 == NULL) ||
1215             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
1216             tmpdir = strdup(mbs_buffer);
1217         }
1218     }
1219 }
1220
1221 #endif /* PARALLEL */
1222
1223 /* find out if stdout and stderr point to the same place */
1224 if (fstat(1, &out_stat) < 0) {
1225     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
1226             "");
1227     if (fstat(2, &err_stat) < 0) {
1228         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
1229             );
1230     if ((out_stat.st_dev == err_stat.st_dev) &&
1231         (out_stat.st_ino == err_stat.st_ino)) {
1232         stdout_stderr_same = true;
1233     } else {
1234         stdout_stderr_same = false;
1235     }
1236 }
1237
1238 /* Make the vroot package scan the path using shell semantics */
1239 set_path_style(0);
1240
1241 setup_char_semantics();
1242
1243 setup_for_projectdir();
1244
1245 /*
1246 * If running with .KEEP_STATE, curdir will be set with
1247 * the connected directory.
1248 */
1249 (void) atexit(cleanup_after_exit);
1250
1251 load_cached_names();
1252
1253 /*
1254 * Set command line flags
1255 */
1256 setup_makeflags_argv();
1257 read_command_options(mf_argc, mf_argv);
1258 read_command_options(argc, argv);
1259
1260 if (debug_level > 0) {
1261     cp = getenv(makeflags->string_mb);
1262     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
1263 }
1264
1265 setup_interrupt(handle_interrupt);
1266
1267 read_files_and_state(argc, argv);
1268
1269 #ifdef TEAMWARE_MAKE_CMN
1270     /*
1271      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
1272      */
1273     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
1274     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
1275     prop2 = get_prop(dmake_name2->prop, macro_prop);
1276     if (prop2 == NULL) {
1277         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
1278         output_mode = txt1_mode;
1279     } else {
1280         dmake_value2 = prop2->body.macro.value;
1281         if ((dmake_value2 == NULL) ||
1282             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
1283             tmpdir = strdup(mbs_buffer);
1284         }
1285     }
1286 }
1287
1288 #endif /* PARALLEL */
1289
1290 /* find out if stdout and stderr point to the same place */
1291 if (fstat(1, &out_stat) < 0) {
1292     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
1293             "");
1294     if (fstat(2, &err_stat) < 0) {
1295         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
1296             );
1297     if ((out_stat.st_dev == err_stat.st_dev) &&
1298         (out_stat.st_ino == err_stat.st_ino)) {
1299         stdout_stderr_same = true;
1300     } else {
1301         stdout_stderr_same = false;
1302     }
1303 }
1304
1305 /* Make the vroot package scan the path using shell semantics */
1306 set_path_style(0);
1307
1308 setup_char_semantics();
1309
1310 setup_for_projectdir();
1311
1312 /*
1313 * If running with .KEEP_STATE, curdir will be set with
1314 * the connected directory.
1315 */
1316 (void) atexit(cleanup_after_exit);
1317
1318 load_cached_names();
1319
1320 /*
1321 * Set command line flags
1322 */
1323 setup_makeflags_argv();
1324 read_command_options(mf_argc, mf_argv);
1325 read_command_options(argc, argv);
1326
1327 if (debug_level > 0) {
1328     cp = getenv(makeflags->string_mb);
1329     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
1330 }
1331
1332 setup_interrupt(handle_interrupt);
1333
1334 read_files_and_state(argc, argv);
1335
1336 #ifdef TEAMWARE_MAKE_CMN
1337     /*
1338      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
1339      */
1340     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
1341     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
1342     prop2 = get_prop(dmake_name2->prop, macro_prop);
1343     if (prop2 == NULL) {
1344         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
1345         output_mode = txt1_mode;
1346     } else {
1347         dmake_value2 = prop2->body.macro.value;
1348         if ((dmake_value2 == NULL) ||
1349             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
1350             tmpdir = strdup(mbs_buffer);
1351         }
1352     }
1353 }
1354
1355 #endif /* PARALLEL */
1356
1357 /* find out if stdout and stderr point to the same place */
1358 if (fstat(1, &out_stat) < 0) {
1359     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
1360             "");
1361     if (fstat(2, &err_stat) < 0) {
1362         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
1363             );
1364     if ((out_stat.st_dev == err_stat.st_dev) &&
1365         (out_stat.st_ino == err_stat.st_ino)) {
1366         stdout_stderr_same = true;
1367     } else {
1368         stdout_stderr_same = false;
1369     }
1370 }
1371
1372 /* Make the vroot package scan the path using shell semantics */
1373 set_path_style(0);
1374
1375 setup_char_semantics();
1376
1377 setup_for_projectdir();
1378
1379 /*
1380 * If running with .KEEP_STATE, curdir will be set with
1381 * the connected directory.
1382 */
1383 (void) atexit(cleanup_after_exit);
1384
1385 load_cached_names();
1386
1387 /*
1388 * Set command line flags
1389 */
1390 setup_makeflags_argv();
1391 read_command_options(mf_argc, mf_argv);
1392 read_command_options(argc, argv);
1393
1394 if (debug_level > 0) {
1395     cp = getenv(makeflags->string_mb);
1396     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
1397 }
1398
1399 setup_interrupt(handle_interrupt);
1400
1401 read_files_and_state(argc, argv);
1402
1403 #ifdef TEAMWARE_MAKE_CMN
1404     /*
1405      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
1406      */
1407     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
1408     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
1409     prop2 = get_prop(dmake_name2->prop, macro_prop);
1410     if (prop2 == NULL) {
1411         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
1412         output_mode = txt1_mode;
1413     } else {
1414         dmake_value2 = prop2->body.m
```

```

540         if (err) {
541             fatal(err->str);
542         }
543         empty_dmakerc = rcfile->was_empty();
544         delete rcfile;
545     }
546
547     if (empty_dmakerc) {
548         if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
549             putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE"
550             (void) fprintf(stdout, catgets(catd, 1,
551             (void) fprintf(stdout, catgets(catd, 1,
552             dmake_mode_type = parallel_mode;
553             no_parallel = false;
554         }
555     }
556
557 #else
558     if(dmake_mode_type == distributed_mode) {
559         (void) fprintf(stdout, NOCATGETS("dmake: Distributed mod
560         (void) fprintf(stdout, NOCATGETS("      Defaulting to p
561         dmake_mode_type = parallel_mode;
562         no_parallel = false;
563     }
564 #endif /* DISTRIBUTED */
565 }
566 }
567#endif

569 #ifdef TEAMWARE_MAKE_CMN
570     parallel_flag = true;
571     /* XXX - This is a major hack for DMake/Licensing. */
572     if (getenv(NOCATGETS("DMAKE_CHILD")) == NULL) {
573         if (!avo_cli_search_license(argv[0], dmake_exit_callback, TRUE,
574             /*
575             * If the user can not get a TeamWare license,
576             * default to serial mode.
577             */
578             dmake_mode_type = serial_mode;
579             no_parallel = true;
580     } else {
581         putenv(NOCATGETS("DMAKE_CHILD=TRUE"));
582     }
583     start_time = time(NULL);
584     /*
585     * XXX - Hack to disable SIGALRM's from licensing library's
586     *       setitimer().
587     */
588     value.it_interval.tv_sec = 0;
589     value.it_interval.tv_usec = 0;
590     value.it_value.tv_sec = 0;
591     value.it_value.tv_usec = 0;
592     (void) setitimer(ITIMER_REAL, &value, NULL);
593 }

595 // If dmake is running with -t option, set dmake_mode_type to serial.
596 // This is done because doname() calls touch_command() that runs serially.
597 // If we do not do that, maketool will have problems.
598 //
599     if(touch) {
600         dmake_mode_type = serial_mode;
601         no_parallel = true;
602     }
603 }
604 #else
605     parallel_flag = false;

```

```

606 #endif

608 #if defined (TEAMWARE_MAKE_CMN) && defined(RDIRECT_ERR)
609     /*
610      * Check whether stdout and stderr are physically same.
611      * This is in order to decide whether we need to redirect
612      * stderr separately from stdout.
613      * This check is performed only if __DMAKE_SEPARATE_STDERR
614      * is not set. This variable may be used in order to preserve
615      * the 'old' behaviour.
616      */
617     out_err_same = true;
618     char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
619     if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS(
620         struct stat stdout_stat;
621         struct stat stderr_stat;
622         if( (fstat(1, &stdout_stat) == 0)
623             && (fstat(2, &stderr_stat) == 0) )
624             {
625                 if( (stdout_stat.st_dev != stderr_stat.st_dev)
626                     || (stdout_stat.st_ino != stderr_stat.st_ino) )
627                     {
628                         out_err_same = false;
629                     }
630             }
631     })
632#endif

634 #ifdef DISTRIBUTED
635     /*
636      * At this point, DMake should startup an r xm with any and all
637      * DMake command line options. Rxm will, among other things,
638      * read the rc file.
639      */
640     if ((!list_all_targets) &&
641         (report_dependencies_level == 0) &&
642         (dmake_mode_type == distributed_mode)) {
643         startup_rxm();
644     }
645 #endif
646
647 /*
648 * Enable interrupt handler for alarms
649 */
650 (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

652 /*
653 * Check if make should report
654 */
655 if (getenv(sunpro_dependencies->string_mb) != NULL) {
656     FILE *report_file;

658     report_dependency("");
659     report_file = get_report_file();
660     if ((report_file != NULL) && (report_file != (FILE*)-1)) {
661         (void) fprintf(report_file, "\n");
662     }
663 }

665 /*
666 * Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
667 */
668 if (keep_state) {
669     maybe_append_prop(sunpro_dependencies, macro_prop)->
670         body.macro.exported = true;
671 } else {

```

```

672         maybe_append_prop(sunpro_dependencies, macro_prop)->
673             body.macro.exported = false;
674     }

675     working_on_targets = true;
676     if (trace_status) {
677         dump_make_state();
678         fclose(stdout);
679         fclose(stderr);
680         exit_status = 0;
681         exit(0);
682     }
683     if (list_all_targets) {
684         dump_target_list();
685         fclose(stdout);
686         fclose(stderr);
687         exit_status = 0;
688         exit(0);
689     }
690     trace_reader = false;

691     /*
692      * Set temp_file_directory to the directory the .make.state
693      * file is written to.
694      */
695     if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
696         temp_file_directory = strdup(get_current_path());
697     } else {
698         *slash_ptr = (int) nul_char;
699         (void) strcpy(make_state_dir, make_state->string_mb);
700         *slash_ptr = (int) slash_char;
701         /* when there is only one slash and it's the first
702          ** character, make_state_dir should point to '/'.
703          */
704         if(make_state_dir[0] == '\0') {
705             make_state_dir[0] = '/';
706             make_state_dir[1] = '\0';
707         }
708         if (make_state_dir[0] == (int) slash_char) {
709             temp_file_directory = strdup(make_state_dir);
710         } else {
711             char    tmp_current_path2[MAXPATHLEN];
712
713             (void) sprintf(tmp_current_path2,
714                           "%s/%s",
715                           get_current_path(),
716                           make_state_dir);
717             temp_file_directory = strdup(tmp_current_path2);
718         }
719     }
720 }

721 #ifdef DISTRIBUTED
722     building_serial = false;
723 #endif

724     report_dir_enter_leave(true);

725     make_targets(argc, argv, parallel_flag);

726     report_dir_enter_leave(false);

727     if (build_failed_ever_seen) {
728         if (posix) {
729             exit_status = 1;
730         }
731     }
732     exit(1);
733 
```

```

738         }
739         exit_status = 0;
740         exit(0);
741         /* NOTREACHED */
742     }
743     unchanged_portion_omitted

1116 /*
1117  *      read_command_options(argc, argv)
1118  *
1119  *      Scan the cmd line options and process the ones that start with "-"
1120  *
1121  *      Return value:
1122  *                      -M argument, if any
1123  *
1124  *      Parameters:
1125  *                      argc           You know what this is
1126  *                      argv           You know what this is
1127  *
1128  *      Global variables used:
1129  */
1130 static void
1131 read_command_options(register int argc, register char **argv)
1132 {
1133     register int          ch;
1134     int                  current_optind = 1;
1135     int                  last_optind_with_double_hyphen = 0;
1136     int                  last_optind;
1137     int                  last_current_optind;
1138     register int          i;
1139     register int          j;
1140     register int          k;
1141     register int          makefile_next = 0; /*
1142                                * flag to note options:
1143                                * -c, f, g, j, m, o
1144                                */
1145     const char            *tptr;
1146     const char            *CMD_OPTS;
1147     extern char           *optarg;
1148     extern int             optind, optarg, opterr, optopt;

1149 #define SUNPRO_CMD_OPTS "~-Bbc:Ddef:g:ij:K:kM:m:NnO:o:PpqRrSsTtuVvwx:"
1150 #ifdef TEAMWARE_MAKE_CMN
1151 #define SVR4_CMD_OPTS   "-c:ef:g:ij:km:nO:o:pqrsTtVv"
1152 #else
1153 #define SVR4_CMD_OPTS   "-ef:iknpqrstV"
1154 #endif

1155 /*
1156  * Added V in SVR4_CMD_OPTS also, which is going to be a hidden
1157  * option, just to make sure that the getopt doesn't fail when some
1158  * users leave their USE_SVR4_MAKE set and try to use the makefiles
1159  * that are designed to issue commands like $(MAKE) -V. Anyway it
1160  * sets the same flag but ensures that getopt doesn't fail.
1161  */
1162
1163 opterr = 0;
1164 optind = 1;
1165 while (1) {
1166     last_optind=optind;
1167     last_current_optind=current_optind; /* Save optind and current_optind
1168     if (svr4) {                                /* in case we have to re
1169         CMD_OPTS=SVR4_CMD_OPTS;
1170     }
1171     if (sunpro) {
1172         CMD_OPTS=SUNPRO_CMD_OPTS;
1173     }
1174 }
```

```

1174         ch = getopt(argc, argv, SVR4_CMD_OPTS);
1175     } else {
1176         CMD_OPTS=SUNPRO_CMD_OPTS;
1177         ch = getopt(argc, argv, SUNPRO_CMD_OPTS);
1178     }
1179     if (ch == EOF) {
1180         if(optind < argc) {
1181             /*
1182             * Fixing bug 4102537:
1183             * Strange behaviour of command make using --
1184             * Not all argv have been processed
1185             * Skip non-flag argv and continue processing.
1186             */
1187             optind++;
1188             current_optind++;
1189             continue;
1190         } else {
1191             break;
1192         }
1193     }
1194     if (ch == '?') {
1195         if (optopt == '--') {
1196             /*
1197             * Bug 5060758: getopt() changed behavior (s10_6
1198             * and now we have to deal with cases when option
1199             * with double hyphen appear here, from -$MAKEF
1200             */
1201             i = current_optind;
1202             if (argv[i][0] == '-') {
1203                 if (argv[i][1] == '-') {
1204                     if (argv[i][2] != '\0') {
1205                         /* Check if this option is allowed */
1206                         tptr = strchr(CMD_OPTS, argv[i][2]);
1207                         if (tptr) {
1208                             if (last_optind_with_double_hyphen != cu
1209                                 /* This is first time we are trying to
1210                                 * problem with this option. If we com
1211                                 * time, we will go to fatal error.
1212                                 */
1213                             last_optind_with_double_hyphen = curre
1214
1215                             /* Eliminate first hyphen character */
1216                             for (j=0; argv[i][j] != '\0'; j++) {
1217                                 argv[i][j] = argv[i][j+1];
1218                             }
1219
1220                             /* Repeat the processing of this argument */
1221                             optind=last_optind;
1222                             current_optind=last_current_optind;
1223                             continue;
1224                         }
1225                     }
1226                 }
1227             }
1228         }
1229     }
1230
1231     if (ch == '?') {
1232         if (svr4) {
1233             ifndef TEAMWARE_MAKE_CMN
1234                 fprintf(stderr,
1235                         catgets(catd, 1, 267, "Usage : dmake [ - "
1236                         fprintf(stderr,
1237                         catgets(catd, 1, 268, " [ -
1238                         fprintf(stderr,

```

```

1240         catgets(catd, 1, 269, " [ -
1241         #else
1242         catgets(catd, 1, 270, "Usage : make [ -f
1243         fprintf(stderr,
1244             catgets(catd, 1, 271, " [ -
1245         #endif
1246         tptr = strchr(SVR4_CMD_OPTS, optopt);
1247         } else {
1248             fprintf(stderr,
1249             catgets(catd, 1, 272, "Usage : dmake [ - "
1250             fprintf(stderr,
1251             catgets(catd, 1, 273, " [ -
1252             fprintf(stderr,
1253             catgets(catd, 1, 274, " [ -
1254             fprintf(stderr,
1255             catgets(catd, 1, 275, " [ -
1256
1257             fprintf(stderr,
1258             catgets(catd, 1, 276, "Usage : make [ -f
1259             fprintf(stderr,
1260             catgets(catd, 1, 277, " [ -
1261             fprintf(stderr,
1262             catgets(catd, 1, 278, " [ -
1263             tptr = strchr(SUNPRO_CMD_OPTS, optopt);
1264             } if (!tptr) {
1265                 fatal(catgets(catd, 1, 279, "Unknown option '-%c
1266             } else {
1267                 fatal(catgets(catd, 1, 280, "Missing argument af
1268             }
1269
1270             makefile_next |= parse_command_option(ch);
1271
1272             /*
1273             * If we're done processing all of the options of
1274             * ONE argument string...
1275             */
1276             if (current_optind < optind) {
1277                 i = current_optind;
1278                 k = 0;
1279                 /* If there's an argument for an option... */
1280                 if ((optind - current_optind) > 1) {
1281                     k = i + 1;
1282                 }
1283                 switch (makefile_next) {
1284                     case 0:
1285                         argv[i] = NULL;
1286                         /* This shouldn't happen */
1287                         if (k) {
1288                             argv[k] = NULL;
1289                         }
1290                         break;
1291                     case 1: /* -f seen */
1292                         argv[i] = (char *)NOCATGETS("-f");
1293                         break;
1294                     case 2: /* -c seen */
1295                         argv[i] = (char *)NOCATGETS("-c");
1296                         warning(catgets(catd, 1, 281, "Ignoring Distribu
1297                         break;
1298
1299             #ifndef TEAMWARE_MAKE_CMN
1300             #endif
1301
1302             warning(catgets(catd, 1, 281, "Ignoring Distribu
1303             break;
1304         }
1305

```

```

1306             case 4: /* -g seen */
1307                 argv[i] = (char *)NOCATGETS("-g");
1308 #ifndef TEAMWARE_MAKE_CMN
1309                 warning(catgets(catd, 1, 282, "Ignoring Distribu
1310 #endif
1311             break;
1312         case 8: /* -j seen */
1313             argv[i] = (char *)NOCATGETS("-j");
1314 #ifndef TEAMWARE_MAKE_CMN
1315             warning(catgets(catd, 1, 283, "Ignoring Distribu
1316 #endif
1317             break;
1318         case 16: /* -M seen */
1319             argv[i] = (char *)NOCATGETS("-M");
1320 #ifndef TEAMWARE_MAKE_CMN
1321             warning(catgets(catd, 1, 284, "Ignoring Parallel
1322 #endif
1323             break;
1324         case 32: /* -m seen */
1325             argv[i] = (char *)NOCATGETS("-m");
1326 #ifndef TEAMWARE_MAKE_CMN
1327             warning(catgets(catd, 1, 285, "Ignoring Distribu
1328 #endif
1329             break;
1330 #ifndef PARALLEL
1331         case 128: /* -O seen */
1332             argv[i] = (char *)NOCATGETS("-O");
1333             break;
1334 #endif
1335         case 256: /* -K seen */
1336             argv[i] = (char *)NOCATGETS("-K");
1337             break;
1338 #ifndef TEAMWARE_MAKE_CMN
1339             warning(catgets(catd, 1, 311, "Ignoring Distribu
1340 #endif
1341             break;
1342         case 1024: /* -x seen */
1343             argv[i] = (char *)NOCATGETS("-x");
1344 #ifndef TEAMWARE_MAKE_CMN
1345             warning(catgets(catd, 1, 353, "Ignoring Distribu
1346 #endif
1347             break;
1348         default: /* > 1 of -c, f, g, j, K, M, m, O, o, x seen */
1349             fatal(catgets(catd, 1, 286, "Illegal command lin
1350         }
1351
1352         makefile_next = 0;
1353         current_optind = optind;
1354     }
1355 }
1356 }



---


unchanged_portion_omitted_

1538 /*
1539 * parse_command_option(ch)
1540 *
1541 * Parse make command line options.
1542 *
1543 * Return value:           Indicates if any -f -c or -M were seen
1544 *
1545 *
1546 * Parameters:            ch
1547 *                         The character to parse
1548 */

```

```

1549 *      Static variables used:
1550 *          dmake_group_specified   Set for make -g
1551 *          dmake_max_jobs_specified Set for make -j
1552 *          dmake_mode_specified    Set for make -m
1553 *          dmake_add_mode_specified Set for make -x
1554 *          dmake_compat_mode_specified Set for make -x SUN_MAKE_COMPAT_
1555 *          dmake_output_mode_specified Set for make -x DMAKE_OUTPUT_MOD
1556 *          dmake_odir_specified    Set for make -o
1557 *          dmake_rcfile_specified  Set for make -c
1558 *          env_wins                Set for make -e
1559 *          ignore_default_mk      Set for make -r
1560 *          trace_status            Set for make -p
1561 *
1562 *      Global variables used:
1563 *          .make.state.path & name set for make -K
1564 *          continue_after_error Set for make -k
1565 *          debug_level            Set for make -d
1566 *          do_not_exec_rule       Set for make -n
1567 *          filter_stderr          Set for make -X
1568 *          ignore_errors_all      Set for make -i
1569 *          no_parallel            Set for make -R
1570 *          quest                  Set for make -q
1571 *          read_trace_level       Set for make -D
1572 *          report_dependencies     Set for make -P
1573 *          send_mtool_msgs        Set for make -K
1574 *          silent_all              Set for make -s
1575 *          touch                  Set for make -t
1576 */
1577 static int
1578 parse_command_option(register char ch)
1579 {
1580     static int               invert_next = 0;
1581     int                      invert_this = invert_next;
1582
1583     invert_next = 0;
1584     switch (ch) {
1585     case '-':               /* Ignore "--" */
1586         return 0;
1587     case '~':               /* Invert next option */
1588         invert_next = 1;
1589         return 0;
1590     case 'B':               /* Obsolete */
1591         return 0;
1592     case 'b':               /* Obsolete */
1593         return 0;
1594     case 'c':               /* Read alternative dmakerc file */
1595         if (invert_this) {
1596             dmake_rcfile_specified = false;
1597         } else {
1598             dmake_rcfile_specified = true;
1599         }
1600         return 2;
1601     case 'D':               /* Show lines read */
1602         if (invert_this) {
1603             read_trace_level--;
1604         } else {
1605             read_trace_level++;
1606         }
1607         return 0;
1608     case 'd':               /* Debug flag */
1609         if (invert_this) {
1610             debug_level--;
1611         } else {
1612             debug_level++;
1613         }
1614     return 0;

```

```

1615     case 'e':                      /* Environment override flag */
1616         if (invert_this) {
1617             env_wins = false;
1618         } else {
1619             env_wins = true;
1620         }
1621         return 0;
1622     case 'f':                      /* Read alternative makefile(s) */
1623         return 1;
1624     case 'g':                      /* Use alternative DMake group */
1625         if (invert_this) {
1626             dmake_group_specified = false;
1627         } else {
1628             dmake_group_specified = true;
1629         }
1630         return 4;
1631     case 'i':                      /* Ignore errors */
1632         if (invert_this) {
1633             ignore_errors_all = false;
1634         } else {
1635             ignore_errors_all = true;
1636         }
1637         return 0;
1638     case 'j':                      /* Use alternative DMake max jobs */
1639         if (invert_this) {
1640             dmake_max_jobs_specified = false;
1641         } else {
1642             dmake_max_jobs_specified = true;
1643         }
1644         return 8;
1645     case 'K':                      /* Read alternative .make.state */
1646         return 256;
1647     case 'k':                      /* Keep making even after errors */
1648         if (invert_this) {
1649             continue_after_error = false;
1650         } else {
1651             continue_after_error = true;
1652             continue_after_error_ever_seen = true;
1653         }
1654         return 0;
1655     case 'M':                      /* Read alternative make.machines file */
1656         if (invert_this) {
1657             pmake_machinesfile_specified = false;
1658         } else {
1659             pmake_machinesfile_specified = true;
1660             dmake_mode_type = parallel_mode;
1661             no_parallel = false;
1662         }
1663         return 16;
1664     case 'm':                      /* Use alternative DMake build mode */
1665         if (invert_this) {
1666             dmake_mode_specified = false;
1667         } else {
1668             dmake_mode_specified = true;
1669         }
1670         return 32;
1671     case 'x':                      /* Use alternative DMake mode */
1672         if (invert_this) {
1673             dmake_add_mode_specified = false;
1674         } else {
1675             dmake_add_mode_specified = true;
1676         }
1677         return 1024;
1678     case 'N':                      /* Reverse -n */
1679         if (invert_this) {
1680             do_not_exec_rule = true;

```

```

1681             } else {
1682                 do_not_exec_rule = false;
1683             }
1684             return 0;
1685         case 'n':                      /* Print, not exec commands */
1686         if (invert_this) {
1687             do_not_exec_rule = false;
1688         } else {
1689             do_not_exec_rule = true;
1690         }
1691         return 0;
1700 #ifndef PARALLEL
1702     case 'O':                      /* Send job start & result msgs */
1703         if (invert_this) {
1704             send_mtool_msgs = false;
1705         } else {
1706             send_mtool_msgs = true;
1707         }
1708 #endif
1710         return 128;
1701     case 'o':                      /* Use alternative dmake output dir */
1702         if (invert_this) {
1703             dmake_odir_specified = false;
1704         } else {
1705             dmake_odir_specified = true;
1706         }
1707         return 512;
1708     case 'P':                      /* Print for selected targets */
1709         if (invert_this) {
1710             report_dependencies_level--;
1711         } else {
1712             report_dependencies_level++;
1713         }
1714         return 0;
1715     case 'p':                      /* Print description */
1716         if (invert_this) {
1717             trace_status = false;
1718             do_not_exec_rule = false;
1719         } else {
1720             trace_status = true;
1721             do_not_exec_rule = true;
1722         }
1723         return 0;
1724     case 'q':                      /* Question flag */
1725         if (invert_this) {
1726             quest = false;
1727         } else {
1728             quest = true;
1729         }
1730         return 0;
1731     case 'R':                      /* Don't run in parallel */
1732 #ifdef TEAMWARE_MAKE_CMN
1733         if (invert_this) {
1734             pmake_cap_r_specified = false;
1735             no_parallel = false;
1736         } else {
1737             pmake_cap_r_specified = true;
1738             dmake_mode_type = serial_mode;
1739             no_parallel = true;
1740         }
1741     #else
1742         warning(catgets(catd, 1, 182, "Ignoring ParallelMake -R option"));
1743     #endif
1744         return 0;

```

```

1745     case 'r':                  /* Turn off internal rules */
1746         if (invert_this) {
1747             ignore_default_mk = false;
1748         } else {
1749             ignore_default_mk = true;
1750         }
1751         return 0;
1752     case 'S':                  /* Reverse -k */
1753         if (invert_this) {
1754             continue_after_error = true;
1755         } else {
1756             continue_after_error = false;
1757             stop_after_error_ever_seen = true;
1758         }
1759         return 0;
1760     case 's':                  /* Silent flag */
1761         if (invert_this) {
1762             silent_all = false;
1763         } else {
1764             silent_all = true;
1765         }
1766         return 0;
1767     case 'T':                  /* Print target list */
1768         if (invert_this) {
1769             list_all_targets = false;
1770             do_not_exec_rule = false;
1771         } else {
1772             list_all_targets = true;
1773             do_not_exec_rule = true;
1774         }
1775         return 0;
1776     case 't':                  /* Touch flag */
1777         if (invert_this) {
1778             touch = false;
1779         } else {
1780             touch = true;
1781         }
1782         return 0;
1783     case 'u':                  /* Unconditional flag */
1784         if (invert_this) {
1785             build_unconditional = false;
1786         } else {
1787             build_unconditional = true;
1788         }
1789         return 0;
1790     case 'V':                  /* SVR4 mode */
1791         svr4 = true;
1792         return 0;
1793     case 'v':                  /* Version flag */
1794         if (invert_this) {
1795         } else {
1796 #ifdef TEAMWARE_MAKE_CMN
1797             fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
1798             exit_status = 0;
1799             exit(0);
1800 #else
1801             warning(catgets(catd, 1, 324, "Ignoring DistributedMake
1802 #endif
1803         }
1804         return 0;
1805     case 'w':                  /* Unconditional flag */
1806         if (invert_this) {
1807             report_cwd = false;
1808         } else {
1809             report_cwd = true;
1810         }

```

```

1811                     return 0;
1812 #if 0
1813     case 'X':                  /* Filter stdout */
1814         if (invert_this) {
1815             filter_stderr = false;
1816         } else {
1817             filter_stderr = true;
1818         }
1819         return 0;
1820 #endif
1821     default:
1822         break;
1823     }
1824     return 0;
1825 }

unchanged_portion_omitted

1938 /*
1939 *      read_files_and_state(argc, argv)
1940 *
1941 *      Read the makefiles we care about and the environment
1942 *      Also read the = style command line options
1943 *
1944 *      Parameters:
1945 *          argc           You know what this is
1946 *          argv           You know what this is
1947 *
1948 *      Static variables used:
1949 *          env_wins        make -e, determines if env vars are RO
1950 *          ignore_default_mk make -r, determines if make.rules is read
1951 *          not_auto_dopen  dwight
1952 *
1953 *      Global variables used:
1954 *          default_target_to_build Set to first proper target from file
1955 *          do_not_exec_rule Set to false when makfile is made
1956 *          dot              The Name ".", used to read current dir
1957 *          empty_name       The Name "", use as macro value
1958 *          keep_state       Set if KEEP_STATE is in environment
1959 *          make_state       The Name ".make.state", used to read file
1960 *          makefile_type    Set to type of file being read
1961 *          makeflags        The Name "MAKEFLAGS", used to set macro value
1962 *          not_auto         dwight
1963 *          read_trace_level Checked to see if the reader should trace
1964 *          report_dependencies If -P is on we do not read .make.state
1965 *          trace_reader    Set if reader should trace
1966 *          virtual_root     The Name "VIRTUAL_ROOT", used to check value
1967 */
1968 static void
1969 read_files_and_state(int argc, char **argv)
1970 {
1971     wchar_t          buffer[1000];
1972     wchar_t          buffer_posix[1000];
1973     register char   ch;
1974     register char   *cp;
1975     Property        def_make_macro = NULL;
1976     Name            def_make_name;
1977     Name            default_makefile;
1978     String_rec      destbuffer[STRING_BUFFER_LENGTH];
1979     wchar_t          i;
1980     register int    j;
1981     register int    keep_state_name;
1982     register int    length;
1983     register int    name;
1984     register Property Makefile;
1985     register struct stat macro;
1986     register struct stat make_state_stat;

```

```

1987     Name          makefile_name;
1988     register int   makefile_next = 0;
1989     register Boolean makefile_read = false;
1990     String_rec    makeflags_string;
1991     String_rec *  makeflags_string_posix;
1992     String_rec *  makeflags_string_current;
1993     Name          makeflags_value_saved;
1994     register Name  name;
1995     Name          new_make_value;
1996     Boolean       save_do_not_exec_rule;
1997     Name          sdotMakefile;
1998     Name          sdotmakefile_name;
1999     static wchar_t state_file_str;
2000     static char    state_file_str_mb[MAXPATHLEN];
2001     static struct _Name state_filename;
2002     Boolean       temp;
2003     char          tmp_char;
2004     wchar_t       *tmp_wcs_buffer;
2005     register Name  value;
2006     ASCII_Dyn_Array makeflags_and_macro;
2007     Boolean       is_xpg4;

2009 */
210  */
211  * Remember current mode. It may be changed after reading makefile
212  * and we will have to correct MAKEFLAGS variable.
213 */
214 is_xpg4 = posix;

215 MBSTOWCS(wcs_buffer, NOCATGETS("KEEP_STATE"));
216 keep_state_name = GETNAME(wcs_buffer, FIND_LENGTH);
217 MBSTOWCS(wcs_buffer, NOCATGETS("Makefile"));
218 Makefile = GETNAME(wcs_buffer, FIND_LENGTH);
219 MBSTOWCS(wcs_buffer, NOCATGETS("makefile"));
220 makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
221 MBSTOWCS(wcs_buffer, NOCATGETS("s.makefile"));
222 sdotmakefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
223 MBSTOWCS(wcs_buffer, NOCATGETS("s.Makefile"));
224 sdotMakefile = GETNAME(wcs_buffer, FIND_LENGTH);

226 */
227 */
228 */
229 Set flag if NSE is active

230 */
231 */
232 */
233 not_auto_depen->next = NULL;
234 not_auto_depen->name = not_auto;
235 not_auto_depen->automatic = not_auto_depen->stale = false;

237 */
238 */
239 */
240 Read internal definitions and rules.

241 if (read_trace_level > 1) {
242     trace_reader = true;
243 }
244 if (!ignore_default_mk) {
245     if (svr4)
246         MBSTOWCS(wcs_buffer, NOCATGETS("svr4.make.rules"));
247     default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
248 } else {
249     MBSTOWCS(wcs_buffer, NOCATGETS("make.rules"));
250     default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
251 }
252 default_makefile->stat.is_file = true;

```

```

2053     (void) read_makefile(default_makefile,
2054                           true,
2055                           false,
2056                           true);
2057 }

2059 */
2060 * If the user did not redefine the MAKE macro in the
2061 * default makefile (make.rules), then we'd like to
2062 * change the macro value of MAKE to be some form
2063 * of argv[0] for recursive MAKE builds.
2064 */
2065 MBSTOWCS(wcs_buffer, NOCATGETS("MAKE"));
2066 def_make_name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2067 def_make_macro = get_prop(def_make_name->prop, macro_prop);
2068 if ((def_make_macro != NULL) &&
2069     (IS_EQUAL(def_make_macro->body.macro.value->string_mb,
2070                NOCATGETS("make")))) {
2071     MBSTOWCS(wcs_buffer, argv_zero_string);
2072     new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2073     (void) SETVAR(def_make_name,
2074                   new_make_value,
2075                   false);
2076 }

2078 default_target_to_build = NULL;
2079 trace_reader = false;

2081 */
2082 */
2083 */
2084 */
2085 */
2086 */
2087 */
2088 */
2089 */
2090 */
2091 */
2092 */
2093 */

2095 */
2096 */
2097 */
2098 */
2099 */
2100 */
2101 */
2102 */
2103 */
2104 */
2105 */
2106 */
2107 */
2108 */
2109 */
2110 */
2111 */
2112 */
2113 */
2114 */
2115 */

2117 */
2118 */

2119 INIT_STRING_FROM_STACK(makeflags_string, buffer);
2120 INIT_STRING_FROM_STACK(makeflags_string_posix, buffer_posix);
2121 append_char((int) hyphen_char, &makeflags_string);
2122 append_char((int) hyphen_char, &makeflags_string_posix);

2123 switch (read_trace_level) {
2124 case 2:

```

```

2119         append_char('D', &makeflags_string);
2120         append_char('D', &makeflags_string_posix);
2121     case 1:
2122         append_char('D', &makeflags_string);
2123         append_char('D', &makeflags_string_posix);
2124     }
2125     switch (debug_level) {
2126     case 2:
2127         append_char('d', &makeflags_string);
2128         append_char('d', &makeflags_string_posix);
2129     case 1:
2130         append_char('d', &makeflags_string);
2131         append_char('d', &makeflags_string_posix);
2132     }
2133     if (env_wins) {
2134         append_char('e', &makeflags_string);
2135         append_char('e', &makeflags_string_posix);
2136     }
2137     if (ignore_errors_all) {
2138         append_char('i', &makeflags_string);
2139         append_char('i', &makeflags_string_posix);
2140     }
2141     if (continue_after_error) {
2142         if (stop_after_error_ever_seen) {
2143             append_char('S', &makeflags_string_posix);
2144             append_char((int) space_char, &makeflags_string_posix);
2145             append_char((int) hyphen_char, &makeflags_string_posix);
2146         }
2147         append_char('k', &makeflags_string);
2148         append_char('k', &makeflags_string_posix);
2149     } else {
2150         if (stop_after_error_ever_seen
2151             && continue_after_error_ever_seen) {
2152             append_char('k', &makeflags_string_posix);
2153             append_char((int) space_char, &makeflags_string_posix);
2154             append_char((int) hyphen_char, &makeflags_string_posix);
2155             append_char('S', &makeflags_string_posix);
2156         }
2157     }
2158     if (do_not_exec_rule) {
2159         append_char('n', &makeflags_string);
2160         append_char('n', &makeflags_string_posix);
2161     }
2162     switch (report_dependencies_level) {
2163     case 4:
2164         append_char('P', &makeflags_string);
2165         append_char('P', &makeflags_string_posix);
2166     case 3:
2167         append_char('P', &makeflags_string);
2168         append_char('P', &makeflags_string_posix);
2169     case 2:
2170         append_char('P', &makeflags_string);
2171         append_char('P', &makeflags_string_posix);
2172     case 1:
2173         append_char('P', &makeflags_string);
2174         append_char('P', &makeflags_string_posix);
2175     }
2176     if (trace_status) {
2177         append_char('p', &makeflags_string);
2178         append_char('p', &makeflags_string_posix);
2179     }
2180     if (quest) {
2181         append_char('q', &makeflags_string);
2182         append_char('q', &makeflags_string_posix);
2183     }
2184     if (silent_all) {

```

```

2185         append_char('s', &makeflags_string);
2186         append_char('s', &makeflags_string_posix);
2187     }
2188     if (touch) {
2189         append_char('t', &makeflags_string);
2190         append_char('t', &makeflags_string_posix);
2191     }
2192     if (build_unconditional) {
2193         append_char('u', &makeflags_string);
2194         append_char('u', &makeflags_string_posix);
2195     }
2196     if (report_cwd) {
2197         append_char('w', &makeflags_string);
2198         append_char('w', &makeflags_string_posix);
2199     }
2200 #ifndef PARALLEL
2201 /* -c dmake_rcfile */
2202 if (dmake_rcfile_specified) {
2203     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2204     dmake_rcfile = GETNAME(wcs_buffer, FIND_LENGTH);
2205     append_makeflags_string(dmake_rcfile, &makeflags_string);
2206     append_makeflags_string(dmake_rcfile, &makeflags_string_posix);
2207 }
2208 /* -g dmake_group */
2209 if (dmake_group_specified) {
2210     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
2211     dmake_group = GETNAME(wcs_buffer, FIND_LENGTH);
2212     append_makeflags_string(dmake_group, &makeflags_string);
2213     append_makeflags_string(dmake_group, &makeflags_string_posix);
2214 }
2215 /* -j dmake_max_jobs */
2216 if (dmake_max_jobs_specified) {
2217     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
2218     dmake_max_jobs = GETNAME(wcs_buffer, FIND_LENGTH);
2219     append_makeflags_string(dmake_max_jobs, &makeflags_string);
2220     append_makeflags_string(dmake_max_jobs, &makeflags_string_posix);
2221 }
2222 /* -m dmake_mode */
2223 if (dmake_mode_specified) {
2224     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2225     dmake_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2226     append_makeflags_string(dmake_mode, &makeflags_string);
2227     append_makeflags_string(dmake_mode, &makeflags_string_posix);
2228 }
2229 /* -x dmake_compat_mode */
2230 if (dmake_compat_mode_specified) {
2231     MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE_COMPAT_MODE"));
2232     dmake_compat_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2233     append_makeflags_string(dmake_compat_mode, &makeflags_string);
2234     append_makeflags_string(dmake_compat_mode, &makeflags_string_posix);
2235 }
2236 /* -x dmake_output_mode */
2237 if (dmake_output_mode_specified) {
2238     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
2239     dmake_output_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2240     append_makeflags_string(dmake_output_mode, &makeflags_string);
2241     append_makeflags_string(dmake_output_mode, &makeflags_string_posix);
2242 }
2243 /* -o dmake_odir */
2244 if (dmake_odir_specified) {
2245     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2246     dmake_odir = GETNAME(wcs_buffer, FIND_LENGTH);
2247     append_makeflags_string(dmake_odir, &makeflags_string);
2248     append_makeflags_string(dmake_odir, &makeflags_string_posix);
2249 }
2250 /* -M pmake_machinesfile */

```

```

2250     if (pmake_machinesfile_specified) {
2251         MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
2252         pmake_machinesfile = GETNAME(wcs_buffer, FIND_LENGTH);
2253         append_makeflags_string(pmake_machinesfile, &makeflags_string);
2254         append_makeflags_string(pmake_machinesfile, &makeflags_string_posix);
2255     }
2256     /* -R */
2257     if (pmake_cap_r_specified) {
2258         append_char((int) space_char, &makeflags_string);
2259         append_char((int) hyphen_char, &makeflags_string);
2260         append_char('R', &makeflags_string);
2261         append_char((int) space_char, &makeflags_string_posix);
2262         append_char((int) hyphen_char, &makeflags_string_posix);
2263         append_char('R', &makeflags_string_posix);
2264     }
2265 #endif
2266 /*
2267 * Make sure MAKEFLAGS is exported
2268 */
2269 maybe_append_prop(makeflags, macro_prop)-
2270     body.macro.exported = true;
2271
2272 if (makeflags_string.buffer.start[1] != (int) nul_char) {
2273     if (makeflags_string.buffer.start[1] != (int) space_char) {
2274         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2275         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2276                      GETNAME(makeflags_string.buffer.start,
2277                               FIND_LENGTH),
2278                      false);
2279     } else {
2280         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2281         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2282                      GETNAME(makeflags_string.buffer.start + 2,
2283                               FIND_LENGTH),
2284                      false);
2285     }
2286 }
2287
2288 /*
2289 * Add command line macro to POSIX makeflags_string
2290 */
2291 if (makeflags_and_macro.start) {
2292     tmp_char = (char) space_char;
2293     cp = makeflags_and_macro.start;
2294     do {
2295         append_char(tmp_char, &makeflags_string_posix);
2296     } while (*tmp_char == *cp++);
2297     retmem_mb(makeflags_and_macro.start);
2298 }
2299
2300 /*
2301 * Now set the value of MAKEFLAGS macro in accordance
2302 * with current mode.
2303 */
2304 macro = maybe_append_prop(makeflags, macro_prop);
2305 temp = (Boolean) macro->body.macro.read_only;
2306 macro->body.macro.read_only = false;
2307 if(posix || gnu_style) {
2308     makeflags_string_current = &makeflags_string_posix;
2309 } else {
2310     makeflags_string_current = &makeflags_string;
2311 }
2312 if (makeflags_string_current->buffer.start[1] == (int) nul_char) {
2313     makeflags_value_saved =
2314         GETNAME( makeflags_string_current->buffer.start + 1

```

```

2315                                         , FIND_LENGTH
2316                                         );
2317     } else {
2318         if (makeflags_string_current->buffer.start[1] != (int) space_cha
2319             makeflags_value_saved =
2320                 GETNAME( makeflags_string_current->buffer.start
2321                               , FIND_LENGTH
2322                               );
2323     } else {
2324         makeflags_value_saved =
2325             GETNAME( makeflags_string_current->buffer.start
2326                               , FIND_LENGTH
2327                               );
2328     }
2329 }
2330 (void) SETVAR( makeflags
2331                 , makeflags_value_saved
2332                 , false
2333                 );
2334 macro->body.macro.read_only = temp;
2335
2336 /*
2337 */
2338 */
2339 save_do_not_exec_rule = do_not_exec_rule;
2340 do_not_exec_rule = false;
2341 if (read_trace_level > 0) {
2342     trace_reader = true;
2343 }
2344 for (i = 1; i < argc; i++) {
2345     if (argv[i] &&
2346         (argv[i][0] == (int) hyphen_char) &&
2347         (argv[i][1] == 'f') &&
2348         (argv[i][2] == (int) nul_char)) {
2349         argv[i] = NULL; /* Remove -f */
2350     if (i >= argc - 1) {
2351         fatal(catgets(catd, 1, 190, "No filename argument
2352     }
2353     MBSTOWCS(wcs_buffer, argv[++i]);
2354     primary_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2355     (void) read_makefile(primary_makefile, true, true, true);
2356     argv[i] = NULL; /* Remove filename */
2357     makefile_read = true;
2358 } else if (argv[i] &&
2359             (argv[i][0] == (int) hyphen_char) &&
2360             (argv[i][1] == 'c' || argv[i][1] == 'g' || argv[i][1] == 'j'
2361             || argv[i][1] == 'K' || argv[i][1] == 'M' || argv[i][1] == 'm'
2362             || argv[i][1] == 'O' || argv[i][1] == 'o')) &&
2363             (argv[i][2] == (int) nul_char)) {
2364     argv[i] = NULL;
2365     argv[++i] = NULL;
2366 }
2367
2368 */
2369 If no command line "-f" args then look for "makefile", and then for
2370 "Makefile" if "makefile" isn't found.
2371 */
2372 if (!makefile_read) {
2373     (void) read_dir(dot,

```

```

2381
2382     (wchar_t *) NULL,
2383     (Property) NULL,
2384     (wchar_t *) NULL);
2385
2386     if (!posix) {
2387         if (makefile_name->stat.is_file) {
2388             if (Makefile->stat.is_file) {
2389                 warning(catgets(catd, 1, 310, "Both 'makefile' a
2390
2391                 primary_makefile = makefile_name;
2392                 makefile_read = read_makefile(makefile_name,
2393
2394                     false,
2395                     false,
2396                     true);
2397
2398             } else {
2399                 primary_makefile = Makefile;
2400                 makefile_read = read_makefile(Makefile,
2401
2402                     false,
2403                     false,
2404                     true);
2405         }
2406     }
2407
2408     enum sccs_stat save_m_has_sccs = NO_SCCS;
2409     enum sccs_stat save_M_has_sccs = NO_SCCS;
2410
2411     if (makefile_name->stat.is_file) {
2412         if (Makefile->stat.is_file) {
2413             warning(catgets(catd, 1, 191, "Both 'makefile' a
2414
2415         } else {
2416             if (makefile_name->stat.has_sccs == NO_SCCS) {
2417                 primary_makefile = makefile_name;
2418                 makefile_read = read_makefile(makefile_name,
2419
2420                     false,
2421                     false,
2422                     true);
2423
2424         }
2425     }
2426
2427     if (!makefile_read &&
2428         Makefile->stat.is_file) {
2429         if (Makefile->stat.has_sccs == NO_SCCS) {
2430             primary_makefile = Makefile;
2431             makefile_read = read_makefile(Makefile,
2432
2433                     false,
2434                     false,
2435                     true);
2436
2437         } else {
2438             save_M_has_sccs = Makefile->stat.has_sccs;
2439             Makefile->stat.has_sccs = NO_SCCS;
2440             primary_makefile = Makefile;
2441             makefile_read = read_makefile(Makefile,
2442
2443                     false,
2444                     false,
2445                     true);
2446     }

```

```

2447
2448     }
2449     if (!makefile_read &&
2450         makefile_name->stat.is_file) {
2451         makefile_name->stat.has_sccs = save_m_has_sccs;
2452         primary_makefile = makefile_name;
2453         makefile_read = read_makefile(makefile_name,
2454
2455                     false,
2456                     false,
2457                     true);
2458     }
2459     if (!makefile_read &&
2460         Makefile->stat.is_file) {
2461         Makefile->stat.has_sccs = save_M_has_sccs;
2462         primary_makefile = Makefile;
2463         makefile_read = read_makefile(Makefile,
2464
2465                     false,
2466                     false,
2467                     true);
2468     }
2469     do_not_exec_rule = save_do_not_exec_rule;
2470     allrules_read = makefile_read;
2471     trace_reader = false;
2472
2473     /*
2474      * Now get current value of MAKEFLAGS and compare it with
2475      * the saved value we set before reading makefile.
2476      * If they are different then MAKEFLAGS is subsequently set by
2477      * makefile, just leave it there. Otherwise, if make mode
2478      * is changed by using .POSIX target in makefile we need
2479      * to correct MAKEFLAGS value.
2480
2481     Name mf_val = getvar(makeflags);
2482     if( (posix != is_xpg4)
2483        && (!strcmp(mf_val->string_mb, makeflags_value_saved->string_mb)))
2484     {
2485         if (makeflags_string_posix.buffer.start[1] == (int) nul_char) {
2486             (void) SETVAR(makeflags,
2487                           GETNAME(makeflags_string_posix.buffer.star
2488                                   FIND_LENGTH),
2489                           false);
2490         } else {
2491             if (makeflags_string_posix.buffer.start[1] != (int) spac
2492                 (void) SETVAR(makeflags,
2493                               GETNAME(makeflags_string_posix.buf
2494                                   FIND_LENGTH),
2495                           false);
2496         }
2497     }
2498
2499     if (makeflags_string.free_after_use) {
2500         retmem(makeflags_string.buffer.start);
2501     }
2502
2503     if (makeflags_string_posix.free_after_use) {
2504         retmem(makeflags_string_posix.buffer.start);
2505     }
2506     makeflags_string.buffer.start = NULL;
2507     makeflags_string_posix.buffer.start = NULL;
2508
2509
2510
2511

```

```

2513     if (posix) {
2514         /*
2515          * If the user did not redefine the ARFLAGS macro in the
2516          * default makefile (make.rules), then we'd like to
2517          * change the macro value of ARFLAGS to be in accordance
2518          * with "POSIX" requirements.
2519         */
2520         MBSTOWCS(wcs_buffer, NOCATGETS("ARFLAGS"));
2521         name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2522         macro = get_prop(name->prop, macro_prop);
2523         if ((macro != NULL) && /* Maybe (macro == NULL) || ? */
2524             (IS_EQUAL(macro->body.macro.value->string_mb,
2525               NOCATGETS("rv")))) {
2526             MBSTOWCS(wcs_buffer, NOCATGETS("-rv"));
2527             value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2528             (void) SETVAR(name,
2529                           value,
2530                           false);
2531         }
2532     }
2533
2534     if (!posix && !svr4) {
2535         set_sgs_support();
2536     }
2537
2538 /* Make sure KEEP_STATE is in the environment if KEEP_STATE is on.
2539 */
2540
2541     macro = get_prop(keep_state_name->prop, macro_prop);
2542     if ((macro != NULL) &&
2543         macro->body.macro.exported) {
2544         keep_state = true;
2545     }
2546     if (keep_state) {
2547         if (macro == NULL) {
2548             macro = maybe_append_prop(keep_state_name,
2549                                       macro_prop);
2550         }
2551         macro->body.macro.exported = true;
2552         (void) SETVAR(keep_state_name,
2553                       empty_name,
2554                       false);
2555
2556         /*
2557          *      Read state file
2558         */
2559
2560         /* Before we read state, let's make sure we have
2561         ** right state file.
2562         */
2563
2564         /* just in case macro references are used in make_state file
2565         ** name, we better expand them at this stage using expand_value.
2566         */
2567         INIT_STRING_FROM_STACK(dest, destbuffer);
2568         expand_value(make_state, &dest, false);
2569
2570         make_state = GETNAME(dest.buffer.start, FIND_LENGTH);
2571
2572         if(!stat(make_state->string_mb, &make_state_stat)) {
2573             if(!(make_state_stat.st_mode & S_IFREG) ) {
2574                 /* copy the make_state structure to the other
2575                  ** and then let make_state point to the new
2576                  ** one.
2577                 */
2578             memcpy(&state_filename, make_state,sizeof(state_filename))

```

```

2579         state_filename.string_mb = state_file_str_mb;
2580         /* Just a kludge to avoid two slashes back to back */
2581         if((make_state->hash.length == 1)&&
2582             (make_state->string_mb[0] == '/')) {
2583             make_state->hash.length = 0;
2584             make_state->string_mb[0] = '\0';
2585         }
2586         sprintf(state_file_str_mb,NOCATGETS("%s%s"),
2587             make_state->string_mb,NOCATGETS("./.make.state"));
2588         make_state = &state_filename;
2589         /* adjust the length to reflect the appended string */
2590         make_state->hash.length += 12;
2591     } else { /* the file doesn't exist or no permission */
2592         char tmp_path[MAXPATHLEN];
2593         char *slashp;
2594
2595         if (slashp = strrchr(make_state->string_mb, '/')) {
2596             strncpy(tmp_path, make_state->string_mb,
2597                     (slashp - make_state->string_mb));
2598             tmp_path[slashp - make_state->string_mb]=0;
2599             if(strlen(tmp_path)) {
2600                 if(stat(tmp_path, &make_state_stat)) {
2601                     warning(catgets(catd, 1, 192, "directory %s for .KEEP_"));
2602                 }
2603                 if (access(tmp_path, F_OK) != 0) {
2604                     warning(catgets(catd, 1, 193, "can't access dir %s"),t
2605                 }
2606             }
2607         }
2608     }
2609     if (report_dependencies_level != 1) {
2610         Makefile_type _makefile_type_temp = makefile_type;
2611         makefile_type = reading_statefile;
2612         if (read_trace_level > 1) {
2613             trace_reader = true;
2614         }
2615         (void) read_simple_file(make_state,
2616                                 false,
2617                                 false,
2618                                 false,
2619                                 false,
2620                                 false,
2621                                 false,
2622                                 true);
2623         trace_reader = false;
2624         makefile_type = _makefile_type_temp;
2625     }
2626 }
2627 } unchanged_portion_omitted

```