```
**********************************************************
    4821 Wed May 20 11:37:20 2015
new/usr/src/cmd/make/bin/globals.cc
make: unifdef for SGE (undefined)
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      globals.cc
 *
 *      This declares all global variables
 */

/*
 * Included files
 */
#include <nl_types.h>
#include <mk/defs.h>
#include <sys/stat.h>

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Global variables used by make only
 */
        FILE            *dependency_report_file;

/*
 * Global variables used by make
 */
        Boolean         allrules_read=false;
        Name            posix_name;
        Name            svr4_name;
        Boolean         sdot_target;    /* used to identify s.m(/M)akefile */
        Boolean         all_parallel;                   /* TEAMWARE_MAKE_CMN */
        Boolean         assign_done;
        int foo;
```



```
        Boolean         build_failed_seen;
#ifdef DISTRIBUTED
        Boolean         building_serial;
#endif
        Name            built_last_make_run;
        Name            c_at;
#ifdef DISTRIBUTED
        Boolean         called_make = false;
#endif
        Boolean         cleanup;
        Boolean         close_report;
        Boolean         command_changed;
        Boolean         commands_done;
        Chain           conditional_targets;
        Name            conditionals;
        Boolean         continue_after_error;           /* '-k' */
        Property        current_line;
        Name            current_make_version;
        Name            current_target;
        short           debug_level;
        Cmd_line        default_rule;
        Name            default_rule_name;
        Name            default_target_to_build;
        Name            dmake_group;
        Name            dmake_max_jobs;
        Name            dmake_mode;
        DMake_mode      dmake_mode_type;
        Name            dmake_output_mode;
        DMake_output_mode       output_mode = txt1_mode;
        Name            dmake_odir;
        Name            dmake_rcfile;
        Name            done;
        Name            dot;
        Name            dot_keep_state;
        Name            dot_keep_state_file;
        Name            empty_name;
        Boolean         fatal_in_progress;
        int             file_number;
#if 0
        Boolean         filter_stderr;                  /* '-X' */
#endif
        Name            force;
        Name            ignore_name;
        Boolean         ignore_errors;                  /* '-i' */
        Boolean         ignore_errors_all;              /* '-i' */
        Name            init;
        int             job_msg_id;
        Boolean         keep_state;
        Name            make_state;
#ifdef TEAMWARE_MAKE_CMN
        timestruc_t     make_state_before;
#endif
        Dependency      makefiles_used;
        Name            makeflags;
//      Boolean         make_state_locked; // Moved to lib/mksh
        Name            make_version;
        char            mbs_buffer2[(MAXPATHLEN * MB_LEN_MAX)];
        char            *mbs_ptr;
        char            *mbs_ptr2;
        int             mtool_msgs_fd;
        Boolean         depinfo_already_read = false;
        Boolean         no_action_was_taken = true;     /* true if we've not **
                                                        ** run any command   */

        Boolean         no_parallel = false;            /* TEAMWARE_MAKE_CMN */
#ifdef SGE_SUPPORT
```

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



```
189 /*
190  * File table of contents
191  */
```

```
**********************************************************
   98530 Wed May 20 11:37:21 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef for SGE (undefined)
**********************************************************
_____unchanged_portion_omitted_
 156 #endif

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

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

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

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

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

 189 jmp_buf jmpbuffer;
 190 extern nl_catd catd;

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



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

 239 #ifndef PARALLEL
 240         struct stat             out_stat, err_stat;
 241 #endif
 242         hostid = gethostid();
 243 #ifdef TEAMWARE_MAKE_CMN
 244         avo_get_user(NULL, NULL); // Initialize user name
 245 #endif
 246         bsd_signals();

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


 251 #ifdef DMAKE_STATISTICS
 252         if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
 253                 getname_stat = true;
 254         }
 255 #endif


 258         /*
 259          * avo_init() sets the umask to 0.  Save it here and restore
 260          * it after the avo_init() call.
 261          */
 262 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
 263         um = umask(0);
 264         avo_init(argv[0]);
 265         umask(um);

 267         cleanup = new Avo_cleanup(NOCATGETS("dmake"), argc, argv);
 268 #endif

 270 #if defined(TEAMWARE_MAKE_CMN)
 271         catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
 272         libcli_init();

 274 #ifdef _CHECK_UPDATE_H
 275         /* This is for dmake only (not for Solaris make).
 276          * Check (in background) if there is an update (dmake patch)
 277          * and inform user
 278          */
 279         {
 280                 Avo_err         *err;
 281                 char            *dir;
```

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

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


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


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

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

315         /*
316          * Set argv_zero_string to some form of argv[0] for
317          * recursive MAKE builds.
318          */

320         if (*argv[0] == (int) slash_char) {
321                 /* argv[0] starts with a slash */
322                 argv_zero_string = strdup(argv[0]);
323         } else if (strchr(argv[0], (int) slash_char) == NULL) {
324                 /* argv[0] contains no slashes */
325                 argv_zero_string = strdup(argv[0]);
326         } else {
327                 /*
328                  * argv[0] contains at least one slash,
329                  * but doesn't start with a slash
330                  */
331                 char    *tmp_current_path;
332                 char    *tmp_string;

334                 tmp_current_path = get_current_path();
335                 tmp_string = getmem(strlen(tmp_current_path) + 1 +
336                                     strlen(argv[0]) + 1);
337                 (void) sprintf(tmp_string,
338                                "%s/%s",
339                                tmp_current_path,
340                                argv[0]);
341                 argv_zero_string = strdup(tmp_string);
342                 retmem_mb(tmp_string);
343         }

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



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


357         /*
358          * look for last slash char in the path to look at the binary
359          * name. This is to resolve the hard link and invoke make
360          * in svr4 mode.
361          */

363         /* Sun OS make standart */
364         svr4 = false;
365         posix = false;
366         if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
367                 svr4 = false;
368                 posix = true;
369         } else {
370                 prognameptr = strrchr(argv[0], '/');
371                 if(prognameptr) {
372                         prognameptr++;
373                 } else {
374                         prognameptr = argv[0];
375                 }
376                 if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
377                         svr4 = true;
378                         posix = false;
379                 }
380         }
381         if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
382            svr4 = true;
383            posix = false;
384         }

386         /*
387          * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
388          */
389         char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
390         if (dmake_compat_mode_var != NULL) {
391                 if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
392                         gnu_style = true;
393                 }
394                 //svr4 = false;
395                 //posix = false;
396         }

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

```
414                                 tmpdir = strdup(mbs_buffer);
415                         }
416                 }
417         }

419 #ifndef PARALLEL
420         /* find out if stdout and stderr point to the same place */
421         if (fstat(1, &out_stat) < 0) {
422                 fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
423         }
424         if (fstat(2, &err_stat) < 0) {
425                 fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
426         }
427         if ((out_stat.st_dev == err_stat.st_dev) &&
428             (out_stat.st_ino == err_stat.st_ino)) {
429                 stdout_stderr_same = true;
430         } else {
431                 stdout_stderr_same = false;
432         }
433 #else
434         stdout_stderr_same = false;
435 #endif
436         /* Make the vroot package scan the path using shell semantics */
437         set_path_style(0);

439         setup_char_semantics();

441         setup_for_projectdir();

443         /*
444          * If running with .KEEP_STATE, curdir will be set with
445          * the connected directory.
446          */
447         (void) atexit(cleanup_after_exit);

449         load_cached_names();

451 /*
452  *      Set command line flags
453  */
454         setup_makeflags_argv();
455         read_command_options(mf_argc, mf_argv);
456         read_command_options(argc, argv);
457         if (debug_level > 0) {
458                 cp = getenv(makeflags->string_mb);
459                 (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
460         }

462         setup_interrupt(handle_interrupt);

464         read_files_and_state(argc, argv);

466 #ifdef TEAMWARE_MAKE_CMN
467         /*
468          * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
469          */
470         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
471         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
472         prop2 = get_prop(dmake_name2->prop, macro_prop);
473         if (prop2 == NULL) {
474                 /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
475                 output_mode = txt1_mode;
476         } else {
477                 dmake_value2 = prop2->body.macro.value;
478                 if ((dmake_value2 == NULL) ||
479                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
```



```
480                         output_mode = txt1_mode;
481                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
482                         output_mode = txt2_mode;
483                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))
484                         output_mode = html1_mode;
485                 } else {
486                         warning(catgets(catd, 1, 352, "Unsupported value ‘%s’ fo
487                               dmake_value2->string_mb);
488                 }
489         }
490         /*
491          * Find the dmake_mode: distributed, parallel, or serial.
492          */
493     if ((!pmake_cap_r_specified) &&
494         (!pmake_machinesfile_specified)) {
495         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
496         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
497         prop2 = get_prop(dmake_name2->prop, macro_prop);
498         if (prop2 == NULL) {
499                 /* DMAKE_MODE not defined, default to distributed mode */
500                 dmake_mode_type = distributed_mode;
501                 no_parallel = false;
502         } else {
503                 dmake_value2 = prop2->body.macro.value;
504                 if ((dmake_value2 == NULL) ||
505                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed"))
506                         dmake_mode_type = distributed_mode;
507                         no_parallel = false;
508                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel
509                         dmake_mode_type = parallel_mode;
510                         no_parallel = false;
511 #ifdef SGE_SUPPORT
512                         grid = false;
513                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("grid")))
514                         dmake_mode_type = parallel_mode;
515                         no_parallel = false;
516                         grid = true;
517 #endif
511                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
512                         dmake_mode_type = serial_mode;
513                         no_parallel = true;
514                 } else {
515                         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
516                 }
517         }

519         if ((!list_all_targets) &&
520             (report_dependencies_level == 0)) {
521                 /*
522                  * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
523                  * They could be defined in the env, in the makefile, or on the
524                  * command line.
525                  * If neither is defined, and $(HOME)/.dmakerc does not exists,
526                  * then print a message, and default to parallel mode.
527                  */
528 #ifdef DISTRIBUTED
529                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
530                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
531                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
532                 dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
533                 if ((((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
534                     ((dmake_value = prop->body.macro.value) == NULL)) &&
535                    (((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL)
536                     ((dmake_value2 = prop2->body.macro.value) == NULL))) {
537                         Boolean empty_dmakerc = true;
538                         char *homedir = getenv(NOCATGETS("HOME"));
```

```
539                         if ((homedir != NULL) && (strlen(homedir) < (sizeof(def_
540                                 sprintf(def_dmakerc_path, NOCATGETS("%s/.dmakerc
541                                 if ((((statval = stat(def_dmakerc_path, &statbuf
542                                         ((statval == 0) && (statbuf.st_size == 0
543                                 } else {
544                                         Avo_dmakerc     *rcfile = new Avo_dmaker
545                                         Avo_err         *err = rcfile->read(def_
546                                         if (err) {
547                                                 fatal(err->str);
548                                         }
549                                         empty_dmakerc = rcfile->was_empty();
550                                         delete rcfile;
551                                 }
552                         }
553                         if (empty_dmakerc) {
554                                 if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
555                                         putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE
556                                         (void) fprintf(stdout, catgets(catd, 1,
557                                         (void) fprintf(stdout, catgets(catd, 1,
558                                 }
559                                 dmake_mode_type = parallel_mode;
560                                 no_parallel = false;
561                         }
562                 }
563 #else
564                 if(dmake_mode_type == distributed_mode) {
565                         (void) fprintf(stdout, NOCATGETS("dmake: Distributed mod
566                         (void) fprintf(stdout, NOCATGETS("       Defaulting to p
567                         dmake_mode_type = parallel_mode;
568                         no_parallel = false;
569                 }
570 #endif  /* DISTRIBUTED */
571         }
572     }
573 #endif

575 #ifdef TEAMWARE_MAKE_CMN
576         parallel_flag = true;
577         /* XXX - This is a major hack for DMake/Licensing. */
578         if (getenv(NOCATGETS("DMAKE_CHILD")) == NULL) {
579                 if (!avo_cli_search_license(argv[0], dmake_exit_callback, TRUE,
580                         /*
581                          * If the user can not get a TeamWare license,
582                          * default to serial mode.
583                          */
584                         dmake_mode_type = serial_mode;
585                         no_parallel = true;
586                 } else {
587                         putenv(NOCATGETS("DMAKE_CHILD=TRUE"));
588                 }
589                 start_time = time(NULL);
590                 /*
591                  * XXX - Hack to disable SIGALRM's from licensing library's
592                  *       setitimer().
593                  */
594                 value.it_interval.tv_sec = 0;
595                 value.it_interval.tv_usec = 0;
596                 value.it_value.tv_sec = 0;
597                 value.it_value.tv_usec = 0;
598                 (void) setitimer(ITIMER_REAL, &value, NULL);
599         }

601 //
602 // If dmake is running with -t option, set dmake_mode_type to serial.
603 // This is done because doname() calls touch_command() that runs serially.
604 // If we do not do that, maketool will have problems.
```



```
605 //
606         if(touch) {
607                 dmake_mode_type = serial_mode;
608                 no_parallel = true;
609         }
610 #else
611         parallel_flag = false;
612 #endif

614 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
615         /*
616          * Check whether stdout and stderr are physically same.
617          * This is in order to decide whether we need to redirect
618          * stderr separately from stdout.
619          * This check is performed only if __DMAKE_SEPARATE_STDERR
620          * is not set. This variable may be used in order to preserve
621          * the 'old' behaviour.
622          */
623         out_err_same = true;
624         char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
625         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
626                 struct stat stdout_stat;
627                 struct stat stderr_stat;
628                 if( (fstat(1, &stdout_stat) == 0)
629                  && (fstat(2, &stderr_stat) == 0) )
630                 {
631                         if( (stdout_stat.st_dev != stderr_stat.st_dev)
632                          || (stdout_stat.st_ino != stderr_stat.st_ino) )
633                         {
634                                 out_err_same = false;
635                         }
636                 }
637         }
638 #endif

640 #ifdef DISTRIBUTED
641         /*
642          * At this point, DMake should startup an rxm with any and all
643          * DMake command line options. Rxm will, among other things,
644          * read the rc file.
645          */
646         if ((!list_all_targets) &&
647             (report_dependencies_level == 0) &&
648             (dmake_mode_type == distributed_mode)) {
649                 startup_rxm();
650         }
651 #endif
652
653 /*
654  *      Enable interrupt handler for alarms
655  */
656         (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

658 /*
659  *      Check if make should report
660  */
661         if (getenv(sunpro_dependencies->string_mb) != NULL) {
662                 FILE    *report_file;

664                 report_dependency("");
665                 report_file = get_report_file();
666                 if ((report_file != NULL) && (report_file != (FILE*)-1)) {
667                         (void) fprintf(report_file, "\n");
668                 }
669         }
```

```
 671 /*
 672  *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
 673  */
 674         if (keep_state) {
 675                 maybe_append_prop(sunpro_dependencies, macro_prop)->
 676                   body.macro.exported = true;
 677         } else {
 678                 maybe_append_prop(sunpro_dependencies, macro_prop)->
 679                   body.macro.exported = false;
 680         }

 682         working_on_targets = true;
 683         if (trace_status) {
 684                 dump_make_state();
 685                 fclose(stdout);
 686                 fclose(stderr);
 687                 exit_status = 0;
 688                 exit(0);
 689         }
 690         if (list_all_targets) {
 691                 dump_target_list();
 692                 fclose(stdout);
 693                 fclose(stderr);
 694                 exit_status = 0;
 695                 exit(0);
 696         }
 697         trace_reader = false;

 699         /*
 700          * Set temp_file_directory to the directory the .make.state
 701          * file is written to.
 702          */
 703         if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
 704                 temp_file_directory = strdup(get_current_path());
 705         } else {
 706                 *slash_ptr = (int) nul_char;
 707                 (void) strcpy(make_state_dir, make_state->string_mb);
 708                 *slash_ptr = (int) slash_char;
 709                    /* when there is only one slash and it's the first
 710                    ** character, make_state_dir should point to '/'.
 711                    */
 712                 if(make_state_dir[0] == '\0') {
 713                    make_state_dir[0] = '/';
 714                    make_state_dir[1] = '\0';
 715                 }
 716                 if (make_state_dir[0] == (int) slash_char) {
 717                         temp_file_directory = strdup(make_state_dir);
 718                 } else {
 719                         char    tmp_current_path2[MAXPATHLEN];
 720
 721                         (void) sprintf(tmp_current_path2,
 722                                        "%s/%s",
 723                                        get_current_path(),
 724                                        make_state_dir);
 725                         temp_file_directory = strdup(tmp_current_path2);
 726                 }
 727         }

 729 #ifdef DISTRIBUTED
 730         building_serial = false;
 731 #endif

 733         report_dir_enter_leave(true);

 735         make_targets(argc, argv, parallel_flag);
```



```
 737         report_dir_enter_leave(false);

 739         if (build_failed_ever_seen) {
 740                 if (posix) {
 741                         exit_status = 1;
 742                 }
 743                 exit(1);
 744         }
 745         exit_status = 0;
 746         exit(0);
 747         /* NOTREACHED */
 748 }
______unchanged_portion_omitted_

 950 /*
 951  *      handle_interrupt()
 952  *
 953  *      This is where C-C traps are caught.
 954  *
 955  *      Parameters:
 956  *
 957  *      Global variables used (except DMake 1.0):
 958  *              current_target          Sometimes the current target is removed
 959  *              do_not_exec_rule        But not if -n is on
 960  *              quest                   or -q
 961  *              running_list            List of parallel running processes
 962  *              touch                   Current target is not removed if -t on
 963  */
 964 void
 965 handle_interrupt(int)
 966 {
 967         Property                member;
 968         Running                 rp;

 970         (void) fflush(stdout);
 971 #ifdef DISTRIBUTED
 972         if (rxmPid > 0) {
 973                 // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
 974                 Avo_AcknowledgeMsg acknowledgeMsg;
 975                 RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;

 977                 int xdrResult = xdr(&xdrs_out, msg);

 979                 if (xdrResult) {
 980                         fflush(dmake_ofp);
 981                 } else {
 982                         kill(rxmPid, SIGTERM);
 983                         rxmPid = 0;
 984                 }
 985         }
 986 #endif
 987         if (childPid > 0) {
 988                 kill(childPid, SIGTERM);
 989                 childPid = -1;
 990         }
 991         for (rp = running_list; rp != NULL; rp = rp->next) {
 992                 if (rp->state != build_running) {
 993                         continue;
 994                 }
 995                 if (rp->pid > 0) {
 996                         kill(rp->pid, SIGTERM);
 997                         rp->pid = -1;
 998                 }
 999         }
1000         if (getpid() == getpgrp()) {
1001                 bsd_signal(SIGTERM, SIG_IGN);
```

```
1002                 kill (-getpid(), SIGTERM);
1003         }
1004 #ifdef TEAMWARE_MAKE_CMN
1005         /* Clean up all parallel/distributed children already finished */
1006         finish_children(false);
1007 #endif

1009         /* Make sure the processes running under us terminate first */

1011         while (wait((int *) NULL) != -1);
1012         /* Delete the current targets unless they are precious */
1013         if ((current_target != NULL) &&
1014             current_target->is_member &&
1015             ((member = get_prop(current_target->prop, member_prop)) != NULL)) {
1016                 current_target = member->body.member.library;
1017         }
1018         if (!do_not_exec_rule &&
1019             !touch &&
1020             !quest &&
1021             (current_target != NULL) &&
1022             !(current_target->stat.is_precious || all_precious)) {

1024 /* BID_1030811 */
1025 /* azv 16 Oct 95 */
1026                 current_target->stat.time = file_no_time;

1028                 if (exists(current_target) != file_doesnt_exist) {
1029                         (void) fprintf(stderr,
1030                                        "\n*** %s ",
1031                                        current_target->string_mb);
1032                         if (current_target->stat.is_dir) {
1033                                 (void) fprintf(stderr,
1034                                                catgets(catd, 1, 168, "not remove
1035                                                current_target->string_mb);
1036                         } else if (unlink(current_target->string_mb) == 0) {
1037                                 (void) fprintf(stderr,
1038                                                catgets(catd, 1, 169, "removed.\n
1039                                                current_target->string_mb);
1040                         } else {
1041                                 (void) fprintf(stderr,
1042                                                catgets(catd, 1, 170, "could not
1043                                                current_target->string_mb,
1044                                                errmsg(errno));
1045                         }
1046                 }
1047         }
1048         for (rp = running_list; rp != NULL; rp = rp->next) {
1049                 if (rp->state != build_running) {
1050                         continue;
1051                 }
1052                 if (rp->target->is_member &&
1053                     ((member = get_prop(rp->target->prop, member_prop)) !=
1054                      NULL)) {
1055                         rp->target = member->body.member.library;
1056                 }
1057                 if (!do_not_exec_rule &&
1058                     !touch &&
1059                     !quest &&
1060                     !(rp->target->stat.is_precious || all_precious)) {

1062                         rp->target->stat.time = file_no_time;
1063                         if (exists(rp->target) != file_doesnt_exist) {
1064                                 (void) fprintf(stderr,
1065                                                "\n*** %s ",
1066                                                rp->target->string_mb);
1067                                 if (rp->target->stat.is_dir) {
```



```
1068                                         (void) fprintf(stderr,
1069                                                        catgets(catd, 1, 171, "no
1070                                                        rp->target->string_mb);
1071                                 } else if (unlink(rp->target->string_mb) == 0) {
1072                                         (void) fprintf(stderr,
1073                                                        catgets(catd, 1, 172, "re
1074                                                        rp->target->string_mb);
1075                                 } else {
1076                                         (void) fprintf(stderr,
1077                                                        catgets(catd, 1, 173, "co
1078                                                        rp->target->string_mb,
1079                                                        errmsg(errno));
1080                                 }
1081                         }
1082                 }
1083         }

1092 #ifdef SGE_SUPPORT
1093         /* Remove SGE script file */
1094         if (grid) {
1095                 unlink(script_file);
1096         }
1097 #endif

1086         /* Have we locked .make.state or .nse_depinfo? */
1087         if ((make_state_lockfile != NULL) && (make_state_locked)) {
1088                 unlink(make_state_lockfile);
1089                 make_state_lockfile = NULL;
1090                 make_state_locked = false;
1091         }
1092         /*
1093          * Re-read .make.state file (it might be changed by recursive make)
1094          */
1095         check_state(NULL);

1097         report_dir_enter_leave(false);

1099         exit_status = 2;
1100         exit(2);
1101 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   53764 Wed May 20 11:37:22 2015
new/usr/src/cmd/make/bin/parallel.cc
make: unifdef for SGE (undefined)
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

#ifdef TEAMWARE_MAKE_CMN

/*
 *      parallel.cc
 *
 *      Deal with the parallel processing
 */

/*
 * Included files
 */
#ifdef DISTRIBUTED
#include <avo/strings.h>        /* AVO_STRDUP() */
#include <dm/Avo_DoJobMsg.h>
#include <dm/Avo_MToolJobResultMsg.h>
#endif
#include <errno.h>              /* errno */
#include <fcntl.h>
#include <avo/util.h>           /* avo_get_user(), avo_hostname() */
#include <mk/defs.h>
#include <mksh/dosys.h>         /* redirect_io() */
#include <mksh/macro.h>         /* expand_value() */
#include <mksh/misc.h>          /* getmem() */
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>
#include <sys/wait.h>
#include <unistd.h>

#ifdef SGE_SUPPORT
#include <dmthread/Avo_PathNames.h>
#endif


/*
```



```
 * Defined macros
 */
#define MAXRULES                100

/*
 * This const should be in avo_dms/include/AvoDmakeCommand.h
 */
const int local_host_mask = 0x20;


/*
 * typedefs & structs
 */


/*
 * Static variables
 */
#ifdef TEAMWARE_MAKE_CMN
static  Boolean         just_did_subtree = false;
static  char            local_host[MAXNAMELEN] = "";
static  char            user_name[MAXNAMELEN] = "";
#endif
static  int             pmake_max_jobs = 0;
static  pid_t           process_running = -1;
static  Running         *running_tail = &running_list;
static  Name            subtree_conflict;
static  Name            subtree_conflict2;


/*
 * File table of contents
 */
#ifdef DISTRIBUTED
static  void            append_dmake_cmd(Avo_DoJobMsg *dmake_job_msg, char *orig
static  void            append_job_result_msg(Avo_MToolJobResultMsg *msg, char *
static  void            send_job_result_msg(Running rp);
#endif
static  void            delete_running_struct(Running rp);
static  Boolean         dependency_conflict(Name target);
static  Doname          distribute_process(char **commands, Property line);
static  void            doname_subtree(Name target, Boolean do_get, Boolean impl
static  void            dump_out_file(char *filename, Boolean err);
static  void            finish_doname(Running rp);
static  void            maybe_reread_make_state(void);
static  void            process_next(void);
static  void            reset_conditionals(int cnt, Name *targets, Property *loc
static  pid_t           run_rule_commands(char *host, char **commands);
static  Property        *set_conditionals(int cnt, Name *targets);
static  void            store_conditionals(Running rp);


/*
 *      execute_parallel(line, waitflg)
 *
 *      DMake 2.x:
 *      parallel mode: spawns a parallel process to execute the command group.
 *      distributed mode: sends the command group down the pipe to rxm.
 *
 *      Return value:
 *                              The result of the execution
 *
 *      Parameters:
 *              line            The command group to execute
 */
Doname
```

```
execute_parallel(Property line, Boolean waitflg, Boolean local)
{
        int                     argcnt;
        int                     cmd_options = 0;
        char                    *commands[MAXRULES + 5];
        char                    *cp;
#ifdef DISTRIBUTED
        Avo_DoJobMsg            *dmake_job_msg = NULL;
#endif
        Name                    dmake_name;
        Name                    dmake_value;
        int                     ignore;
        Name                    make_machines_name;
        char                    **p;
        Property                prop;
        Doname                  result = build_ok;
        Cmd_line                rule;
        Boolean                 silent_flag;
        Name                    target = line->body.line.target;
        Boolean                 wrote_state_file = false;

        if ((pmake_max_jobs == 0) &&
            (dmake_mode_type == parallel_mode)) {
                if (user_name[0] == '\0') {
                        avo_get_user(user_name, NULL);
                }
                if (local_host[0] == '\0') {
                        strcpy(local_host, avo_hostname());
                }
                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
                dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
                if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
                    ((dmake_value = prop->body.macro.value) != NULL)) {
                        pmake_max_jobs = atoi(dmake_value->string_mb);
                        if (pmake_max_jobs <= 0) {
                                warning(catgets(catd, 1, 308, "DMAKE_MAX_JOBS ca
                                warning(catgets(catd, 1, 309, "setting DMAKE_MAX
                                pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
                        }
                } else {
                        /*
                         * For backwards compatibility w/ PMake 1.x, when
                         * DMake 2.x is being run in parallel mode, DMake
                         * should parse the PMake startup file
                         * $(HOME)/.make.machines to get the pmake_max_jobs.
                         */
                        MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
                        dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
                        if (((prop = get_prop(dmake_name->prop, macro_prop)) !=
                            ((dmake_value = prop->body.macro.value) != NULL)) {
                                make_machines_name = dmake_value;
                        } else {
                                make_machines_name = NULL;
                        }
                        if ((pmake_max_jobs = read_make_machines(make_machines_n
                                pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
                        }
                }
#ifdef DISTRIBUTED
                if (send_mtool_msgs) {
                        send_rsrc_info_msg(pmake_max_jobs, local_host, user_name
                }
#endif
        }

        if ((dmake_mode_type == serial_mode) ||
```



```
            ((dmake_mode_type == parallel_mode) && (waitflg))) {
                return (execute_serial(line));
        }

#ifdef DISTRIBUTED
        if (dmake_mode_type == distributed_mode) {
                if(local) {
//                      return (execute_serial(line));
                        waitflg = true;
                }
                dmake_job_msg = new Avo_DoJobMsg();
                dmake_job_msg->setJobId(++job_msg_id);
                dmake_job_msg->setTarget(target->string_mb);
                dmake_job_msg->setImmediateOutput(0);
                called_make = false;
        } else
#endif
        {
                p = commands;
        }

        argcnt = 0;
        for (rule = line->body.line.command_used;
             rule != NULL;
             rule = rule->next) {
                if (posix && (touch || quest) && !rule->always_exec) {
                        continue;
                }
                if (vpath_defined) {
                        rule->command_line =
                          vpath_translation(rule->command_line);
                }
                if (dmake_mode_type == distributed_mode) {
                        cmd_options = 0;
                        if(local) {
                                cmd_options |= local_host_mask;
                        }
                } else {
                        silent_flag = false;
                        ignore = 0;
                }
                if (rule->command_line->hash.length > 0) {
                        if (++argcnt == MAXRULES) {
                                if (dmake_mode_type == distributed_mode) {
                                        /* XXX - tell rxm to execute on local ho
                                        /* I WAS HERE!!! */
                                } else {
                                        /* Too many rules, run serially instead.
                                        return build_serial;
                                }
                        }
#ifdef DISTRIBUTED
                        if (dmake_mode_type == distributed_mode) {
                                /*
                                 * XXX - set assign_mask to tell rxm
                                 * to do the following.
                                 */
/* From execute_serial():
                                if (rule->assign) {
                                        result = build_ok;
                                        do_assign(rule->command_line, target);
 */
                                if (0) {
                                } else if (report_dependencies_level == 0) {
                                        if (rule->ignore_error) {
                                                cmd_options |= ignore_mask;
```

```
257                                     }
258                                     if (rule->silent) {
259                                             cmd_options |= silent_mask;
260                                     }
261                                     if (rule->command_line->meta) {
262                                             cmd_options |= meta_mask;
263                                     }
264                                     if (rule->make_refd) {
265                                             cmd_options |= make_refd_mask;
266                                     }
267                                     if (do_not_exec_rule) {
268                                             cmd_options |= do_not_exec_mask;
269                                     }
270                                     append_dmake_cmd(dmake_job_msg,
271                                                      rule->command_line->str
272                                                      cmd_options);
273                                     /* Copying dosys()... */
274                                     if (rule->make_refd) {
275                                             if (waitflg) {
276                                                     dmake_job_msg->setImmedi
277                                             }
278                                             called_make = true;
279                                             if (command_changed &&
280                                                 !wrote_state_file) {
281                                                     write_state_file(0, fals
282                                                     wrote_state_file = true;
283                                             }
284                                     }
285                             }
286                     } else
287 #endif
288                     {
289                             if (rule->silent && !silent) {
290                                     silent_flag = true;
291                             }
292                             if (rule->ignore_error) {
293                                     ignore++;
294                             }
295                             /* XXX - need to add support for + prefix */
296                             if (silent_flag || ignore) {
297                                     *p = getmem((silent_flag ? 1 : 0) +
298                                                 ignore +
299                                                 (strlen(rule->
300                                                         command_line->
301                                                         string_mb)) +
302                                                 1);
303                                     cp = *p++;
304                                     if (silent_flag) {
305                                             *cp++ = (int) at_char;
306                                     }
307                                     if (ignore) {
308                                             *cp++ = (int) hyphen_char;
309                                     }
310                                     (void) strcpy(cp, rule->command_line->st
311                             } else {
312                                     *p++ = rule->command_line->string_mb;
313                             }
314                     }
315             }
316         }
317         if ((argcnt == 0) ||
318             (report_dependencies_level > 0)) {
319 #ifdef DISTRIBUTED
320                 if (dmake_job_msg) {
321                         delete dmake_job_msg;
322                 }
```



```
323 #endif
324                 return build_ok;
325         }
326 #ifdef DISTRIBUTED
327         if (dmake_mode_type == distributed_mode) {
328                 // Send  a DoJob message to the rxm process.
329                 distribute_rxm(dmake_job_msg);
330
331                 // Wait for an acknowledgement.
332                 Avo_AcknowledgeMsg *ackMsg = getAcknowledgeMsg();
333                 if (ackMsg) {
334                         delete ackMsg;
335                 }
336
337                 if (waitflg) {
338                         // Wait for, and process a job result.
339                         result = await_dist(waitflg);
340                         if (called_make) {
341                                 maybe_reread_make_state();
342                         }
343                         check_state(temp_file_name);
344                         if (result == build_failed) {
345                                 if (!continue_after_error) {
346
347 #ifdef PRINT_EXIT_STATUS
348                                         warning(NOCATGETS("I'm in execute_parall
349 #endif
350
351                                         fatal(catgets(catd, 1, 252, "Command fai
352                                               target->string_mb);
353                                 }
354                                 /*
355                                  * Make sure a failing command is not
356                                  * saved in .make.state.
357                                  */
358                                 line->body.line.command_used = NULL;
359                         }
360                         if (temp_file_name != NULL) {
361                                 free_name(temp_file_name);
362                         }
363                         temp_file_name = NULL;
364                         Property spro = get_prop(sunpro_dependencies->prop, macr
365                         if(spro != NULL) {
366                                 Name val = spro->body.macro.value;
367                                 if(val != NULL) {
368                                         free_name(val);
369                                         spro->body.macro.value = NULL;
370                                 }
371                         }
372                         spro = get_prop(sunpro_dependencies->prop, env_mem_prop)
373                         if(spro) {
374                                 char *val = spro->body.env_mem.value;
375                                 if(val != NULL) {
376                                         retmem_mb(val);
377                                         spro->body.env_mem.value = NULL;
378                                 }
379                         }
380                         return result;
381                 } else {
382                         parallel_process_cnt++;
383                         return build_running;
384                 }
385         } else
386 #endif
387         {
388                 *p = NULL;
```

```
390                 Doname res = distribute_process(commands, line);
391                 if (res == build_running) {
392                         parallel_process_cnt++;
393                 }
395                 /*
396                  * Return only those memory that were specially allocated
397                  * for part of commands.
398                  */
399                 for (int i = 0; commands[i] != NULL; i++) {
400                         if ((commands[i][0] == (int) at_char) ||
401                             (commands[i][0] == (int) hyphen_char)) {
402                                 retmem_mb(commands[i]);
403                         }
404                 }
405                 return res;
406         }
407 }
```

*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) {
```

```
                                tmp_index_str_ptr++;
                        }
                        (void) fprintf(stdout, "%s\n", tmp_index_str_ptr);
                    }
                    tmp_index++;
                }
                (void) fflush(stdout);
        }

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

        mktemp(mbstring);

        stdout_file = strdup(mbstring);
        stderr_file = NULL;
#if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
        if (!out_err_same) {
                (void) sprintf(mbstring,
                                NOCATGETS("%s/dmake.stderr.%d.%d.XXXXXX"),
                                tmpdir,
                                getpid(),
                                file_number++);

                mktemp(mbstring);

                stderr_file = strdup(mbstring);
        }
#endif

#ifdef SGE_SUPPORT
        if (grid) {
                static char *dir4gridscripts = NULL;
                static char *hostName = NULL;
                if (dir4gridscripts == NULL) {
                        Name            dmakeOdir_name, dmakeOdir_value;
                        Property        prop;
                        MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
                        dmakeOdir_name = GETNAME(wcs_buffer, FIND_LENGTH);
                        if (((prop = get_prop(dmakeOdir_name->prop, macro_prop))
                            ((dmakeOdir_value = prop->body.macro.value) != NULL)
                                dir4gridscripts = dmakeOdir_value->string_mb;
                        }
                        dir4gridscripts = Avo_PathNames::pathname_output_directo
                        hostName = Avo_PathNames::pathname_local_host();
                }
                (void) sprintf(script_file,
                                NOCATGETS("%s/dmake.script.%s.%d.%d.XXXXXX"),
                                dir4gridscripts,
                                hostName,
                                getpid(),
                                file_number++);
        }
#endif /* SGE_SUPPORT */
        process_running = run_rule_commands(local_host, commands);

        return build_running;
}
```

unchanged_portion_omitted_

```
/*
 * This function replaces the makesh binary.
 */
```



```

#ifdef SGE_SUPPORT
#define DO_CHECK(f)     if (f <= 0) { \
                                fprintf(stderr, \
                                        catgets(catd, 1, 347, "Could not write t
                                                script_file, errmsg(errno)); \
                                _exit(1); \
                        }
#endif /* SGE_SUPPORT */

static pid_t
run_rule_commands(char *host, char **commands)
{
        Boolean         always_exec;
        Name            command;
        Boolean         ignore;
        int             length;
        Doname          result;
        Boolean         silent_flag;
#ifdef SGE_SUPPORT
        wchar_t         *wcmd, *tmp_wcs_buffer = NULL;
        char            *cmd, *tmp_mbs_buffer = NULL;
        FILE            *scrfp;
        Name            shell = getvar(shell_name);
#else
        wchar_t         *tmp_wcs_buffer;
#endif /* SGE_SUPPORT */

        childPid = fork();
        switch (childPid) {
        case -1:        /* Error */
                fatal(catgets(catd, 1, 337, "Could not fork child process for dm
                      errmsg(errno));
                break;
        case 0:         /* Child */
                /* To control the processed targets list is not the child's busi
                running_list = NULL;
#if defined(REDIRECT_ERR)
                if(out_err_same) {
                        redirect_io(stdout_file, (char*)NULL);
                } else {
                        redirect_io(stdout_file, stderr_file);
                }
#else
                redirect_io(stdout_file, (char*)NULL);
#endif
#ifdef SGE_SUPPORT
                if (grid) {
                        int fdes = mkstemp(script_file);
                        if ((fdes < 0) || (scrfp = fdopen(fdes, "w")) == NULL) {
                                fprintf(stderr,
                                        catgets(catd, 1, 341, "Could not create
                                        script_file, errmsg(errno));
                                _exit(1);
                        }
                        if (IS_EQUAL(shell->string_mb, "")) {
                                shell = shell_name;
                        }
                }
#endif /* SGE_SUPPORT */
                for (commands = commands;
                     (*commands != (char *)NULL);
                     commands++) {
                        silent_flag = silent;
                        ignore = false;
                        always_exec = false;
```

```
2062                        while ((**commands == (int) at_char) ||
2063                               (**commands == (int) hyphen_char) ||
2064                               (**commands == (int) plus_char)) {
2065                                if (**commands == (int) at_char) {
2066                                        silent_flag = true;
2067                                }
2068                                if (**commands == (int) hyphen_char) {
2069                                        ignore = true;
2070                                }
2071                                if (**commands == (int) plus_char) {
2072                                        always_exec = true;
2073                                }
2074                                (*commands)++;
2075                        }
2132 #ifdef SGE_SUPPORT
2133                        if (grid) {
2134                                if ((length = strlen(*commands)) >= MAXPATHLEN /
2135                                        wcmd = tmp_wcs_buffer = ALLOC_WC(length
2136                                        (void) mbstowcs(tmp_wcs_buffer, *command
2137                                } else {
2138                                        MBSTOWCS(wcs_buffer, *commands);
2139                                        wcmd = wcs_buffer;
2140                                        cmd = mbs_buffer;
2141                                }
2142                                wchar_t *from = wcmd + wslen(wcmd);
2143                                wchar_t *to = from + (from - wcmd);
2144                                *to = (int) nul_char;
2145                                while (from > wcmd) {
2146                                        *--to = *--from;
2147                                        if (*from == (int) newline_char) { // ne
2148                                                *--to = *--from;
2149                                        } else if (wschr(char_semantics_char, *f
2150                                                *--to = (int) backslash_char;
2151                                        }
2152                                }
2153                                if (length >= MAXPATHLEN*MB_LEN_MAX/2) { // size
2154                                        cmd = tmp_mbs_buffer = getmem((length *
2155                                        (void) wcstombs(tmp_mbs_buffer, to, (len
2156                                } else {
2157                                        WCSTOMBS(mbs_buffer, to);
2158                                        cmd = mbs_buffer;
2159                                }
2160                                char *mbst, *mbend;
2161                                if ((length > 0) &&
2162                                    !silent_flag) {
2163                                        for (mbst = cmd; (mbend = strstr(mbst, "
2164                                                *mbend = '\0';
2165                                                DO_CHECK(fprintf(scrfp, NOCATGET
2166                                                *mbend = '\\';
2167                                        }
2168                                        DO_CHECK(fprintf(scrfp, NOCATGETS("/usr/
2169                                }
2170                                if (!do_not_exec_rule ||
2171                                    !working_on_targets ||
2172                                    always_exec) {
2173                                        DO_CHECK(fprintf(scrfp, NOCATGETS("%s -c
2174                                        DO_CHECK(fputs(NOCATGETS("__DMAKECMDEXIT
2175                                        if (ignore) {
2176                                                DO_CHECK(fprintf(scrfp, NOCATGET
2177                                                        catgets(catd, 1, 343, "\
2178                                                        catgets(catd, 1, 344, "(
2179                                        } else {
2180                                                DO_CHECK(fprintf(scrfp, NOCATGET
2181                                                        catgets(catd, 1, 342, "\
2182                                        }
2183                                        if (silent_flag) {
```



```
2184                                                DO_CHECK(fprintf(scrfp, NOCATGET
2185                                                        catgets(catd, 1, 345, "T
2186                                                for (mbst = cmd; (mbend = strstr
2187                                                        *mbend = '\0';
2188                                                        DO_CHECK(fprintf(scrfp,
2189                                                        *mbend = '\\';
2190                                                }
2191                                                DO_CHECK(fprintf(scrfp, NOCATGET
2192                                        }
2193                                        if (!ignore) {
2194                                                DO_CHECK(fputs(NOCATGETS("\texit
2195                                        }
2196                                        DO_CHECK(fputs(NOCATGETS("fi\n"), scrfp)
2197                                }
2198                                if (tmp_wcs_buffer) {
2199                                        retmem_mb(tmp_mbs_buffer);
2200                                        tmp_mbs_buffer = NULL;
2201                                }
2202                                if (tmp_wcs_buffer) {
2203                                        retmem(tmp_wcs_buffer);
2204                                        tmp_wcs_buffer = NULL;
2205                                }
2206                                continue;
2207                        }
2208 #endif /* SGE_SUPPORT */
2076                        if ((length = strlen(*commands)) >= MAXPATHLEN) {
2077                                tmp_wcs_buffer = ALLOC_WC(length + 1);
2078                                (void) mbstowcs(tmp_wcs_buffer, *commands, lengt
2079                                command = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2080                                retmem(tmp_wcs_buffer);
2081                        } else {
2082                                MBSTOWCS(wcs_buffer, *commands);
2083                                command = GETNAME(wcs_buffer, FIND_LENGTH);
2084                        }
2085                        if ((command->hash.length > 0) &&
2086                            !silent_flag) {
2087                                (void) printf("%s\n", command->string_mb);
2088                        }
2089                        result = dosys(command,
2090                                       ignore,
2091                                       false,
2092                                       false, /* bugs #4085164 & #4990057 */
2093                                       /* BOOLEAN(silent_flag && ignore), */
2094                                       always_exec,
2095                                       (Name) NULL,
2096                                       false);
2097                        if (result == build_failed) {
2098                                if (silent_flag) {
2099                                        (void) printf(catgets(catd, 1, 152, "The
2100                                }
2101                                if (!ignore) {
2102                                        _exit(1);
2103                                }
2104                        }
2105                }
2239 #ifndef SGE_SUPPORT
2240                _exit(0);
2241 #else
2242                if (!grid) {
2106                _exit(0);
2244                }
2245                DO_CHECK(fputs(NOCATGETS("exit 0\n"), scrfp));
2246                if (fclose(scrfp) != 0) {
2247                        fprintf(stderr,
2248                                catgets(catd, 1, 346, "Could not close file: %s:
2249                                script_file, errmsg(errno));
```

```
2250                        _exit(1);
2251                }
2252                {

2254 #define DEFAULT_QRSH_TRIES_NUMBER       1
2255 #define DEFAULT_QRSH_TIMEOUT            0

2257                static char     *sge_env_var = NULL;
2258                static int      qrsh_tries_number = DEFAULT_QRSH_TRIES_NUMBER;
2259                static int      qrsh_timeout = DEFAULT_QRSH_TIMEOUT;
2260 #define SGE_DEBUG
2261 #ifdef SGE_DEBUG
2262                static  Boolean do_not_remove = false;
2263 #endif /* SGE_DEBUG */
2264                if (sge_env_var == NULL) {
2265                        sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_TRIES")
2266                        if (sge_env_var != NULL) {
2267                                qrsh_tries_number = atoi(sge_env_var);
2268                                if (qrsh_tries_number < 1 || qrsh_tries_number >
2269                                        qrsh_tries_number = DEFAULT_QRSH_TRIES_N
2270                                }
2271                        }
2272                        sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_TIMEOUT
2273                        if (sge_env_var != NULL) {
2274                                qrsh_timeout = atoi(sge_env_var);
2275                                if (qrsh_timeout <= 0) {
2276                                        qrsh_timeout = DEFAULT_QRSH_TIMEOUT;
2277                                }
2278                        } else {
2279                                sge_env_var = "";
2280                        }
2281 #ifdef SGE_DEBUG
2282                        sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_DEBUG")
2283                        if (sge_env_var == NULL) {
2284                                sge_env_var = "";
2285                        }
2286                        if (strstr(sge_env_var, NOCATGETS("noqrsh")) != NULL)
2287                                qrsh_tries_number = 0;
2288                        if (strstr(sge_env_var, NOCATGETS("donotremove")) != NUL
2289                                do_not_remove = true;
2290 #endif /* SGE_DEBUG */
2291                }
2292                for (int i = qrsh_tries_number; ; i--)
2293                if ((childPid = fork()) < 0) {
2294                        fatal(catgets(catd, 1, 348, "Could not fork child proces
2295                                errmsg(errno));
2296                        _exit(1);
2297                } else if (childPid == 0) {
2298                        enable_interrupt((void (*) (int))SIG_DFL);
2299                        if (i > 0) {
2300                                static char qrsh_cmd[50+MAXPATHLEN] = NOCATGETS(
2301                                static char *fname_ptr = NULL;
2302                                static char *argv[] = { NOCATGETS("sh"),
2303                                                        NOCATGETS("-fce"),
2304                                                        qrsh_cmd,
2305                                                        NULL};
2306                                if (fname_ptr == NULL) {
2307                                        fname_ptr = qrsh_cmd + strlen(qrsh_cmd);
2308                                }
2309                                strcpy(fname_ptr, script_file);
2310                                (void) execve(NOCATGETS("/bin/sh"), argv, enviro
2311                        } else {
2312                                static char *argv[] = { NOCATGETS("sh"),
2313                                                        script_file,
2314                                                        NULL};
2315                                (void) execve(NOCATGETS("/bin/sh"), argv, enviro
```



```
2316                        }
2317                        fprintf(stderr,
2318                                catgets(catd, 1, 349, "Could not load ‘qrsh’: %s
2319                                errmsg(errno));
2320                        _exit(1);
2321                } else {
2322                        int             status;
2323                        pid_t pid;
2324                        while ((pid = wait(&status)) != childPid) {
2325                                if (pid == -1) {
2326                                        fprintf(stderr,
2327                                                catgets(catd, 1, 350, "wait() fa
2328                                                errmsg(errno));
2329                                        _exit(1);
2330                                }
2331                        }
2332                        if (status != 0 && i > 0) {
2333                                if (i > 1) {
2334                                        sleep(qrsh_timeout);
2335                                }
2336                                continue;
2337                        }
2338 #ifdef SGE_DEBUG
2339                        if (do_not_remove) {
2340                                if (status) {
2341                                        fprintf(stderr,
2342                                                NOCATGETS("SGE script failed: %s
2343                                                script_file);
2344                                }
2345                                _exit(status ? 1 : 0);
2346                        }
2347 #endif /* SGE_DEBUG */
2348                        (void) unlink(script_file);
2349                        _exit(status ? 1 : 0);
2350                }
2351                }
2352 #endif /* SGE_SUPPORT */
2107                break;
2108        default:
2109                break;
2110        }
2111        return childPid;
2112 }
```

_____***unchanged_portion_omitted_***

****
15375 Wed May 20 11:37:23 2015
new/usr/src/cmd/make/include/mk/defs.h
make: unifdef for SGE (undefined)
****
_____***unchanged_portion_omitted_***

```
177 /*
178  * Typedefs for all structs
179  */
180 typedef struct _Cmd_line        *Cmd_line, Cmd_line_rec;
181 typedef struct _Dependency      *Dependency, Dependency_rec;
182 typedef struct _Macro           *Macro, Macro_rec;
183 typedef struct _Name_vector     *Name_vector, Name_vector_rec;
184 typedef struct _Percent         *Percent, Percent_rec;
185 typedef struct _Dyntarget       *Dyntarget;
186 typedef struct _Recursive_make  *Recursive_make, Recursive_make_rec;
187 typedef struct _Running         *Running, Running_rec;


190 /*
191  *      extern declarations for all global variables.
192  *      The actual declarations are in globals.cc
193  */
194 extern  Boolean         allrules_read;
195 extern  Name            posix_name;
196 extern  Name            svr4_name;
197 extern  Boolean         sdot_target;
198 extern  Boolean         all_parallel;
199 extern  Boolean         assign_done;
200 extern  Boolean         build_failed_seen;
201 #ifdef DISTRIBUTED
202 extern  Boolean         building_serial;
203 #endif
204 extern  Name            built_last_make_run;
205 extern  Name            c_at;
206 #ifdef DISTRIBUTED
207 extern  Boolean         called_make;
208 #endif
209 extern  Boolean         command_changed;
210 extern  Boolean         commands_done;
211 extern  Chain           conditional_targets;
212 extern  Name            conditionals;
213 extern  Boolean         continue_after_error;
214 extern  Property        current_line;
215 extern  Name            current_make_version;
216 extern  Name            current_target;
217 extern  short           debug_level;
218 extern  Cmd_line        default_rule;
219 extern  Name            default_rule_name;
220 extern  Name            default_target_to_build;
221 extern  Boolean         depinfo_already_read;
222 extern  Name            dmake_group;
223 extern  Name            dmake_max_jobs;
224 extern  Name            dmake_mode;
225 extern  DMake_mode      dmake_mode_type;
226 extern  Name            dmake_output_mode;
227 extern  DMake_output_mode       output_mode;
228 extern  Name            dmake_odir;
229 extern  Name            dmake_rcfile;
230 extern  Name            done;
231 extern  Name            dot;
232 extern  Name            dot_keep_state;
233 extern  Name            dot_keep_state_file;
234 extern  Name            empty_name;
```



```
235 extern  Boolean         fatal_in_progress;
236 extern  int             file_number;
237 extern  Name            force;
238 extern  Name            ignore_name;
239 extern  Boolean         ignore_errors;
240 extern  Boolean         ignore_errors_all;
241 extern  Name            init;
242 extern  int             job_msg_id;
243 extern  Boolean         keep_state;
244 extern  Name            make_state;
245 #ifdef TEAMWARE_MAKE_CMN
246 extern  timestruc_t     make_state_before;
247 #endif
248 extern  Boolean         make_state_locked;
249 extern  Dependency      makefiles_used;
250 extern  Name            makeflags;
251 extern  Name            make_version;
252 extern  char            mbs_buffer2[];
253 extern  char            *mbs_ptr;
254 extern  char            *mbs_ptr2;
255 extern  Boolean         no_action_was_taken;
256 extern  int             mtool_msgs_fd;
257 extern  Boolean         no_parallel;
258 #ifdef SGE_SUPPORT
259 extern  Boolean         grid;
260 #endif
258 extern  Name            no_parallel_name;
259 extern  Name            not_auto;
260 extern  Boolean         only_parallel;
261 extern  Boolean         parallel;
262 extern  Name            parallel_name;
263 extern  Name            localhost_name;
264 extern  int             parallel_process_cnt;
265 extern  Percent         percent_list;
266 extern  Dyntarget       dyntarget_list;
267 extern  Name            plus;
268 extern  Name            pmake_machinesfile;
269 extern  Name            precious;
270 extern  Name            primary_makefile;
271 extern  Boolean         quest;
272 extern  short           read_trace_level;
273 extern  Boolean         reading_dependencies;
274 extern  int             recursion_level;
275 extern  Name            recursive_name;
276 extern  short           report_dependencies_level;
277 extern  Boolean         report_pwd;
278 extern  Boolean         rewrite_statefile;
279 extern  Running         running_list;
280 extern  char            *sccs_dir_path;
281 extern  Name            sccs_get_name;
282 extern  Name            sccs_get_posix_name;
283 extern  Cmd_line        sccs_get_rule;
284 extern  Cmd_line        sccs_get_org_rule;
285 extern  Cmd_line        sccs_get_posix_rule;
286 extern  Name            get_name;
287 extern  Name            get_posix_name;
288 extern  Cmd_line        get_rule;
289 extern  Cmd_line        get_posix_rule;
290 extern  Boolean         send_mtool_msgs;
291 extern  Boolean         all_precious;
292 extern  Boolean         report_cwd;
293 extern  Boolean         silent_all;
294 extern  Boolean         silent;
295 extern  Name            silent_name;
296 extern  char            *stderr_file;
297 extern  char            *stdout_file;
```

```
#ifdef SGE_SUPPORT
extern  char            script_file[];
#endif
extern  Boolean         stdout_stderr_same;
extern  Dependency      suffixes;
extern  Name            suffixes_name;
extern  Name            sunpro_dependencies;
extern  Boolean         target_variants;
extern  const char      *tmpdir;
extern  const char      *temp_file_directory;
extern  Name            temp_file_name;
extern  short           temp_file_number;
extern  wchar_t         *top_level_target;
extern  Boolean         touch;
extern  Boolean         trace_reader;
extern  Boolean         build_unconditional;
extern  pathpt          vroot_path;
extern  Name            wait_name;
extern  wchar_t         wcs_buffer2[];
extern  wchar_t         *wcs_ptr;
extern  wchar_t         *wcs_ptr2;
extern  nl_catd         catd;
extern  long int        hostid;

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

/*
 * Declarations of system supplied functions
 */
extern  int             file_lock(char *, char *, int *, int);

/*
 * Declarations of functions declared and used by make
 */
extern  void            add_pending(Name target, int recursion_level, Boolean do
extern  void            add_running(Name target, Name true_target, Property comm
extern  void            add_serial(Name target, int recursion_level, Boolean do_
extern  void            add_subtree(Name target, int recursion_level, Boolean do
extern  void            append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar
#ifdef DISTRIBUTED
extern  Doname          await_dist(Boolean waitflg);
#endif
#ifdef TEAMWARE_MAKE_CMN
extern  void            await_parallel(Boolean waitflg);
#endif
extern  void            build_suffix_list(Name target_suffix);
extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
extern  void            check_state(Name temp_file_name);
extern  void            cond_macros_into_string(Name np, String_rec *buffer);
extern  void            construct_target_string();
extern  void            create_xdrs_ptr(void);
extern  void            depvar_add_to_list (Name name, Boolean cmdline);
#ifdef DISTRIBUTED
extern  void            distribute_rxm(Avo_DoJobMsg *dmake_job_msg);
extern  int             getRxmMessage(void);
extern  Avo_JobResultMsg* getJobResultMsg(void);
extern  Avo_AcknowledgeMsg* getAcknowledgeMsg(void);
#endif
extern  Doname          doname(register Name target, register Boolean do_get, re
extern  Doname          doname_check(register Name target, register Boolean do_g
extern  Doname          doname_parallel(Name target, Boolean do_get, Boolean imp
extern  Doname          dosys(register Name command, register Boolean ignore_err
```



```
extern  void            dump_make_state(void);
extern  void            dump_target_list(void);
extern  void            enter_conditional(register Name target, Name name, Name
extern  void            enter_dependencies(register Name target, Chain target_gr
extern  void            enter_dependency(Property line, register Name depe, Bool
extern  void            enter_equal(Name name, Name value, register Boolean appe
extern  Percent         enter_percent(register Name target, Chain target_group,
extern  Dyntarget       enter_dyntarget(register Name target);
extern  Name_vector     enter_name(String string, Boolean tail_present, register
extern  Boolean         exec_vp(register char *name, register char **argv, char
extern  Doname          execute_parallel(Property line, Boolean waitflg, Boolean
extern  Doname          execute_serial(Property line);
extern  timestruc_t&    exists(register Name target);
extern  void            fatal(char *, ...);
extern  void            fatal_reader(char *, ...);
extern  Doname          find_ar_suffix_rule(register Name target, Name true_targ
extern  Doname          find_double_suffix_rule(register Name target, Property *
extern  Doname          find_percent_rule(register Name target, Property *comman
extern  int             find_run_directory (char *cmd, char *cwd, char *dir, cha
extern  Doname          find_suffix_rule(Name target, Name target_body, Name tar
extern  Chain           find_target_groups(register Name_vector target_list, reg
extern  void            finish_children(Boolean docheck);
extern  void            finish_running(void);
extern  void            free_chain(Name_vector ptr);
extern  void            gather_recursive_deps(void);
extern  char            *get_current_path(void);
extern  int             get_job_msg_id(void);
extern  FILE            *get_mtool_msgs_fp(void);
#ifdef DISTRIBUTED
extern  Boolean         get_dmake_group_specified(void);
extern  Boolean         get_dmake_max_jobs_specified(void);
extern  Boolean         get_dmake_mode_specified(void);
extern  Boolean         get_dmake_odir_specified(void);
extern  Boolean         get_dmake_rcfile_specified(void);
extern  Boolean         get_pmake_machinesfile_specified(void);
#endif
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
extern  XDR             *get_xdrs_ptr(void);
#endif
extern  wchar_t         *getmem_wc(register int size);
/* On linux getwd(char *) is defined in 'unistd.h' */
#ifdef __cplusplus
extern "C" {
#endif
extern  char            *getwd(char *);
#ifdef __cplusplus
}
```

_____unchanged_portion_omitted_

```
**********************************************************
   22848 Wed May 20 11:37:23 2015
new/usr/src/cmd/make/include/mksh/defs.h
make: unifdef for SGE (undefined)
**********************************************************
______unchanged_portion_omitted_

 116 /*
 117  * For make i18n. Codeset independent.
 118  * Setup character semantics by identifying all the special characters
 119  * of make, and assigning each an entry in the char_semantics[] vector.
 120  */
 121 enum {
 122         ampersand_char_entry = 0,       /*  0 */
 123         asterisk_char_entry,            /*  1 */
 124         at_char_entry,                  /*  2 */
 125         backquote_char_entry,           /*  3 */
 126         backslash_char_entry,           /*  4 */
 127         bar_char_entry,                 /*  5 */
 128         bracketleft_char_entry,         /*  6 */
 129         bracketright_char_entry,        /*  7 */
 130         colon_char_entry,               /*  8 */
 131         dollar_char_entry,              /*  9 */
 132         doublequote_char_entry,         /* 10 */
 133         equal_char_entry,               /* 11 */
 134         exclam_char_entry,              /* 12 */
 135         greater_char_entry,             /* 13 */
 136         hat_char_entry,                 /* 14 */
 137         hyphen_char_entry,              /* 15 */
 138         less_char_entry,                /* 16 */
 139         newline_char_entry,             /* 17 */
 140         numbersign_char_entry,          /* 18 */
 141         parenleft_char_entry,           /* 19 */
 142         parenright_char_entry,          /* 20 */
 143         percent_char_entry,             /* 21 */
 144         plus_char_entry,                /* 22 */
 145         question_char_entry,            /* 23 */
 146         quote_char_entry,               /* 24 */
 147         semicolon_char_entry,           /* 25 */
 148 #ifdef SGE_SUPPORT
 149         space_char_entry,               /* 26 */
 150         tab_char_entry,                 /* 27 */
 151         no_semantics_entry              /* 28 */
 152 #else
 148         no_semantics_entry              /* 26 */
 154 #endif /* SGE_SUPPORT */
 149 };

 151 /*
 152  * CHAR_SEMANTICS_ENTRIES should be the number of entries above.
 153  * The last entry in char_semantics[] should be blank.
 154  */
 161 #ifdef SGE_SUPPORT
 162 #define CHAR_SEMANTICS_ENTRIES  29
 163 /*
 164 #define CHAR_SEMANTICS_STRING   "&*@'\\|[]:$=!>-\n#()%+?;^<'\" \t"
 165  */
 166 #else
 155 #define CHAR_SEMANTICS_ENTRIES  27
 156 /*
 157 #define CHAR_SEMANTICS_STRING   "&*@'\\|[]:$=!>-\n#()%+?;^<'\""
 158  */
 171 #endif /* SGE_SUPPORT */

 160 /*
 161  * Some utility macros
```



```
 162  */
 163 #define ALLOC(x)                ((struct _##x *)getmem(sizeof (struct _##x)))
 164 #define ALLOC_WC(x)             ((wchar_t *)getmem((x) * SIZEOFWCHAR_T))
 165 #define FIND_LENGTH             -1
 166 #define GETNAME(a,b)            getname_fn((a), (b), false)
 167 #define IS_EQUAL(a,b)           (!strcmp((a), (b)))
 168 #define IS_EQUALN(a,b,n)        (!strncmp((a), (b), (n)))
 169 #define IS_WEQUAL(a,b)          (!wscmp((a), (b)))
 170 #define IS_WEQUALN(a,b,n)       (!wsncmp((a), (b), (n)))
 171 #define MBLEN(a)                mblen((a), MB_LEN_MAX)
 172 #define MBSTOWCS(a,b)           (void) mbstowcs_with_check((a), (b), MAXPATHLEN)
 173 #define MBTOWC(a,b)             mbtowc((a), (b), MB_LEN_MAX)
 174 #define SIZEOFWCHAR_T           (sizeof (wchar_t))
 175 #define VSIZEOF(v)              (sizeof (v) / sizeof ((v)[0]))
 176 #define WCSTOMBS(a,b)           (void) wcstombs((a), (b), (MAXPATHLEN * MB_LEN_M
 177 #define WCTOMB(a,b)             (void) wctomb((a), (b))
 178 #define HASH(v, c)              (v = (v)*31 + (unsigned int)(c))

 180 extern void mbstowcs_with_check(wchar_t *pwcs, const char *s, size_t n);

 182 /*
 183  * Bits stored in funny vector to classify chars
 184  */
 185 enum {
 186         dollar_sem =            0001,
 187         meta_sem =              0002,
 188         percent_sem =           0004,
 189         wildcard_sem =          0010,
 190         command_prefix_sem =    0020,
 191         special_macro_sem =     0040,
 192         colon_sem =             0100,
 193         parenleft_sem =         0200
 194 };
______unchanged_portion_omitted_
```

```
**********************************************************
    3035 Wed May 20 11:37:24 2015
new/usr/src/cmd/make/lib/mksh/globals.cc
make: unifdef for SGE (undefined)
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */


/*
 *      globals.cc
 *
 *      This declares all global variables
 */

/*
 * Included files
 */
#include <mksh/globals.h>

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Global variables
 */
char            char_semantics[CHAR_SEMANTICS_ENTRIES];
wchar_t         char_semantics_char[] = {
        ampersand_char,
        asterisk_char,
        at_char,
        backquote_char,
        backslash_char,
        bar_char,
        bracketleft_char,
        bracketright_char,
        colon_char,
        dollar_char,
        doublequote_char,
```



```
        equal_char,
        exclam_char,
        greater_char,
        hat_char,
        hyphen_char,
        less_char,
        newline_char,
        numbersign_char,
        parenleft_char,
        parenright_char,
        percent_char,
        plus_char,
        question_char,
        quote_char,
        semicolon_char,
#ifdef SGE_SUPPORT
        space_char,
        tab_char,
#endif
        nul_char
};
```

_____unchanged_portion_omitted_