```
**********************************************************
   15270 Wed May 20 11:48:47 2015
new/usr/src/cmd/make/bin/dist.cc
make: unifdef for REDIRECT_ERR (defined)
**********************************************************
______unchanged_portion_omitted_

 271 /*
 272  * static void
 273  * set_dmake_env_vars()
 274  *
 275  * Sets the DMAKE_* environment variables for rxm and rxs.
 276  *      DMAKE_PWD
 277  *      DMAKE_NPWD
 278  *      DMAKE_UMASK
 279  *      DMAKE_SHELL
 280  */
 281 static void
 282 set_dmake_env_vars()
 283 {
 284         char            *current_netpath;
 285         char            *current_path;
 286         static char     *env;
 287         int             length;
 288         char            netpath[MAXPATHLEN];
 289         mode_t          um;
 290         char            um_buf[MAXPATHLEN];
 291         Name            dmake_name;
 292         Name            dmake_value;
 293         Property        prop;

 295 #ifdef REDIRECT_ERR
 295         /* Set __DMAKE_REDIRECT_STDERR */
 296         length = 2 + strlen(NOCATGETS("__DMAKE_REDIRECT_STDERR")) + 1;
 297         env = getmem(length);
 298         (void) sprintf(env,
 299                         "%s=%s",
 300                         NOCATGETS("__DMAKE_REDIRECT_STDERR"),
 301                         out_err_same ? NOCATGETS("0") : NOCATGETS("1"));
 302         (void) putenv(env);
 304 #endif

 304         /* Set DMAKE_PWD to the current working directory */
 305         current_path = get_current_path();
 306         length = 2 + strlen(NOCATGETS("DMAKE_PWD")) + strlen(current_path);
 307         env = getmem(length);
 308         (void) sprintf(env,
 309                         "%s=%s",
 310                         NOCATGETS("DMAKE_PWD"),
 311                         current_path);
 312         (void) putenv(env);

 314         /* Set DMAKE_NPWD to machine:pathname */
 315         if (avo_path_to_netpath(current_path, netpath)) {
 316                 current_netpath = netpath;
 317         } else {
 318                 current_netpath = current_path;
 319         }
 320         length = 2 + strlen(NOCATGETS("DMAKE_NPWD")) + strlen(current_netpath);
 321         env = getmem(length);
 322         (void) sprintf(env,
 323                         "%s=%s",
 324                         NOCATGETS("DMAKE_NPWD"),
 325                         current_netpath);
 326         (void) putenv(env);
```



```
 328         /* Set DMAKE_UMASK to the value of umask */
 329         um = umask(0);
 330         umask(um);
 331         (void) sprintf(um_buf, NOCATGETS("%ul"), um);
 332         length = 2 + strlen(NOCATGETS("DMAKE_UMASK")) + strlen(um_buf);
 333         env = getmem(length);
 334         (void) sprintf(env,
 335                         "%s=%s",
 336                         NOCATGETS("DMAKE_UMASK"),
 337                         um_buf);
 338         (void) putenv(env);

 340         if (((prop = get_prop(shell_name->prop, macro_prop)) != NULL) &&
 341             ((dmake_value = prop->body.macro.value) != NULL)) {
 342                 length = 2 + strlen(NOCATGETS("DMAKE_SHELL")) +
 343                         strlen(dmake_value->string_mb);
 344                 env = getmem(length);
 345                 (void) sprintf(env,
 346                                 "%s=%s",
 347                                 NOCATGETS("DMAKE_SHELL"),
 348                                 dmake_value->string_mb);
 349                 (void) putenv(env);
 350         }
 351         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
 352         dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
 353         if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
 354             ((dmake_value = prop->body.macro.value) != NULL)) {
 355                 length = 2 + strlen(NOCATGETS("DMAKE_OUTPUT_MODE")) +
 356                         strlen(dmake_value->string_mb);
 357                 env = getmem(length);
 358                 (void) sprintf(env,
 359                                 "%s=%s",
 360                                 NOCATGETS("DMAKE_OUTPUT_MODE"),
 361                                 dmake_value->string_mb);
 362                 (void) putenv(env);
 363         }
 364 }
______unchanged_portion_omitted_
```

```
**********************************************************
   97819 Wed May 20 11:48:48 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef for REDIRECT_ERR (defined)
**********************************************************
______unchanged_portion_omitted_
 147 #endif

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

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

 153 static  void            append_makeflags_string(Name, String);
 154 static  void            doalarm(int);
 155 static  void            enter_argv_values(int , char **, ASCII_Dyn_Array *);
 156 static  void            make_targets(int, char **, Boolean);
 157 static  int             parse_command_option(char);
 158 static  void            read_command_options(int, char **);
 159 static  void            read_environment(Boolean);
 160 static  void            read_files_and_state(int, char **);
 161 static  Boolean         read_makefile(Name, Boolean, Boolean, Boolean);
 162 static  void            report_recursion(Name);
 163 static  void            set_sgs_support(void);
 164 static  void            setup_for_projectdir(void);
 165 static  void            setup_makeflags_argv(void);
 166 static  void            report_dir_enter_leave(Boolean entering);

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

 170 #ifdef DISTRIBUTED
 171         extern  int             dmake_ofd;
 172         extern  FILE*           dmake_ofp;
 173         extern  int             rxmPid;
 174         extern  XDR             xdrs_out;
 175 #endif
 176 #ifdef TEAMWARE_MAKE_CMN
 177         extern  char            verstring[];
 178 #endif

 180 jmp_buf jmpbuffer;
 181 extern nl_catd catd;

 183 /*
 184  *      main(argc, argv)
 185  *
 186  *      Parameters:
 187  *              argc                    You know what this is
 188  *              argv                    You know what this is
 189  *
 190  *      Static variables used:
 191  *              list_all_targets        make -T seen
 192  *              trace_status            make -p seen
 193  *
 194  *      Global variables used:
 195  *              debug_level             Should we trace make actions?
 196  *              keep_state              Set if .KEEP_STATE seen
 197  *              makeflags               The Name "MAKEFLAGS", used to get macro
 198  *              remote_command_name     Name of remote invocation cmd ("on")
 199  *              running_list            List of parallel running processes
 200  *              stdout_stderr_same      true if stdout and stderr are the same
 201  *              auto_dependencies       The Name "SUNPRO_DEPENDENCIES"
 202  *              temp_file_directory     Set to the dir where we create tmp file
 203  *              trace_reader            Set to reflect tracing status
 204  *              working_on_targets      Set when building user targets
 205  */
 206 int
```



```
 207 main(int argc, char *argv[])
 208 {
 209         /*
 210          * cp is a -> to the value of the MAKEFLAGS env var,
 211          * which has to be regular chars.
 212          */
 213         register char           *cp;
 214         char                    make_state_dir[MAXPATHLEN];
 215         Boolean                 parallel_flag = false;
 216         char                    *prognameptr;
 217         char                    *slash_ptr;
 218         mode_t                  um;
 219         int                     i;
 220 #ifdef TEAMWARE_MAKE_CMN
 221         struct itimerval        value;
 222         char                    def_dmakerc_path[MAXPATHLEN];
 223         Name                    dmake_name, dmake_name2;
 224         Name                    dmake_value, dmake_value2;
 225         Property                prop, prop2;
 226         struct stat             statbuf;
 227         int                     statval;
 228 #endif

 230         struct stat             out_stat, err_stat;
 231         hostid = gethostid();
 232 #ifdef TEAMWARE_MAKE_CMN
 233         avo_get_user(NULL, NULL); // Initialize user name
 234 #endif
 235         bsd_signals();

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


 240 #ifdef DMAKE_STATISTICS
 241         if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
 242                 getname_stat = true;
 243         }
 244 #endif


 247         /*
 248          * avo_init() sets the umask to 0.  Save it here and restore
 249          * it after the avo_init() call.
 250          */
 251 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
 252         um = umask(0);
 253         avo_init(argv[0]);
 254         umask(um);

 256         cleanup = new Avo_cleanup(NOCATGETS("dmake"), argc, argv);
 257 #endif

 259 #if defined(TEAMWARE_MAKE_CMN)
 260         catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
 261         libcli_init();
 262 #endif

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


 267 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
 268 /*
 269  * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
 270  * from avo_util library.
 271  */
 272         libmksdmsi18n_init();
```

```
#endif


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

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

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

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

                tmp_current_path = get_current_path();
                tmp_string = getmem(strlen(tmp_current_path) + 1 +
                                    strlen(argv[0]) + 1);
                (void) sprintf(tmp_string,
                               "%s/%s",
                               tmp_current_path,
                               argv[0]);
                argv_zero_string = strdup(tmp_string);
                retmem_mb(tmp_string);
        }

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


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

        /* Sun OS make standart */
        svr4 = false;
```



```
        posix = false;
        if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
                svr4 = false;
                posix = true;
        } else {
                prognameptr = strrchr(argv[0], '/');
                if(prognameptr) {
                        prognameptr++;
                } else {
                        prognameptr = argv[0];
                }
                if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
                        svr4 = true;
                        posix = false;
                }
        }
        if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
           svr4 = true;
           posix = false;
        }

        /*
         * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
         */
        char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
        if (dmake_compat_mode_var != NULL) {
                if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
                        gnu_style = true;
                }
                //svr4 = false;
                //posix = false;
        }

        /*
         * Temporary directory set up.
         */
        char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
        if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
                strcpy(mbs_buffer, tmpdir_var);
                for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
                        *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
                        *tmpdir_var = '\0');
                if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
                        sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
                                mbs_buffer, getpid());
                        int fd = mkstemp(mbs_buffer2);
                        if (fd >= 0) {
                                close(fd);
                                unlink(mbs_buffer2);
                                tmpdir = strdup(mbs_buffer);
                        }
                }
        }

        /* find out if stdout and stderr point to the same place */
        if (fstat(1, &out_stat) < 0) {
                fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
        }
        if (fstat(2, &err_stat) < 0) {
                fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
        }
        if ((out_stat.st_dev == err_stat.st_dev) &&
            (out_stat.st_ino == err_stat.st_ino)) {
                stdout_stderr_same = true;
        } else {
                stdout_stderr_same = false;
```

```
405         }
406         /* Make the vroot package scan the path using shell semantics */
407         set_path_style(0);

409         setup_char_semantics();

411         setup_for_projectdir();

413         /*
414          * If running with .KEEP_STATE, curdir will be set with
415          * the connected directory.
416          */
417         (void) atexit(cleanup_after_exit);

419         load_cached_names();

421 /*
422  *      Set command line flags
423  */
424         setup_makeflags_argv();
425         read_command_options(mf_argc, mf_argv);
426         read_command_options(argc, argv);
427         if (debug_level > 0) {
428                 cp = getenv(makeflags->string_mb);
429                 (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
430         }

432         setup_interrupt(handle_interrupt);

434         read_files_and_state(argc, argv);

436 #ifdef TEAMWARE_MAKE_CMN
437         /*
438          * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
439          */
440         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
441         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
442         prop2 = get_prop(dmake_name2->prop, macro_prop);
443         if (prop2 == NULL) {
444                 /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
445                 output_mode = txt1_mode;
446         } else {
447                 dmake_value2 = prop2->body.macro.value;
448                 if ((dmake_value2 == NULL) ||
449                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
450                         output_mode = txt1_mode;
451                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
452                         output_mode = txt2_mode;
453                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))
454                         output_mode = html1_mode;
455                 } else {
456                         warning(catgets(catd, 1, 352, "Unsupported value ‘%s’ fo
457                               dmake_value2->string_mb);
458                 }
459         }
460         /*
461          * Find the dmake_mode: distributed, parallel, or serial.
462          */
463     if ((!pmake_cap_r_specified) &&
464         (!pmake_machinesfile_specified)) {
465         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
466         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
467         prop2 = get_prop(dmake_name2->prop, macro_prop);
468         if (prop2 == NULL) {
469                 /* DMAKE_MODE not defined, default to distributed mode */
470                 dmake_mode_type = distributed_mode;
```



```
471                 no_parallel = false;
472         } else {
473                 dmake_value2 = prop2->body.macro.value;
474                 if ((dmake_value2 == NULL) ||
475                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed"))
476                         dmake_mode_type = distributed_mode;
477                         no_parallel = false;
478                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel
479                         dmake_mode_type = parallel_mode;
480                         no_parallel = false;
481                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
482                         dmake_mode_type = serial_mode;
483                         no_parallel = true;
484                 } else {
485                         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
486                 }
487         }

489         if ((!list_all_targets) &&
490             (report_dependencies_level == 0)) {
491                 /*
492                  * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
493                  * They could be defined in the env, in the makefile, or on the
494                  * command line.
495                  * If neither is defined, and $(HOME)/.dmakerc does not exists,
496                  * then print a message, and default to parallel mode.
497                  */
498 #ifdef DISTRIBUTED
499                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
500                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
501                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
502                 dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
503                 if ((((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
504                      ((dmake_value = prop->body.macro.value) == NULL)) &&
505                     (((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL)
506                      ((dmake_value2 = prop2->body.macro.value) == NULL))) {
507                         Boolean empty_dmakerc = true;
508                         char *homedir = getenv(NOCATGETS("HOME"));
509                         if ((homedir != NULL) && (strlen(homedir) < (sizeof(def_
510                                 sprintf(def_dmakerc_path, NOCATGETS("%s/.dmakerc
511                                 if ((((statval = stat(def_dmakerc_path, &statbuf
512                                       ((statval == 0) && (statbuf.st_size == 0
513                                 } else {
514                                         Avo_dmakerc     *rcfile = new Avo_dmaker
515                                         Avo_err         *err = rcfile->read(def_
516                                         if (err) {
517                                                 fatal(err->str);
518                                         }
519                                         empty_dmakerc = rcfile->was_empty();
520                                         delete rcfile;
521                                 }
522                         }
523                         if (empty_dmakerc) {
524                                 if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
525                                         putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE
526                                         (void) fprintf(stdout, catgets(catd, 1,
527                                         (void) fprintf(stdout, catgets(catd, 1,
528                                 }
529                                 dmake_mode_type = parallel_mode;
530                                 no_parallel = false;
531                         }
532                 }
533 #else
534                 if(dmake_mode_type == distributed_mode) {
535                         (void) fprintf(stdout, NOCATGETS("dmake: Distributed mod
536                         (void) fprintf(stdout, NOCATGETS("       Defaulting to p
```

```
537                         dmake_mode_type = parallel_mode;
538                         no_parallel = false;
539                 }
540 #endif  /* DISTRIBUTED */
541         }
542     }
543 #endif

545 #ifdef TEAMWARE_MAKE_CMN
546         parallel_flag = true;
547         /* XXX - This is a major hack for DMake/Licensing. */
548         if (getenv(NOCATGETS("DMAKE_CHILD")) == NULL) {
549                 if (!avo_cli_search_license(argv[0], dmake_exit_callback, TRUE,
550                         /*
551                          * If the user can not get a TeamWare license,
552                          * default to serial mode.
553                          */
554                         dmake_mode_type = serial_mode;
555                         no_parallel = true;
556                 } else {
557                         putenv(NOCATGETS("DMAKE_CHILD=TRUE"));
558                 }
559                 start_time = time(NULL);
560                 /*
561                  * XXX - Hack to disable SIGALRM's from licensing library's
562                  *       setitimer().
563                  */
564                 value.it_interval.tv_sec = 0;
565                 value.it_interval.tv_usec = 0;
566                 value.it_value.tv_sec = 0;
567                 value.it_value.tv_usec = 0;
568                 (void) setitimer(ITIMER_REAL, &value, NULL);
569         }

571 //
572 // If dmake is running with -t option, set dmake_mode_type to serial.
573 // This is done because doname() calls touch_command() that runs serially.
574 // If we do not do that, maketool will have problems.
575 //
576         if(touch) {
577                 dmake_mode_type = serial_mode;
578                 no_parallel = true;
579         }
580 #else
581         parallel_flag = false;
582 #endif

584 #if defined (TEAMWARE_MAKE_CMN)
584 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
585         /*
586          * Check whether stdout and stderr are physically same.
587          * This is in order to decide whether we need to redirect
588          * stderr separately from stdout.
589          * This check is performed only if __DMAKE_SEPARATE_STDERR
590          * is not set. This variable may be used in order to preserve
591          * the 'old' behaviour.
592          */
593         out_err_same = true;
594         char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
595         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
596                 struct stat stdout_stat;
597                 struct stat stderr_stat;
598                 if( (fstat(1, &stdout_stat) == 0)
599                  && (fstat(2, &stderr_stat) == 0) )
600                 {
601                         if( (stdout_stat.st_dev != stderr_stat.st_dev)
```



```
602                          || (stdout_stat.st_ino != stderr_stat.st_ino) )
603                         {
604                                 out_err_same = false;
605                         }
606                 }
607         }
608 #endif

610 #ifdef DISTRIBUTED
611         /*
612          * At this point, DMake should startup an rxm with any and all
613          * DMake command line options. Rxm will, among other things,
614          * read the rc file.
615          */
616         if ((!list_all_targets) &&
617             (report_dependencies_level == 0) &&
618             (dmake_mode_type == distributed_mode)) {
619                 startup_rxm();
620         }
621 #endif
622
623 /*
624  *      Enable interrupt handler for alarms
625  */
626         (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

628 /*
629  *      Check if make should report
630  */
631         if (getenv(sunpro_dependencies->string_mb) != NULL) {
632                 FILE    *report_file;

634                 report_dependency("");
635                 report_file = get_report_file();
636                 if ((report_file != NULL) && (report_file != (FILE*)-1)) {
637                         (void) fprintf(report_file, "\n");
638                 }
639         }

641 /*
642  *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
643  */
644         if (keep_state) {
645                 maybe_append_prop(sunpro_dependencies, macro_prop)->
646                   body.macro.exported = true;
647         } else {
648                 maybe_append_prop(sunpro_dependencies, macro_prop)->
649                   body.macro.exported = false;
650         }

652         working_on_targets = true;
653         if (trace_status) {
654                 dump_make_state();
655                 fclose(stdout);
656                 fclose(stderr);
657                 exit_status = 0;
658                 exit(0);
659         }
660         if (list_all_targets) {
661                 dump_target_list();
662                 fclose(stdout);
663                 fclose(stderr);
664                 exit_status = 0;
665                 exit(0);
666         }
667         trace_reader = false;
```

```
669         /*
670          * Set temp_file_directory to the directory the .make.state
671          * file is written to.
672          */
673         if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
674                 temp_file_directory = strdup(get_current_path());
675         } else {
676                 *slash_ptr = (int) nul_char;
677                 (void) strcpy(make_state_dir, make_state->string_mb);
678                 *slash_ptr = (int) slash_char;
679                    /* when there is only one slash and it's the first
680                    ** character, make_state_dir should point to '/'.
681                    */
682                 if(make_state_dir[0] == '\0') {
683                    make_state_dir[0] = '/';
684                    make_state_dir[1] = '\0';
685                 }
686                 if (make_state_dir[0] == (int) slash_char) {
687                         temp_file_directory = strdup(make_state_dir);
688                 } else {
689                         char    tmp_current_path2[MAXPATHLEN];
690
691                         (void) sprintf(tmp_current_path2,
692                                         "%s/%s",
693                                         get_current_path(),
694                                         make_state_dir);
695                         temp_file_directory = strdup(tmp_current_path2);
696                 }
697         }

699 #ifdef DISTRIBUTED
700         building_serial = false;
701 #endif

703         report_dir_enter_leave(true);

705         make_targets(argc, argv, parallel_flag);

707         report_dir_enter_leave(false);

709         if (build_failed_ever_seen) {
710                 if (posix) {
711                         exit_status = 1;
712                 }
713                 exit(1);
714         }
715         exit_status = 0;
716         exit(0);
717         /* NOTREACHED */
718 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   53335 Wed May 20 11:48:49 2015
new/usr/src/cmd/make/bin/parallel.cc
make: unifdef for REDIRECT_ERR (defined)
**********************************************************
______unchanged_portion_omitted_

 722 #endif /* MAXJOBS_ADJUST_RFE4694000 */
 723 #endif /* TEAMWARE_MAKE_CMN */

 725 /*
 726  *      distribute_process(char **commands, Property line)
 727  *
 728  *      Parameters:
 729  *              commands        argv vector of commands to execute
 730  *
 731  *      Return value:
 732  *                              The result of the execution
 733  *
 734  *      Static variables used:
 735  *              process_running Set to the pid of the process set running
 736  * #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
 737  *              job_adjust_mode Current job adjust mode
 738  * #endif
 739  */
 740 static Doname
 741 distribute_process(char **commands, Property line)
 742 {
 743         static unsigned file_number = 0;
 744         wchar_t         string[MAXPATHLEN];
 745         char            mbstring[MAXPATHLEN];
 746         int             filed;
 747         int             res;
 748         int             tmp_index;
 749         char            *tmp_index_str_ptr;

 751 #if !defined (TEAMWARE_MAKE_CMN) || !defined (MAXJOBS_ADJUST_RFE4694000)
 752         while (parallel_process_cnt >= pmake_max_jobs) {
 753                 await_parallel(false);
 754                 finish_children(true);
 755         }
 756 #else /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
 757         /* initialize adjust mode, if not initialized */
 758         if (job_adjust_mode == ADJUST_UNKNOWN) {
 759                 job_adjust_init();
 760         }

 762         /* actions depend on adjust mode */
 763         switch (job_adjust_mode) {
 764         case ADJUST_M1:
 765                 while (parallel_process_cnt >= adjust_pmake_max_jobs (pmake_max_
 766                         await_parallel(false);
 767                         finish_children(true);
 768                 }
 769                 break;
 770         case ADJUST_M2:
 771                 if ((res = m2_acquire_job()) == 0) {
 772                         if (parallel_process_cnt > 0) {
 773                                 await_parallel(false);
 774                                 finish_children(true);

 776                                 if ((res = m2_acquire_job()) == 0) {
 777                                         return build_serial;
 778                                 }
 779                         } else {
 780                                 return build_serial;
```



```
 781                         }
 782                 }
 783                 if (res < 0) {
 784                         /* job adjustment error */
 785                         job_adjust_error();

 787                         /* no adjustment */
 788                         while (parallel_process_cnt >= pmake_max_jobs) {
 789                                 await_parallel(false);
 790                                 finish_children(true);
 791                         }
 792                 }
 793                 break;
 794         default:
 795                 while (parallel_process_cnt >= pmake_max_jobs) {
 796                         await_parallel(false);
 797                         finish_children(true);
 798                 }
 799         }
 800 #endif /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
 801 #ifdef DISTRIBUTED
 802         if (send_mtool_msgs) {
 803                 send_job_start_msg(line);
 804         }
 805 #endif
 806 #ifdef DISTRIBUTED
 807         setvar_envvar((Avo_DoJobMsg *)NULL);
 808 #else
 809         setvar_envvar();
 810 #endif
 811         /*
 812          * Tell the user what DMake is doing.
 813          */
 814         if (!silent && output_mode != txt2_mode) {
 815                 /*
 816                  * Print local_host --> x job(s).
 817                  */
 818                 (void) fprintf(stdout,
 819                                 catgets(catd, 1, 325, "%s --> %d %s\n"),
 820                                 local_host,
 821                                 parallel_process_cnt + 1,
 822                                 (parallel_process_cnt == 0) ? catgets(catd, 1, 12

 824                 /* Print command line(s). */
 825                 tmp_index = 0;
 826                 while (commands[tmp_index] != NULL) {
 827                     /* No @ char. */
 828                     /* XXX - need to add [2] when + prefix is added */
 829                     if ((commands[tmp_index][0] != (int) at_char) &&
 830                         (commands[tmp_index][1] != (int) at_char)) {
 831                         tmp_index_str_ptr = commands[tmp_index];
 832                         if (*tmp_index_str_ptr == (int) hyphen_char) {
 833                                 tmp_index_str_ptr++;
 834                         }
 835                         (void) fprintf(stdout, "%s\n", tmp_index_str_ptr);
 836                     }
 837                     tmp_index++;
 838                 }
 839                 (void) fflush(stdout);
 840         }

 842         (void) sprintf(mbstring,
 843                         NOCATGETS("%s/dmake.stdout.%d.%d.XXXXXX"),
 844                         tmpdir,
 845                         getpid(),
 846                         file_number++);
```

```
 848         mktemp(mbstring);

 850         stdout_file = strdup(mbstring);
 851         stderr_file = NULL;
 852 #if defined (TEAMWARE_MAKE_CMN)
 852 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
 853         if (!out_err_same) {
 854                 (void) sprintf(mbstring,
 855                                 NOCATGETS("%s/dmake.stderr.%d.%d.XXXXXX"),
 856                                 tmpdir,
 857                                 getpid(),
 858                                 file_number++);

 860                 mktemp(mbstring);

 862                 stderr_file = strdup(mbstring);
 863         }
 864 #endif

 866         process_running = run_rule_commands(local_host, commands);

 868         return build_running;
 869 }
_____unchanged_portion_omitted_

1381 /*
1382  *      finish_children(docheck)
1383  *
1384  *      Finishes the processing for all targets which were running
1385  *      and have now completed.
1386  *
1387  *      Parameters:
1388  *              docheck         Completely check the finished target
1389  *
1390  *      Static variables used:
1391  *              running_tail    The tail of the running list
1392  *
1393  *      Global variables used:
1394  *              continue_after_error  -k flag
1395  *              fatal_in_progress  True if we are finishing up after fatal err
1396  *              running_list    List of running processes
1397  */
1398 void
1399 finish_children(Boolean docheck)
1400 {
1401         int             cmds_length;
1402         Property        line;
1403         Property        line2;
1404         struct stat     out_buf;
1405         Running         rp;
1406         Running         *rp_prev;
1407         Cmd_line        rule;
1408         Boolean         silent_flag;

1410         for (rp_prev = &running_list, rp = running_list;
1411              rp != NULL;
1412              rp = rp->next) {
1413 bypass_for_loop_inc_4:
1414                 /*
1415                  * If the state is ok or failed, then this target has
1416                  * finished building.
1417                  * In parallel_mode, output the accumulated stdout/stderr.
1418                  * Read the auto dependency stuff, handle a failed build,
1419                  * update the target, then finish the doname process for
1420                  * that target.
```



```
1421                  */
1422                 if (rp->state == build_ok || rp->state == build_failed) {
1423                         *rp_prev = rp->next;
1424                         if (rp->next == NULL) {
1425                                 running_tail = rp_prev;
1426                         }
1427                         if ((line2 = rp->command) == NULL) {
1428                                 line2 = get_prop(rp->target->prop, line_prop);
1429                         }
1430                         if (dmake_mode_type == distributed_mode) {
1431                                 if (rp->make_refd) {
1432                                         maybe_reread_make_state();
1433                                 }
1434                         } else {
1435                                 /*
1436                                  * Send an Avo_MToolJobResultMsg to maketool.
1437                                  */
1438 #ifdef DISTRIBUTED
1439                                 if (send_mtool_msgs) {
1440                                         send_job_result_msg(rp);
1441                                 }
1442 #endif
1443                                 /*
1444                                  * Check if there were any job output
1445                                  * from the parallel build.
1446                                  */
1447                                 if (rp->stdout_file != NULL) {
1448                                         if (stat(rp->stdout_file, &out_buf) < 0)
1449                                                 fatal(catgets(catd, 1, 130, "sta
1450                                                       rp->stdout_file,
1451                                                       errmsg(errno));
1452                                         }
1453                                         if ((line2 != NULL) &&
1454                                             (out_buf.st_size > 0)) {
1455                                                 cmds_length = 0;
1456                                                 for (rule = line2->body.line.com
1457                                                      silent_flag = silent;
1458                                                      rule != NULL;
1459                                                      rule = rule->next) {
1460                                                         cmds_length += rule->com
1461                                                         silent_flag = BOOLEAN(si
1462                                                 }
1463                                                 if (out_buf.st_size != cmds_leng
1464                                                     output_mode == txt2_mode) {
1465                                                         dump_out_file(rp->stdout
1466                                                 }
1467                                         }
1468                                         (void) unlink(rp->stdout_file);
1469                                         retmem_mb(rp->stdout_file);
1470                                         rp->stdout_file = NULL;
1471                                 }

1472 #if defined(REDIRECT_ERR)
1473                                 if (!out_err_same && (rp->stderr_file != NULL))
1474                                         if (stat(rp->stderr_file, &out_buf) < 0)
1475                                                 fatal(catgets(catd, 1, 130, "sta
1476                                                       rp->stderr_file,
1477                                                       errmsg(errno));
1478                                         }
1479                                         if ((line2 != NULL) &&
1480                                             (out_buf.st_size > 0)) {
1481                                                 dump_out_file(rp->stderr_file, t
1482                                         }
1483                                         (void) unlink(rp->stderr_file);
1484                                         retmem_mb(rp->stderr_file);
1485                                         rp->stderr_file = NULL;
```

```
1486                                }
1487 #endif
1487                        }
1488                        check_state(rp->temp_file);
1489                        if (rp->temp_file != NULL) {
1490                                free_name(rp->temp_file);
1491                        }
1492                        rp->temp_file = NULL;
1493                        if (rp->state == build_failed) {
1494                                line = get_prop(rp->target->prop, line_prop);
1495                                if (line != NULL) {
1496                                        line->body.line.command_used = NULL;
1497                                }
1498                                if (continue_after_error ||
1499                                    fatal_in_progress ||
1500                                    !docheck) {
1501                                        warning(catgets(catd, 1, 256, "Command f
1502                                                rp->command ? line2->body.line.t
1503                                        build_failed_seen = true;
1504                                } else {
1505                                        /*
1506                                         * XXX??? - DMake needs to exit(),
1507                                         * but shouldn't call fatal().
1508                                         */
1509 #ifdef PRINT_EXIT_STATUS
1510                                        warning(NOCATGETS("I'm in finish_childre
1511 #endif
1513                                        fatal(catgets(catd, 1, 258, "Command fai
1514                                               rp->command ? line2->body.line.t
1515                                }
1516                        }
1517                        if (!docheck) {
1518                                delete_running_struct(rp);
1519                                rp = *rp_prev;
1520                                if (rp == NULL) {
1521                                        break;
1522                                } else {
1523                                        goto bypass_for_loop_inc_4;
1524                                }
1525                        }
1526                        update_target(get_prop(rp->target->prop, line_prop),
1527                                      rp->state);
1528                        finish_doname(rp);
1529                        delete_running_struct(rp);
1530                        rp = *rp_prev;
1531                        if (rp == NULL) {
1532                                break;
1533                        } else {
1534                                goto bypass_for_loop_inc_4;
1535                        }
1536                } else {
1537                        rp_prev = &rp->next;
1538                }
1539        }
1540 }
```

*unchanged_portion_omitted_*

```
2011 /*
2012  * This function replaces the makesh binary.
2013  */
2014

2016 static pid_t
2017 run_rule_commands(char *host, char **commands)
2018 {
```



```
2019        Boolean         always_exec;
2020        Name            command;
2021        Boolean         ignore;
2022        int             length;
2023        Doname          result;
2024        Boolean         silent_flag;
2025        wchar_t         *tmp_wcs_buffer;

2027        childPid = fork();
2028        switch (childPid) {
2029        case -1:        /* Error */
2030                fatal(catgets(catd, 1, 337, "Could not fork child process for dm
2031                      errmsg(errno));
2032                break;
2033        case 0:         /* Child */
2034                /* To control the processed targets list is not the child's busi
2035                running_list = NULL;
2037 #if defined(REDIRECT_ERR)
2036                if(out_err_same) {
2037                        redirect_io(stdout_file, (char*)NULL);
2038                } else {
2039                        redirect_io(stdout_file, stderr_file);
2040                }
2043 #else
2044                redirect_io(stdout_file, (char*)NULL);
2045 #endif
2041                for (commands = commands;
2042                     (*commands != (char *)NULL);
2043                     commands++) {
2044                        silent_flag = silent;
2045                        ignore = false;
2046                        always_exec = false;
2047                        while ((**commands == (int) at_char) ||
2048                               (**commands == (int) hyphen_char) ||
2049                               (**commands == (int) plus_char)) {
2050                                if (**commands == (int) at_char) {
2051                                        silent_flag = true;
2052                                }
2053                                if (**commands == (int) hyphen_char) {
2054                                        ignore = true;
2055                                }
2056                                if (**commands == (int) plus_char) {
2057                                        always_exec = true;
2058                                }
2059                                (*commands)++;
2060                        }
2061                        if ((length = strlen(*commands)) >= MAXPATHLEN) {
2062                                tmp_wcs_buffer = ALLOC_WC(length + 1);
2063                                (void) mbstowcs(tmp_wcs_buffer, *commands, lengt
2064                                command = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2065                                retmem(tmp_wcs_buffer);
2066                        } else {
2067                                MBSTOWCS(wcs_buffer, *commands);
2068                                command = GETNAME(wcs_buffer, FIND_LENGTH);
2069                        }
2070                        if ((command->hash.length > 0) &&
2071                            !silent_flag) {
2072                                (void) printf("%s\n", command->string_mb);
2073                        }
2074                        result = dosys(command,
2075                                       ignore,
2076                                       false,
2077                                       false, /* bugs #4085164 & #4990057 */
2078                                       /* BOOLEAN(silent_flag && ignore), */
2079                                       always_exec,
2080                                       (Name) NULL,
```

```
2081                                        false);
2082                        if (result == build_failed) {
2083                                if (silent_flag) {
2084                                        (void) printf(catgets(catd, 1, 152, "The
2085                                }
2086                                if (!ignore) {
2087                                        _exit(1);
2088                                }
2089                        }
2090                }
2091                _exit(0);
2092                break;
2093        default:
2094                break;
2095        }
2096        return childPid;
2097 }
```

_____***unchanged_portion_omitted_***

****
22816 Wed May 20 11:48:50 2015
new/usr/src/cmd/make/include/mksh/defs.h
make: unifdef for REDIRECT_ERR (defined)
****
_____***unchanged_portion_omitted_***

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

911 /*
912  * RFE 1257407: make does not use fine granularity time info available from stat
913  * High resolution time comparison.
914  */

916 inline int
917 operator==(const timestruc_t &t1, const timestruc_t &t2) {
918         return ((t1.tv_sec == t2.tv_sec) && (t1.tv_nsec == t2.tv_nsec));
919 }
```

_____***unchanged_portion_omitted_***

****
3010 Wed May 20 11:48:50 2015
new/usr/src/cmd/make/lib/mksh/globals.cc
make: unifdef for REDIRECT_ERR (defined)
****
*****unchanged_portion_omitted_*****

```
  79 Macro_list      cond_macro_list;
  80 Boolean         conditional_macro_used;
  81 Boolean         do_not_exec_rule;              /* '-n' */
  82 Boolean         dollarget_seen;
  83 Boolean         dollarless_flag;
  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        hashtab;
  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)
 117 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
 118 Boolean         out_err_same;
 119 #endif
 120 pid_t           childPid = -1;  // This variable is used for killing child's pro
 121                                 // Such as qrsh, running command, etc.

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