

new/usr/src/cmd/make/Makefile.com

1

579 Wed May 20 11:58:26 2015

new/usr/src/cmd/make/Makefile.com

make: undef for TEAMWARE_MAKE_CMN (defined)

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License ("CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #
```

12 # Copyright 2015, Richard Lowe.

14 MAKE_INCLUDE= \$(SRC)/cmd/make/include

15 MAKE_DEFS= -DSYSV -DINTER -DTEAMWARE_MAKE_CMN

15 \$(RELEASE_BUILD)MAKE_DEFS += -DNDEBUG

16 CFLAGS += \$(CCVERBOSE)

17 CPPFLAGS += -I\$(MAKE_INCLUDE) \$(MAKE_DEFS)

```

*****
94650 Wed May 20 11:58:27 2015
new/usr/src/cmd/make/bin/doname.cc
make: undef for TEAMWARE_MAKE_CMN (defined)
*****
_____unchanged_portion_omitted_____

260 /*
261 * DONE.
262 *
263 * doname(target, do_get, implicit)
264 *
265 * Chases all files the target depends on and builds any that
266 * are out of date. If the target is out of date it is then rebuilt.
267 *
268 * Return value:
269 *
270 *          Indicates if build failed or nt
271 *
272 * Parameters:
273 *     target      Target to build
274 *     do_get      Run sccs get is nessecary
275 *     implicit    doname is trying to find an implicit rule
276 *
277 * Global variables used:
278 *     assign_done True if command line assignmt has happened
279 *     commands_done Preserved for the case that we need local value
280 *     debug_level  Should we trace make's actions?
281 *     default_rule The rule for ".DEFAULT", used as last resort
282 *     empty_name   The Name "", used when looking for single sfx
283 *     keep_state   Indicates that .KEEP_STATE is on
284 *     parallel     True if building in parallel
285 *     recursion_level Used for tracing
286 *     report_dependencies make -P is on
287 */
288 Doname
289 doname(register Name target, register Boolean do_get, register Boolean implicit,
290 {
291     Doname      result = build_dont_know;
292     Chain       out_of_date_list = NULL;
293 #ifdef TEAMWARE_MAKE_CMN
294     Chain       target_group;
295 #endif
296 Property      old_locals = NULL;
297 register Property line;
298 Property      command = NULL;
299 register Dependency dependency;
300 Name          less = NULL;
301 Name          true_target = target;
302 Name          *automatics = NULL;
303 register int  auto_count;
304 Boolean      rechecking_target = false;
305 Boolean      saved_commands_done;
306 Boolean      restart = false;
307 Boolean      save_parallel = parallel;
308 Boolean      doing_subtree = false;
309
310 Boolean      recheck_conditionals = false;
311
312 if (target->state == build_running) {
313     return build_running;
314 }
315 line = get_prop(target->prop, line_prop);
316 #ifdef TEAMWARE_MAKE_CMN
317 if (line != NULL) {
318     /*
319     * If this target is a member of target group and one of the

```

```

316     * other members of the group is running, mark this target
317     * as running.
318     */
319     for (target_group = line->body.line.target_group;
320          target_group != NULL;
321          target_group = target_group->next) {
322         if (is_running(target_group->name)) {
323             target->state = build_running;
324             add_pending(target,
325                 recursion_level,
326                 do_get,
327                 implicit,
328                 false);
329             return build_running;
330         }
331     }
332 }
333 #endif
334 /*
335 * If the target is a constructed one for a "::" target,
336 * we need to consider that.
337 */
338 if (target->has_target_prop) {
339     true_target = get_prop(target->prop,
340         target_prop->body.target.target;
341     if (true_target->colon_splits > 0) {
342         /* Make sure we have a valid time for :: targets */
343         Property time;
344         time = get_prop(true_target->prop, time_prop);
345         if (time != NULL) {
346             true_target->stat.time = time->body.time.time;
347         }
348     }
349 }
350 (void) exists(true_target);
351 /*
352 * If the target has been processed, we don't need to do it again,
353 * unless it depends on conditional macros or a delayed assignment,
354 * or it has been done when KEEP_STATE is on.
355 */
356 if (target->state == build_ok) {
357     if (!(keep_state || (!target->depends_on_conditional && !assign_d
358         return build_ok;
359     } else {
360         recheck_conditionals = true;
361     }
362 }
363 if (target->state == build_subtree) {
364     /* A dynamic macro subtree is being built */
365     target->state = build_dont_know;
366     doing_subtree = true;
367     if (!target->checking_subtree) {
368         /*
369         * This target has been started before and therefore
370         * not all dependencies have to be built.
371         */
372         restart = true;
373     }
374 } else if (target->state == build_pending) {
375     target->state = build_dont_know;
376     restart = true;
377 }
378 #ifdef TEAMWARE_MAKE_CMN
379 } else if (parallel &&
380     keep_state &&

```

```

380         (target->conditional_cnt > 0)) {
381     if (!parallel_ok(target, false)) {
382         add_subtree(target, recursion_level, do_get, implicit);
383         target->state = build_running;
384         return build_running;
385     }
391 #endif
386 */
387 }
388 /*
389  * If KEEP_STATE is on, we have to rebuild the target if the
390  * building of it caused new automatic dependencies to be reported.
391  * This is where we restart the build.
392  */
393 if (line != NULL) {
394     line->body.line.percent = NULL;
395 }
396 recheck_target:
397 /* Init all local variables */
398 result = build_dont_know;
399 out_of_date_list = NULL;
400 command = NULL;
401 less = NULL;
402 auto_count = 0;
403 if (!restart && line != NULL) {
404     /*
405      * If this target has never been built before, mark all
406      * of the dependencies as never built.
407      */
408     for (dependency = line->body.line.dependencies;
409          dependency != NULL;
410          dependency = dependency->next) {
411         dependency->built = false;
412     }
413 }
414 /* Save the set of automatic depes defined for this target */
415 if (keep_state &&
416     (line != NULL) &&
417     (line->body.line.dependencies != NULL)) {
418     Name *p;
419
420     /*
421      * First run thru the dependency list to see how many
422      * autos there are.
423      */
424     for (dependency = line->body.line.dependencies;
425          dependency != NULL;
426          dependency = dependency->next) {
427         if (dependency->automatic && !dependency->stale) {
428             auto_count++;
429         }
430     }
431     /* Create vector to hold the current autos */
432     automatics =
433         (Name *) alloca((int) (auto_count * sizeof (Name)));
434     /* Copy them */
435     for (p = automatics, dependency = line->body.line.dependencies;
436          dependency != NULL;
437          dependency = dependency->next) {
438         if (dependency->automatic && !dependency->stale) {
439             *p++ = dependency->name;
440         }
441     }
442 }
443 if (debug_level > 1) {
444     (void) printf(NOCATGETS("%*sdoname(%s)\n"),

```

```

445         recursion_level,
446         "",
447         target->string_mb);
448     }
449     recursion_level++;
450     /* Avoid infinite loops */
451     if (target->state == build_in_progress) {
452         warning(catgets(catd, 1, 16, "Infinite loop: Target '%s' depends
453             target->string_mb);
454         return build_ok;
455     }
456     target->state = build_in_progress;
457
458     /* Activate conditional macros for the target */
459     if (!target->added_pattern_conditionals) {
460         add_pattern_conditionals(target);
461         target->added_pattern_conditionals = true;
462     }
463     if (target->conditional_cnt > 0) {
464         old_locals = (Property) alloca(target->conditional_cnt *
465             sizeof (Property_rec));
466         set_locals(target, old_locals);
467     }
468
469     /*
470      * after making the call to dynamic_dependencies unconditional we can handle
471      * target names that are same as file name. In this case $$@ in the
472      * dependencies did not mean anything. With this change it expands it
473      * as expected.
474      */
475     if (!target->has_depe_list_expanded)
476     {
477         dynamic_dependencies(target);
478     }
479
480     /*
481      * FIRST SECTION -- GO THROUGH DEPENDENCIES AND COLLECT EXPLICIT
482      * COMMANDS TO RUN
483      */
484     if ((line = get_prop(target->prop, line_prop)) != NULL) {
485         if (check_dependencies(&result,
486             line,
487             do_get,
488             target,
489             true_target,
490             doing_subtree,
491             &out_of_date_list,
492             old_locals,
493             implicit,
494             &command,
495             less,
496             rechecking_target,
497             recheck_conditionals)) {
498             return build_running;
499         }
500     if (line->body.line.query != NULL) {
501         delete_query_chain(line->body.line.query);
502     }
503     line->body.line.query = out_of_date_list;
504 }
505
506 /*
507  * If the target is a :: type, do not try to find the rule for the target,
508  * all actions will be taken by separate branches.
509  * Else, we try to find an implicit rule using various methods,

```

```

511 * we quit as soon as one is found.
512 *
513 * [tolik, 12 Sep 2002] Do not try to find implicit rule for the target
514 * being rechecked - the target is being rechecked means that it already
515 * has explicit dependencies derived from an implicit rule found
516 * in previous step.
517 */
518     if (target->colon_splits == 0 && !rechecking_target) {
519         /* Look for percent matched rule */
520         if ((result == build_dont_know) &&
521             (command == NULL)) {
522             switch (find_percent_rule(
523                 target,
524                 &command,
525                 recheck_conditionals)) {
526             case build_failed:
527                 result = build_failed;
528                 break;
529 #ifdef TEAMWARE_MAKE_CMN
530             case build_running:
531                 target->state = build_running;
532                 add_pending(target,
533                     --recursion_level,
534                     do_get,
535                     implicit,
536                     false);
537                 if (target->conditional_cnt > 0) {
538                     reset_locals(target,
539                         old_locals,
540                         get_prop(target->prop,
541                             conditional_prop),
542                     0);
543                 }
544                 return build_running;
545 #endif
546             case build_ok:
547                 result = build_ok;
548                 break;
549         }
550         /* Look for double suffix rule */
551         if (result == build_dont_know) {
552             Property member;
553
554             if (target->is_member &&
555                 ((member = get_prop(target->prop, member_prop)) !=
556                  NULL)) {
557                 switch (find_ar_suffix_rule(target,
558                     member->body.
559                     member.member,
560                     &command,
561                     recheck_conditionals)) {
562             case build_failed:
563                 result = build_failed;
564                 break;
565 #ifdef TEAMWARE_MAKE_CMN
566             case build_running:
567                 target->state = build_running;
568                 add_pending(target,
569                     --recursion_level,
570                     do_get,
571                     implicit,
572                     false);
573                 if (target->conditional_cnt > 0) {
574                     reset_locals(target,
575                         old_locals,

```

```

574                                     get_prop(target->prop,
575                                     conditional_prop),
576                                     0);
577                                 }
578                                 return build_running;
579 #endif
580                                 default:
581                                     /* ALWAYS bind $$% for old style */
582                                     /* ar rules */
583                                     if (line == NULL) {
584                                         line =
585                                             maybe_append_prop(target,
586                                                 line_prop);
587                                     }
588                                     line->body.line.percent =
589                                         member->body.member.member;
590                                     break;
591                                 } else {
592                                     switch (find_double_suffix_rule(target,
593                                         &command,
594                                         recheck_conditionals)) {
595             case build_failed:
596                 result = build_failed;
597                 break;
598 #ifdef TEAMWARE_MAKE_CMN
599             case build_running:
600                 target->state = build_running;
601                 add_pending(target,
602                     --recursion_level,
603                     do_get,
604                     implicit,
605                     false);
606                 if (target->conditional_cnt > 0) {
607                     reset_locals(target,
608                         old_locals,
609                         get_prop(target->
610                             prop,
611                             conditiona
612                                 0);
613                 }
614                 return build_running;
615 #endif
616             }
617         }
618         /* Look for single suffix rule */
619
620         /* /tolik/
621         * I commented !implicit to fix bug 1247448: Suffix Rules failed when combine wi
622         * This caused problem with SVR4 tilde rules (infinite recursion). So I made som
623         */
624         /* /tolik, 06.21.96/
625         * Regression! See BugId 1255360
626         * If more than one percent rules are defined for the same target then
627         * the behaviour of 'make' with my previous fix may be different from one
628         * of the 'old make'.
629         * The global variable second_pass (maybe it should be an argument to doname())
630         * is intended to avoid this regression. It is set in doname_check().
631         * First, 'make' will work as it worked before. Only when it is
632         * going to say "don't know how to make target" it sets second_pass to true and
633         * run 'doname' again but now trying to use Single Suffix Rules.
634         */
635         if ((result == build_dont_know) && !automatic && (!implicit || s
636             ((line == NULL) ||
637              ((line->body.line.target != NULL) &&

```

```

637         !line->body.line.target->has_regular_dependency))) {
638             switch (find_suffix_rule(target,
639                                     target,
640                                     empty_name,
641                                     &command,
642                                     recheck_conditionals)) {
643                 case build_failed:
644                     result = build_failed;
645                     break;
646 #ifdef TEAMWARE_MAKE_CMN
647                 case build_running:
648                     target->state = build_running;
649                     add_pending(target,
650                                 --recursion_level,
651                                 do_get,
652                                 implicit,
653                                 false);
654                     if (target->conditional_cnt > 0) {
655                         reset_locals(target,
656                                     old_locals,
657                                     get_prop(target->prop,
658                                             conditional_prop),
659                                     0);
660                     }
661                     return build_running;
662 #endif
663             }
664             /* Try to sccs get */
665             if ((command == NULL) &&
666                 (result == build_dont_know) &&
667                 do_get) {
668                 result = sccs_get(target, &command);
669             }
670             /* Use .DEFAULT rule if it is defined. */
671             if ((command == NULL) &&
672                 (result == build_dont_know) &&
673                 (true_target->colons == no_colon) &&
674                 default_rule &&
675                 !implicit) {
676                 /* Make sure we have a line prop */
677                 line = maybe_append_prop(target, line_prop);
678                 command = line;
679                 Boolean out_of_date;
680                 if (true_target->is_member) {
681                     out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
682                                                                line->bo
683                                                                ) else {
684                     out_of_date = (Boolean) OUT_OF_DATE(true_target-
685                                                                line->body.l
686                                                                )
687                 }
688                 if (build_unconditional || out_of_date) {
689                     line->body.line.is_out_of_date = true;
690                     if (debug_level > 0) {
691                         (void) printf(catgets(catd, 1, 17, "%s*B
692                                     recursion_level,
693                                     "",
694                                     true_target->string_mb);
695                     }
696                 }
697                 line->body.line.sccs_command = false;
698                 line->body.line.command_template = default_rule;
699                 line->body.line.target = true_target;
700                 line->body.line.star = NULL;
701                 line->body.line.less = true_target;

```

```

701         line->body.line.percent = NULL;
702     }
703 }
704
705 /* We say "target up to date" if no cmd were executed for the target */
706 if (!target->is_double_colon_parent) {
707     commands_done = false;
708 }
709
710 silent = silent_all;
711 ignore_errors = ignore_errors_all;
712 if (posix)
713 {
714     if (!silent)
715     {
716         silent = (Boolean) target->silent_mode;
717     }
718     if (!ignore_errors)
719     {
720         ignore_errors = (Boolean) target->ignore_error_mode;
721     }
722 }
723
724 int doname_dyntarget = 0;
725 r_command:
726 /* Run commands if any. */
727 if ((command != NULL) &&
728     (command->body.line.command_template != NULL)) {
729     if (result != build_failed) {
730         result = run_command(command,
731                               (Boolean) ((parallel || save_parall
732                                           )
733                                           )
734                               );
735     }
736     switch (result) {
737 #ifdef TEAMWARE_MAKE_CMN
738     case build_running:
739         add_running(target,
740                     true_target,
741                     command,
742                     --recursion_level,
743                     auto_count,
744                     automatics,
745                     do_get,
746                     implicit);
747         target->state = build_running;
748         if ((line = get_prop(target->prop,
749                               line_prop)) != NULL) {
750             if (line->body.line.query != NULL) {
751                 delete_query_chain(line->body.line.query);
752             }
753             line->body.line.query = NULL;
754         }
755         if (target->conditional_cnt > 0) {
756             reset_locals(target,
757                           old_locals,
758                           get_prop(target->prop,
759                                   conditional_prop),
760                           0);
761         }
762         return build_running;
763     case build_serial:
764         add_serial(target,
765                   --recursion_level,
766                   do_get,
767                   implicit);
768         target->state = build_running;
769         line = get_prop(target->prop, line_prop);

```

```

766         if (line != NULL) {
767             if (line->body.line.query != NULL) {
768                 delete_query_chain(line->body.line.query);
769             }
770             line->body.line.query = NULL;
771         }
772         if (target->conditional_cnt > 0) {
773             reset_locals(target,
774                 old_locals,
775                 get_prop(target->prop,
776                     conditional_prop),
777                 0);
778         }
779         return build_running;
795 #endif
780     case build_ok:
781         /* If all went OK set a nice timestamp */
782         if (true_target->stat.time == file_doesnt_exist) {
783             true_target->stat.time = file_max_time;
784         }
785         break;
786     }
787 } else {
788     /*
789     * If no command was found for the target, and it doesn't
790     * exist, and it is mentioned as a target in the makefile,
791     * we say it is extremely new and that it is OK.
792     */
793     if (target->colons != no_colon) {
794         if (true_target->stat.time == file_doesnt_exist) {
795             true_target->stat.time = file_max_time;
796         }
797         result = build_ok;
798     }
799     /*
800     * Trying dynamic targets.
801     */
802     if (!doname_dyntarget) {
803         doname_dyntarget = 1;
804         Name dtarg = find_dyntarget(target);
805         if (dtarg != NULL) {
806             if (!target->has_depe_list_expanded) {
807                 dynamic_dependencies(target);
808             }
809             if ((line = get_prop(target->prop, line_prop)) !
810                 if (check_dependencies(&result,
811                     line,
812                     do_get,
813                     target,
814                     true_target,
815                     doing_subtree,
816                     &out_of_date_list,
817                     old_locals,
818                     implicit,
819                     &command,
820                     less,
821                     rechecking_target,
822                     recheck_condition
823                 )
824                 {
825                     return build_running;
826                 }
827             if (line->body.line.query != NULL) {
828                 delete_query_chain(line->body.li
829             }
830             line->body.line.query = out_of_date_list

```

```

831         goto r_command;
832     }
833     }
834     /*
835     * If the file exists, it is OK that we couldnt figure
836     * out how to build it.
837     */
838     (void) exists(target);
839     if ((target->stat.time != file_doesnt_exist) &&
840         (result == build_dont_know)) {
841         result = build_ok;
842     }
843 }
844
845 /*
846 * Some of the following is duplicated in the function finish_doname.
847 * If anything is changed here, check to see if it needs to be
848 * changed there.
849 */
850 if ((line = get_prop(target->prop, line_prop)) != NULL) {
851     if (line->body.line.query != NULL) {
852         delete_query_chain(line->body.line.query);
853     }
854     line->body.line.query = NULL;
855 }
856 target->state = result;
857 parallel = save_parallel;
858 if (target->conditional_cnt > 0) {
859     reset_locals(target,
860         old_locals,
861         get_prop(target->prop, conditional_prop),
862         0);
863 }
864 recursion_level--;
865 if (target->is_member) {
866     Property member;
867
868     /* Propagate the timestamp from the member file to the member */
869     if ((target->stat.time != file_max_time) &&
870         ((member = get_prop(target->prop, member_prop)) != NULL) &&
871         (exists(member->body.member.member) > file_doesnt_exist)) {
872         target->stat.time =
873             member->body.member.member->stat.time;
874     }
875 }
876 /*
877 * Check if we found any new auto dependencies when we
878 * built the target.
879 */
880 if ((result == build_ok) && check_auto_dependencies(target,
881     auto_count,
882     automatics)) {
883     if (debug_level > 0) {
884         (void) printf(catgets(catd, 1, 18, "%sTarget '%s' acqui
885             recursion_level,
886             ",
887             true_target->string_mb);
888     }
889     rechecking_target = true;
890     saved_commands_done = commands_done;
891     goto recheck_target;
892 }
893
894 if (rechecking_target && !commands_done) {
895     commands_done = saved_commands_done;
896 }

```

```

898     return result;
899 }

901 /*
902  * DONE.
903  *
904  * check_dependencies(result, line, do_get,
905  *                   target, true_target, doing_subtree, out_of_date_tail,
906  *                   old_locals, implicit, command, less, rechecking_target)
907  *
908  * Return value:
909  *             True returned if some dependencies left running
910  *
911  * Parameters:
912  *   result      Pointer to cell we update if build failed
913  *   line        We get the dependencies from here
914  *   do_get      Allow use of sccs get in recursive doname()
915  *   target      The target to chase dependencies for
916  *   true_target The real one for :: and lib(member)
917  *   doing_subtree True if building a conditional macro subtree
918  *   out_of_date_tail Used to set the $? list
919  *   old_locals   Used for resetting the local macros
920  *   implicit     Called when scanning for implicit rules?
921  *   command     Place to stuff command
922  *   less        Set to $< value
923  *
924  * Global variables used:
925  *   command_changed Set if we suspect .make.state needs rewrite
926  *   debug_level     Should we trace actions?
927  *   force           The Name " FORCE", compared against
928  *   recursion_level Used for tracing
929  *   rewrite_statefile Set if .make.state needs rewriting
930  *   wait_name       The Name ".WAIT", compared against
931  */
932 static Boolean
933 #ifdef TEAMWARE_MAKE_CMN
934 check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
935 #else
936 check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
937 #endif
938 {
939     Boolean dependencies_running;
940     register Dependency dependency;
941     Doname dep_result;
942     Boolean dependency_changed = false;
943
944     line->body.line.dependency_time = file_doesnt_exist;
945     if (line->body.line.query != NULL) {
946         delete_query_chain(line->body.line.query);
947     }
948     line->body.line.query = NULL;
949     line->body.line.is_out_of_date = false;
950     dependencies_running = false;
951     /*
952      * Run thru all the dependencies and call doname() recursively
953      * on each of them.
954      */
955     for (dependency = line->body.line.dependencies;
956          dependency != NULL;
957          dependency = dependency->next) {
958         Boolean this_dependency_changed = false;

```

```

959         * We only bother with the autos when rechecking
960         */
961         continue;
962     }
963
964     if (dependency->name == wait_name) {
965         /*
966          * The special target .WAIT means finish all of
967          * the prior dependencies before continuing.
968          */
969         if (dependencies_running) {
970             break;
971         }
972     } else {
973         timestruc_t depe_time = file_doesnt_exist;
974
975         if (true_target->is_member) {
976             depe_time = exists(dependency->name);
977         }
978         if (dependency->built ||
979             (dependency->name->state == build_failed)) {
980             dep_result = (Doname) dependency->name->state;
981         } else {
982             dep_result = doname_check(dependency->name,
983                                     do_get,
984                                     false,
985                                     (Boolean) dependency->
986                                     );
987         }
988         if (true_target->is_member || dependency->name->is_membe
989             /* should compare only secs, cause lib members d
990             if (depe_time.tv_sec != dependency->name->stat.t
991                 this_dependency_changed =
992                 dependency_changed =
993                 true;
994             }
995         } else {
996             if (depe_time != dependency->name->stat.time) {
997                 this_dependency_changed =
998                 dependency_changed =
999                 true;
1000             }
1001         }
1002         dependency->built = true;
1003         switch (dep_result) {
1004             case build_running:
1005                 dependencies_running = true;
1006                 continue;
1007             case build_failed:
1008                 *result = build_failed;
1009                 break;
1010             case build_dont_know:
1011                 /*
1012                  * If make can't figure out how to make a dependency, maybe the dependency
1013                  * is out of date. In this case, we just declare the target out of date
1014                  * and go on. If we really need the dependency, the make'ing of the target
1015                  * will fail. This will only happen for automatic (hidden) dependencies.
1016                  */
1017                 if (!recheck_conditionals) {
1018                     line->body.line.is_out_of_date = true;
1019                 }
1020                 /*
1021                  * Make sure the dependency is not saved
1022                  * in the state file.
1023                  */
1024                 dependency->stale = true;

```

```

1025         rewrite_statefile =
1026             command_changed =
1027             true;
1028         if (debug_level > 0) {
1029             (void) printf(catgets(catd, 1, 19, "Targ
1030                 true_target->string_mb,
1031                 dependency->name->string_mb
1032             );
1033         }
1034         break;
1035     }
1036     if (dependency->name->depends_on_conditional) {
1037         target->depends_on_conditional = true;
1038     }
1039     if (dependency->name == force) {
1040         target->stat.time =
1041             dependency->name->stat.time;
1042     }
1043     /*
1044     * Propagate new timestamp from "member" to
1045     * "lib.a(member)".
1046     */
1047     (void) exists(dependency->name);
1048
1049     /* Collect the timestamp of the youngest dependency */
1050     line->body.line.dependency_time =
1051         MAX(dependency->name->stat.time,
1052             line->body.line.dependency_time);
1053
1054     /* Correction: do not consider nanosecs for members */
1055     if (true_target->is_member || dependency->name->is_member)
1056         line->body.line.dependency_time.tv_nsec = 0;
1057
1058     if (debug_level > 1) {
1059         (void) printf(catgets(catd, 1, 20, "%sDate(%s)=
1060             recursion_level,
1061             "",
1062             dependency->name->string_mb,
1063             time_to_string(dependency->name->
1064                 stat.time));
1065         if (dependency->name->stat.time > line->body.lin
1066             (void) printf(catgets(catd, 1, 21, "%sD
1067                 recursion_level,
1068                 "",
1069                 true_target->string_mb,
1070                 time_to_string(line->body.
1071                     dependency_
1072                 );
1073     }
1074
1075     /* Build the $? list */
1076     if (true_target->is_member) {
1077         if (this_dependency_changed == true) {
1078             true_target->stat.time = dependency->nam
1079             true_target->stat.time.tv_sec--;
1080         }
1081         else {
1082             /* Dina:
1083              * The next statement is commented
1084              * out as a fix for bug #1051032.
1085              * if dependency hasn't changed
1086              * then there's no need to invalidate
1087              * true_target. This statemnt causes
1088              * make to take much longer to process
1089              * an already-built archive. Soren
1090              * said it was a quick fix for some
1091              * problem he doesn't remember.

```

```

1091         true_target->stat.time = file_no_time;
1092         */
1093         (void) exists(true_target);
1094     }
1095     } else {
1096         (void) exists(true_target);
1097     }
1098     Boolean out_of_date;
1099     if (true_target->is_member || dependency->name->is_membe
1100         out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
1101             dependen
1102     ) else {
1103         out_of_date = (Boolean) OUT_OF_DATE(true_target->
1104             dependency->
1105     );
1106     if ((build_unconditional || out_of_date) &&
1107         (dependency->name != force) &&
1108         (dependency->stale == false)) {
1109         *out_of_date_tail = ALLOC(Chain);
1110         if (dependency->name->is_member &&
1111             (get_prop(dependency->name->prop,
1112                 member_prop) != NULL)) {
1113             (*out_of_date_tail)->name =
1114                 get_prop(dependency->name->prop,
1115                     member_prop)->
1116                     body.member.member;
1117         } else {
1118             (*out_of_date_tail)->name =
1119                 dependency->name;
1120         }
1121         (*out_of_date_tail)->next = NULL;
1122         out_of_date_tail = &(*out_of_date_tail)->next;
1123         if (debug_level > 0) {
1124             if (dependency->name->stat.time == file_
1125                 (void) printf(catgets(catd, 1, 2
1126                     recursion_level,
1127                     "",
1128                     true_target->strin
1129                     dependency->name->
1130             );
1131         } else {
1132             (void) printf(catgets(catd, 1, 2
1133                 recursion_level,
1134                 "",
1135                 true_target->strin
1136                 dependency->name->
1137             );
1138         }
1139     }
1140     if (dependency->name == force) {
1141         force->stat.time =
1142             file_max_time;
1143         force->state = build_dont_know;
1144     }
1145     }
1146     #ifdef TEAMWARE_MAKE_CMN
1147     if (dependencies_running) {
1148         if (doing_subtree) {
1149             if (target->conditional_cnt > 0) {
1150                 reset_locals(target,
1151                     old_locals,
1152                     get_prop(target->prop,
1153                         conditional_prop),
1154                     0);
1155             }
1156         }
1157     }
1158     return true;

```



```

1156     } else {
1157         target->state = build_running;
1158         add_pending(target,
1159             --recursion_level,
1160             do_get,
1161             implicit,
1162             false);
1163         if (target->conditional_cnt > 0) {
1164             reset_locals(target,
1165                 old_locals,
1166                 get_prop(target->prop,
1167                     conditional_prop),
1168                 0);
1169         }
1170         return true;
1171     }
1172 }
1194 #endif
1173 /*
1174  * Collect the timestamp of the youngest double colon target
1175  * dependency.
1176  */
1177 if (target->is_double_colon_parent) {
1178     for (dependency = line->body.line.dependencies;
1179         dependency != NULL;
1180         dependency = dependency->next) {
1181         Property tmp_line;
1182
1183         if ((tmp_line = get_prop(dependency->name->prop, line_pr
1184             if(tmp_line->body.line.dependency_time != file_m
1185                 target->stat.time =
1186                 MAX(tmp_line->body.line.dependency_tim
1187                 target->stat.time);
1188         }
1189     }
1190 }
1191 }
1192 if ((true_target->is_member) && (dependency_changed == true)) {
1193     true_target->stat.time = file_no_time;
1194 }
1195 /*
1196  * After scanning all the dependencies, we check the rule
1197  * if we found one.
1198  */
1199 if (line->body.line.command_template != NULL) {
1200     if (line->body.line.command_template_redefined) {
1201         warning(catgets(catd, 1, 24, "Too many rules defined for
1202             target->string_mb);
1203     }
1204     *command = line;
1205     /* Check if the target is out of date */
1206     Boolean out_of_date;
1207     if (true_target->is_member) {
1208         out_of_date = (Boolean) OUT_OF_DATE_SEC(true_target->sta
1209             line->body.line.
1210     } else {
1211         out_of_date = (Boolean) OUT_OF_DATE(true_target->stat.ti
1212             line->body.line.depe
1213     }
1214     if (build_unconditional || out_of_date){
1215         if(!recheck_conditionals) {
1216             line->body.line.is_out_of_date = true;
1217         }
1218     }
1219     line->body.line.sccs_command = false;
1220     line->body.line.target = true_target;

```

```

1221     if(gnu_style) {
1222
1223         // set $< for explicit rule
1224         if(line->body.line.dependencies != NULL) {
1225             less = line->body.line.dependencies->name;
1226         }
1227
1228         // set $* for explicit rule
1229         Name target_body;
1230         Name tt = true_target;
1231         Property member;
1232         register wchar_t *target_end;
1233         register Dependency suffix;
1234         register int suffix_length;
1235         Wstring targ_string;
1236         Wstring suf_string;
1237
1238         if (true_target->is_member &&
1239             ((member = get_prop(target->prop, member_prop)) !=
1240             NULL) ) {
1241             tt = member->body.member.member;
1242         }
1243         targ_string.init(tt);
1244         target_end = targ_string.get_string() + tt->hash.length;
1245         for (suffix = suffixes; suffix != NULL; suffix = suffix-
1246             suffix_length = suffix->name->hash.length;
1247             suf_string.init(suffix->name);
1248             if (tt->hash.length < suffix_length) {
1249                 continue;
1250             } else if (!IS_WEQUALN(suf_string.get_string(),
1251                 (target_end - suffix_length),
1252                 suffix_length)) {
1253                 continue;
1254             }
1255         }
1256         target_body = GETNAME(
1257             targ_string.get_string(),
1258             (int)(tt->hash.length - suffix_length)
1259         );
1260         line->body.line.star = target_body;
1261
1262         // set result = build_ok so that implicit rules are not
1263         if(*result == build_dont_know) {
1264             *result = build_ok;
1265         }
1266     }
1267     if (less != NULL) {
1268         line->body.line.less = less;
1269     }
1270 }
1271
1272     return false;
1273 }

```

unchanged portion omitted

```

1536 /*
1537  * DONE.
1538  *
1539  * run_command(line)
1540  *
1541  * Takes one Cmd_line and runs the commands from it.
1542  *
1543  * Return value:
1544  * Indicates if the command failed or not
1545  *
1546  * Parameters:

```

```

1547 *           line           The command line to run
1548 *
1549 *   Global variables used:
1550 *   commands_done      Set if we do run command
1551 *   current_line       Set to the line we run a command from
1552 *   current_target     Set to the target we run a command for
1553 *   file_number        Used to form temp file name
1554 *   keep_state         Indicates that .KEEP_STATE is on
1555 *   make_state         The Name ".make.state", used to check timestamp
1556 *   parallel          True if currently building in parallel
1557 *   parallel_process_cnt Count of parallel processes running
1558 *   quest             Indicates that make -q is on
1559 *   rewrite_statefile Set if we do run a command
1560 *   sunpro_dependencies The Name "SUNPRO_DEPENDENCIES", set value
1561 *   temp_file_directory Used to form temp file name
1562 *   temp_file_name     Set to the name of the temp file
1563 *   touch             Indicates that make -t is on
1564 */
1565 static Doname
1566 run_command(register Property line, Boolean)
1567 {
1568     register Doname      result = build_ok;
1569     register Boolean     remember_only = false;
1570     register Name       target = line->body.line.target;
1571     wchar_t             *string;
1572     char                tmp_file_path[MAXPATHLEN];
1573
1574     if (!line->body.line.is_out_of_date && target->rechecking_target) {
1575         target->rechecking_target = false;
1576         return build_ok;
1577     }
1578
1579     /*
1580     * Build the command if we know the target is out of date,
1581     * or if we want to check cmd consistency.
1582     */
1583     if (line->body.line.is_out_of_date || keep_state) {
1584         /* Hack for handling conditional macros in DMake. */
1585         if (!line->body.line.dont_rebuild_command_used) {
1586             build_command_strings(target, line);
1587         }
1588     }
1589     /* Never mind */
1590     if (!line->body.line.is_out_of_date) {
1591         return build_ok;
1592     }
1593     /* If quest, then exit(1) because the target is out of date */
1594     if (quest) {
1595         if (posix) {
1596             #ifdef TEAMWARE_MAKE_CMN
1597                 result = execute_parallel(line, true);
1598             #else
1599                 result = execute_serial(line);
1600             #endif
1601         }
1602         /* We actually had to do something this time */
1603         rewrite_statefile = commands_done = true;
1604     }
1605     /*
1606     * If this is an sccs command, we have to do some extra checking
1607     * and possibly complain. If the file can't be gotten because it's
1608     * checked out, we complain and behave as if the command was
1609     * executed eventhough we ignored the command.
1610     */

```

```

1609     if (!touch &&
1610         line->body.line.sccs_command &&
1611         (target->stat.time != file_doesnt_exist) &&
1612         ((target->stat.mode & 0222) != 0)) {
1613         fatal(catgets(catd, 1, 27, "%s is writable so it cannot be sccs
1614             target->string_mb);
1615         target->has_complained = remember_only = true;
1616     }
1617     /*
1618     * If KEEP_STATE is on, we make sure we have the timestamp for
1619     * .make.state. If .make.state changes during the command run,
1620     * we reread .make.state after the command. We also setup the
1621     * environment variable that asks utilities to report dependencies.
1622     */
1623     if (!touch &&
1624         keep_state &&
1625         !remember_only) {
1626         (void) exists(make_state);
1627         if ((strlen(temp_file_directory) == 1) &&
1628             (temp_file_directory[0] == '/')) {
1629             tmp_file_path[0] = '\0';
1630         } else {
1631             strcpy(tmp_file_path, temp_file_directory);
1632         }
1633         sprintf(mbs_buffer,
1634             NOCATGETS("%s/.make.dependency.%08x.%d.%d"),
1635             tmp_file_path,
1636             hostid,
1637             getpid(),
1638             file_number++);
1639         MBSTOWCS(wcs_buffer, mbs_buffer);
1640         Boolean fnd;
1641         temp_file_name = getname_fn(wcs_buffer, FIND_LENGTH, false, &fnd);
1642         temp_file_name->stat.is_file = true;
1643         int len = 2*MAXPATHLEN + strlen(target->string_mb) + 2;
1644         wchar_t *to = string = ALLOC_WC(len);
1645         for (wchar_t *from = wcs_buffer; *from != (int) nul_char; ) {
1646             if (*from == (int) space_char) {
1647                 *to++ = (int) backslash_char;
1648             }
1649             *to++ = *from++;
1650         }
1651         *to++ = (int) space_char;
1652         MBSTOWCS(to, target->string_mb);
1653         Name sprodep_name = getname_fn(string, FIND_LENGTH, false, &fnd);
1654         (void) SETVAR(sunpro_dependencies,
1655             sprodep_name,
1656             false);
1657         retmem(string);
1658     } else {
1659         temp_file_name = NULL;
1660     }
1661
1662     /*
1663     * In case we are interrupted, we need to know what was going on.
1664     */
1665     current_target = target;
1666     /*
1667     * We also need to be able to save an empty command instead of the
1668     * interrupted one in .make.state.
1669     */
1670     current_line = line;
1671     if (remember_only) {
1672         /* Empty block!!! */
1673     } else if (touch) {
1674         result = touch_command(line, target, result);

```

```

1675         if (posix) {
1702 #ifdef TEAMWARE_MAKE_CMN
1676             result = execute_parallel(line, true);
1704 #else
1675             result = execute_serial(line);
1706 #endif
1677     }
1678 } else {
1679     /*
1680     * If this is not a touch run, we need to execute the
1681     * proper command(s) for the target.
1682     */
1713 #ifdef TEAMWARE_MAKE_CMN
1683     if (parallel) {
1684         if (!parallel_ok(target, true)) {
1685             /*
1686             * We are building in parallel, but
1687             * this target must be built in serial.
1688             */
1689             /*
1690             * If nothing else is building,
1691             * do this one, else wait.
1692             */
1693             if (parallel_process_cnt == 0) {
1725 #ifdef TEAMWARE_MAKE_CMN
1694                 result = execute_parallel(line, true, ta
1727 #else
1728                 result = execute_serial(line);
1729 #endif
1695             } else {
1696                 current_target = NULL;
1697                 current_line = NULL;
1698             /*
1699             */
1700             *
1701                 line->body.line.command_used = NULL;
1702                 line->body.line.dont_rebuild_command_use
1703                 return build_serial;
1704             } else {
1705                 result = execute_parallel(line, false);
1706                 switch (result) {
1707                 case build_running:
1708                     return build_running;
1709                 case build_serial:
1710                     if (parallel_process_cnt == 0) {
1746 #ifdef TEAMWARE_MAKE_CMN
1711                         result = execute_parallel(line,
1748 #else
1749                         result = execute_serial(line);
1750 #endif
1712                     } else {
1713                         current_target = NULL;
1714                         current_line = NULL;
1715                         target->parallel = false;
1716                         line->body.line.command_used =
1717                             NULL;
1718                         return build_serial;
1719                     }
1720                 }
1721             } else {
1722         }
1762 #endif
1763 #ifdef TEAMWARE_MAKE_CMN
1723         result = execute_parallel(line, true, target->localhost)
1765 #else
1766         result = execute_serial(line);

```

```

1767 #endif
1768 #ifdef TEAMWARE_MAKE_CMN
1724     }
1770 #endif
1725 }
1726 temp_file_name = NULL;
1727 if (report_dependencies_level == 0){
1728     update_target(line, result);
1729 }
1730 current_target = NULL;
1731 current_line = NULL;
1732 return result;
1733 }

```

unchanged_portion_omitted

```

*****
4576 Wed May 20 11:58:27 2015
new/usr/src/cmd/make/bin/globals.cc
make: undef for TEAMWARE_MAKE_CMN (defined)
*****
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 2004 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  *      globals.cc
28  *
29  *      This declares all global variables
30  */

32 /*
33  * Included files
34  */
35 #include <nl_types.h>
36 #include <mk/defs.h>
37 #include <sys/stat.h>

39 /*
40  * Defined macros
41  */

43 /*
44  * typedefs & structs
45  */

47 /*
48  * Global variables used by make only
49  */
50 FILE          *dependency_report_file;

52 /*
53  * Global variables used by make
54  */
55 Boolean      allrules_read=false;
56 Name        posix_name;
57 Name        svr4_name;
58 Boolean      sdot_target; /* used to identify s.m(/M)akefile */
59 Boolean     all_parallel;
59 Boolean      all_parallel; /* TEAMWARE_MAKE_CMN */
60 Boolean      assign_done;

```

```

61 int foo;
62 Boolean      build_failed_seen;
63 Name        built_last_make_run;
64 Name        c_at;
65 Boolean      cleanup;
66 Boolean      close_report;
67 Boolean      command_changed;
68 Boolean      commands_done;
69 Chain       conditional_targets;
70 Name        conditionals;
71 Boolean      continue_after_error; /* '-k' */
72 Property    current_line;
73 Name        current_make_version;
74 Name        current_target;
75 short       debug_level;
76 Cmd_line    default_rule;
77 Name        default_rule_name;
78 Name        default_target_to_build;
79 Name        dmake_group;
80 Name        dmake_max_jobs;
81 Name        dmake_mode;
82 DMake_mode  dmake_mode_type;
83 Name        dmake_output_mode;
84 DMake_output_mode output_mode = txt1_mode;
85 Name        dmake_odir;
86 Name        dmake_rcfile;
87 Name        done;
88 Name        dot;
89 Name        dot_keep_state;
90 Name        dot_keep_state_file;
91 Name        empty_name;
92 Boolean      fatal_in_progress;
93 int         file_number;
94 #if 0
95 Boolean      filter_stderr; /* '-X' */
96 #endif
97 Name        force;
98 Name        ignore_name;
99 Boolean      ignore_errors; /* '-i' */
100 Boolean      ignore_errors_all; /* '-i' */
101 Name        init;
102 int         job_msg_id;
103 Boolean      keep_state;
104 Name        make_state;
105 #ifdef TEAMWARE_MAKE_CMN
105 timestruc_t make_state_before;
107 #endif
106 Dependency  makefiles_used;
107 Name        makeflags;
108 // Boolean   make_state_locked; // Moved to lib/mksh
109 Name        make_version;
110 char        mbs_buffer2[(MAXPATHLEN * MB_LEN_MAX)];
111 char        *mbs_ptr;
112 char        *mbs_ptr2;
113 int         mtool_msgs_fd;
114 Boolean      depinfo_already_read = false;
115 Boolean      no_action_was_taken = true; /* true if we've not **
116                                           ** run any command */

118 Boolean     no_parallel = false;
120 Boolean      no_parallel = false; /* TEAMWARE_MAKE_CMN */
119 Name        no_parallel_name;
120 Name        not_auto;
121 Boolean     only_parallel;
122 Boolean     parallel;
123 Boolean      only_parallel; /* TEAMWARE_MAKE_CMN */

```

```

124 Boolean parallel; /* TEAMWARE_MAKE_CMN */
123 Name parallel_name;
124 Name localhost_name;
125 int parallel_process_cnt;
126 Percent percent_list;
127 Dyntarget dyntarget_list;
128 Name plus;
129 Name pmake_machinesfile;
130 Name precious;
131 Name primary_makefile;
132 Boolean quest; /* '-q' */
133 short read_trace_level;
134 Boolean reading_dependencies = false;
135 Name recursive_name;
136 int recursion_level;
137 short report_dependencies_level = 0; /* -P */
138 Boolean report_pwd;
139 Boolean rewrite_statefile;
140 Running running_list;
141 char *sccs_dir_path;
142 Name sccs_get_name;
143 Name sccs_get_posix_name;
144 Cmd_line sccs_get_rule;
145 Cmd_line sccs_get_org_rule;
146 Cmd_line sccs_get_posix_rule;
147 Name get_name;
148 Cmd_line get_rule;
149 Name get_posix_name;
150 Cmd_line get_posix_rule;
151 Boolean send_mtool_msgs; /* '-K' */
152 Boolean all_precious;
153 Boolean silent_all; /* '-s' */
154 Boolean report_cwd; /* '-w' */
155 Boolean silent; /* '-s' */
156 Name silent_name;
157 char *stderr_file = NULL;
158 char *stdout_file = NULL;
159 Boolean stdout_stderr_same;
160 Dependency suffixes;
161 Name suffixes_name;
162 Name sunpro_dependencies;
163 Boolean target_variants;
164 const char *tmpdir = NOCATGETS("/tmp");
165 const char *temp_file_directory = NOCATGETS(".");
166 Name temp_file_name;
167 short temp_file_number;
168 time_t timing_start;
169 wchar_t *top_level_target;
170 Boolean touch; /* '-t' */
171 Boolean trace_reader; /* '-D' */
172 Boolean build_unconditional; /* '-u' */
173 pathpt vroot_path = VROOT_DEFAULT;
174 Name wait_name;
175 wchar_t wcs_buffer2[MAXPATHLEN];
176 wchar_t *wcs_ptr;
177 wchar_t *wcs_ptr2;
178 nl_catd catd;
179 long int hostid;

181 /*
182 * File table of contents
183 */

```



```

895         );
896     }
897     if (less == NULL) {
898         less = depe_to_check;
899     }
900 }
901
902 if (depe_to_check == empty_name) {
903     result = build_ok;
904 } else {
905     if (debug_level > 1) {
906         (void) printf(catgets(catd, 1, 2
907             recursion_level,
908             "",
909             depe_to_check->str
910         ));
911
912         pat_rule->being_expanded = true;
913
914         /* suppress message output */
915         int save_debug_level = debug_level;
916         debug_level = 0;
917
918         /* check whether dependency can be built
919         if (dependency_exists(depe_to_check,
920             get_prop(target->prop,
921                 line_prop)))
922         {
923             result = (Doname) depe_to_check-
924         } else {
925             if(actual_doname) {
926                 result = doname(depe_to_
927             } else {
928                 result = target_can_be_b
929             }
930             if(!dep_name_found) {
931                 if(result != build_ok &&
932                     free_name(depe_t
933                 ) else {
934                     store_name(depe_
935                 )
936             }
937         }
938         if(result != build_ok && is_pattern) {
939             rule_maybe_ok = false;
940         }
941
942         /* restore debug_level */
943         debug_level = save_debug_level;
944     }
945
946     if (pat_depe->name->percent) {
947         if (string.free_after_use) {
948             retmem(string.buffer.start);
949         }
950     }
951     /* make can't figure out how to make this depend
952     if (result != build_ok && result != build_runnin
953         pat_rule->being_expanded = false;
954         break;
955     }
956 }
957 } else {
958     result = build_ok;
959 }

```

```

961         /* this pattern rule is the needed one since all dependencies co
962         if (result == build_ok || result == build_running) {
963             break;
964         }
965
966         /* Make does not know how to build some of dependencies from thi
967         But if all "pattern" dependencies can be built, we remember t
968         as a candidate for the case if no other pattern rule found.
969         */
970         if(rule_maybe_ok && rule_candidate == NULL) {
971             rule_candidate = pat_rule;
972         }
973     }
974
975     /* if no pattern matching rule was found, use the remembered candidate
976     or return build_dont_know if there is no candidate.
977     */
978     if (result != build_ok && result != build_running) {
979         if(rule_candidate) {
980             pat_rule = rule_candidate;
981         } else {
982             return build_dont_know;
983         }
984     }
985
986     /* if we are performing only check whether dependency could be built wit
987     return success */
988     if (command == NULL) {
989         if(pat_rule != NULL) {
990             pat_rule->being_expanded = false;
991         }
992         return result;
993     }
994
995     if (debug_level > 1) {
996         (void) printf(catgets(catd, 1, 224, "%sMatched %s:"),
997             recursion_level,
998             "",
999             target->string_mb);
1000
1001     for (pat_depe = pat_rule->dependencies;
1002         pat_depe != NULL;
1003         pat_depe = pat_depe->next) {
1004         if (pat_depe->name->percent) {
1005             INIT_STRING_FROM_STACK(string, string_buf);
1006             construct_string_from_pattern(pat_depe, &percent
1007             depe_to_check = GETNAME(string.buffer.start, FIN
1008         } else {
1009             depe_to_check = pat_depe->name;
1010             if(depe_to_check->dollar) {
1011                 INIT_STRING_FROM_STACK(string, string_bu
1012                 expand_value(depe_to_check, &string, fal
1013                 depe_to_check = GETNAME(string.buffer.st
1014             )
1015         }
1016
1017         if (depe_to_check != empty_name) {
1018             (void) printf(" %s", depe_to_check->string_mb);
1019         }
1020     }
1021
1022     (void) printf(catgets(catd, 1, 225, " from: %s:"),
1023         pat_rule->name->string_mb);
1024
1025     for (pat_depe = pat_rule->dependencies;
1026         pat_depe != NULL;

```

```

1027         pat_depe = pat_depe->next) {
1028             (void) printf(" %s", pat_depe->name->string_mb);
1029         }
1031     (void) printf("\n");
1032 }
1034 if (true_target->colons == no_colon) {
1035     true_target->colons = one_colon;
1036 }
1038 /* create dependency list and target group from matched pattern rule */
1039 create_target_group_and_dependencies_list(target, pat_rule, &percent);
1041 /* save command */
1042 line = get_prop(target->prop, line_prop);
1043 *command = line;
1045 /* free query chain if one exist */
1046 while(line->body.line.query != NULL) {
1047     Chain to_free = line->body.line.query;
1048     line->body.line.query = line->body.line.query->next;
1049     retmem_mb((char *) to_free);
1050 }
1052 if (line->body.line.dependencies != NULL) {
1053     /* build all collected dependencies */
1054     for (depe = line->body.line.dependencies;
1055          depe != NULL;
1056          depe = depe->next) {
1057         actual_doname = true;
1058         result = doname_check(depe->name, true, true, depe->auto
1060         actual_doname = false;
1061         if (result == build_failed) {
1062             pat_rule->being_expanded = false;
1063             return build_failed;
1064         }
1065         if (result == build_running) {
1066             pat_rule->being_expanded = false;
1067             return build_running;
1068         }
1070         if ((depe->name->stat.time > line->body.line.dependency_
1071             (debug_level > 1)) {
1072             (void) printf(catgets(catd, 1, 226, "%sDate(%s)
1073                 recursion_level,
1074                 "",
1075                 depe->name->string_mb,
1076                 time_to_string(depe->name->stat.ti
1077                 true_target->string_mb,
1078                 time_to_string(line->body.line.dep
1079             )
1081         line->body.line.dependency_time =
1082             MAX(line->body.line.dependency_time, depe->name->stat.
1084         /* determine whether this dependency made target out of
1085         Boolean out_of_date;
1086         if (target->is_member || depe->name->is_member) {
1087             out_of_date = (Boolean) OUT_OF_DATE_SEC(target->
1088         } else {
1089             out_of_date = (Boolean) OUT_OF_DATE(target->stat
1090         }
1091         if (build_unconditional || out_of_date) {
1092             if(!rechecking) {

```

```

1093         line->body.line.is_out_of_date = true;
1094     }
1095     add_target_to_chain(depe->name, &(line->body.lin
1097     if (debug_level > 0) {
1098         (void) printf(catgets(catd, 1, 227, "%*s
1099             recursion_level,
1100             "",
1101             true_target->string_mb,
1102             pat_rule->name->string_mb)
1104         for (pat_depe = pat_rule->dependencies;
1105              pat_depe != NULL;
1106              pat_depe = pat_depe->next) {
1107             (void) printf(" %s", pat_depe->n
1108         }
1110         (void) printf(catgets(catd, 1, 228, " be
1111             depe->name->string_mb);
1112     }
1113 } else {
1114     if ((true_target->stat.time <= file_doesnt_exist) ||
1115         (true_target->stat.time < line->body.line.dependency_time))
1116         if(!rechecking) {
1117             line->body.line.is_out_of_date = true;
1118         }
1119         if (debug_level > 0) {
1120             (void) printf(catgets(catd, 1, 229, "%*sBuilding
1121                 recursion_level,
1122                 "",
1123                 true_target->string_mb,
1124                 pat_rule->name->string_mb,
1125                 (target->stat.time > file_doesnt_e
1126                 catgets(catd, 1, 230, "because it
1127                 catgets(catd, 1, 236, "because it
1128         }
1129     }
1130 }
1131 }
1132 }
1134 /* enter explicit rule from percent rule */
1135 Name lmn_target = true_target;
1136 if (true_target->has_long_member_name) {
1137     lmn_target = get_prop(true_target->prop, long_member_name_prop)-
1138 }
1139 line->body.line.sccs_command = false;
1140 line->body.line.target = true_target;
1141 line->body.line.command_template = pat_rule->command_template;
1142 line->body.line.star = GETNAME(percent.buffer.start, FIND_LENGTH);
1143 line->body.line.less = less;
1145 if (lmn_target->parenleft) {
1146     Wstring lmn_string(lmn_target);
1148     wchar_t *left = (wchar_t *) wschr(lmn_string.get_string(), (int)
1149     wchar_t *right = (wchar_t *) wschr(lmn_string.get_string(), (int)
1151     if ((left == NULL) || (right == NULL)) {
1152         line->body.line.percent = NULL;
1153     } else {
1154         line->body.line.percent = GETNAME(left + 1, right - left
1155     }
1156 } else {
1157     line->body.line.percent = NULL;
1158 }

```



```
1159     pat_rule->being_expanded = false;
```

```
1161 #ifdef TEAMWARE_MAKE_CMN
```

```
1162     /*
```

```
1163      * This #ifdef fixes a dmake bug, but introduces bugid 1136156.
```

```
1164     */
```

```
1161     return result;
```

```
1166 #else
```

```
1167     return build_ok;
```

```
1168 #endif
```

```
1162 }
```

```
_____unchanged_portion_omitted_____
```

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

1

```
*****
90998 Wed May 20 11:58:28 2015
new/usr/src/cmd/make/bin/main.cc
make: undef for TEAMWARE_MAKE_CMN (defined)
*****
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 */

26 /*
27  *      main.cc
28  *
29  *      make program main routine plus some helper routines
30  */
31
32 /*
33  * Included files
34  */
35 #if defined(Teamware_MAKE_CMN)
36 #include <avo/intl.h>
37 #endif

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

42 #include <locale.h>          /* setlocale() */
43 #include <mk/defs.h>
44 #include <mkdmsi18n/mksdmsi18n.h> /* libmkdmsi18n_init() */
45 #include <mksh/macro.h>      /* getvar() */
46 #include <mksh/misc.h>       /* getmem(), setup_char_semantics() */

48 #if defined(Teamware_MAKE_CMN)
49 #endif

51 #include <pwd.h>              /* getpwnam() */
52 #include <setjmp.h>
53 #include <signal.h>
54 #include <stdlib.h>
55 #include <sys/errno.h>        /* ENOENT */
56 #include <sys/stat.h>         /* fstat() */
57 #include <fcntl.h>            /* open() */

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

61 #include <sys/types.h>        /* stat() */
```

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

2

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

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

69 // From parallel.cc
70 #if defined(Teamware_MAKE_CMN)
71 #define MAXJOBS_ADJUST_RFE4694000

72 #ifdef MAXJOBS_ADJUST_RFE4694000
73 extern void job_adjust_fini();
74 #endif /* MAXJOBS_ADJUST_RFE4694000 */
75 #endif /* Teamware_MAKE_CMN */

77 /*
78  * Defined macros
79  */
80 #define MAKE_PREFIX           NOCATGETS("/usr")
81 #define LD_SUPPORT_ENV_VAR    NOCATGETS("SGS_SUPPORT_32")
82 #define LD_SUPPORT_ENV_VAR_32 NOCATGETS("SGS_SUPPORT_32")
83 #define LD_SUPPORT_ENV_VAR_64 NOCATGETS("SGS_SUPPORT_64")
84 #define LD_SUPPORT_MAKE_LIB   NOCATGETS("libmakestate.so.1")
85 #define LD_SUPPORT_MAKE_LIB_DIR NOCATGETS("/lib")
86 #define LD_SUPPORT_MAKE_LIB_DIR_64 NOCATGETS("/64")

88 /*
89  * typedefs & structs
90  */

92 /*
93  * Static variables
94  */
95 static char      *argv_zero_string;
96 static Boolean   build_failed_ever_seen;
97 static Boolean   continue_after_error_ever_seen; /* '-k' */
98 static Boolean   dmake_group_specified;         /* '-g' */
99 static Boolean   dmake_max_jobs_specified;      /* '-j' */
100 static Boolean   dmake_mode_specified;          /* '-m' */
101 static Boolean   dmake_add_mode_specified;      /* '-x' */
102 static Boolean   dmake_output_mode_specified;   /* '-x DMAKE_OUTPUT_MODE */
103 static Boolean   dmake_compat_mode_specified;   /* '-x SUN_MAKE_COMPAT_M */
104 static Boolean   dmake_odir_specified;          /* '-o' */
105 static Boolean   dmake_rcfile_specified;        /* '-c' */
106 static Boolean   env_wins;                       /* '-e' */
107 static Boolean   ignore_default_mk;             /* '-r' */
108 static Boolean   list_all_targets;              /* '-T' */
109 static int       mf_argc;
110 static char      **mf_argv;
111 static Dependency_rec not_auto_depen_struct;
112 static Dependency   not_auto_depen = &not_auto_depen_struct;
113 static Boolean      pmake_cap_r_specified;      /* '-R' */
114 static Boolean      pmake_machinesfile_specified; /* '-M' */
115 static Boolean      stop_after_error_ever_seen; /* '-S' */
116 static Boolean      trace_status;               /* '-p' */

118 #ifdef DMAKE_STATISTICS
119 static Boolean      getname_stat = false;
120 #endif

124 #if defined(Teamware_MAKE_CMN)
122 static time_t      start_time;
123 static int         g_argc;
124 static char        **g_argv;
```

```

128 #endif

126 /*
127 * File table of contents
128 */
129     extern "C" void         cleanup_after_exit(void);

135 #ifndef TEAMWARE_MAKE_CMN
131 extern "C" {
132     extern void             dmake_exit_callback(void);
133     extern void             dmake_message_callback(char *);
134 }
140 #endif

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

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

140 static void               append_makeflags_string(Name, String);
141 static void               doalarm(int);
142 static void               enter_argv_values(int , char **, ASCII_Dyn_Array *);
143 static void               make_targets(int, char **, Boolean);
144 static int                parse_command_option(char);
145 static void               read_command_options(int, char **);
146 static void               read_environment(Boolean);
147 static void               read_files_and_state(int, char **);
148 static Boolean            read_makefile(Name, Boolean, Boolean, Boolean);
149 static void               report_recursion(Name);
150 static void               set_sgs_support(void);
151 static void               setup_for_projectdir(void);
152 static void               setup_makeflags_argv(void);
153 static void               report_dir_enter_leave(Boolean entering);

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

157 static const char         verstring[] = "illumos make";
163 #ifndef TEAMWARE_MAKE_CMN
164     static const char     verstring[] = "illumos make";
165 #endif

159 jmp_buf jmpbuffer;
160 extern nl_catd catd;

162 /*
163 *     main(argc, argv)
164 *
165 *     Parameters:
166 *         argc         You know what this is
167 *         argv         You know what this is
168 *
169 *     Static variables used:
170 *         list_all_targets    make -T seen
171 *         trace_status        make -p seen
172 *
173 *     Global variables used:
174 *         debug_level         Should we trace make actions?
175 *         keep_state          Set if .KEEP_STATE seen
176 *         makeflags           The Name "MAKEFLAGS", used to get macro
177 *         remote_command_name Name of remote invocation cmd ("on")
178 *         running_list        List of parallel running processes
179 *         stdout_stderr_same   true if stdout and stderr are the same
180 *         auto_dependencies    The Name "SUNPRO_DEPENDENCIES"
181 *         temp_file_directory  Set to the dir where we create tmp file
182 *         trace_reader         Set to reflect tracing status
183 *         working_on_targets   Set when building user targets
184 */

```

```

185 int
186 main(int argc, char *argv[])
187 {
188     /*
189     * cp is a -> to the value of the MAKEFLAGS env var,
190     * which has to be regular chars.
191     */
192     register char         *cp;
193     char                  make_state_dir[MAXPATHLEN];
194     Boolean                parallel_flag = false;
195     char                  *prognameptr;
196     char                  *slash_ptr;
197     mode_t                um;
198     int                    i;
199 #ifndef TEAMWARE_MAKE_CMN
200     struct itimerval       value;
201     char                   def_dmakerc_path[MAXPATHLEN];
202     Name                   dmake_name, dmake_name2;
203     Name                   dmake_value, dmake_value2;
204     Property               prop, prop2;
205     struct stat            statbuf;
206     int                    statval;
207 #endif
207     struct stat            out_stat, err_stat;
208     hostid = gethostid();
209     bsd_signals();

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

214 #ifndef DMAKE_STATISTICS
215     if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
216         getname_stat = true;
217     }
218 #endif

230 #if defined(TEAMWARE_MAKE_CMN)
220     catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
232 #endif

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

237 #if defined(TEAMWARE_MAKE_CMN)
225 /*
226 * I put libmksdmsil8n_init() under #ifndef because it requires avo_il8n_init()
227 * from avo_util library.
228 */
229     libmksdmsil8n_init();
243 #endif

246 #ifndef TEAMWARE_MAKE_CMN
232     textdomain(NOCATGETS("SUNW_SPRO_MAKE"));
248 #endif /* TEAMWARE_MAKE_CMN */

250 #ifndef TEAMWARE_MAKE_CMN
234     g_argc = argc;
235     g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
236     for (i = 0; i < argc; i++) {
237         g_argv[i] = argv[i];
238     }
239     g_argv[i] = NULL;
257 #endif /* TEAMWARE_MAKE_CMN */

```

```

241  /*
242  * Set argv_zero_string to some form of argv[0] for
243  * recursive MAKE builds.
244  */

246  if (*argv[0] == (int) slash_char) {
247      /* argv[0] starts with a slash */
248      argv_zero_string = strdup(argv[0]);
249  } else if (strchr(argv[0], (int) slash_char) == NULL) {
250      /* argv[0] contains no slashes */
251      argv_zero_string = strdup(argv[0]);
252  } else {
253      /*
254      * argv[0] contains at least one slash,
255      * but doesn't start with a slash
256      */
257      char    *tmp_current_path;
258      char    *tmp_string;

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

271  /*
272  * The following flags are reset if we don't have the
273  * (.nse_depinfo or .make.state) files locked and only set
274  * AFTER the file has been locked. This ensures that if the user
275  * interrupts the program while file_lock() is waiting to lock
276  * the file, the interrupt handler doesn't remove a lock
277  * that doesn't belong to us.
278  */
279  make_state_lockfile = NULL;
280  make_state_locked = false;

283  /*
284  * look for last slash char in the path to look at the binary
285  * name. This is to resolve the hard link and invoke make
286  * in svr4 mode.
287  */

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

```

```

307      if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
308          svr4 = true;
309          posix = false;
310      }

312  /*
313  * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
314  */
315  char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"));
316  if (dmake_compat_mode_var != NULL) {
317      if (0 == strcmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
318          gnu_style = true;
319      }
320      //svr4 = false;
321      //posix = false;
322  }

324  /*
325  * Temporary directory set up.
326  */
327  char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
328  if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
329      strcpy(mbs_buffer, tmpdir_var);
330      for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
331          *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
332          *tmpdir_var = '\0');
333      if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
334          sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
335                mbs_buffer, getpid());
336          int fd = mkstemp(mbs_buffer2);
337          if (fd >= 0) {
338              close(fd);
339              unlink(mbs_buffer2);
340              tmpdir = strdup(mbs_buffer);
341          }
342      }
343  }

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

361  setup_char_semantics();

363  setup_for_projectdir();

365  /*
366  * If running with .KEEP_STATE, curdir will be set with
367  * the connected directory.
368  */
369  (void) atexit(cleanup_after_exit);

371  load_cached_names();

```

```

373 /*
374 *      Set command line flags
375 */
376     setup_makeflags_argv();
377     read_command_options(mf_argc, mf_argv);
378     read_command_options(argc, argv);
379     if (debug_level > 0) {
380         cp = getenv(makeflags->string_mb);
381         (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp);
382     }

384     setup_interrupt(handle_interrupt);

386     read_files_and_state(argc, argv);

406 #ifdef TEAMWARE_MAKE_CMN
388     /*
389     * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
390     */
391     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
392     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
393     prop2 = get_prop(dmake_name2->prop, macro_prop);
394     if (prop2 == NULL) {
395         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
396         output_mode = txt1_mode;
397     } else {
398         dmake_value2 = prop2->body.macro.value;
399         if ((dmake_value2 == NULL) ||
400             (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
401             output_mode = txt1_mode;
402         } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
403             output_mode = txt2_mode;
404         } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1")))
405             output_mode = html1_mode;
406         } else {
407             warning(catgets(catd, 1, 352, "Unsupported value '%s' fo
408                 dmake_value2->string_mb);
409         }
410     }
411     /*
412     * Find the dmake_mode: distributed, parallel, or serial.
413     */
414     if ((!pmake_cap_r_specified) &&
415         (!pmake_machinesfile_specified)) {
416         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
417         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
418         prop2 = get_prop(dmake_name2->prop, macro_prop);
419         if (prop2 == NULL) {
420             /* DMAKE_MODE not defined, default to distributed mode */
421             dmake_mode_type = distributed_mode;
422             no_parallel = false;
423         } else {
424             dmake_value2 = prop2->body.macro.value;
425             if ((dmake_value2 == NULL) ||
426                 (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed"))
427                     dmake_mode_type = distributed_mode;
428                     no_parallel = false;
429                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel
430                     dmake_mode_type = parallel_mode;
431                     no_parallel = false;
432                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
433                     dmake_mode_type = serial_mode;
434                     no_parallel = true;
435                 } else {
436                     fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
437

```

```

438     }

440     if ((!list_all_targets) &&
441         (report_dependencies_level == 0)) {
442         /*
443         * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
444         * They could be defined in the env, in the makefile, or on the
445         * command line.
446         * If neither is defined, and $(HOME)/.dmake.rc does not exist,
447         * then print a message, and default to parallel mode.
448         */
449         if(dmake_mode_type == distributed_mode) {
450             dmake_mode_type = parallel_mode;
451             no_parallel = false;
452         }
453     }
454 }
474 #endif

476 #ifdef TEAMWARE_MAKE_CMN
456     parallel_flag = true;
457     putenv(strdup(NOCATGETS("DMAKE_CHILD=TRUE")));

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

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

493
494 /*
495 *      Enable interrupt handler for alarms
496 */

```

```

497     (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);
499 /*
500 *   Check if make should report
501 */
502 if (getenv(sunpro_dependencies->string_mb) != NULL) {
503     FILE    *report_file;
504
505     report_dependency("");
506     report_file = get_report_file();
507     if ((report_file != NULL) && (report_file != (FILE*)-1)) {
508         (void) fprintf(report_file, "\n");
509     }
510 }
511
512 /*
513 *   Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
514 */
515 if (keep_state) {
516     maybe_append_prop(sunpro_dependencies, macro_prop->
517         body.macro.exported = true;
518 } else {
519     maybe_append_prop(sunpro_dependencies, macro_prop->
520         body.macro.exported = false;
521 }
522
523 working_on_targets = true;
524 if (trace_status) {
525     dump_make_state();
526     fclose(stdout);
527     fclose(stderr);
528     exit_status = 0;
529     exit(0);
530 }
531 if (list_all_targets) {
532     dump_target_list();
533     fclose(stdout);
534     fclose(stderr);
535     exit_status = 0;
536     exit(0);
537 }
538 trace_reader = false;
539
540 /*
541 *   Set temp_file_directory to the directory the .make.state
542 *   file is written to.
543 */
544 if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
545     temp_file_directory = strdup(get_current_path());
546 } else {
547     *slash_ptr = (int) nul_char;
548     (void) strcpy(make_state_dir, make_state->string_mb);
549     *slash_ptr = (int) slash_char;
550     /* when there is only one slash and it's the first
551     ** character, make_state_dir should point to '/'.
552     */
553     if (make_state_dir[0] == '\0') {
554         make_state_dir[0] = '/';
555         make_state_dir[1] = '\0';
556     }
557     if (make_state_dir[0] == (int) slash_char) {
558         temp_file_directory = strdup(make_state_dir);
559     } else {
560         char    tmp_current_path2[MAXPATHLEN];
561
562         (void) sprintf(tmp_current_path2,

```

```

563         "%s/%s",
564         get_current_path(),
565         make_state_dir);
566         temp_file_directory = strdup(tmp_current_path2);
567     }
568 }
569
570 report_dir_enter_leave(true);
571
572 make_targets(argc, argv, parallel_flag);
573
574 report_dir_enter_leave(false);
575
576 if (build_failed_ever_seen) {
577     if (posix) {
578         exit_status = 1;
579     }
580     exit(1);
581 }
582 exit_status = 0;
583 exit(0);
584 /* NOTREACHED */
585 }
586 }
587
588 unchanged_portion_omitted
589 #endif
590
591 parallel = false;
592 /* If we used the SVR4_MAKE, don't build .DONE or .FAILED */
593 if (!getenv(USE_SVR4_MAKE)) {
594     /* Build the target .DONE or .FAILED if we caught an error */
595     if (!quest && !list_all_targets) {
596         Name        failed_name;
597
598         MBSTOWCS(wcs_buffer, NOCATGETS(".FAILED"));
599         failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
600         if ((exit_status != 0) && (failed_name->prop != NULL)) {
601 #ifdef TEAMWARE_MAKE_CMN
602             /*
603              * [tolik] switch DMake to serial mode
604              */
605             dmake_mode_type = serial_mode;
606             no_parallel = true;
607 #endif
608         (void) doname(failed_name, false, true);
609     } else {
610         if (!trace_status) {
611 #ifdef TEAMWARE_MAKE_CMN
612             /*
613              * Switch DMake to serial mode
614              */
615             dmake_mode_type = serial_mode;
616             no_parallel = true;
617 #endif
618         (void) doname(done, false, true);
619     }
620 }
621 }
622
623 /*
624 * Remove the temp file utilities report dependencies thru if it
625 * is still around
626 */
627 if (temp_file_name != NULL) {
628     (void) unlink(temp_file_name->string_mb);
629 }

```



```

823     }
824   }
825   for (rp = running_list; rp != NULL; rp = rp->next) {
826     if (rp->state != build_running) {
827       continue;
828     }
829     if (rp->target->is_member &&
830         ((member = get_prop(rp->target->prop, member_prop)) !=
831          NULL)) {
832       rp->target = member->body.member.library;
833     }
834     if (!do_not_exec_rule &&
835         !touch &&
836         !quest &&
837         !(rp->target->stat.is_precious || all_precious)) {
838
839       rp->target->stat.time = file_no_time;
840       if (exists(rp->target) != file_doesnt_exist) {
841         (void) fprintf(stderr,
842                        "\n*** %s ",
843                        rp->target->string_mb);
844         if (rp->target->stat.is_dir) {
845           (void) fprintf(stderr,
846                          catgets(catd, 1, 171, "no
847                          rp->target->string_mb);
848         } else if (unlink(rp->target->string_mb) == 0) {
849           (void) fprintf(stderr,
850                          catgets(catd, 1, 172, "re
851                          rp->target->string_mb);
852         } else {
853           (void) fprintf(stderr,
854                          catgets(catd, 1, 173, "co
855                          rp->target->string_mb,
856                          errmsg(errno));
857         }
858       }
859     }
860   }
861
862   /* Have we locked .make.state or .nse_depinfo? */
863   if ((make_state_lockfile != NULL) && (make_state_locked)) {
864     unlink(make_state_lockfile);
865     make_state_lockfile = NULL;
866     make_state_locked = false;
867   }
868   /*
869   * Re-read .make.state file (it might be changed by recursive make)
870   */
871   check_state(NULL);
872
873   report_dir_enter_leave(false);
874
875   exit_status = 2;
876   exit(2);
877 }
878 }
879
880 _____unchanged_portion_omitted_____
881
882 899 /*
899 *   read_command_options(argc, argv)
900 *
901 *   Scan the cmd line options and process the ones that start with "--"
902 *
903 *   Return value:
904 *
905 *       -M argument, if any

```

```

906 *
907 *   Parameters:
908 *       argc       You know what this is
909 *       argv       You know what this is
910 *
911 *   Global variables used:
912 */
913 static void
914 read_command_options(register int argc, register char **argv)
915 {
916     register int     ch;
917     int              current_optind = 1;
918     int              last_optind_with_double_hyphen = 0;
919     int              last_optind;
920     int              last_current_optind;
921     register int     i;
922     register int     j;
923     register int     k;
924     register int     makefile_next = 0; /*
925                                     * flag to note options:
926                                     * -c, f, g, j, m, o
927                                     */
928     const char       *tptr;
929     const char       *CMD_OPTS;
930
931     extern char       *optarg;
932     extern int        optind, opterr, optopt;
933
934 #define SUNPRO_CMD_OPTS "--Bbc:Ddef:g:ij:K:kM:m:nNo:o:PpqRrSsTtuVvwx:"
935
936 #ifdef TEAMWARE_MAKE_CMN
937 #define SVR4_CMD_OPTS "-c:ef:g:ij:km:nO:o:pqrsTtVv"
938 #else
939 #define SVR4_CMD_OPTS "-ef:iknpqrstV"
940 #endif
941
942     /*
943     * Added V in SVR4_CMD_OPTS also, which is going to be a hidden
944     * option, just to make sure that the getopt doesn't fail when some
945     * users leave their USE_SVR4_MAKE set and try to use the makefiles
946     * that are designed to issue commands like $(MAKE) -V. Anyway it
947     * sets the same flag but ensures that getopt doesn't fail.
948     */
949
950     opterr = 0;
951     optind = 1;
952     while (1) {
953         last_optind=optind; /* Save optind and curre
954         last_current_optind=current_optind; /* in case we have to re
955         if (svr4) {
956             CMD_OPTS=SVR4_CMD_OPTS;
957             ch = getopt(argc, argv, SVR4_CMD_OPTS);
958         } else {
959             CMD_OPTS=SUNPRO_CMD_OPTS;
960             ch = getopt(argc, argv, SUNPRO_CMD_OPTS);
961         }
962         if (ch == EOF) {
963             if (optind < argc) {
964                 /*
965                 * Fixing bug 4102537:
966                 * Strange behaviour of command make using --
967                 * Not all argv have been processed
968                 * Skip non-flag argv and continue processing.
969                 */
970                 optind++;
971                 current_optind++;

```



```

968         continue;
969     } else {
970         break;
971     }
972 }
973 }
974 if (ch == '?') {
975     if (optopt == '-') {
976         /* Bug 5060758: getopt() changed behavior (sl0_6
977          * and now we have to deal with cases when optio
978          * with double hyphen appear here, from -(MAKEF
979          */
980         i = current_optind;
981         if (argv[i][0] == '-') {
982             if (argv[i][1] == '-') {
983                 if (argv[i][2] != '\0') {
984                     /* Check if this option is allowed */
985                     tptr = strchr(CMD_OPTS, argv[i][2]);
986                     if (tptr) {
987                         if (last_optind_with_double_hyphen != cu
988                             /* This is first time we are trying to
989                              * problem with this option. If we com
990                              * time, we will go to fatal error.
991                              */
992                         last_optind_with_double_hyphen = curre
993
994                         /* Eliminate first hyphen character */
995                         for (j=0; argv[i][j] != '\0'; j++) {
996                             argv[i][j] = argv[i][j+1];
997                         }
998
999                         /* Repeat the processing of this argum
1000                        optind=last_optind;
1001                        current_optind=last_current_optind;
1002                        continue;
1003                    }
1004                }
1005            }
1006        }
1007    }
1008 }
1009 }
1010
1011     if (ch == '?') {
1012         if (svr4) {
1013             #ifdef TEAMWARE_MAKE_CMN
1014                 fprintf(stderr,
1015                     catgets(catd, 1, 267, "Usage : dmake [ -
1016                     fprintf(stderr,
1017                         catgets(catd, 1, 268, "
1018                     fprintf(stderr,
1019                         catgets(catd, 1, 269, "
1020                 #else
1021                     fprintf(stderr,
1022                         catgets(catd, 1, 270, "Usage : make [ -f
1023                     fprintf(stderr,
1024                         catgets(catd, 1, 271, "
1025                 #endif
1026             tptr = strchr(SVR4_CMD_OPTS, optopt);
1027         } else {
1028             #ifdef TEAMWARE_MAKE_CMN
1029                 fprintf(stderr,
1030                     catgets(catd, 1, 272, "Usage : dmake [ -
1031                 fprintf(stderr,
1032                     catgets(catd, 1, 273, "
1033             #endif

```

```

1026         catgets(catd, 1, 274, "
1027     fprintf(stderr,
1028         catgets(catd, 1, 275, "
1029 #else
1030     fprintf(stderr,
1031         catgets(catd, 1, 276, "Usage : make [ -f
1032     fprintf(stderr,
1033         catgets(catd, 1, 277, "
1034     fprintf(stderr,
1035         catgets(catd, 1, 278, "
1036 #endif
1037     tptr = strchr(SUNPRO_CMD_OPTS, optopt);
1038     if (!tptr) {
1039         fatal(catgets(catd, 1, 279, "Unknown option '-%c
1040     } else {
1041         fatal(catgets(catd, 1, 280, "Missing argument af
1042     }
1043 }
1044
1045     makefile_next |= parse_command_option(ch);
1046     /*
1047     * If we're done processing all of the options of
1048     * ONE argument string...
1049     */
1050     if (current_optind < optind) {
1051         i = current_optind;
1052         k = 0;
1053         /* If there's an argument for an option... */
1054         if ((optind - current_optind) > 1) {
1055             k = i + 1;
1056         }
1057         switch (makefile_next) {
1058             case 0:
1059                 argv[i] = NULL;
1060                 /* This shouldn't happen */
1061                 if (k) {
1062                     argv[k] = NULL;
1063                 }
1064                 break;
1065             case 1: /* -f seen */
1066                 argv[i] = (char *)NOCATGETS("-f");
1067                 break;
1068             case 2: /* -c seen */
1069                 argv[i] = (char *)NOCATGETS("-c");
1070                 #ifdef TEAMWARE_MAKE_CMN
1071                 warning(catgets(catd, 1, 281, "Ignoring Distribu
1072                 #endif
1073                 break;
1074             case 4: /* -g seen */
1075                 argv[i] = (char *)NOCATGETS("-g");
1076                 #ifdef TEAMWARE_MAKE_CMN
1077                 warning(catgets(catd, 1, 282, "Ignoring Distribu
1078                 #endif
1079                 break;
1080             case 8: /* -j seen */
1081                 argv[i] = (char *)NOCATGETS("-j");
1082                 #ifdef TEAMWARE_MAKE_CMN
1083                 warning(catgets(catd, 1, 283, "Ignoring Distribu
1084                 #endif
1085                 break;
1086             case 16: /* -M seen */
1087                 argv[i] = (char *)NOCATGETS("-M");
1088                 #ifdef TEAMWARE_MAKE_CMN

```

```

1136 warning(catgets(catd, 1, 284, "Ignoring Parallel
1137 #endif
1074 break;
1075 case 32: /* -m seen */
1076 argv[i] = (char *)NOCATGETS("-m");
1141 #ifndef TEAMWARE_MAKE_CMN
1142 warning(catgets(catd, 1, 285, "Ignoring Distribu
1143 #endif
1077 break;
1078 case 128: /* -O seen */
1079 argv[i] = (char *)NOCATGETS("-O");
1080 break;
1081 case 256: /* -K seen */
1082 argv[i] = (char *)NOCATGETS("-K");
1083 break;
1084 case 512: /* -o seen */
1085 argv[i] = (char *)NOCATGETS("-o");
1153 #ifndef TEAMWARE_MAKE_CMN
1154 warning(catgets(catd, 1, 311, "Ignoring Distribu
1155 #endif
1086 break;
1087 case 1024: /* -x seen */
1088 argv[i] = (char *)NOCATGETS("-x");
1159 #ifndef TEAMWARE_MAKE_CMN
1160 warning(catgets(catd, 1, 353, "Ignoring Distribu
1161 #endif
1089 break;
1090 default: /* > 1 of -c, f, g, j, K, M, m, O, o, x seen */
1091 fatal(catgets(catd, 1, 286, "Illegal command lin
1092 }
1094 makefile_next = 0;
1095 current_optind = optind;
1096 }
1097 }
1098 }

```

unchanged_portion_omitted_

```

1280 /*
1281 * parse_command_option(ch)
1282 *
1283 * Parse make command line options.
1284 *
1285 * Return value:
1286 * Indicates if any -f -c or -M were seen
1287 *
1288 * Parameters:
1289 * ch The character to parse
1290 *
1291 * Static variables used:
1292 * dmake_group_specified Set for make -g
1293 * dmake_max_jobs_specified Set for make -j
1294 * dmake_mode_specified Set for make -m
1295 * dmake_add_mode_specified Set for make -x
1296 * dmake_compat_mode_specified Set for make -x SUN_MAKE_COMPAT_
1297 * dmake_output_mode_specified Set for make -x DMAKE_OUTPUT_MOD
1298 * dmake_odir_specified Set for make -o
1299 * dmake_rcfile_specified Set for make -c
1300 * env_wins Set for make -e
1301 * ignore_default_mk Set for make -r
1302 * trace_status Set for make -p
1303 *
1304 * Global variables used:
1305 * .make.state path & name set for make -K
1306 * continue_after_error Set for make -k
1307 * debug_level Set for make -d

```

```

1308 * do_not_exec_rule Set for make -n
1309 * filter_stderr Set for make -X
1310 * ignore_errors_all Set for make -i
1311 * no_parallel Set for make -R
1312 * quest Set for make -g
1313 * read_trace_level Set for make -D
1314 * report_dependencies Set for make -P
1315 * send_mtool_msgs Set for make -K
1316 * silent_all Set for make -s
1317 * touch Set for make -t
1318 */
1319 static int
1320 parse_command_option(register char ch)
1321 {
1322     static int invert_next = 0;
1323     int invert_this = invert_next;
1325     invert_next = 0;
1326     switch (ch) {
1327     case '-': /* Ignore "--" */
1328         return 0;
1329     case '~': /* Invert next option */
1330         invert_next = 1;
1331         return 0;
1332     case 'B': /* Obsolete */
1333         return 0;
1334     case 'b': /* Obsolete */
1335         return 0;
1336     case 'c': /* Read alternative dmake.rc file */
1337         if (invert_this) {
1338             dmake_rcfile_specified = false;
1339         } else {
1340             dmake_rcfile_specified = true;
1341         }
1342         return 2;
1343     case 'D': /* Show lines read */
1344         if (invert_this) {
1345             read_trace_level--;
1346         } else {
1347             read_trace_level++;
1348         }
1349         return 0;
1350     case 'd': /* Debug flag */
1351         if (invert_this) {
1352             debug_level--;
1353         } else {
1354             debug_level++;
1355         }
1356         return 0;
1357     case 'e': /* Environment override flag */
1358         if (invert_this) {
1359             env_wins = false;
1360         } else {
1361             env_wins = true;
1362         }
1363         return 0;
1364     case 'f': /* Read alternative makefile(s) */
1365         return 1;
1366     case 'g': /* Use alternative DMake group */
1367         if (invert_this) {
1368             dmake_group_specified = false;
1369         } else {
1370             dmake_group_specified = true;
1371         }
1372         return 4;
1373     case 'i': /* Ignore errors */

```

```

1374         if (invert_this) {
1375             ignore_errors_all = false;
1376         } else {
1377             ignore_errors_all = true;
1378         }
1379         return 0;
1380     case 'j': /* Use alternative DMake max jobs */
1381         if (invert_this) {
1382             dmake_max_jobs_specified = false;
1383         } else {
1384             dmake_max_jobs_specified = true;
1385         }
1386         return 8;
1387     case 'K': /* Read alternative .make.state */
1388         return 256;
1389     case 'k': /* Keep making even after errors */
1390         if (invert_this) {
1391             continue_after_error = false;
1392         } else {
1393             continue_after_error = true;
1394             continue_after_error_ever_seen = true;
1395         }
1396         return 0;
1397     case 'M': /* Read alternative make.machines file
1398         if (invert_this) {
1399             pmake_machinesfile_specified = false;
1400         } else {
1401             pmake_machinesfile_specified = true;
1402             dmake_mode_type = parallel_mode;
1403             no_parallel = false;
1404         }
1405         return 16;
1406     case 'm': /* Use alternative DMake build mode */
1407         if (invert_this) {
1408             dmake_mode_specified = false;
1409         } else {
1410             dmake_mode_specified = true;
1411         }
1412         return 32;
1413     case 'x': /* Use alternative DMake mode */
1414         if (invert_this) {
1415             dmake_add_mode_specified = false;
1416         } else {
1417             dmake_add_mode_specified = true;
1418         }
1419         return 1024;
1420     case 'N': /* Reverse -n */
1421         if (invert_this) {
1422             do_not_exec_rule = true;
1423         } else {
1424             do_not_exec_rule = false;
1425         }
1426         return 0;
1427     case 'n': /* Print, not exec commands */
1428         if (invert_this) {
1429             do_not_exec_rule = false;
1430         } else {
1431             do_not_exec_rule = true;
1432         }
1433         return 0;
1434     case 'O': /* Send job start & result msgs */
1435         if (invert_this) {
1436             send_mtool_msgs = false;
1437         } else {
1438             send_mtool_msgs = true;
1439         }
1440         return 128;

```

```

1440     case 'o': /* Use alternative dmake output dir */
1441         if (invert_this) {
1442             dmake_odir_specified = false;
1443         } else {
1444             dmake_odir_specified = true;
1445         }
1446         return 512;
1447     case 'p': /* Print for selected targets */
1448         if (invert_this) {
1449             report_dependencies_level--;
1450         } else {
1451             report_dependencies_level++;
1452         }
1453         return 0;
1454     case 'p': /* Print description */
1455         if (invert_this) {
1456             trace_status = false;
1457             do_not_exec_rule = false;
1458         } else {
1459             trace_status = true;
1460             do_not_exec_rule = true;
1461         }
1462         return 0;
1463     case 'q': /* Question flag */
1464         if (invert_this) {
1465             quest = false;
1466         } else {
1467             quest = true;
1468         }
1469         return 0;
1470     case 'R': /* Don't run in parallel */
1471     #ifdef TEAMWARE_MAKE_CMN
1472         if (invert_this) {
1473             pmake_cap_r_specified = false;
1474             no_parallel = false;
1475         } else {
1476             pmake_cap_r_specified = true;
1477             dmake_mode_type = serial_mode;
1478             no_parallel = true;
1479         }
1480     #else
1481         warning(catgets(catd, 1, 182, "Ignoring ParallelMake -R option")
1482     #endif
1483         return 0;
1484     case 'r': /* Turn off internal rules */
1485         if (invert_this) {
1486             ignore_default_mk = false;
1487         } else {
1488             ignore_default_mk = true;
1489         }
1490         return 0;
1491     case 'S': /* Reverse -k */
1492         if (invert_this) {
1493             continue_after_error = true;
1494         } else {
1495             continue_after_error = false;
1496             stop_after_error_ever_seen = true;
1497         }
1498         return 0;
1499     case 's': /* Silent flag */
1500         if (invert_this) {
1501             silent_all = false;
1502         } else {
1503             silent_all = true;
1504         }
1505         return 0;

```

```

1502     case 'T':                /* Print target list */
1503         if (invert_this) {
1504             list_all_targets = false;
1505             do_not_exec_rule = false;
1506         } else {
1507             list_all_targets = true;
1508             do_not_exec_rule = true;
1509         }
1510         return 0;
1511     case 't':                /* Touch flag */
1512         if (invert_this) {
1513             touch = false;
1514         } else {
1515             touch = true;
1516         }
1517         return 0;
1518     case 'u':                /* Unconditional flag */
1519         if (invert_this) {
1520             build_unconditional = false;
1521         } else {
1522             build_unconditional = true;
1523         }
1524         return 0;
1525     case 'V':                /* SVR4 mode */
1526         svr4 = true;
1527         return 0;
1528     case 'v':                /* Version flag */
1529         if (invert_this) {
1530             } else {
1608 #ifndef TEAMWARE_MAKE_CMN
1531             fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
1532             exit_status = 0;
1533             exit(0);
1612 #else
1613             warning(catgets(catd, 1, 324, "Ignoring DistributedMake
1614 #endif
1534         }
1535         return 0;
1536     case 'w':                /* Unconditional flag */
1537         if (invert_this) {
1538             report_cwd = false;
1539         } else {
1540             report_cwd = true;
1541         }
1542         return 0;
1543 #if 0
1544     case 'X':                /* Filter stdout */
1545         if (invert_this) {
1546             filter_stderr = false;
1547         } else {
1548             filter_stderr = true;
1549         }
1550         return 0;
1551 #endif
1552     default:
1553         break;
1554 }
1555 return 0;
1556 }

```

unchanged_portion_omitted

```

2764 /*
2765 *   make_targets(argc, argv, parallel_flag)
2766 *
2767 *   Call doname on the specified targets
2768 *

```

```

2769 *   Parameters:
2770 *       argc           You know what this is
2771 *       argv           You know what this is
2772 *       parallel_flag  True if building in parallel
2773 *
2774 *   Global variables used:
2775 *       build_failed_seen Used to generated message after failed -k
2776 *       commands_done     Used to generate message "Up to date"
2777 *       default_target_to_build First proper target in makefile
2778 *       init              The Name ".INIT", use to run command
2779 *       parallel          Global parallel building flag
2780 *       quest             make -q, suppresses messages
2781 *       recursion_level   Initialized, used for tracing
2782 *       report_dependencies make -P, regroves whole process
2783 */
2784 static void
2785 make_targets(int argc, char **argv, Boolean parallel_flag)
2786 {
2787     int             i;
2788     char            *cp;
2789     Doname          result;
2790     register Boolean target_to_make_found = false;
2791
2792     (void) doname(init, true, true);
2793     recursion_level = 1;
2794     parallel = parallel_flag;
2795 /*
2796 *   make remaining args
2797 */
2879 #ifndef TEAMWARE_MAKE_CMN
2798 /*
2799     if ((report_dependencies_level == 0) && parallel) {
2800 *
2801     if (parallel) {
2802         /*
2803         * If building targets in parallel, start all of the
2804         * remaining args to build in parallel.
2805         */
2806         for (i = 1; i < argc; i++) {
2807             if ((cp = argv[i]) != NULL) {
2808                 commands_done = false;
2809                 if ((cp[0] == (int) period_char) &&
2810                     (cp[1] == (int) slash_char)) {
2811                     cp += 2;
2812                 }
2813                 if((cp[0] == (int) ' ') &&
2814                     (cp[1] == (int) '-') &&
2815                     (cp[2] == (int) ' ') &&
2816                     (cp[3] == (int) '-')) {
2817                     argv[i] = NULL;
2818                     continue;
2819                 }
2820                 MBSTOWCS(wcs_buffer, cp);
2821                 //default_target_to_build = GETNAME(wcs_buffer,
2822                 //                                     FIND_LENGTH);
2823                 default_target_to_build = normalize_name(wcs_buff
2824                                                         wslen(wcs_buff
2825                 if (default_target_to_build == wait_name) {
2826                     if (parallel_process_cnt > 0) {
2827                         finish_running();
2828                     }
2829                     continue;
2830                 }
2831                 top_level_target = get_wstring(default_target_to
2832 /*
2833         * If we can't execute the current target in

```

```

2834     * parallel, hold off the target processing
2835     * to preserve the order of the targets as they
2836     * in command line.
2837     */
2838     if (!parallel_ok(default_target_to_build, false)
2839         && parallel_process_cnt > 0) {
2840         finish_running();
2841     }
2842     result = doname_check(default_target_to_build,
2843                          true,
2844                          false,
2845                          false);
2846     gather_recursive_deps();
2847     if (/* !commands_done && */
2848         (result == build_ok) &&
2849         !quest &&
2850         (report_dependencies_level == 0) /* &&
2851         (exists(default_target_to_build) > file_does
2852         if (posix) {
2853             if (!commands_done) {
2854                 (void) printf(catgets(ca
2855                                 default_ta
2856             } else {
2857                 if (no_action_was_taken)
2858                     (void) printf(ca
2859                                 de
2860             }
2861         } else {
2862             default_target_to_build->stat.ti
2863             if (!commands_done &&
2864                 (exists(default_target_to_bu
2865                     (void) printf(catgets(ca
2866                                 default_ta
2867             }
2868         }
2869     }
2870 }
2871 }
2872 }
2873 /* Now wait for all of the targets to finish running */
2874 finish_running();
2875 // setjmp(jmpbuffer);
2876 }
2877 }
2878 #endif
2879 for (i = 1; i < argc; i++) {
2880     if ((cp = argv[i]) != NULL) {
2881         target_to_make_found = true;
2882         if ((cp[0] == (int) period_char) &&
2883             (cp[1] == (int) slash_char)) {
2884             cp += 2;
2885         }
2886         if ((cp[0] == (int) ' ') &&
2887             (cp[1] == (int) '-') &&
2888             (cp[2] == (int) ' ') &&
2889             (cp[3] == (int) '-')) {
2890             argv[i] = NULL;
2891             continue;
2892         }
2893         MBSTOWCS(wcs_buffer, cp);
2894         default_target_to_build = normalize_name(wcs_buffer, wsl
2895         top_level_target = get_wstring(default_target_to_build->
2896         report_recursion(default_target_to_build);
2897         commands_done = false;
2898         if (parallel) {
2899             result = (Doname) default_target_to_build->state

```

```

2899     } else {
2900         result = doname_check(default_target_to_build,
2901                              true,
2902                              false,
2903                              false);
2904     }
2905     gather_recursive_deps();
2906     if (build_failed_seen) {
2907         build_failed_ever_seen = true;
2908         warning(catgets(catd, 1, 200, "Target '%s' not r
2909                     default_target_to_build->string_mb);
2910     }
2911     build_failed_seen = false;
2912     if (report_dependencies_level > 0) {
2913         print_dependencies(default_target_to_build,
2914                            get_prop(default_target_to_bu
2915                                    line_prop));
2916     }
2917     default_target_to_build->stat.time =
2918         file_no_time;
2919     if (default_target_to_build->colon_splits > 0) {
2920         default_target_to_build->state =
2921             build_dont_know;
2922     }
2923     if (!parallel &&
2924         /* !commands_done && */
2925         (result == build_ok) &&
2926         !quest &&
2927         (report_dependencies_level == 0) /* &&
2928         (exists(default_target_to_build) > file_doesnt_exist
2929         if (posix) {
2930             if (!commands_done) {
2931                 (void) printf(catgets(catd, 1, 2
2932                                 default_target_to_
2933             } else {
2934                 if (no_action_was_taken) {
2935                     (void) printf(catgets(ca
2936                                 default_ta
2937                 }
2938             }
2939         } else {
2940             if (!commands_done &&
2941                 (exists(default_target_to_build) > f
2942                     (void) printf(catgets(catd, 1, 2
2943                                 default_target_to_
2944             }
2945         }
2946     }
2947 }
2948 }
2949 }
2950 /*
2951 * If no file arguments have been encountered,
2952 * make the first name encountered that doesnt start with a dot
2953 */
2954 if (!target_to_make_found) {
2955     if (default_target_to_build == NULL) {
2956         fatal(catgets(catd, 1, 202, "No arguments to build"));
2957     }
2958     commands_done = false;
2959     top_level_target = get_wstring(default_target_to_build->string_m
2960     report_recursion(default_target_to_build);
2961 }
2962 }
2963 if (getenv(NOCATGETS("SPRO_EXPAND_ERRORS"))){
2964     (void) printf(NOCATGETS("::(%s)\n"),

```

```

2965         default_target_to_build->string_mb);
2966     }

3052 #ifdef TEAMWARE_MAKE_CMN
2969         result = doname_parallel(default_target_to_build, true, false);
3054 #else
3055         result = doname_check(default_target_to_build, true,
3056                             false, false);
3057 #endif
2970     gather_recursive_deps();
2971     if (build_failed_seen) {
2972         build_failed_ever_seen = true;
2973         warning(catgets(catd, 1, 203, "Target '%s' not remade be
2974                 default_target_to_build->string_mb);
2975     }
2976     build_failed_seen = false;
2977     if (report_dependencies_level > 0) {
2978         print_dependencies(default_target_to_build,
2979                           get_prop(default_target_to_build->
2980                                   prop,
2981                                   line_prop));
2982     }
2983     default_target_to_build->stat.time = file_no_time;
2984     if (default_target_to_build->colon_splits > 0) {
2985         default_target_to_build->state = build_dont_know;
2986     }
2987     if (/* !commands_done && */
2988         (result == build_ok) &&
2989         !quest &&
2990         (report_dependencies_level == 0) /* &&
2991         (exists(default_target_to_build) > file_doesnt_exist) */) {
2992         if (posix) {
2993             if (!commands_done) {
2994                 (void) printf(catgets(catd, 1, 299, "%s
2995                                 default_target_to_build->s
2996                                 ) else {
2997                 if (no_action_was_taken) {
2998                     (void) printf(catgets(catd, 1, 3
2999                                 default_target_to_
3000                                 )
3001                 }
3002             } else {
3003                 if (!commands_done &&
3004                     (exists(default_target_to_build) > file_does
3005                     (void) printf(catgets(catd, 1, 301, "%s
3006                                 default_target_to_build->s
3007                                 )
3008                 }
3009             }
3010         }
3011     }
unchanged portion omitted
3208 #endif

3211 static void
3212 report_dir_enter_leave(Boolean entering)
3213 {
3214     char    rcwd[MAXPATHLEN];
3215     static char * mlev = NULL;
3216     static char * make_level_str = NULL;
3217     int     make_level_val = 0;

3219     make_level_str = getenv(NOCATGETS("MAKELEVEL"));
3220     if (make_level_str) {

```

```

3221         make_level_val = atoi(make_level_str);
3222     }
3223     if (mlev == NULL) {
3224         mlev = (char*) malloc(MAXPATHLEN);
3225     }
3226     if (entering) {
3227         sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val + 1);
3228     } else {
3229         make_level_val--;
3230         sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val);
3231     }
3232     putenv(mlev);

3234     if (report_rcwd) {
3235         if (make_level_val <= 0) {
3236             if (entering) {
3237                 #ifdef TEAMWARE_MAKE_CMN
3238                 sprintf( rcwd
3239                         , catgets(catd, 1, 329, "dmake: Entering
3240                         , get_current_path());
3241                 #else
3242                 sprintf( rcwd
3243                         , catgets(catd, 1, 330, "make: Entering d
3244                         , get_current_path());
3245                 #endif
3246             } else {
3247                 #ifdef TEAMWARE_MAKE_CMN
3248                 sprintf( rcwd
3249                         , catgets(catd, 1, 331, "dmake: Leaving d
3250                         , get_current_path());
3251                 #else
3252                 sprintf( rcwd
3253                         , catgets(catd, 1, 332, "make: Leaving di
3254                         , get_current_path());
3255                 #endif
3256             } else {
3257                 #ifdef TEAMWARE_MAKE_CMN
3258                 sprintf( rcwd
3259                         , catgets(catd, 1, 333, "dmake[%d]: Enter
3260                         , make_level_val, get_current_path());
3261                 #else
3262                 sprintf( rcwd
3263                         , catgets(catd, 1, 334, "make[%d]: Enteri
3264                         , make_level_val, get_current_path());
3265                 #endif
3266             } else {
3267                 #ifdef TEAMWARE_MAKE_CMN
3268                 sprintf( rcwd
3269                         , catgets(catd, 1, 335, "dmake[%d]: Leavi
3270                         , make_level_val, get_current_path());
3271                 #else
3272                 sprintf( rcwd
3273                         , catgets(catd, 1, 336, "make[%d]: Leavin
3274                         , make_level_val, get_current_path());
3275                 #endif
3276             }
3277         }
3278         printf(NOCATGETS("%s"), rcwd);
3279     }
unchanged portion omitted

```

```

*****
25535 Wed May 20 11:58:29 2015
new/usr/src/cmd/make/bin/misc.cc
make: undef for TEAMWARE_MAKE_CMN (defined)
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  *      misc.cc
28  *
29  *      This file contains various unclassified routines. Some main groups:
30  *      getname
31  *      Memory allocation
32  *      String handling
33  *      Property handling
34  *      Error message handling
35  *      Make internal state dumping
36  *      main routine support
37  */

39 /*
40  * Included files
41  */
42 #include <errno.h>
43 #include <mk/defs.h>
44 #include <mksh/macro.h>      /* SETVAR() */
45 #include <mksh/misc.h>      /* enable_interrupt() */
46 #include <stdarg.h>         /* va_list, va_start(), va_end() */
47 #include <vroot/report.h>    /* SUNPRO_DEPENDENCIES */

50 #ifdef TEAMWARE_MAKE_CMN
51 #define MAXJOBS_ADJUST_RFE4694000

52 #ifdef MAXJOBS_ADJUST_RFE4694000
53 extern void job_adjust_fini();
54 #endif /* MAXJOBS_ADJUST_RFE4694000 */
55 #endif /* TEAMWARE_MAKE_CMN */

57 /*
58  * Defined macros
59  */

```

```

61 /*
62  * typedefs & structs
63  */

65 /*
66  * Static variables
67  */

69 /*
70  * File table of contents
71  */
72 static void      print_rule(register Name target);
73 static void      print_target_n_deps(register Name target);

75 /*****
76  *
77  *      getname
78  */

80 /*****
81  *
82  *      Memory allocation
83  */

85 /*
86  *      free_chain()
87  *
88  *      frees a chain of Name_vector's
89  *
90  *      Parameters:
91  *          ptr          Pointer to the first element in the chain
92  *                      to be freed.
93  *
94  *      Global variables used:
95  */
96 void
97 free_chain(Name_vector ptr)
98 {
99     if (ptr != NULL) {
100         if (ptr->next != NULL) {
101             free_chain(ptr->next);
102         }
103         free((char *) ptr);
104     }
105 }

107 /*****
108  *
109  *      String manipulation
110  */

112 /*****
113  *
114  *      Nameblock property handling
115  */

117 /*****
118  *
119  *      Error message handling
120  */

122 /*
123  *      fatal(format, args...)
124  *
125  *      Print a message and die

```

```
126 *
127 *   Parameters:
128 *       format      printf type format string
129 *       args        Arguments to match the format
130 *
131 *   Global variables used:
132 *       fatal_in_progress Indicates if this is a recursive call
133 *       parallel_process_cnt Do we need to wait for anything?
134 *       report_pwd      Should we report the current path?
135 */
136 /*VARARGS*/
137 void
138 fatal(const char *message, ...)
139 {
140     va_list args;
141
142     va_start(args, message);
143     (void) fflush(stdout);
144     (void) fprintf(stderr, catgets(catd, 1, 263, "make: Fatal error: "));
145     (void) vfprintf(stderr, message, args);
146     (void) fprintf(stderr, "\n");
147     va_end(args);
148     if (report_pwd) {
149         (void) fprintf(stderr,
150             catgets(catd, 1, 156, "Current working directory
151             get_current_path());
152     }
153     (void) fflush(stderr);
154     if (fatal_in_progress) {
155         exit_status = 1;
156         exit(1);
157     }
158     fatal_in_progress = true;
159 #ifdef TEAMWARE_MAKE_CMN
160     /* Let all parallel children finish */
161     if ((dmake_mode_type == parallel_mode) &&
162         (parallel_process_cnt > 0)) {
163         (void) fprintf(stderr,
164             catgets(catd, 1, 157, "Waiting for %d %s to finis
165             parallel_process_cnt,
166             parallel_process_cnt == 1 ?
167             catgets(catd, 1, 158, "job") : catgets(catd, 1, 1
168         (void) fflush(stderr);
169     }
170     while (parallel_process_cnt > 0) {
171         await_parallel(true);
172         finish_children(false);
173     }
174 #endif
175 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
176     job_adjust_fini();
177 #endif
178
179     exit_status = 1;
180     exit(1);
181 }
182
183 unchanged_portion_omitted
```


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

1

```
*****
46703 Wed May 20 11:58:29 2015
new/usr/src/cmd/make/bin/parallel.cc
make: unifdef for TEAMWARE_MAKE_CMN (defined)
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 #ifdef TEAMWARE_MAKE_CMN

27 /*
28  * parallel.cc
29  *
30  * Deal with the parallel processing
31  */

33 /*
34  * Included files
35  */
36 #include <errno.h>          /* errno */
37 #include <fcntl.h>
38 #include <mk/defs.h>
39 #include <mksh/dosys.h>    /* redirect_io() */
40 #include <mksh/macro.h>    /* expand_value() */
41 #include <mksh/misc.h>     /* getmem() */
42 #include <sys/signal.h>
43 #include <sys/stat.h>
44 #include <sys/types.h>
45 #include <sys/utsname.h>
46 #include <sys/wait.h>
47 #include <unistd.h>
48 #include <netdb.h>

52 /*
53  * Defined macros
54  */
55 #define MAXRULES          100

57 /*
58  * This const should be in avo_dms/include/AvoDmakeCommand.h
59  */
60 const int local_host_mask = 0x20;
```

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

2

```
63 /*
64  * typedefs & structs
65  */

68 /*
69  * Static variables
70  */
72 #ifdef TEAMWARE_MAKE_CMN
71 static Boolean just_did_subtree = false;
72 static char local_host[MAXNAMELEN] = "";
73 static char user_name[MAXNAMELEN] = "";
76 #endif
74 static int pmake_max_jobs = 0;
75 static pid_t process_running = -1;
76 static Running *running_tail = &running_list;
77 static Name subtree_conflict;
78 static Name subtree_conflict2;

81 /*
82  * File table of contents
83  */
84 static void delete_running_struct(Running rp);
85 static Boolean dependency_conflict(Name target);
86 static Doname distribute_process(char **commands, Property line);
87 static void doname_subtree(Name target, Boolean do_get, Boolean impl);
88 static void dump_out_file(char *filename, Boolean err);
89 static void finish_doname(Running rp);
90 static void maybe_reread_make_state(void);
91 static void process_next(void);
92 static void reset_conditionals(int cnt, Name *targets, Property *loc);
93 static pid_t run_rule_commands(char *host, char **commands);
94 static Property *set_conditionals(int cnt, Name *targets);
95 static void store_conditionals(Running rp);

98 /*
99  * execute_parallel(line, waitflg)
100 *
101 * DMake 2.x:
102 * parallel mode: spawns a parallel process to execute the command group.
103 * distributed mode: sends the command group down the pipe to rxm.
104 *
105 * Return value:
106 * The result of the execution
107 *
108 * Parameters:
109 * line The command group to execute
110 */
111 Doname
112 execute_parallel(Property line, Boolean waitflg, Boolean local)
113 {
114     int argcnt;
115     int cmd_options = 0;
116     char *commands[MAXRULES + 5];
117     char *cp;
118     Name dmake_name;
119     Name dmake_value;
120     int ignore;
121     Name make_machines_name;
122     char **p;
123     Property prop;
124     Doname result = build_ok;
```

```

125 Cmd_line      rule;
126 Boolean      silent_flag;
127 Name         target = line->body.line.target;
128 Boolean      wrote_state_file = false;

130 if ((pmake_max_jobs == 0) &&
131     (dmake_mode_type == parallel_mode)) {
132     if (local_host[0] == '\0') {
133         (void) gethostname(local_host, MAXNAMELEN);
134     }
135     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
136     dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
137     if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
138         ((dmake_value = prop->body.macro.value) != NULL)) {
139         pmake_max_jobs = atoi(dmake_value->string_mb);
140         if (pmake_max_jobs <= 0) {
141             warning(catgets(catd, 1, 308, "DMAKE_MAX_JOBS ca
142             warning(catgets(catd, 1, 309, "setting DMAKE_MAX
143             pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
144         } else {
145             /*
146             * For backwards compatibility w/ PMake 1.x, when
147             * DMake 2.x is being run in parallel mode, DMake
148             * should parse the PMake startup file
149             * $(HOME)/.make.machines to get the pmake_max_jobs.
150             */
151             MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
152             dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
153             if (((prop = get_prop(dmake_name->prop, macro_prop)) !=
154                 ((dmake_value = prop->body.macro.value) != NULL)) {
155                 make_machines_name = dmake_value;
156             } else {
157                 make_machines_name = NULL;
158             }
159             if ((pmake_max_jobs = read_make_machines(make_machines_n
160             pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
161         }
162     }
163 }
164
166 if ((dmake_mode_type == serial_mode) ||
167     ((dmake_mode_type == parallel_mode) && (waitflg))) {
168     return (execute_serial(line));
169 }

171 {
172     p = commands;
173 }

175 argcnt = 0;
176 for (rule = line->body.line.command_used;
177     rule != NULL;
178     rule = rule->next) {
179     if (posix && (touch || quest) && !rule->always_exec) {
180         continue;
181     }
182     if (vpath_defined) {
183         rule->command_line =
184         vpath_translation(rule->command_line);
185     }
186     if (dmake_mode_type == distributed_mode) {
187         cmd_options = 0;
188         if (local) {
189             cmd_options |= local_host_mask;
190         }

```

```

191     } else {
192         silent_flag = false;
193         ignore = 0;
194     }
195     if (rule->command_line->hash.length > 0) {
196         if (++argcnt == MAXRULES) {
197             if (dmake_mode_type == distributed_mode) {
198                 /* XXX - tell rxm to execute on local ho
199                 /* I WAS HERE!!! */
200             } else {
201                 /* Too many rules, run serially instead.
202                 return build_serial;
203             }
204         }
205     }
206     if (rule->silent && !silent) {
207         silent_flag = true;
208     }
209     if (rule->ignore_error) {
210         ignore++;
211     }
212     /* XXX - need to add support for + prefix */
213     if (silent_flag || ignore) {
214         *p = getmem((silent_flag ? 1 : 0) +
215                 ignore +
216                 (strlen(rule->
217                     command_line->
218                     string_mb)) +
219                 1);
220         cp = *p++;
221         if (silent_flag) {
222             *cp++ = (int) at_char;
223         }
224         if (ignore) {
225             *cp++ = (int) hyphen_char;
226         }
227         (void) strcpy(cp, rule->command_line->st
228     } else {
229         *p++ = rule->command_line->string_mb;
230     }
231 }
232 }
233
234 if ((argcnt == 0) ||
235     (report_dependencies_level > 0)) {
236     return build_ok;
237 }
238 {
239     *p = NULL;
240 }

241 Doname res = distribute_process(commands, line);
242 if (res == build_running) {
243     parallel_process_cnt++;
244 }

246 /*
247 * Return only those memory that were specially allocated
248 * for part of commands.
249 */
250 for (int i = 0; commands[i] != NULL; i++) {
251     if ((commands[i][0] == (int) at_char) ||
252         (commands[i][0] == (int) hyphen_char)) {
253         retmem_mb(commands[i]);
254     }
255 }
256 return res;

```

```

257     }
258 }

264 #ifdef TEAMWARE_MAKE_CMN
261 #define MAXJOBS_ADJUST_RFE4694000

263 #ifdef MAXJOBS_ADJUST_RFE4694000

265 #include <unistd.h> /* sysconf(_SC_NPROCESSORS_ONLN) */
266 #include <sys/ipc.h> /* ftok() */
267 #include <sys/shm.h> /* shmget(), shmat(), shmdt(), shmctl() */
268 #include <semaphore.h> /* sem_init(), sem_trywait(), sem_post(), sem_de
269 #include <sys/loadavg.h> /* getloadavg() */

271 /*
272 * adjust_pmake_max_jobs (int pmake_max_jobs)
273 *
274 * Parameters:
275 *     pmake_max_jobs - max jobs limit set by user
276 *
277 * External functions used:
278 *     sysconf()
279 *     getloadavg()
280 */
281 static int
282 adjust_pmake_max_jobs (int pmake_max_jobs)
283 {
284     static int ncpu = 0;
285     double loadavg[3];
286     int adjustment;
287     int adjusted_max_jobs;

289     if (ncpu <= 0) {
290         if ((ncpu = sysconf(_SC_NPROCESSORS_ONLN)) <= 0) {
291             ncpu = 1;
292         }
293     }
294     if (getloadavg(loadavg, 3) != 3) return(pmake_max_jobs);
295     adjustment = ((int)loadavg[LOADAVG_LMIN]);
296     if (adjustment < 2) return(pmake_max_jobs);
297     if (ncpu > 1) {
298         adjustment = adjustment / ncpu;
299     }
300     adjusted_max_jobs = pmake_max_jobs - adjustment;
301     if (adjusted_max_jobs < 1) adjusted_max_jobs = 1;
302     return(adjusted_max_jobs);
303 }

unchanged_portion_omitted

549 #endif /* MAXJOBS_ADJUST_RFE4694000 */
554 #endif /* TEAMWARE_MAKE_CMN */

551 /*
552 * distribute_process(char **commands, Property line)
553 *
554 * Parameters:
555 *     commands      argv vector of commands to execute
556 *
557 * Return value:
558 *     The result of the execution
559 *
560 * Static variables used:
561 *     process_running Set to the pid of the process set running
562 * #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
563 *     job_adjust_mode Current job adjust mode

```

```

564 * #endif
565 */
566 static Doname
567 distribute_process(char **commands, Property line)
568 {
569     static unsigned file_number = 0;
570     wchar_t string[MAXPATHLEN];
571     char mbstring[MAXPATHLEN];
572     int filed;
573     int res;
574     int tmp_index;
575     char *tmp_index_str_ptr;

577 #if !defined (TEAMWARE_MAKE_CMN) || !defined (MAXJOBS_ADJUST_RFE4694000)
578     while (parallel_process_cnt >= pmake_max_jobs) {
579         await_parallel(false);
580         finish_children(true);
581     }
582 #else /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
583     /* initialize adjust mode, if not initialized */
584     if (job_adjust_mode == ADJUST_UNKNOWN) {
585         job_adjust_init();
586     }

588     /* actions depend on adjust mode */
589     switch (job_adjust_mode) {
590     case ADJUST_M1:
591         while (parallel_process_cnt >= adjust_pmake_max_jobs (pmake_max_
592             await_parallel(false);
593             finish_children(true);
594         }
595         break;
596     case ADJUST_M2:
597         if ((res = m2_acquire_job()) == 0) {
598             if (parallel_process_cnt > 0) {
599                 await_parallel(false);
600                 finish_children(true);

602                 if ((res = m2_acquire_job()) == 0) {
603                     return build_serial;
604                 }
605             } else {
606                 return build_serial;
607             }
608         }
609         if (res < 0) {
610             /* job adjustment error */
611             job_adjust_error();

613             /* no adjustment */
614             while (parallel_process_cnt >= pmake_max_jobs) {
615                 await_parallel(false);
616                 finish_children(true);
617             }
618         }
619         break;
620     default:
621         while (parallel_process_cnt >= pmake_max_jobs) {
622             await_parallel(false);
623             finish_children(true);
624         }
625     }
626 #endif /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
627     setvar_envvar();
628     /*
629     * Tell the user what DMake is doing.

```

```

630     */
631     if (!silent && output_mode != txt2_mode) {
632         /*
633          * Print local_host --> x job(s).
634          */
635         (void) fprintf(stdout,
636                       catgets(catd, 1, 325, "%s --> %d %s\n"),
637                       local_host,
638                       parallel_process_cnt + 1,
639                       (parallel_process_cnt == 0) ? catgets(catd, 1, 12

641         /* Print command line(s). */
642         tmp_index = 0;
643         while (commands[tmp_index] != NULL) {
644             /* No @ char. */
645             /* XXX - need to add [2] when + prefix is added */
646             if ((commands[tmp_index][0] != (int) at_char) &&
647                 (commands[tmp_index][1] != (int) at_char)) {
648                 tmp_index_str_ptr = commands[tmp_index];
649                 if (*tmp_index_str_ptr == (int) hyphen_char) {
650                     tmp_index_str_ptr++;
651                 }
652                 (void) fprintf(stdout, "%s\n", tmp_index_str_ptr);
653             }
654             tmp_index++;
655         }
656         (void) fflush(stdout);
657     }

659     (void) sprintf(mbstring,
660                  NOCATGETS("%s/dmake.stdout.%d.%d.XXXXXX"),
661                  tmpdir,
662                  getpid(),
663                  file_number++);

665     mktemp(mbstring);

667     stdout_file = strdup(mbstring);
668     stderr_file = NULL;

674 #if defined (TEAMWARE_MAKE_CMN)
670     if (!out_err_same) {
671         (void) sprintf(mbstring,
672                       NOCATGETS("%s/dmake.stderr.%d.%d.XXXXXX"),
673                       tmpdir,
674                       getpid(),
675                       file_number++);

677         mktemp(mbstring);

679         stderr_file = strdup(mbstring);
680     }
686 #endif

682     process_running = run_rule_commands(local_host, commands);

684     return build_running;
685 }
unchanged_portion_omitted

1930 #endif

```

```

*****
11199 Wed May 20 11:58:30 2015
new/usr/src/cmd/make/bin/pmake.cc
make: undef for TEAMWARE_MAKE_CMN (defined)
*****
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 2004 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 #ifdef TEAMWARE_MAKE_CMN

27 /*
28  * Included files
29  */
30 #include <arpa/inet.h>
31 #include <mk/defs.h>
32 #include <mksh/misc.h>
33 #include <netdb.h>
34 #include <netinet/in.h>
35 #include <sys/socket.h>
36 #include <sys/stat.h>
37 #include <sys/types.h>
38 #include <sys/utsname.h>
39 #include <rpc/rpc.h>          /* host2netname(), netname2host() */

41 /*
42  * Defined macros
43  */

45 /*
46  * typedefs & structs
47  */

49 /*
50  * Static variables
51  */

53 /*
54  * File table of contents
55  */
56 static int      get_max(wchar_t **ms_address, wchar_t *hostname);
57 static Boolean  pskip_comment(wchar_t **cp_address);
58 static void     pskip_till_next_word(wchar_t **cp);
59 static Boolean  pskip_white_space(wchar_t **cp_address);

```

```

62 /*
63  *      read_make_machines(Name make_machines_name)
64  *
65  *      For backwards compatibility w/ PMake 1.x, when DMake 2.x is
66  *      being run in parallel mode, DMake should parse the PMake startup
67  *      file $(HOME)/.make.machines to get the PMake max jobs.
68  *
69  *      Return value:
70  *          int of PMake max jobs
71  *
72  *      Parameters:
73  *          make_machines_name      Name of .make.machines file
74  *
75  */
76 int
77 read_make_machines(Name make_machines_name)
78 {
79     wchar_t      c;
80     Boolean      default_make_machines;
81     struct hostent *hp;
82     wchar_t      local_host[MAX_HOSTNAMELEN + 1];
83     char          local_host_mb[MAX_HOSTNAMELEN + 1] = "";
84     int          local_host_wslen;
85     wchar_t      full_host[MAXNETNAMELEN + 1];
86     int          full_host_wslen = 0;
87     char         *homedir;
88     Name         MAKE_MACHINES;
89     struct stat  make_machines_buf;
90     FILE         *make_machines_file;
91     wchar_t      *make_machines_list = NULL;
92     char         *make_machines_list_mb = NULL;
93     wchar_t      make_machines_path[MAXPATHLEN];
94     char         mb_make_machines_path[MAXPATHLEN];
95     *mp;
96     wchar_t      *ms;
97     int          pmake_max_jobs = 0;
98     struct utsname
99     uts_info;

101     MBSTOWCS(wcs_buffer, NOCATGETS("MAKE_MACHINES"));
102     MAKE_MACHINES = GETNAME(wcs_buffer, FIND_LENGTH);
103     /* Did the user specify a .make.machines file on the command line? */
104     default_make_machines = false;
105     if (make_machines_name == NULL) {
106         /* Try reading the default .make.machines file, in $(HOME). */
107         homedir = getenv(NOCATGETS("HOME"));
108         if ((homedir != NULL) && (strlen(homedir) < (sizeof(mb_make_mach
109             sprintf(mb_make_machines_path,
110                 NOCATGETS("%s/.make.machines"), homedir);
111             MBSTOWCS(make_machines_path, mb_make_machines_path);
112             make_machines_name = GETNAME(make_machines_path, FIND_LE
113             default_make_machines = true;
114         }
115         if (make_machines_name == NULL) {
116             /*
117              * No $(HOME)/.make.machines file.
118              * Return 0 for PMake max jobs.
119              */
120             return(0);
121         }
122     }
123 /*
124     make_machines_list_mb = getenv(MAKE_MACHINES->string_mb);
125  */
126     /* Open the .make.machines file. */

```

```

127     if ((make_machines_file = fopen(make_machines_name->string_mb, "r")) ==
128         if (!default_make_machines) {
129             /* Error opening .make.machines file. */
130             fatal(catgets(catd, 1, 314, "Open of %s failed: %s"),
131                 make_machines_name->string_mb,
132                 errmsg(errno));
133         } else {
134             /*
135              * No $(HOME)/.make.machines file.
136              * Return 0 for PMake max jobs.
137              */
138             return(0);
139         }
140     /* Stat the .make.machines file to get the size of the file. */
141     } else if (fstat(fileno(make_machines_file), &make_machines_buf) < 0) {
142         /* Error stat'ing .make.machines file. */
143         fatal(catgets(catd, 1, 315, "Stat of %s failed: %s"),
144             make_machines_name->string_mb,
145             errmsg(errno));
146     } else {
147         /* Allocate memory for "MAKE_MACHINES=<contents of .m.m>" */
148         make_machines_list_mb =
149             (char *) getmem((int) (strlen(MAKE_MACHINES->string_mb) +
150                 2 +
151                 make_machines_buf.st_size));
152         sprintf(make_machines_list_mb,
153             "%s=",
154             MAKE_MACHINES->string_mb);
155         /* Read in the .make.machines file. */
156         if (fread(make_machines_list_mb + strlen(MAKE_MACHINES->string_m
157             sizeof(char),
158             (int) make_machines_buf.st_size,
159             make_machines_file) != make_machines_buf.st_size) {
160             /*
161              * Error reading .make.machines file.
162              * Return 0 for PMake max jobs.
163              */
164             warning(catgets(catd, 1, 316, "Unable to read %s"),
165                 make_machines_name->string_mb);
166             (void) fclose(make_machines_file);
167             retmem_mb((caddr_t) make_machines_list_mb);
168             return(0);
169         } else {
170             (void) fclose(make_machines_file);
171             /* putenv "MAKE_MACHINES=<contents of .m.m>" */
172             *(make_machines_list_mb +
173                 strlen(MAKE_MACHINES->string_mb) +
174                 1 +
175                 make_machines_buf.st_size) = (int) nul_char;
176             if (putenv(make_machines_list_mb) != 0) {
177                 warning(catgets(catd, 1, 317, "Couldn't put cont
178                     make_machines_name->string_mb);
179             } else {
180                 make_machines_list_mb += strlen(MAKE_MACHINES->s
181                     make_machines_list = ALLOC_WC(strlen(make_machin
182                     (void) mbstowcs(make_machines_list,
183                         make_machines_list_mb,
184                         (strlen(make_machines_list_mb) +
185                     )
186             }
187         }
188     }
189     uname(&uts_info);
190     strcpy(local_host_mb, &uts_info.nodename[0]);
191     MBSTOWCS(local_host, local_host_mb);
192     local_host_wslen = wslen(local_host);

```

```

194     // There is no getdomainname() function on Solaris.
195     // And netname2host() function does not work on Linux.
196     // So we have to use different APIs.
197     if (host2netname(mbs_buffer, NULL, NULL) &&
198         netname2host(mbs_buffer, mbs_buffer2, MAXNETNAMELEN+1)) {
199         MBSTOWCS(full_host, mbs_buffer2);
200         full_host_wslen = wslen(full_host);
201     }
202
203     for (ms = make_machines_list;
204         (ms) && (*ms);
205         ) {
206         /*
207          * Skip white space and comments till you reach
208          * a machine name.
209          */
210         pskip_till_next_word(&ms);
211
212         /*
213          * If we haven't reached the end of file, process the
214          * machine name.
215          */
216         if (*ms) {
217             /*
218              * If invalid machine name decrement counter
219              * and skip line.
220              */
221             mp = ms;
222             SKIPWORD(ms);
223             c = *ms;
224             *ms++ = '\0'; /* Append null to machine name. */
225             /*
226              * If this was the beginning of a comment
227              * (we overwrote a # sign) and it's not
228              * end of line yet, shift the # sign.
229              */
230             if ((c == '#' ) && (*ms != '\n') && (*ms)) {
231                 *ms = '#';
232             }
233             WCSTOMBS(mbs_buffer, mp);
234             /*
235              * Print "Ignoring unknown host" if:
236              * 1) hostname is longer than MAX_HOSTNAMELEN, or
237              * 2) hostname is unknown
238              */
239             if ((wslen(mp) > MAX_HOSTNAMELEN) ||
240                 ((hp = gethostbyname(mbs_buffer)) == NULL)) {
241                 warning(catgets(catd, 1, 318, "Ignoring unknown
242                     mbs_buffer);
243                 SKIPTOEND(ms);
244                 /* Increment ptr if not end of file. */
245                 if (*ms) {
246                     ms++;
247                 }
248             } else {
249                 /* Compare current hostname with local_host. */
250                 if (wslen(mp) == local_host_wslen &&
251                     IS_WEQUALN(mp, local_host, local_host_wslen)
252                     /*
253                      * Bingo, local_host is in .make.machine
254                      * Continue reading.
255                      */
256                     pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
257                 /* Compare current hostname with full_host. */
258                 } else if (wslen(mp) == full_host_wslen &&

```

```
259         IS_WEQUALN(mp, full_host, full_host_w
260         /*
261         * Bingo, full_host is in .make.machines
262         * Continue reading.
263         */
264         pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
265     } else {
266         if (c != '\n') {
267             SKIPTOEND(ms);
268             if (*ms) {
269                 ms++;
270             }
271         }
272         continue;
273     }
274     /* If we get here, local_host is in .make.machin
275     if (c != '\n') {
276         /* Now look for keyword 'max'. */
277         MBSTOWCS(wcs_buffer, NOCATGETS("max"));
278         SKIPSPACE(ms);
279         while ((*ms != '\n') && (*ms)) {
280             if (*ms == '#') {
281                 pskip_comment(&ms);
282             } else if (IS_WEQUALN(ms, wcs_bu
283                 /* Skip "max". */
284                 ms += 3;
285                 pmake_max_jobs = get_max
286                 SKIPSPACE(ms);
287             } else {
288                 warning(catgets(catd, 1,
289                 SKIPTOEND(ms);
290                 break;
291             }
292         }
293     }
294     break; /* out of outermost for() loop. */
295 }
296 }
297 }
298 retmem(make_machines_list);
299 return(pmake_max_jobs);
300 }
```

unchanged_portion_omitted

```

*****
14099 Wed May 20 11:58:30 2015
new/usr/src/cmd/make/include/mk/defs.h
make: undef for TEAMWARE_MAKE_CMN (defined)
*****
_____unchanged_portion_omitted_____

108 struct _Running {
109     struct _Running    *next;
110     Doname              state;
111     struct _Name       *target;
112     struct _Name       *true_target;
113     struct _Property   *command;
114     struct _Name       *sprodep_value;
115     char                *sprodep_env;
116     int                recursion_level;
117     Boolean            do_get;
118     Boolean            implicit;
119     Boolean            redo;
120     int                auto_count;
121     struct _Name       **automatics;
122     pid_t              pid;
123 #ifdef TEAMWARE_MAKE_CMN
123     int                job_msg_id;
125 #else
126     int                host;
127 #endif
124     char                *stdout_file;
125     char                *stderr_file;
126     struct _Name       *temp_file;
127     int                conditional_cnt;
128     struct _Name       **conditional_targets;
133 #ifdef TEAMWARE_MAKE_CMN
129     Boolean            make_refd;
135 #endif
130 };
_____unchanged_portion_omitted_____

163 /*
164 * Typedefs for all structs
165 */
166 typedef struct _Cmd_line    *Cmd_line, Cmd_line_rec;
167 typedef struct _Dependency *Dependency, Dependency_rec;
168 typedef struct _Macro      *Macro, Macro_rec;
169 typedef struct _Name_vector *Name_vector, Name_vector_rec;
170 typedef struct _Percent    *Percent, Percent_rec;
171 typedef struct _Dyntarget  *Dyntarget;
172 typedef struct _Recursive_make *Recursive_make, Recursive_make_rec;
173 typedef struct _Running    *Running, Running_rec;

176 /*
177 * extern declarations for all global variables.
178 * The actual declarations are in globals.cc
179 */
180 extern Boolean    allrules_read;
181 extern Name      posix_name;
182 extern Name      svr4_name;
183 extern Boolean   sdot_target;
184 extern Boolean   all_parallel;
185 extern Boolean   assign_done;
186 extern Boolean   build_failed_seen;
187 extern Name      built_last_make_run;
188 extern Name      c_at;
189 extern Boolean   command_changed;

```

```

190 extern Boolean    commands_done;
191 extern Chain      conditional_targets;
192 extern Name       conditionals;
193 extern Boolean    continue_after_error;
194 extern Property   current_line;
195 extern Name       current_make_version;
196 extern Name       current_target;
197 extern short     debug_level;
198 extern Cmd_line   default_rule;
199 extern Name       default_rule_name;
200 extern Name       default_target_to_build;
201 extern Boolean    depinfo_already_read;
202 extern Name       dmake_group;
203 extern Name       dmake_max_jobs;
204 extern Name       dmake_mode;
205 extern DMake_mode dmake_mode_type;
206 extern Name       dmake_output_mode;
207 extern DMake_output_mode output_mode;
208 extern Name       dmake_odir;
209 extern Name       dmake_rcfile;
210 extern Name       done;
211 extern Name       dot;
212 extern Name       dot_keep_state;
213 extern Name       dot_keep_state_file;
214 extern Name       empty_name;
215 extern Boolean    fatal_in_progress;
216 extern int       file_number;
217 extern Name       force;
218 extern Name       ignore_name;
219 extern Boolean    ignore_errors;
220 extern Boolean    ignore_errors_all;
221 extern Name       init;
222 extern int       job_msg_id;
223 extern Boolean    keep_state;
224 extern Name       make_state;
231 #ifdef TEAMWARE_MAKE_CMN
225 extern timestruc_t make_state_before;
233 #endif
226 extern Boolean    make_state_locked;
227 extern Dependency makefiles_used;
228 extern Name       makeflags;
229 extern Name       make_version;
230 extern char       mbs_buffer2[];
231 extern char       *mbs_ptr;
232 extern char       *mbs_ptr2;
233 extern Boolean    no_action_was_taken;
234 extern int       mtool_msgs_fd;
235 extern Boolean    no_parallel;
236 extern Name       no_parallel_name;
237 extern Name       not_auto;
238 extern Boolean    only_parallel;
239 extern Boolean    parallel;
240 extern Name       parallel_name;
241 extern Name       localhost_name;
242 extern int       parallel_process_cnt;
243 extern Percent   percent_list;
244 extern Dyntarget dyntarget_list;
245 extern Name       plus;
246 extern Name       pmake_machinesfile;
247 extern Name       precious;
248 extern Name       primary_makefile;
249 extern Boolean    quest;
250 extern short     read_trace_level;
251 extern Boolean    reading_dependencies;
252 extern int       recursion_level;
253 extern Name       recursive_name;

```



```

254 extern short      report_dependencies_level;
255 extern Boolean    report_pwd;
256 extern Boolean    rewrite_statefile;
257 extern Running    running_list;
258 extern char       *sccs_dir_path;
259 extern Name       sccs_get_name;
260 extern Name       sccs_get_posix_name;
261 extern Cmd_line   sccs_get_rule;
262 extern Cmd_line   sccs_get_org_rule;
263 extern Cmd_line   sccs_get_posix_rule;
264 extern Name       get_name;
265 extern Name       get_posix_name;
266 extern Cmd_line   get_rule;
267 extern Cmd_line   get_posix_rule;
268 extern Boolean    send_mtool_msgs;
269 extern Boolean    all_precious;
270 extern Boolean    report_cwd;
271 extern Boolean    silent_all;
272 extern Boolean    silent;
273 extern Name       silent_name;
274 extern char       *stderr_file;
275 extern char       *stdout_file;
276 extern Boolean    stdout_stderr_same;
277 extern Dependency suffixes;
278 extern Name       suffixes_name;
279 extern Name       sunpro_dependencies;
280 extern Boolean    target_variants;
281 extern const char *tmpdir;
282 extern const char *temp_file_directory;
283 extern Name       temp_file_name;
284 extern short      temp_file_number;
285 extern wchar_t    *top_level_target;
286 extern Boolean    touch;
287 extern Boolean    trace_reader;
288 extern Boolean    build_unconditional;
289 extern pathpt     vroot_path;
290 extern Name       wait_name;
291 extern wchar_t    wcs_buffer2[];
292 extern wchar_t    *wcs_ptr;
293 extern wchar_t    *wcs_ptr2;
294 extern nl_catd     catd;
295 extern long int   hostid;

297 /*
298  * Declarations of system defined variables
299  */
300 /* On linux this variable is defined in 'signal.h' */
301 extern char        *sys_siglist[];

303 /*
304  * Declarations of system supplied functions
305  */
306 extern int         file_lock(char *, char *, int *, int);

308 /*
309  * Declarations of functions declared and used by make
310  */
311 extern void        add_pending(Name target, int recursion_level, Boolean do
312 extern void        add_running(Name target, Name true_target, Property comm
313 extern void        add_serial(Name target, int recursion_level, Boolean do_
314 extern void        add_subtree(Name target, int recursion_level, Boolean do
315 extern void        append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar
316 extern void        await_parallel(Boolean waitflg);
317 extern void        build_suffix_list(Name target_suffix);

```

```

318 extern Boolean    check_auto_dependencies(Name target, int auto_count, Nam
319 extern void       check_state(Name temp_file_name);
320 extern void       cond_macros_into_string(Name np, String_rec *buffer);
321 extern void       construct_target_string();
322 extern void       create_xdrs_ptr(void);
323 extern void       depvar_add_to_list(Name name, Boolean cmdline);
324 extern Doname     doname(register Name target, register Boolean do_get, re
325 extern Doname     doname_check(register Name target, register Boolean do_g
326 extern Doname     doname_parallel(Name target, Boolean do_get, Boolean imp
327 extern Doname     dosys(register Name command, register Boolean ignore_err
328 extern void       dump_make_state(void);
329 extern void       dump_target_list(void);
330 extern void       enter_conditional(register Name target, Name name, Name
331 extern void       enter_dependencies(register Name target, Chain target_gr
332 extern void       enter_dependency(Property line, register Name depe, Bool
333 extern void       enter_equal(Name name, Name value, register Boolean appe
334 extern Percent    enter_percent(register Name target, Chain target_group,
335 extern Dyntaxet   enter_dyntarget(register Name target);
336 extern Name_vector enter_name(String string, Boolean tail_present, register
337 extern Boolean    exec_vp(register char *name, register char **argv, char
338 extern Doname     execute_parallel(Property line, Boolean waitflg, Boolean
339 extern Doname     execute_serial(Property line);
340 extern timestruc_t& exists(register Name target);
341 extern void       fatal(const char *, ...);
342 extern void       fatal_reader(char *, ...);
343 extern Doname     find_ar_suffix_rule(register Name target, Name true_targ
344 extern Doname     find_double_suffix_rule(register Name target, Property *
345 extern Doname     find_percent_rule(register Name target, Property *comman
346 extern int        find_run_directory(char *cmd, char *cwd, char *dir, cha
347 extern Doname     find_suffix_rule(Name target, Name target_body, Name tar
348 extern Chain      find_target_groups(register Name_vector target_list, reg
349 extern void       finish_children(Boolean docheck);
350 extern void       finish_running(void);
351 extern void       free_chain(Name_vector ptr);
352 extern void       gather_recursive_deps(void);
353 extern char       *get_current_path(void);
354 extern int        get_job_msg_id(void);
355 extern FILE       *get_mtool_msgs_fp(void);
356 extern wchar_t    *getmem_wc(register int size);
357 /* On linux getwd(char *) is defined in 'unistd.h' */
358 #ifdef __cplusplus
359 extern "C" {
360 #endif
361 extern char       *getwd(char *);
362 #ifdef __cplusplus
363 }

```

unchanged portion omitted

```

*****
22598 Wed May 20 11:58:32 2015
new/usr/src/cmd/make/include/mksh/defs.h
make: unifdef for TEAMWARE_MAKE_CMN (defined)
*****
1 #ifndef _MKSH_DEFS_H
2 #define _MKSH_DEFS_H
3 /*
4  * CDDL HEADER START
5  *
6  * The contents of this file are subject to the terms of the
7  * Common Development and Distribution License (the "License").
8  * You may not use this file except in compliance with the License.
9  *
10 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11 * or http://www.opensolaris.org/os/licensing.
12 * See the License for the specific language governing permissions
13 * and limitations under the License.
14 *
15 * When distributing Covered Code, include this CDDL HEADER in each
16 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17 * If applicable, add the following below this CDDL HEADER, with the
18 * fields enclosed by brackets "[]" replaced with your own identifying
19 * information: Portions Copyright [yyyy] [name of copyright owner]
20 *
21 * CDDL HEADER END
22 */
23 /*
24 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
25 * Use is subject to license terms.
26 */

28 /*
29 * This is not "#ifdef TEAMWARE_MAKE_CMN" because we're currently
30 * using the TW fake i18n headers and libraries to build both
31 * SMake and PMake on SPARC/S1 and x86/S2.
32 */

28 #include <avo/intl.h>
29 #include <limits.h>          /* MB_LEN_MAX */
30 #include <stdio.h>
31 #include <stdlib.h>         /* wchar_t */
32 #include <string.h>        /* strcmp() */
33 #include <nl_types.h>      /* catgets() */
34 #include <sys/param.h>    /* MAXPATHLEN */
35 #include <sys/types.h>    /* time_t, caddr_t */
36 #include <vroot/vroot.h> /* pathpt */
37 #include <sys/time.h>    /* timestruc_t */
38 #include <errno.h>      /* errno */

40 #include <wctype.h>
41 #include <wchar.h>

44 /*
45 * A type and some utilities for boolean values
46 */

48 #define false    BOOLEAN_false
49 #define true     BOOLEAN_true

51 typedef enum {
52     false =    0,
53     true  =    1,
54     failed =    0,
55     succeeded = 1

```

```

56 } Boolean;
_____ unchanged_portion_omitted _____

856 /*
857 *     extern declarations for all global variables.
858 *     The actual declarations are in globals.cc
859 */
860 extern char          char_semantics[];
861 extern wchar_t       char_semantics_char[];
862 extern Macro_list    cond_macro_list;
863 extern Boolean       conditional_macro_used;
864 extern Boolean       do_not_exec_rule;          /* '-n' */
865 extern Boolean       dollarget_seen;
866 extern Boolean       dollarless_flag;
867 extern Name          dollarless_value;
868 extern char          **environ;
869 extern Envvar        envvar;
870 extern int           exit_status;
871 extern wchar_t       *file_being_read;
872 /* Variable gnu_style=true if env. var. SUN_MAKE_COMPAT_MODE=GNU (RFE 4866328) */
873 extern Boolean       gnu_style;
874 extern Name_set      hashtab;
875 extern Name          host_arch;
876 extern Name          host_mach;
877 extern int           line_number;
878 extern char          *make_state_lockfile;
879 extern Boolean       make_word_mentioned;
880 extern Makefile_type makefile_type;
881 extern char          mbs_buffer[];
882 extern Name          path_name;
883 extern Boolean       posix;
884 extern Name          query;
885 extern Boolean       query_mentioned;
886 extern Name          hat;
887 extern Boolean       reading_environment;
888 extern Name          shell_name;
889 extern Boolean       svr4;
890 extern Name          target_arch;
891 extern Name          target_mach;
892 extern Boolean       tilde_rule;
893 extern wchar_t       wcs_buffer[];
894 extern Boolean       working_on_targets;
895 extern Name          virtual_root;
896 extern Boolean       vpath_defined;
897 extern Name          vpath_name;
898 extern Boolean       make_state_locked;
899 extern Boolean       out_err_same;
900 #endif
901 extern pid_t         childPid;
902 extern nl_catd       libmksh_catd;

903 /*
904 * RFE 1257407: make does not use fine granularity time info available from stat
905 * High resolution time comparison.
906 */

908 inline int
909 operator==(const timestruc_t &t1, const timestruc_t &t2) {
910     return ((t1.tv_sec == t2.tv_sec) && (t1.tv_nsec == t2.tv_nsec));
911 }
_____ unchanged_portion_omitted _____

```

```

*****
2971 Wed May 20 11:58:32 2015
new/usr/src/cmd/make/lib/mksh/globals.cc
make: undef for TEAMWARE_MAKE_CMN (defined)
*****
  unchanged portion omitted
79 Macro_list      cond_macro_list;
80 Boolean         conditional_macro_used;
81 Boolean         do_not_exec_rule;          /* '\-n' */
82 Boolean         dollarless_flag;
83 Boolean         dollarless_value;
84 Name            dollarless_value;
85 Envvar          envvar;
86 #ifdef lint
87 char            **environ;
88 #endif
89 int             exit_status;
90 wchar_t         *file_being_read;
91 /* Variable gnu_style=true if env. var. SUN_MAKE_COMPAT_MODE=GNU (RFE 4866328) *
92 Boolean         gnu_style = false;
93 Name_set        hashtable;
94 Name            host_arch;
95 Name            host_mach;
96 int             line_number;
97 char            *make_state_lockfile;
98 Boolean         make_word_mentioned;
99 Makefile_type   makefile_type = reading_nothing;
100 char           mbs_buffer[(MAXPATHLEN * MB_LEN_MAX)];
101 Name            path_name;
102 Boolean         posix = true;
103 Name            hat;
104 Name            query;
105 Boolean         query_mentioned;
106 Boolean         reading_environment;
107 Name            shell_name;
108 Boolean         svr4 = false;
109 Name            target_arch;
110 Name            target_mach;
111 Boolean         tilde_rule;
112 Name            virtual_root;
113 Boolean         vpath_defined;
114 Name            vpath_name;
115 wchar_t         wcs_buffer[MAXPATHLEN];
116 Boolean         working_on_targets;
117 #if defined (TEAMWARE_MAKE_CMN)
118 Boolean         out_err_same;
119 #endif
118 pid_t          childPid = -1; // This variable is used for killing child's pro
119                               // Such as qrsh, running command, etc.

121 /*
122 * timestamps defined in defs.h
123 */
124 const timestruc_t file_no_time      = { -1, 0 };
125 const timestruc_t file_doesnt_exist = { 0, 0 };
126 const timestruc_t file_is_dir      = { 1, 0 };
127 const timestruc_t file_min_time    = { 2, 0 };
128 const timestruc_t file_max_time    = { INT_MAX, 0 };

```