```
**********************************************************
   91576 Wed May 20 12:17:18 2015
new/usr/src/cmd/make/bin/doname.cc
make: remove maketool support
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      doname.c
 *
 *      Figure out which targets are out of date and rebuild them
 */

/*
 * Included files
 */
#include <alloca.h>             /* alloca() */


#include <fcntl.h>
#include <mk/defs.h>
#include <mksh/i18n.h>          /* get_char_semantics_value() */
#include <mksh/macro.h>         /* getvar(), expand_value() */
#include <mksh/misc.h>          /* getmem() */
#include <poll.h>


#include <signal.h>

#       include <stropts.h>

#include <sys/errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>        /* uname() */
#include <sys/wait.h>
#include <unistd.h>             /* close() */

/*
 * Defined macros
 */
#       define LOCALHOST "localhost"
```



```
#define MAXRULES 100

#define SEND_MTOOL_MSG(cmds)

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

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static char     hostName[MAXNAMELEN] = "";
static char     userName[MAXNAMELEN] = "";


static int      second_pass = 0;

/*
 * File table of contents
 */
extern  Doname          doname_check(register Name target, register Boolean do_g
extern  Doname          doname(register Name target, register Boolean do_get, re
static  Boolean         check_dependencies(Doname *result, Property line, Boolea
void            dynamic_dependencies(Name target);
static  Doname          run_command(register Property line, Boolean print_machin
extern  Doname          execute_serial(Property line);
extern  Name            vpath_translation(register Name cmd);
extern  void            check_state(Name temp_file_name);
static  void            read_dependency_file(register Name filename);
static  void            check_read_state_file(void);
static  void            do_assign(register Name line, register Name target);
static  void            build_command_strings(Name target, register Property lin
static  Doname          touch_command(register Property line, register Name targ
extern  void            update_target(Property line, Doname result);
static  Doname          sccs_get(register Name target, register Property *comman
extern  void            read_directory_of_file(register Name file);
static  void            add_pattern_conditionals(register Name target);
extern  void            set_locals(register Name target, register Property old_l
extern  void            reset_locals(register Name target, register Property old
extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
static  void            delete_query_chain(Chain ch);

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



/*
 * DONE.
 *
 *      doname_check(target, do_get, implicit, automatic)
 *
 *      Will call doname() and then inspect the return value
 *
 *      Return value:
 *                              Indication if the build failed or not
 *
 *      Parameters:
 *              target          The target to build
 *              do_get          Passed thru to doname()
 *              implicit        Passed thru to doname()
 *              automatic       Are we building a hidden dependency?
 *
```

```
 126  *      Global variables used:
 127  *              build_failed_seen       Set if -k is on and error occurs
 128  *              continue_after_error    Indicates that -k is on
 129  *              report_dependencies     No error msg if -P is on
 130  */
 131 Doname
 132 doname_check(register Name target, register Boolean do_get, register Boolean imp
 133 {
 134         int first_time = 1;
 135         (void) fflush(stdout);
 136 try_again:
 137         switch (doname(target, do_get, implicit, automatic)) {
 138         case build_ok:
 139                 second_pass = 0;
 140                 return build_ok;
 141         case build_running:
 142                 second_pass = 0;
 143                 return build_running;
 144         case build_failed:
 145                 if (!continue_after_error) {
 146                         fatal(catgets(catd, 1, 13, "Target '%s' not remade becau
 147                               target->string_mb);
 148                 }
 149                 build_failed_seen = true;
 150                 second_pass = 0;
 151                 return build_failed;
 152         case build_dont_know:
 153                 /*
 154                  * If we can't figure out how to build an automatic
 155                  * (hidden) dependency, we just ignore it.
 156                  * We later declare the target to be out of date just in
 157                  * case something changed.
 158                  * Also, don't complain if just reporting the dependencies
 159                  * and not building anything.
 160                  */
 161                 if (automatic || (report_dependencies_level > 0)) {
 162                         second_pass = 0;
 163                         return build_dont_know;
 164                 }
 165                 if(first_time) {
 166                         first_time = 0;
 167                         second_pass = 1;
 168                         goto try_again;
 169                 }
 170                 second_pass = 0;
 171                 if (continue_after_error && !svr4) {
 172                         warning(catgets(catd, 1, 14, "Don't know how to make tar
 173                                 target->string_mb);
 174                         build_failed_seen = true;
 175                         return build_failed;
 176                 }
 177                 fatal(catgets(catd, 1, 15, "Don't know how to make target '%s'")
 178                 break;
 179         }
 180 #ifdef lint
 181         return build_failed;
 182 #endif
 183 }
_____unchanged_portion_omitted_

1741 /*
1742  *      execute_serial(line)
1743  *
1744  *      Runs thru the command line for the target and
1745  *      executes the rules one by one.
1746  *
```



```
1747  *      Return value:
1748  *                              The result of the command build
1749  *
1750  *      Parameters:
1751  *              line            The command to execute
1752  *
1753  *      Static variables used:
1754  *
1755  *      Global variables used:
1756  *              continue_after_error -k flag
1757  *              do_not_exec_rule -n flag
1758  *              report_dependencies -P flag
1759  *              silent          Don't echo commands before executing
1760  *              temp_file_name  Temp file for auto dependencies
1761  *              vpath_defined   If true, translate path for command
1762  */
1763 Doname
1764 execute_serial(Property line)
1765 {
1766         int                     child_pid = 0;
1767         Boolean                 printed_serial;
1768         Doname                  result = build_ok;
1769         Cmd_line                rule, cmd_tail, command = NULL;
1770         char                    mbstring[MAXPATHLEN];
1771         int                     filed;
1772         Name                    target = line->body.line.target;

1776         SEND_MTOOL_MSG(
1777                 if (!sent_rsrc_info_msg) {
1778                         if (userName[0] == '\0') {
1779                                 avo_get_user(userName, NULL);
1780                         }
1781                         if (hostName[0] == '\0') {
1782                                 strcpy(hostName, avo_hostname());
1783                         }
1784                         send_rsrc_info_msg(1, hostName, userName);
1785                         sent_rsrc_info_msg = 1;
1786                 }
1787                 send_job_start_msg(line);
1788                 job_result_msg = new Avo_MToolJobResultMsg();
1789         );

1774         target->has_recursive_dependency = false;
1775         // We have to create a copy of the rules chain for processing because
1776         // the original one can be destroyed during .make.state file rereading.
1777         for (rule = line->body.line.command_used;
1778              rule != NULL;
1779              rule = rule->next) {
1780                 if (command == NULL) {
1781                         command = cmd_tail = ALLOC(Cmd_line);
1782                 } else {
1783                         cmd_tail->next = ALLOC(Cmd_line);
1784                         cmd_tail = cmd_tail->next;
1785                 }
1786                 *cmd_tail = *rule;
1787         }
1788         if (command) {
1789                 cmd_tail->next = NULL;
1790         }
1791         for (rule = command; rule != NULL; rule = rule->next) {
1792                 if (posix && (touch || quest) && !rule->always_exec) {
1793                         continue;
1794                 }
1795                 if (vpath_defined) {
1796                         rule->command_line =
1797                           vpath_translation(rule->command_line);
```

```
1798                }
1799                /* Echo command line, maybe. */
1800                if ((rule->command_line->hash.length > 0) &&
1801                    !silent &&
1802                    (!rule->silent || do_not_exec_rule) &&
1803                    (report_dependencies_level == 0)) {
1804                        (void) printf("%s\n", rule->command_line->string_mb);
1822                        SEND_MTOOL_MSG(
1823                                job_result_msg->appendOutput(AVO_STRDUP(rule->co
1824                        );
1805                }
1806                if (rule->command_line->hash.length > 0) {
1827                        SEND_MTOOL_MSG(
1828                                (void) sprintf(mbstring,
1829                                                NOCATGETS("%s/make.stdout.%d.%d.
1830                                                tmpdir,
1831                                                getpid(),
1832                                                file_number++);

1834                                int tmp_fd = mkstemp(mbstring);
1835                                if(tmp_fd) {
1836                                        (void) close(tmp_fd);
1837                                }

1839                                stdout_file = strdup(mbstring);
1840                                stderr_file = NULL;
1841                                child_pid = pollResults(stdout_file,
1842                                                        (char *)NULL,
1843                                                        (char *)NULL);
1844                        );
1807                        /* Do assignment if command line prefixed with "=" */
1808                        if (rule->assign) {
1809                                result = build_ok;
1810                                do_assign(rule->command_line, target);
1811                        } else if (report_dependencies_level == 0) {
1812                                /* Execute command line. */
1813                                setvar_envvar();
1814                                result = dosys(rule->command_line,
1815                                               (Boolean) rule->ignore_error,
1816                                               (Boolean) rule->make_refd,
1817                                               /* ds 98.04.23 bug #4085164. make
1818                                               false,
1819                                               /* BOOLEAN(rule->silent &&
1820                                                       rule->ignore_error), */
1821                                               (Boolean) rule->always_exec,
1822                                               target);
1860                                               target,
1861                                               send_mtool_msgs);
1823                                check_state(temp_file_name);
1824                        }
1864                        SEND_MTOOL_MSG(
1865                                append_job_result_msg(job_result_msg);
1866                                if (child_pid > 0) {
1867                                        kill(child_pid, SIGUSR1);
1868                                        while (!((waitpid(child_pid, 0, 0) == -1
1869                                                 && (errno == ECHILD)));
1870                                }
1871                                child_pid = 0;
1872                                (void) unlink(stdout_file);
1873                                retmem_mb(stdout_file);
1874                                stdout_file = NULL;
1875                        );
1825                } else {
1826                        result = build_ok;
1827                }
1828                if (result == build_failed) {
```



```
1829                        if (silent || rule->silent) {
1830                                (void) printf(catgets(catd, 1, 242, "The followi
1831                                              rule->command_line->string_mb);
1883                                SEND_MTOOL_MSG(
1884                                        job_result_msg->appendOutput(AVO_STRDUP(
1885                                        job_result_msg->appendOutput(AVO_STRDUP(
1886                                );
1832                        }
1833                        if (!rule->ignore_error && !ignore_errors) {
1834                                if (!continue_after_error) {
1890                                        SEND_MTOOL_MSG(
1891                                                job_result_msg->setResult(job_ms
1892                                                xdr_msg = (RWCollectable*)
1893                                                        job_result_msg;
1894                                                xdr(&xdrs, xdr_msg);
1895                                                (void) fflush(mtool_msgs_fp);
1896                                                delete job_result_msg;
1897                                        );
1835                                        fatal(catgets(catd, 1, 244, "Command fai
1836                                              target->string_mb);
1837                                }
1838                                /*
1839                                 * Make sure a failing command is not
1840                                 * saved in .make.state.
1841                                 */
1842                                line->body.line.command_used = NULL;
1843                                break;
1844                        } else {
1845                                result = build_ok;
1846                        }
1847                }
1848        }
1849        for (rule = command; rule != NULL; rule = cmd_tail) {
1850                cmd_tail = rule->next;
1851                free(rule);
1852        }
1853        command = NULL;
1917        SEND_MTOOL_MSG(
1918                job_result_msg->setResult(job_msg_id, (result == build_ok) ? 0 :
1919                xdr_msg = (RWCollectable*) job_result_msg;
1920                xdr(&xdrs, xdr_msg);
1921                (void) fflush(mtool_msgs_fp);

1923                delete job_result_msg;
1924        );
1854        if (temp_file_name != NULL) {
1855                free_name(temp_file_name);
1856        }
1857        temp_file_name = NULL;

1859        Property spro = get_prop(sunpro_dependencies->prop, macro_prop);
1860        if(spro != NULL) {
1861                Name val = spro->body.macro.value;
1862                if(val != NULL) {
1863                        free_name(val);
1864                        spro->body.macro.value = NULL;
1865                }
1866        }
1867        spro = get_prop(sunpro_dependencies->prop, env_mem_prop);
1868        if(spro) {
1869                char *val = spro->body.env_mem.value;
1870                if(val != NULL) {
1871                        /*
1872                         * Do not return memory allocated for SUNPRO_DEPENDENCIE
1873                         * It will be returned in setvar_daemon() in macro.cc
1874                         */
```

```
1875                        //      retmem_mb(val);
1876                        spro->body.env_mem.value = NULL;
1877                }
1878        }
1879
1880        return result;
1881 }
_____unchanged_portion_omitted_

2487 /*
2488  *      touch_command(line, target, result)
2489  *
2490  *      If this is an "make -t" run we do this.
2491  *      We touch all targets in the target group ("foo + fie:") if any.
2492  *
2493  *      Return value:
2494  *                              Indicates if the command failed or not
2495  *
2496  *      Parameters:
2497  *              line            The command line to update
2498  *              target          The target we are touching
2499  *              result          Initial value for the result we return
2500  *
2501  *      Global variables used:
2502  *              do_not_exec_rule Indicates that -n is on
2503  *              silent          Do not echo commands
2504  */
2505 static Doname
2506 touch_command(register Property line, register Name target, Doname result)
2507 {
2508         Name                    name;
2509         register Chain          target_group;
2510         String_rec              touch_string;
2511         wchar_t                 buffer[MAXPATHLEN];
2512         Name                    touch_cmd;
2513         Cmd_line                rule;

2587         SEND_MTOOL_MSG(
2588                 if (!sent_rsrc_info_msg) {
2589                         if (userName[0] == '\0') {
2590                                 avo_get_user(userName, NULL);
2591                         }
2592                         if (hostName[0] == '\0') {
2593                                 strcpy(hostName, avo_hostname());
2594                         }
2595                         send_rsrc_info_msg(1, hostName, userName);
2596                         sent_rsrc_info_msg = 1;
2597                 }
2598                 send_job_start_msg(line);
2599                 job_result_msg = new Avo_MToolJobResultMsg();
2600         );
2515         for (name = target, target_group = NULL; name != NULL;) {
2516                 if (!name->is_member) {
2517                         /*
2518                          * Build a touch command that can be passed
2519                          * to dosys(). If KEEP_STATE is on, "make -t"
2520                          * will save the proper command, not the
2521                          * "touch" in .make.state.
2522                          */
2523                         INIT_STRING_FROM_STACK(touch_string, buffer);
2524                         MBSTOWCS(wcs_buffer, NOCATGETS("touch "));
2525                         append_string(wcs_buffer, &touch_string, FIND_LENGTH);
2526                         touch_cmd = name;
2527                         if (name->has_vpath_alias_prop) {
2528                                 touch_cmd = get_prop(name->prop,
```



```
2529                                                 vpath_alias_prop)->
2530                                                   body.vpath_alias.alias;
2531                         }
2532                         APPEND_NAME(touch_cmd,
2533                                     &touch_string,
2534                                     FIND_LENGTH);
2535                         touch_cmd = GETNAME(touch_string.buffer.start,
2536                                             FIND_LENGTH);
2537                         if (touch_string.free_after_use) {
2538                                 retmem(touch_string.buffer.start);
2539                         }
2540                         if (!silent ||
2541                             do_not_exec_rule &&
2542                             (target_group == NULL)) {
2543                                 (void) printf("%s\n", touch_cmd->string_mb);
2630                                 SEND_MTOOL_MSG(
2631                                         job_result_msg->appendOutput(AVO_STRDUP(
2632                                 );
2544                         }
2545                         /* Run the touch command, or simulate it */
2546                         if (!do_not_exec_rule) {

2637                                 SEND_MTOOL_MSG(
2638                                         (void) sprintf(mbstring,
2639                                                        NOCATGETS("%s/make.stdou
2640                                                        tmpdir,
2641                                                        getpid(),
2642                                                        file_number++);
2643
2644                                         int tmp_fd = mkstemp(mbstring);
2645                                         if(tmp_fd) {
2646                                                 (void) close(tmp_fd);
2647                                         }

2649                                         stdout_file = strdup(mbstring);
2650                                         stderr_file = NULL;
2651                                         child_pid = pollResults(stdout_file,
2652                                                                 (char *)NULL,
2653                                                                 (char *)NULL);
2654                                 );

2547                                 result = dosys(touch_cmd,
2548                                                false,
2549                                                false,
2550                                                false,
2551                                                false,
2552                                                name);
2661                                                name,
2662                                                send_mtool_msgs);

2664                                 SEND_MTOOL_MSG(
2665                                         append_job_result_msg(job_result_msg);
2666                                         if (child_pid > 0) {
2667                                                 kill(child_pid, SIGUSR1);
2668                                                 while (!((waitpid(child_pid, 0,
2669                                                           && (errno == ECHILD)));
2670                                         }
2671                                         child_pid = 0;
2672                                         (void) unlink(stdout_file);
2673                                         retmem_mb(stdout_file);
2674                                         stdout_file = NULL;
2675                                 );

2553                         } else {
2554                                 result = build_ok;
2555                         }
```

```
2556                 } else {
2557                         result = build_ok;
2558                 }
2559                 if (target_group == NULL) {
2560                         target_group = line->body.line.target_group;
2561                 } else {
2562                         target_group = target_group->next;
2563                 }
2564                 if (target_group != NULL) {
2565                         name = target_group->name;
2566                 } else {
2567                         name = NULL;
2568                 }
2569         }
2694         SEND_MTOOL_MSG(
2695                 job_result_msg->setResult(job_msg_id, (result == build_ok) ? 0 :
2696                 xdr_msg = (RWCollectable*) job_result_msg;
2697                 xdr(&xdrs, xdr_msg);
2698                 (void) fflush(mtool_msgs_fp);
2699                 delete job_result_msg;
2700         );
2570         return result;
2571 }
```

unchanged_portion_omitted_

```
**********************************************************
    4094 Wed May 20 12:17:19 2015
new/usr/src/cmd/make/bin/dosys.cc
make: remove maketool support
**********************************************************
```

```
/*
 * 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.
 */

/*
 *      dosys.cc
 *
 *      Execute one commandline
 */

/*
 * Included files
 */
#include <fcntl.h>              /* open() */
#include <mk/defs.h>
#include <mksh/dosys.h>         /* doshell(), doexec() */
#include <mksh/misc.h>          /* getmem() */
#include <sys/stat.h>           /* open() */
#include <unistd.h>             /* getpid() */

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static  int             filter_file;
static  char            *filter_file_name;

/*
 * File table of contents
 */
static  void            redirect_stderr(void);

/*
```



```
 *      dosys(command, ignore_error, call_make, silent_error, target)
 *
 *      Check if command string contains meta chars and dispatch to
 *      the proper routine for executing one command line.
 *
 *      Return value:
 *                              Indicates if the command execution failed
 *
 *      Parameters:
 *              command         The command to run
 *              ignore_error    Should make abort when an error is seen?
 *              call_make       Did command reference $(MAKE) ?
 *              silent_error    Should error messages be suppressed for pmake?
 *              target          Target we are building
 *
 *      Global variables used:
 *              do_not_exec_rule Is -n on?
 *              working_on_targets We started processing real targets
 */
Doname
dosys(register Name command, register Boolean ignore_error, register Boolean cal
dosys(register Name command, register Boolean ignore_error, register Boolean cal
{
        timestruc_t             before;
        register int            length = command->hash.length;
        Wstring                 wcb(command);
        register wchar_t        *p = wcb.get_string();
        register wchar_t        *q;
        Doname                  result;

        /* Strip spaces from head of command string */
        while (iswspace(*p)) {
                p++, length--;
        }
        if (*p == (int) nul_char) {
                return build_failed;
        }
        /* If we are faking it we just return */
        if (do_not_exec_rule &&
            working_on_targets &&
            !call_make &&
            !always_exec) {
                return build_ok;
        }
        /* no_action_was_taken is used to print special message */
        no_action_was_taken = false;

        /* Copy string to make it OK to write it. */
        q = ALLOC_WC(length + 1);
        (void) wscpy(q, p);
        /* Write the state file iff this command uses make. */
        if (call_make && command_changed) {
                write_state_file(0, false);
        }
        make_state->stat.time = file_no_time;
        (void)exists(make_state);
        before = make_state->stat.time;
        /*
         * Run command directly if it contains no shell meta chars,
         * else run it using the shell.
         */
        if (await(ignore_error,
                  silent_error,
                  target,
                  wcb.get_string(),
                  command->meta ?
```

```
127                     doshell(q, ignore_error,
127                     doshell(q, ignore_error, redirect_out_err,
128                             stdout_file, stderr_file, 0) :
129                     doexec(q, ignore_error,
129                     doexec(q, ignore_error, redirect_out_err,
130                             stdout_file, stderr_file,
131                             vroot_path, 0),
132                   send_mtool_msgs,
132                   NULL,
133                   -1
134                   )) {
135                 result = build_ok;
136         } else {
137                 result = build_failed;
138         }
139         retmem(q);

141         if ((report_dependencies_level == 0) &&
142             call_make) {
143                 make_state->stat.time = file_no_time;
144                 (void)exists(make_state);
145                 if (before == make_state->stat.time) {
146                         return result;
147                 }
148                 makefile_type = reading_statefile;
149                 if (read_trace_level > 1) {
150                         trace_reader = true;
151                 }
152                 temp_file_number++;
153                 (void) read_simple_file(make_state,
154                                         false,
155                                         false,
156                                         false,
157                                         false,
158                                         false,
159                                         true);
160                 trace_reader = false;
161         }
162         return result;
163 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4516 Wed May 20 12:17:19 2015
new/usr/src/cmd/make/bin/globals.cc
make: remove maketool support
**********************************************************
```

```
/*
 * 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;
        Boolean         assign_done;
        int foo;
```



```
        Boolean         build_failed_seen;
        Name            built_last_make_run;
        Name            c_at;
        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;
        timestruc_t     make_state_before;
        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;
        Name            no_parallel_name;
        Name            not_auto;
        Boolean         only_parallel;
        Boolean         parallel;
        Name            parallel_name;
        Name            localhost_name;
        int             parallel_process_cnt;
        Percent         percent_list;
        Dyntarget       dyntarget_list;
```

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

179 /*
180  * File table of contents
181  */
```

```
**********************************************************
   89391 Wed May 20 12:17:19 2015
new/usr/src/cmd/make/bin/main.cc
make: remove maketool support
**********************************************************
______unchanged_portion_omitted_

1282 /*
1283  *      parse_command_option(ch)
1284  *
1285  *      Parse make command line options.
1286  *
1287  *      Return value:
1288  *                              Indicates if any -f -c or -M were seen
1289  *
1290  *      Parameters:
1291  *              ch              The character to parse
1292  *
1293  *      Static variables used:
1294  *              dmake_group_specified   Set for make -g
1295  *              dmake_max_jobs_specified        Set for make -j
1296  *              dmake_mode_specified    Set for make -m
1297  *              dmake_add_mode_specified        Set for make -x
1298  *              dmake_compat_mode_specified     Set for make -x SUN_MAKE_COMPAT_
1299  *              dmake_output_mode_specified     Set for make -x DMAKE_OUTPUT_MOD
1300  *              dmake_odir_specified    Set for make -o
1301  *              dmake_rcfile_specified  Set for make -c
1302  *              env_wins                Set for make -e
1303  *              ignore_default_mk       Set for make -r
1304  *              trace_status            Set for make -p
1305  *
1306  *      Global variables used:
1307  *              .make.state path & name set for make -K
1308  *              continue_after_error    Set for make -k
1309  *              debug_level             Set for make -d
1310  *              do_not_exec_rule        Set for make -n
1311  *              filter_stderr           Set for make -X
1312  *              ignore_errors_all       Set for make -i
1313  *              no_parallel             Set for make -R
1314  *              quest                   Set for make -q
1315  *              read_trace_level        Set for make -D
1316  *              report_dependencies     Set for make -P
1317  *              send_mtool_msgs         Set for make -K
1317  *              silent_all              Set for make -s
1318  *              touch                   Set for make -t
1319  */
1320 static int
1321 parse_command_option(register char ch)
1322 {
1323         static int              invert_next = 0;
1324         int                     invert_this = invert_next;

1326         invert_next = 0;
1327         switch (ch) {
1328         case '-':                       /* Ignore "--" */
1329                 return 0;
1330         case '~':                       /* Invert next option */
1331                 invert_next = 1;
1332                 return 0;
1333         case 'B':                       /* Obsolete */
1334                 return 0;
1335         case 'b':                       /* Obsolete */
1336                 return 0;
1337         case 'c':                       /* Read alternative dmakerc file */
1338                 if (invert_this) {
1339                         dmake_rcfile_specified = false;
```



```
1340                 } else {
1341                         dmake_rcfile_specified = true;
1342                 }
1343                 return 2;
1344         case 'D':                       /* Show lines read */
1345                 if (invert_this) {
1346                         read_trace_level--;
1347                 } else {
1348                         read_trace_level++;
1349                 }
1350                 return 0;
1351         case 'd':                       /* Debug flag */
1352                 if (invert_this) {
1353                         debug_level--;
1354                 } else {
1355                         debug_level++;
1356                 }
1357                 return 0;
1358         case 'e':                       /* Environment override flag */
1359                 if (invert_this) {
1360                         env_wins = false;
1361                 } else {
1362                         env_wins = true;
1363                 }
1364                 return 0;
1365         case 'f':                       /* Read alternative makefile(s) */
1366                 return 1;
1367         case 'g':                       /* Use alternative DMake group */
1368                 if (invert_this) {
1369                         dmake_group_specified = false;
1370                 } else {
1371                         dmake_group_specified = true;
1372                 }
1373                 return 4;
1374         case 'i':                       /* Ignore errors */
1375                 if (invert_this) {
1376                         ignore_errors_all = false;
1377                 } else {
1378                         ignore_errors_all = true;
1379                 }
1380                 return 0;
1381         case 'j':                       /* Use alternative DMake max jobs */
1382                 if (invert_this) {
1383                         dmake_max_jobs_specified = false;
1384                 } else {
1385                         dmake_mode_type = parallel_mode;
1386                         no_parallel = false;
1387                         dmake_max_jobs_specified = true;
1388                 }
1389                 return 8;
1390         case 'K':                       /* Read alternative .make.state */
1391                 return 256;
1392         case 'k':                       /* Keep making even after errors */
1393                 if (invert_this) {
1394                         continue_after_error = false;
1395                 } else {
1396                         continue_after_error = true;
1397                         continue_after_error_ever_seen = true;
1398                 }
1399                 return 0;
1400         case 'M':                       /* Read alternative make.machines file
1401                 if (invert_this) {
1402                         pmake_machinesfile_specified = false;
1403                 } else {
1404                         pmake_machinesfile_specified = true;
1405                         dmake_mode_type = parallel_mode;
```

```
1406                            no_parallel = false;
1407                    }
1408                    return 16;
1409            case 'm':                       /* Use alternative DMake build mode */
1410                    if (invert_this) {
1411                            dmake_mode_specified = false;
1412                    } else {
1413                            dmake_mode_specified = true;
1414                    }
1415                    return 32;
1416            case 'x':                       /* Use alternative DMake mode */
1417                    if (invert_this) {
1418                            dmake_add_mode_specified = false;
1419                    } else {
1420                            dmake_add_mode_specified = true;
1421                    }
1422                    return 1024;
1423            case 'N':                       /* Reverse -n */
1424                    if (invert_this) {
1425                            do_not_exec_rule = true;
1426                    } else {
1427                            do_not_exec_rule = false;
1428                    }
1429                    return 0;
1430            case 'n':                       /* Print, not exec commands */
1431                    if (invert_this) {
1432                            do_not_exec_rule = false;
1433                    } else {
1434                            do_not_exec_rule = true;
1435                    }
1436                    return 0;
1437            case 'O':                       /* Integrate with maketool, obsolete */
1438                    return 0;
1438            case 'O':                       /* Send job start & result msgs */
1439                    if (invert_this) {
1440                            send_mtool_msgs = false;
1441                    } else {
1442                    }
1443                    return 128;
1439            case 'o':                       /* Use alternative dmake output dir */
1440                    if (invert_this) {
1441                            dmake_odir_specified = false;
1442                    } else {
1443                            dmake_odir_specified = true;
1444                    }
1445                    return 512;
1446            case 'P':                       /* Print for selected targets */
1447                    if (invert_this) {
1448                            report_dependencies_level--;
1449                    } else {
1450                            report_dependencies_level++;
1451                    }
1452                    return 0;
1453            case 'p':                       /* Print description */
1454                    if (invert_this) {
1455                            trace_status = false;
1456                            do_not_exec_rule = false;
1457                    } else {
1458                            trace_status = true;
1459                            do_not_exec_rule = true;
1460                    }
1461                    return 0;
1462            case 'q':                       /* Question flag */
1463                    if (invert_this) {
1464                            quest = false;
1465                    } else {
```



```
1466                            quest = true;
1467                    }
1468                    return 0;
1469            case 'R':                       /* Don't run in parallel */
1470                    if (invert_this) {
1471                            pmake_cap_r_specified = false;
1472                            no_parallel = false;
1473                    } else {
1474                            pmake_cap_r_specified = true;
1475                            dmake_mode_type = serial_mode;
1476                            no_parallel = true;
1477                    }
1478                    return 0;
1479            case 'r':                       /* Turn off internal rules */
1480                    if (invert_this) {
1481                            ignore_default_mk = false;
1482                    } else {
1483                            ignore_default_mk = true;
1484                    }
1485                    return 0;
1486            case 'S':                       /* Reverse -k */
1487                    if (invert_this) {
1488                            continue_after_error = true;
1489                    } else {
1490                            continue_after_error = false;
1491                            stop_after_error_ever_seen = true;
1492                    }
1493                    return 0;
1494            case 's':                       /* Silent flag */
1495                    if (invert_this) {
1496                            silent_all = false;
1497                    } else {
1498                            silent_all = true;
1499                    }
1500                    return 0;
1501            case 'T':                       /* Print target list */
1502                    if (invert_this) {
1503                            list_all_targets = false;
1504                            do_not_exec_rule = false;
1505                    } else {
1506                            list_all_targets = true;
1507                            do_not_exec_rule = true;
1508                    }
1509                    return 0;
1510            case 't':                       /* Touch flag */
1511                    if (invert_this) {
1512                            touch = false;
1513                    } else {
1514                            touch = true;
1515                    }
1516                    return 0;
1517            case 'u':                       /* Unconditional flag */
1518                    if (invert_this) {
1519                            build_unconditional = false;
1520                    } else {
1521                            build_unconditional = true;
1522                    }
1523                    return 0;
1524            case 'V':                      /* SVR4 mode */
1525                    svr4 = true;
1526                    return 0;
1527            case 'v':                       /* Version flag */
1528                    if (invert_this) {
1529                    } else {
1530                            fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
1531                            exit_status = 0;
```

```
1532                        exit(0);
1533                }
1534                return 0;
1535        case 'w':                       /* Unconditional flag */
1536                if (invert_this) {
1537                        report_cwd = false;
1538                } else {
1539                        report_cwd = true;
1540                }
1541                return 0;
1542 #if 0
1543        case 'X':                       /* Filter stdout */
1544                if (invert_this) {
1545                        filter_stderr = false;
1546                } else {
1547                        filter_stderr = true;
1548                }
1549                return 0;
1550 #endif
1551        default:
1552                break;
1553        }
1554        return 0;
1555 }
______unchanged_portion_omitted_

2423 /*
2424  * Scan the argv for options and "=" type args and make them readonly.
2425  */
2426 static void
2427 enter_argv_values(int argc, char *argv[], ASCII_Dyn_Array *makeflags_and_macro)
2428 {
2429        register char           *cp;
2430        register int            i;
2431        int                     length;
2432        register Name           name;
2433        int                     opt_separator = argc;
2434        char                    tmp_char;
2435        wchar_t                 *tmp_wcs_buffer;
2436        register Name           value;
2437        Boolean                 append = false;
2438        Property                macro;
2439        struct stat             statbuf;


2442        /* Read argv options and "=" type args and make them readonly. */
2443        makefile_type = reading_nothing;
2444        for (i = 1; i < argc; ++i) {
2445                append = false;
2446                if (argv[i] == NULL) {
2447                        continue;
2448                } else if (((argv[i][0] == '-') && (argv[i][1] == '-')) ||
2449                           ((argv[i][0] == (int) ' ') &&
2450                            (argv[i][1] == (int) '-') &&
2451                            (argv[i][2] == (int) ' ') &&
2452                            (argv[i][3] == (int) '-'))) {
2453                        argv[i] = NULL;
2454                        opt_separator = i;
2455                        continue;
2456                } else if ((i < opt_separator) && (argv[i][0] == (int) hyphen_ch
2457                        switch (parse_command_option(argv[i][1])) {
2458                        case 1: /* -f seen */
2459                                ++i;
2460                                continue;
2461                        case 2: /* -c seen */
2462                                if (argv[i+1] == NULL) {
```



```
2463                                        fatal(catgets(catd, 1, 194, "No dmake rc
2464                                }
2465                                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2466                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2467                                break;
2468                        case 4: /* -g seen */
2469                                if (argv[i+1] == NULL) {
2470                                        fatal(catgets(catd, 1, 195, "No dmake gr
2471                                }
2472                                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
2473                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2474                                break;
2475                        case 8: /* -j seen */
2476                                if (argv[i+1] == NULL) {
2477                                        fatal(catgets(catd, 1, 196, "No dmake ma
2478                                }
2479                                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS")
2480                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2481                                break;
2482                        case 16: /* -M seen */
2483                                if (argv[i+1] == NULL) {
2484                                        fatal(catgets(catd, 1, 323, "No pmake ma
2485                                }
2486                                MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFI
2487                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2488                                break;
2489                        case 32: /* -m seen */
2490                                if (argv[i+1] == NULL) {
2491                                        fatal(catgets(catd, 1, 197, "No dmake mo
2492                                }
2493                                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2494                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2495                                break;
2501                        case 128: /* -O seen */
2502                                if (argv[i+1] == NULL) {
2503                                        fatal(catgets(catd, 1, 287, "No file des
2504                                }
2505                                mtool_msgs_fd = atoi(argv[i+1]);
2506                                /* find out if mtool_msgs_fd is a valid file des
2507                                if (fstat(mtool_msgs_fd, &statbuf) < 0) {
2508                                        fatal(catgets(catd, 1, 355, "Invalid fil
2509                                }
2510                                argv[i] = NULL;
2511                                argv[i+1] = NULL;
2512                                continue;
2496                        case 256: /* -K seen */
2497                                if (argv[i+1] == NULL) {
2498                                        fatal(catgets(catd, 1, 288, "No makestat
2499                                }
2500                                MBSTOWCS(wcs_buffer, argv[i+1]);
2501                                make_state = GETNAME(wcs_buffer, FIND_LENGTH);
2502                                keep_state = true;
2503                                argv[i] = NULL;
2504                                argv[i+1] = NULL;
2505                                continue;
2506                        case 512:       /* -o seen */
2507                                if (argv[i+1] == NULL) {
2508                                        fatal(catgets(catd, 1, 312, "No dmake ou
2509                                }
2510                                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2511                                name = GETNAME(wcs_buffer, FIND_LENGTH);
2512                                break;
2513                        case 1024: /* -x seen */
2514                                if (argv[i+1] == NULL) {
2515                                        fatal(catgets(catd, 1, 351, "No argument
2516                                }
```

```
2517                                length = strlen( NOCATGETS("SUN_MAKE_COMPAT_MODE
2518                                if (strncmp(argv[i+1], NOCATGETS("SUN_MAKE_COMPA
2519                                        argv[i+1] = &argv[i+1][length];
2520                                        MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE
2521                                        name = GETNAME(wcs_buffer, FIND_LENGTH);
2522                                        dmake_compat_mode_specified = dmake_add_
2523                                        break;
2524                                }
2525                                length = strlen( NOCATGETS("DMAKE_OUTPUT_MODE=")
2526                                if (strncmp(argv[i+1], NOCATGETS("DMAKE_OUTPUT_M
2527                                        argv[i+1] = &argv[i+1][length];
2528                                        MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OU
2529                                        name = GETNAME(wcs_buffer, FIND_LENGTH);
2530                                        dmake_output_mode_specified = dmake_add_
2531                                } else {
2532                                        warning(catgets(catd, 1, 354, "Unknown a
2533                                              argv[i+1]);
2534                                        argv[i] = argv[i + 1] = NULL;
2535                                        continue;
2536                                }
2537                                break;
2538                        default: /* Shouldn't reach here */
2539                                argv[i] = NULL;
2540                                continue;
2541                        }
2542                        argv[i] = NULL;
2543                        if (i == (argc - 1)) {
2544                                break;
2545                        }
2546                        if ((length = strlen(argv[i+1])) >= MAXPATHLEN) {
2547                                tmp_wcs_buffer = ALLOC_WC(length + 1);
2548                                (void) mbstowcs(tmp_wcs_buffer, argv[i+1], lengt
2549                                value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2550                                retmem(tmp_wcs_buffer);
2551                        } else {
2552                                MBSTOWCS(wcs_buffer, argv[i+1]);
2553                                value = GETNAME(wcs_buffer, FIND_LENGTH);
2554                        }
2555                        argv[i+1] = NULL;
2556                } else if ((cp = strchr(argv[i], (int) equal_char)) != NULL) {
2557 /*
2558  * Combine all macro in dynamic array
2559  */
2560                        if(*(cp-1) == (int) plus_char)
2561                        {
2562                                if(isspace(*(cp-2))) {
2563                                        append = true;
2564                                        cp--;
2565                                }
2566                        }
2567                        if(!append)
2568                                append_or_replace_macro_in_dyn_array(makeflags_a

2570                        while (isspace(*(cp-1))) {
2571                                cp--;
2572                        }
2573                        tmp_char = *cp;
2574                        *cp = (int) nul_char;
2575                        MBSTOWCS(wcs_buffer, argv[i]);
2576                        *cp = tmp_char;
2577                        name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2578                        while (*cp != (int) equal_char) {
2579                                cp++;
2580                        }
2581                        cp++;
2582                        while (isspace(*cp) && (*cp != (int) nul_char)) {
```



```
2583                                cp++;
2584                        }
2585                        if ((length = strlen(cp)) >= MAXPATHLEN) {
2586                                tmp_wcs_buffer = ALLOC_WC(length + 1);
2587                                (void) mbstowcs(tmp_wcs_buffer, cp, length + 1);
2588                                value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2589                                retmem(tmp_wcs_buffer);
2590                        } else {
2591                                MBSTOWCS(wcs_buffer, cp);
2592                                value = GETNAME(wcs_buffer, FIND_LENGTH);
2593                        }
2594                        argv[i] = NULL;
2595                } else {
2596                        /* Illegal MAKEFLAGS argument */
2597                        continue;
2598                }
2599                if(append) {
2600                        setvar_append(name, value);
2601                        append = false;
2602                } else {
2603                        macro = maybe_append_prop(name, macro_prop);
2604                        macro->body.macro.exported = true;
2605                        SETVAR(name, value, false)->body.macro.read_only = true;
2606                }
2607        }
2608 }
```

unchanged_portion_omitted_

```
3208 #ifdef TEAMWARE_MAKE_CMN
3209 /*
3210  * This function, if registered w/ avo_cli_get_license(), will be called
3211  * if the application is about to exit because:
3212  *   1) there has been certain unrecoverable error(s) that cause the
3213  *      application to exit immediately.
3214  *   2) the user has lost a license while the application is running.
3215  */
3216 extern "C" void
3217 dmake_exit_callback(void)
3218 {
3219         fatal(catgets(catd, 1, 306, "can not get a license, exiting..."));
3220         exit(1);
3221 }

3223 /*
3224  * This function, if registered w/ avo_cli_get_license(), will be called
3225  * if the application can not get a license.
3226  */
3227 extern "C" void
3228 dmake_message_callback(char *err_msg)
3229 {
3230         static Boolean  first = true;

3232         if (!first) {
3233                 return;
3234         }
3235         first = false;
3236         if ((!list_all_targets) &&
3237             (report_dependencies_level == 0) &&
3238             (dmake_mode_type != serial_mode)) {
3239                 warning(catgets(catd, 1, 313, "can not get a TeamWare license, d
3240         }
3241 }
3242 #endif


3191 static void
```

```
3192 report_dir_enter_leave(Boolean entering)
3193 {
3194         char    rcwd[MAXPATHLEN];
3195 static  char *  mlev = NULL;
3196         char *  make_level_str = NULL;
3197         int     make_level_val = 0;

3199         make_level_str = getenv(NOCATGETS("MAKELEVEL"));
3200         if(make_level_str) {
3201                 make_level_val = atoi(make_level_str);
3202         }
3203         if(mlev == NULL) {
3204                 mlev = (char*) malloc(MAXPATHLEN);
3205         }
3206         if(entering) {
3207                 sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val + 1);
3208         } else {
3209                 make_level_val--;
3210                 sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val);
3211         }
3212         putenv(mlev);

3214         if(report_cwd) {
3215                 if(make_level_val <= 0) {
3216                         if(entering) {
3217                                 sprintf( rcwd
3218                                        , catgets(catd, 1, 329, "dmake: Entering
3219                                        , get_current_path());
3220                         } else {
3221                                 sprintf( rcwd
3222                                        , catgets(catd, 1, 331, "dmake: Leaving d
3223                                        , get_current_path());
3224                         }
3225                 } else {
3226                         if(entering) {
3227                                 sprintf( rcwd
3228                                        , catgets(catd, 1, 333, "dmake[%d]: Enter
3229                                        , make_level_val, get_current_path());
3230                         } else {
3231                                 sprintf( rcwd
3232                                        , catgets(catd, 1, 335, "dmake[%d]: Leavi
3233                                        , make_level_val, get_current_path());
3234                         }
3235                 }
3236                 printf(NOCATGETS("%s"), rcwd);
3237         }
3238 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   46053 Wed May 20 12:17:20 2015
new/usr/src/cmd/make/bin/parallel.cc
make: remove maketool support
**********************************************************
```

_____*unchanged_portion_omitted_*

```
1100 /*
1101  *      await_parallel(waitflg)
1102  *
1103  *      Waits for parallel children to exit and finishes their processing.
1104  *      If waitflg is false, the function returns after update_delay.
1105  *
1106  *      Parameters:
1107  *              waitflg         dwight
1108  */
1109 void
1110 await_parallel(Boolean waitflg)
1111 {
1112         Boolean         nohang;
1113         pid_t           pid;
1114         int             status;
1115         Running         rp;
1116         int             waiterr;

1118         nohang = false;
1119         for ( ; ; ) {
1120                 if (!nohang) {
1121                         (void) alarm((int) update_delay);
1122                 }
1123                 pid = waitpid((pid_t)-1,
1124                               &status,
1125                               nohang ? WNOHANG : 0);
1126                 waiterr = errno;
1127                 if (!nohang) {
1128                         (void) alarm(0);
1129                 }
1130                 if (pid <= 0) {
1131                         if (waiterr == EINTR) {
1132                                 if (waitflg) {
1133                                         continue;
1134                                 } else {
1135                                         return;
1136                                 }
1137                         } else {
1138                                 return;
1139                         }
1140                 }
1141                 for (rp = running_list;
1142                      (rp != NULL) && (rp->pid != pid);
1143                      rp = rp->next) {
1144                         ;
1145                 }
1146                 if (rp == NULL) {
1147                         if (send_mtool_msgs) {
1148                                 continue;
1149                         } else {
1147                         fatal(catgets(catd, 1, 128, "Internal error: returned ch
1151                         }
1148                 } else {
1149                         rp->state = (WIFEXITED(status) && WEXITSTATUS(status) ==
1150                 }
1151                 nohang = true;
1152                 parallel_process_cnt--;

1154 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
```



```
1155                 if (job_adjust_mode == ADJUST_M2) {
1156                         if (m2_release_job()) {
1157                                 job_adjust_error();
1158                         }
1159                 }
1160 #endif
1161         }
1162 }
```

_____*unchanged_portion_omitted_*

```
1768 /*
1769  * This function replaces the makesh binary.
1770  */
1771

1773 static pid_t
1774 run_rule_commands(char *host, char **commands)
1775 {
1776         Boolean         always_exec;
1777         Name            command;
1778         Boolean         ignore;
1779         int             length;
1780         Doname          result;
1781         Boolean         silent_flag;
1782         wchar_t         *tmp_wcs_buffer;

1784         childPid = fork();
1785         switch (childPid) {
1786         case -1:        /* Error */
1787                 fatal(catgets(catd, 1, 337, "Could not fork child process for dm
1788                       errmsg(errno));
1789                 break;
1790         case 0:         /* Child */
1791                 /* To control the processed targets list is not the child's busi
1792                 running_list = NULL;
1793                 if(out_err_same) {
1794                         redirect_io(stdout_file, (char*)NULL);
1795                 } else {
1796                         redirect_io(stdout_file, stderr_file);
1797                 }
1798                 for (commands = commands;
1799                      (*commands != (char *)NULL);
1800                      commands++) {
1801                         silent_flag = silent;
1802                         ignore = false;
1803                         always_exec = false;
1804                         while ((**commands == (int) at_char) ||
1805                                (**commands == (int) hyphen_char) ||
1806                                (**commands == (int) plus_char)) {
1807                                 if (**commands == (int) at_char) {
1808                                         silent_flag = true;
1809                                 }
1810                                 if (**commands == (int) hyphen_char) {
1811                                         ignore = true;
1812                                 }
1813                                 if (**commands == (int) plus_char) {
1814                                         always_exec = true;
1815                                 }
1816                                 (*commands)++;
1817                         }
1818                         if ((length = strlen(*commands)) >= MAXPATHLEN) {
1819                                 tmp_wcs_buffer = ALLOC_WC(length + 1);
1820                                 (void) mbstowcs(tmp_wcs_buffer, *commands, lengt
1821                                 command = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
1822                                 retmem(tmp_wcs_buffer);
1823                         } else {
```

```
1824                                    MBSTOWCS(wcs_buffer, *commands);
1825                                    command = GETNAME(wcs_buffer, FIND_LENGTH);
1826                            }
1827                            if ((command->hash.length > 0) &&
1828                                !silent_flag) {
1829                                    (void) printf("%s\n", command->string_mb);
1830                            }
1831                            result = dosys(command,
1832                                           ignore,
1833                                           false,
1834                                           false, /* bugs #4085164 & #4990057 */
1835                                           /* BOOLEAN(silent_flag && ignore), */
1836                                           always_exec,
1837                                           (Name) NULL);
1841                                           (Name) NULL,
1842                                           false);
1838                            if (result == build_failed) {
1839                                    if (silent_flag) {
1840                                            (void) printf(catgets(catd, 1, 152, "The
1841                                    }
1842                                    if (!ignore) {
1843                                            _exit(1);
1844                                    }
1845                            }
1846                    }
1847                    _exit(0);
1848                    break;
1849            default:
1850                    break;
1851            }
1852            return childPid;
1853 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   13997 Wed May 20 12:17:21 2015
new/usr/src/cmd/make/include/mk/defs.h
make: remove maketool support
**********************************************************
______unchanged_portion_omitted_


 162 /*
 163  * Typedefs for all structs
 164  */
 165 typedef struct _Cmd_line        *Cmd_line, Cmd_line_rec;
 166 typedef struct _Dependency      *Dependency, Dependency_rec;
 167 typedef struct _Macro           *Macro, Macro_rec;
 168 typedef struct _Name_vector     *Name_vector, Name_vector_rec;
 169 typedef struct _Percent         *Percent, Percent_rec;
 170 typedef struct _Dyntarget       *Dyntarget;
 171 typedef struct _Recursive_make  *Recursive_make, Recursive_make_rec;
 172 typedef struct _Running         *Running, Running_rec;


 175 /*
 176  *      extern declarations for all global variables.
 177  *      The actual declarations are in globals.cc
 178  */
 179 extern  Boolean         allrules_read;
 180 extern  Name            posix_name;
 181 extern  Name            svr4_name;
 182 extern  Boolean         sdot_target;
 183 extern  Boolean         all_parallel;
 184 extern  Boolean         assign_done;
 185 extern  Boolean         build_failed_seen;
 186 extern  Name            built_last_make_run;
 187 extern  Name            c_at;
 188 extern  Boolean         command_changed;
 189 extern  Boolean         commands_done;
 190 extern  Chain           conditional_targets;
 191 extern  Name            conditionals;
 192 extern  Boolean         continue_after_error;
 193 extern  Property        current_line;
 194 extern  Name            current_make_version;
 195 extern  Name            current_target;
 196 extern  short           debug_level;
 197 extern  Cmd_line        default_rule;
 198 extern  Name            default_rule_name;
 199 extern  Name            default_target_to_build;
 200 extern  Boolean         depinfo_already_read;
 201 extern  Name            dmake_group;
 202 extern  Name            dmake_max_jobs;
 203 extern  Name            dmake_mode;
 204 extern  DMake_mode      dmake_mode_type;
 205 extern  Name            dmake_output_mode;
 206 extern  DMake_output_mode       output_mode;
 207 extern  Name            dmake_odir;
 208 extern  Name            dmake_rcfile;
 209 extern  Name            done;
 210 extern  Name            dot;
 211 extern  Name            dot_keep_state;
 212 extern  Name            dot_keep_state_file;
 213 extern  Name            empty_name;
 214 extern  Boolean         fatal_in_progress;
 215 extern  int             file_number;
 216 extern  Name            force;
 217 extern  Name            ignore_name;
 218 extern  Boolean         ignore_errors;
 219 extern  Boolean         ignore_errors_all;
```



```
 220 extern  Name            init;
 221 extern  int             job_msg_id;
 222 extern  Boolean         keep_state;
 223 extern  Name            make_state;
 224 extern  timestruc_t     make_state_before;
 225 extern  Boolean         make_state_locked;
 226 extern  Dependency      makefiles_used;
 227 extern  Name            makeflags;
 228 extern  Name            make_version;
 229 extern  char            mbs_buffer2[];
 230 extern  char            *mbs_ptr;
 231 extern  char            *mbs_ptr2;
 232 extern  Boolean         no_action_was_taken;
 233 extern  int             mtool_msgs_fd;
 233 extern  Boolean         no_parallel;
 234 extern  Name            no_parallel_name;
 235 extern  Name            not_auto;
 236 extern  Boolean         only_parallel;
 237 extern  Boolean         parallel;
 238 extern  Name            parallel_name;
 239 extern  Name            localhost_name;
 240 extern  int             parallel_process_cnt;
 241 extern  Percent         percent_list;
 242 extern  Dyntarget       dyntarget_list;
 243 extern  Name            plus;
 244 extern  Name            pmake_machinesfile;
 245 extern  Name            precious;
 246 extern  Name            primary_makefile;
 247 extern  Boolean         quest;
 248 extern  short           read_trace_level;
 249 extern  Boolean         reading_dependencies;
 250 extern  int             recursion_level;
 251 extern  Name            recursive_name;
 252 extern  short           report_dependencies_level;
 253 extern  Boolean         report_pwd;
 254 extern  Boolean         rewrite_statefile;
 255 extern  Running         running_list;
 256 extern  char            *sccs_dir_path;
 257 extern  Name            sccs_get_name;
 258 extern  Name            sccs_get_posix_name;
 259 extern  Cmd_line        sccs_get_rule;
 260 extern  Cmd_line        sccs_get_org_rule;
 261 extern  Cmd_line        sccs_get_posix_rule;
 262 extern  Name            get_name;
 263 extern  Name            get_posix_name;
 264 extern  Cmd_line        get_rule;
 265 extern  Cmd_line        get_posix_rule;
 267 extern  Boolean         send_mtool_msgs;
 266 extern  Boolean         all_precious;
 267 extern  Boolean         report_cwd;
 268 extern  Boolean         silent_all;
 269 extern  Boolean         silent;
 270 extern  Name            silent_name;
 271 extern  char            *stderr_file;
 272 extern  char            *stdout_file;
 273 extern  Boolean         stdout_stderr_same;
 274 extern  Dependency      suffixes;
 275 extern  Name            suffixes_name;
 276 extern  Name            sunpro_dependencies;
 277 extern  Boolean         target_variants;
 278 extern  const char      *tmpdir;
 279 extern  const char      *temp_file_directory;
 280 extern  Name            temp_file_name;
 281 extern  short           temp_file_number;
 282 extern  wchar_t         *top_level_target;
 283 extern  Boolean         touch;
```

```
284 extern  Boolean         trace_reader;
285 extern  Boolean         build_unconditional;
286 extern  pathpt          vroot_path;
287 extern  Name            wait_name;
288 extern  wchar_t         wcs_buffer2[];
289 extern  wchar_t         *wcs_ptr;
290 extern  wchar_t         *wcs_ptr2;
291 extern  nl_catd         catd;
292 extern  long int        hostid;

294 /*
295  * Declarations of system defined variables
296  */
297 /* On linux this variable is defined in 'signal.h' */
298 extern  char            *sys_siglist[];

300 /*
301  * Declarations of system supplied functions
302  */
303 extern  int             file_lock(char *, char *, int *, int);

305 /*
306  * Declarations of functions declared and used by make
307  */
308 extern  void            add_pending(Name target, int recursion_level, Boolean do
309 extern  void            add_running(Name target, Name true_target, Property comm
310 extern  void            add_serial(Name target, int recursion_level, Boolean do_
311 extern  void            add_subtree(Name target, int recursion_level, Boolean do
312 extern  void            append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar
313 extern  void            await_parallel(Boolean waitflg);
314 extern  void            build_suffix_list(Name target_suffix);
315 extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
316 extern  void            check_state(Name temp_file_name);
317 extern  void            cond_macros_into_string(Name np, String_rec *buffer);
318 extern  void            construct_target_string();
319 extern  void            create_xdrs_ptr(void);
320 extern  void            depvar_add_to_list (Name name, Boolean cmdline);
321 extern  Doname          doname(register Name target, register Boolean do_get, re
322 extern  Doname          doname_check(register Name target, register Boolean do_g
323 extern  Doname          doname_parallel(Name target, Boolean do_get, Boolean imp
324 extern  Doname          dosys(register Name command, register Boolean ignore_err
326 extern  Doname          dosys(register Name command, register Boolean ignore_err
325 extern  void            dump_make_state(void);
326 extern  void            dump_target_list(void);
327 extern  void            enter_conditional(register Name target, Name name, Name
328 extern  void            enter_dependencies(register Name target, Chain target_gr
329 extern  void            enter_dependency(Property line, register Name depe, Bool
330 extern  void            enter_equal(Name name, Name value, register Boolean appe
331 extern  Percent         enter_percent(register Name target, Chain target_group,
332 extern  Dyntarget       enter_dyntarget(register Name target);
333 extern  Name_vector     enter_name(String string, Boolean tail_present, register
334 extern  Boolean         exec_vp(register char *name, register char **argv, char
335 extern  Doname          execute_parallel(Property line, Boolean waitflg, Boolean
336 extern  Doname          execute_serial(Property line);
337 extern  timestruc_t&    exists(register Name target);
338 extern  void            fatal(const char *, ...);
339 extern  void            fatal_reader(char *, ...);
340 extern  Doname          find_ar_suffix_rule(register Name target, Name true_targ
341 extern  Doname          find_double_suffix_rule(register Name target, Property *
342 extern  Doname          find_percent_rule(register Name target, Property *comman
343 extern  int             find_run_directory (char *cmd, char *cwd, char *dir, cha
344 extern  Doname          find_suffix_rule(Name target, Name target_body, Name tar
345 extern  Chain           find_target_groups(register Name_vector target_list, reg
346 extern  void            finish_children(Boolean docheck);
347 extern  void            finish_running(void);
348 extern  void            free_chain(Name_vector ptr);
```



```
349 extern  void            gather_recursive_deps(void);
350 extern  char            *get_current_path(void);
351 extern  int             get_job_msg_id(void);
354 extern  FILE            *get_mtool_msgs_fp(void);
352 extern  wchar_t         *getmem_wc(register int size);
353 /* On linux getwd(char *) is defined in 'unistd.h' */
354 #ifdef __cplusplus
355 extern "C" {
356 #endif
357 extern  char            *getwd(char *);
358 #ifdef __cplusplus
359 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    1844 Wed May 20 12:17:21 2015
new/usr/src/cmd/make/include/mksh/dosys.h
make: remove maketool support
**********************************************************
```

```
#ifndef _MKSH_DOSYS_H
#define _MKSH_DOSYS_H
/*
 * 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.
 */

#include <mksh/defs.h>
#include <vroot/vroot.h>

extern Boolean  await(register Boolean ignore_error, register Boolean silent_err
extern int      doexec(register wchar_t *command, register Boolean ignore_error,
extern int      doshell(wchar_t *command, register Boolean ignore_error, char *s
extern Doname   dosys_mksh(register Name command, register Boolean ignore_error,
extern Boolean  await(register Boolean ignore_error, register Boolean silent_err
extern int      doexec(register wchar_t *command, register Boolean ignore_error,
extern int      doshell(wchar_t *command, register Boolean ignore_error, Boolean
extern Doname   dosys_mksh(register Name command, register Boolean ignore_error,
extern void     redirect_io(char *stdout_file, char *stderr_file);
extern void     sh_command2string(register String command, register String desti

#endif
```

```
**********************************************************
   15683 Wed May 20 12:17:21 2015
new/usr/src/cmd/make/lib/mksh/dosys.cc
make: remove maketool support
**********************************************************
```

```
/*
 * 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.
 */


/*
 *      dosys.cc
 *
 *      Execute one commandline
 */

/*
 * Included files
 */
#include <sys/wait.h>                   /* WIFEXITED(status) */
#include <alloca.h>             /* alloca() */

#include <stdio.h>              /* errno */
#include <errno.h>              /* errno */
#include <fcntl.h>              /* open() */
#include <mksh/dosys.h>
#include <mksh/macro.h>         /* getvar() */
#include <mksh/misc.h>          /* getmem(), fatal_mksh(), errmsg() */
#include <mksdmsi18n/mksdmsi18n.h>      /* libmksdmsi18n_init() */
#include <sys/signal.h>         /* SIG_DFL */
#include <sys/stat.h>           /* open() */
#include <sys/wait.h>           /* wait() */
#include <ulimit.h>             /* ulimit() */
#include <unistd.h>             /* close(), dup2() */


/*
 * Defined macros
 */
#define SEND_MTOOL_MSG(cmds)

/*
 * typedefs & structs
 */
```



```
/*
 * Static variables
 */

/*
 * File table of contents
 */
static Boolean  exec_vp(register char *name, register char **argv, char **envp,

/*
 * Workaround for NFS bug. Sometimes, when running 'open' on a remote
 * dmake server, it fails with "Stale NFS file handle" error.
 * The second attempt seems to work.
 */
int
my_open(const char *path, int oflag, mode_t mode) {
        int res = open(path, oflag, mode);
        if (res < 0 && (errno == ESTALE || errno == EAGAIN)) {
                /* Stale NFS file handle. Try again */
                res = open(path, oflag, mode);
        }
        return res;
}
```

_____*unchanged_portion_omitted_*

```
/*
 *      dosys_mksh(command, ignore_error, call_make, silent_error, target)
 *
 *      Check if command string contains meta chars and dispatch to
 *      the proper routine for executing one command line.
 *
 *      Return value:
 *                              Indicates if the command execution failed
 *
 *      Parameters:
 *              command         The command to run
 *              ignore_error    Should we abort when an error is seen?
 *              call_make       Did command reference $(MAKE) ?
 *              silent_error    Should error messages be suppressed for dmake?
 *              target          Target we are building
 *
 *      Global variables used:
 *              do_not_exec_rule Is -n on?
 *              working_on_targets We started processing real targets
 */
Doname
dosys_mksh(register Name command, register Boolean ignore_error, register Boolea
{
        register int            length = command->hash.length;
        register wchar_t        *p;
        register wchar_t        *q;
        register wchar_t        *cmd_string;
        struct stat             before;
        Doname                  result;
        Boolean                 working_on_targets_mksh = true;
        Wstring wcb(command);
        p = wcb.get_string();
        cmd_string = p;

        /* Strip spaces from head of command string */
        while (iswspace(*p)) {
                p++, length--;
        }
        if (*p == (int) nul_char) {
                return build_failed;
```

```
178         }
179         /* If we are faking it we just return */
180         if (do_not_exec_rule &&
181             working_on_targets_mksh &&
182             !call_make &&
183             !always_exec) {
184                 return build_ok;
185         }

187         /* Copy string to make it OK to write it. */
188         q = ALLOC_WC(length + 1);
189         (void) wscpy(q, p);
190         /* Write the state file iff this command uses make. */
191 /* XXX - currently does not support recursive make's, $(MAKE)'s
192         if (call_make && command_changed) {
193                 write_state_file(0, false);
194         }
195         (void) stat(make_state->string_mb, &before);
196  */
197         /*
198          * Run command directly if it contains no shell meta chars,
199          * else run it using the shell.
200          */
201         /* XXX - command->meta *may* not be set correctly */
202         if (await(ignore_error,
203                   silent_error,
204                   target,
205                   cmd_string,
206                   command->meta ?
207                     doshell(q, ignore_error, redirect_out_err, stdout_file, stde
208                     doexec(q, ignore_error, redirect_out_err, stdout_file, stder
209                   false,
210                   NULL,
211                   -1)) {

213 #ifdef PRINT_EXIT_STATUS
214                 warning_mksh(NOCATGETS("I'm in dosys_mksh(), and await() returne
215 #endif

217                 result = build_ok;
218         } else {

220 #ifdef PRINT_EXIT_STATUS
221                 warning_mksh(NOCATGETS("I'm in dosys_mksh(), and await() returne
222 #endif

224                 result = build_failed;
225         }
226         retmem(q);

228 /* XXX - currently does not support recursive make's, $(MAKE)'s
229         if ((report_dependencies_level == 0) &&
230             call_make) {
231                 make_state->stat.time = (time_t)file_no_time;
232                 (void)exists(make_state);
233                 if (before.st_mtime == make_state->stat.time) {
234                         return result;
235                 }
236                 makefile_type = reading_statefile;
237                 if (read_trace_level > 1) {
238                         trace_reader = true;
239                 }
240                 (void) read_simple_file(make_state,
241                                         false,
242                                         false,
243                                         false,
```



```
244                                         false,
245                                         false,
246                                         true);
247                 trace_reader = false;
248         }
249  */
250         return result;
251 }

253 /*
132  *      doshell(command, ignore_error)
133  *
134  *      Used to run command lines that include shell meta-characters.
135  *      The make macro SHELL is supposed to contain a path to the shell.
136  *
137  *      Return value:
138  *                              The pid of the process we started
139  *
140  *      Parameters:
141  *              command         The command to run
142  *              ignore_error    Should we abort on error?
143  *
144  *      Global variables used:
145  *              filter_stderr   If -X is on we redirect stderr
146  *              shell_name      The Name "SHELL", used to get the path to shell
147  */
148 int
149 doshell(wchar_t *command, register Boolean ignore_error, char *stdout_file, char
271 doshell(wchar_t *command, register Boolean ignore_error, Boolean redirect_out_er
150 {
151         char                    *argv[6];
152         int                     argv_index = 0;
153         int                     cmd_argv_index;
154         int                     length;
155         char                    nice_prio_buf[MAXPATHLEN];
156         register Name           shell = getvar(shell_name);
157         register char           *shellname;
158         char                    *tmp_mbs_buffer;


161         if (IS_EQUAL(shell->string_mb, "")) {
162                 shell = shell_name;
163         }
164         if ((shellname = strrchr(shell->string_mb, (int) slash_char)) == NULL) {
165                 shellname = shell->string_mb;
166         } else {
167                 shellname++;
168         }

170         /*
171          * Only prepend the /usr/bin/nice command to the original command
172          * if the nice priority, nice_prio, is NOT zero (0).
173          * Nice priorities can be a positive or a negative number.
174          */
175         if (nice_prio != 0) {
176                 argv[argv_index++] = (char *)NOCATGETS("nice");
177                 (void) sprintf(nice_prio_buf, NOCATGETS("-%d"), nice_prio);
178                 argv[argv_index++] = strdup(nice_prio_buf);
179         }
180         argv[argv_index++] = shellname;
181         argv[argv_index++] = (char*)(ignore_error ? NOCATGETS("-c") : NOCATGETS(
182         if ((length = wslen(command)) >= MAXPATHLEN) {
183                 tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
184                 (void) wcstombs(tmp_mbs_buffer, command, (length * MB_LEN_MAX) +
185                 cmd_argv_index = argv_index;
186                 argv[argv_index++] = strdup(tmp_mbs_buffer);
```

```
187                 retmem_mb(tmp_mbs_buffer);
188         } else {
189                 WCSTOMBS(mbs_buffer, command);
190                 cmd_argv_index = argv_index;
191                 argv[argv_index++] = strdup(mbs_buffer);
192         }
193         argv[argv_index] = NULL;
194         (void) fflush(stdout);
195         if ((childPid = fork()) == 0) {
196                 enable_interrupt((void (*) (int)) SIG_DFL);
319                 if (redirect_out_err) {
320                         redirect_io(stdout_file, stderr_file);
321                 }
197 #if 0
198                 if (filter_stderr) {
199                         redirect_stderr();
200                 }
201 #endif
202                 if (nice_prio != 0) {
203                         (void) execve(NOCATGETS("/usr/bin/nice"), argv, environ)
204                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 92, "Could not
205                               errmsg(errno));
206                 } else {
207                         (void) execve(shell->string_mb, argv, environ);
208                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 93, "Could not
209                               shell->string_mb,
210                               errmsg(errno));
211                 }
212         }
213         if (childPid  == -1) {
214                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 94, "fork failed: %s")
215                       errmsg(errno));
216         }
217         retmem_mb(argv[cmd_argv_index]);
218         return childPid;
219 }
_____unchanged_portion_omitted_

297 /*
298  *      doexec(command, ignore_error)
299  *
300  *      Will scan an argument string and split it into words
301  *      thus building an argument list that can be passed to exec_ve()
302  *
303  *      Return value:
304  *                              The pid of the process started here
305  *
306  *      Parameters:
307  *              command         The command to run
308  *              ignore_error    Should we abort on error?
309  *
310  *      Global variables used:
311  *              filter_stderr   If -X is on we redirect stderr
312  */
313 int
314 doexec(register wchar_t *command, register Boolean ignore_error, char *stdout_fi
439 doexec(register wchar_t *command, register Boolean ignore_error, Boolean redirec
315 {
316         int                     arg_count = 5;
317         char                    **argv;
318         int                     length;
319         char                    nice_prio_buf[MAXPATHLEN];
320         register char           **p;
321         wchar_t                 *q;
322         register wchar_t        *t;
323         char                    *tmp_mbs_buffer;
```



```
325         /*
326          * Only prepend the /usr/bin/nice command to the original command
327          * if the nice priority, nice_prio, is NOT zero (0).
328          * Nice priorities can be a positive or a negative number.
329          */
330         if (nice_prio != 0) {
331                 arg_count += 2;
332         }
333         for (t = command; *t != (int) nul_char; t++) {
334                 if (iswspace(*t)) {
335                         arg_count++;
336                 }
337         }
338         argv = (char **)alloca(arg_count * (sizeof(char *)));
339         /*
340          * Reserve argv[0] for sh in case of exec_vp failure.
341          * Don't worry about prepending /usr/bin/nice command to argv[0].
342          * In fact, doing it may cause the sh command to fail!
343          */
344         p = &argv[1];
345         if ((length = wslen(command)) >= MAXPATHLEN) {
346                 tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
347                 (void) wcstombs(tmp_mbs_buffer, command, (length * MB_LEN_MAX) +
348                 argv[0] = strdup(tmp_mbs_buffer);
349                 retmem_mb(tmp_mbs_buffer);
350         } else {
351                 WCSTOMBS(mbs_buffer, command);
352                 argv[0] = strdup(mbs_buffer);
353         }

355         if (nice_prio != 0) {
356                 *p++ = strdup(NOCATGETS("/usr/bin/nice"));
357                 (void) sprintf(nice_prio_buf, NOCATGETS("-%d"), nice_prio);
358                 *p++ = strdup(nice_prio_buf);
359         }
360         /* Build list of argument words. */
361         for (t = command; *t;) {
362                 if (p >= &argv[arg_count]) {
363                         /* This should never happen, right? */
364                         WCSTOMBS(mbs_buffer, command);
365                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 95, "Command '
366                               mbs_buffer,
367                               arg_count);
368                 }
369                 q = t;
370                 while (!iswspace(*t) && (*t != (int) nul_char)) {
371                         t++;
372                 }
373                 if (*t) {
374                         for (*t++ = (int) nul_char; iswspace(*t); t++);
375                 }
376                 if ((length = wslen(q)) >= MAXPATHLEN) {
377                         tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
378                         (void) wcstombs(tmp_mbs_buffer, q, (length * MB_LEN_MAX)
379                         *p++ = strdup(tmp_mbs_buffer);
380                         retmem_mb(tmp_mbs_buffer);
381                 } else {
382                         WCSTOMBS(mbs_buffer, q);
383                         *p++ = strdup(mbs_buffer);
384                 }
385         }
386         *p = NULL;

388         /* Then exec the command with that argument list. */
389         (void) fflush(stdout);
```

```
390         if ((childPid = fork()) == 0) {
391                 enable_interrupt((void (*) (int)) SIG_DFL);
517                 if (redirect_out_err) {
518                         redirect_io(stdout_file, stderr_file);
519                 }
392 #if 0
393                 if (filter_stderr) {
394                         redirect_stderr();
395                 }
396 #endif
397                 (void) exec_vp(argv[1], argv, environ, ignore_error, vroot_path)
398                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 96, "Cannot load comma
399         }
400         if (childPid  == -1) {
401                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 97, "fork failed: %s")
402                       errmsg(errno));
403         }
404         for (int i = 0; argv[i] != NULL; i++) {
405                 retmem_mb(argv[i]);
406         }
407         return childPid;
408 }

410 /*
411  *      await(ignore_error, silent_error, target, command, running_pid)
412  *
413  *      Wait for one child process and analyzes
414  *      the returned status when the child process terminates.
415  *
416  *      Return value:
417  *                              Returns true if commands ran OK
418  *
419  *      Parameters:
420  *              ignore_error    Should we abort on error?
421  *              silent_error    Should error messages be suppressed for dmake?
422  *              target          The target we are building, for error msgs
423  *              command         The command we ran, for error msgs
424  *              running_pid     The pid of the process we are waiting for
425  *
426  *      Static variables used:
427  *              filter_file     The fd for the filter file
428  *              filter_file_name The name of the filter file
429  *
430  *      Global variables used:
431  *              filter_stderr   Set if -X is on
432  */
433 Boolean
434 await(register Boolean ignore_error, register Boolean silent_error, Name target,
562 await(register Boolean ignore_error, register Boolean silent_error, Name target,
435 {
436         int                     status;
437         char                    *buffer;
438         int                     core_dumped;
439         int                     exit_status;
440         FILE                    *outfp;
441         register pid_t          pid;
442         struct stat             stat_buff;
443         int                     termination_signal;
444         char                    tmp_buf[MAXPATHLEN];

446         while ((pid = wait(&status)) != running_pid) {
447                 if (pid == -1) {
448                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 98, "wait() fa
449                 }
450         }
451         (void) fflush(stdout);
```



```
452         (void) fflush(stderr);

454         if (status == 0) {

456 #ifdef PRINT_EXIT_STATUS
457                 warning_mksh(NOCATGETS("I'm in await(), and status is 0."));
458 #endif

460                 return succeeded;
461         }

463 #ifdef PRINT_EXIT_STATUS
464         warning_mksh(NOCATGETS("I'm in await(), and status is *NOT* 0."));
465 #endif


468         exit_status = WEXITSTATUS(status);

470 #ifdef PRINT_EXIT_STATUS
471         warning_mksh(NOCATGETS("I'm in await(), and exit_status is %d."), exit_s
472 #endif

474         termination_signal = WTERMSIG(status);
475         core_dumped = WCOREDUMP(status);

477         /*
478          * If the child returned an error, we now try to print a
479          * nice message about it.
480          */
609         SEND_MTOOL_MSG(
610                 make_output_msg = new Avo_CmdOutput();
611                 (void) sprintf(tmp_buf, "%d", job_msg_id);
612                 make_output_msg->appendOutput(strdup(tmp_buf));
613         );
481
482         tmp_buf[0] = (int) nul_char;
483         if (!silent_error) {
484                 if (exit_status != 0) {
485                         (void) fprintf(stdout,
486                                 catgets(libmksdmsi18n_catd, 1, 103, "***
487                                 exit_status);
621                         SEND_MTOOL_MSG(
622                                 (void) sprintf(&tmp_buf[strlen(tmp_buf)],
623                                         catgets(libmksdmsi18n_catd, 1, 10
624                                         exit_status);
625                         );
488                 } else {
489                         (void) fprintf(stdout,
490                                 catgets(libmksdmsi18n_catd, 1, 10
491                                 termination_signal);
630                         SEND_MTOOL_MSG(
631                                 (void) sprintf(&tmp_buf[strlen(tmp_buf)]
632                                         catgets(libmksdmsi18n_cat
633                                         termination_signal);
634                         );
492                         if (core_dumped) {
493                                 (void) fprintf(stdout,
494                                         catgets(libmksdmsi18n_catd, 1, 10
638                                 SEND_MTOOL_MSG(
639                                         (void) sprintf(&tmp_buf[strlen(tmp_buf)]
640                                                 catgets(libmksdmsi18n_cat
641                                 );
495                         }
496                 }
497                 if (ignore_error) {
498                         (void) fprintf(stdout,
```

```
499                                     catgets(libmksdmsi18n_catd, 1, 109, " (ig
647                         SEND_MTOOL_MSG(
648                                 (void) sprintf(&tmp_buf[strlen(tmp_buf)],
649                                         catgets(libmksdmsi18n_catd, 1, 11
650                         );
500                 }
501                 (void) fprintf(stdout, "\n");
502                 (void) fflush(stdout);
503         }
654                 SEND_MTOOL_MSG(
655                         make_output_msg->appendOutput(strdup(tmp_buf));
656                 );
657         }
658         SEND_MTOOL_MSG(
659                 xdr_msg = (RWCollectable*) make_output_msg;
660                 xdr(xdrs_p, xdr_msg);
661                 delete make_output_msg;
662         );

505 #ifdef PRINT_EXIT_STATUS
506         warning_mksh(NOCATGETS("I'm in await(), returning failed."));
507 #endif

509         return failed;
510 }
```

_____unchanged_portion_omitted_