

```

*****
23115 Fri May 22 11:19:39 2015
new/usr/src/cmd/make/bin/ar.cc
make: use the more modern wchar routines, not widec.h
*****
_____unchanged_portion_omitted_____

150 /*
151  * Static variables
152  */

154 /*
155  * File table of contents
156  */
157 extern timestruc_t& read_archive(register Name target);
158 static Boolean open_archive(char *filename, register Ar *arp);
159 static void close_archive(register Ar *arp);
160 static Boolean read_archive_dir(register Ar *arp, Name library, char **
161 static void translate_entry(register Ar *arp, Name target, register
162 static long sgetl(char *);

164 /*
165  * read_archive(target)
166  *
167  * Read the contents of an ar file.
168  *
169  * Return value:
170  *           The time the member was created
171  *
172  * Parameters:
173  *   target   The member to find time for
174  *
175  * Global variables used:
176  *   empty_name The Name ""
177  */

179 int read_member_header (Ar_port *header, FILE *fd, char* filename);
180 int process_long_names_member (register Ar *arp, char **long_names_table, char *

182 timestruc_t&
183 read_archive(register Name target)
184 {
185     register Property member;
186     wchar_t *slash;
187     String_rec true_member_name;
188     wchar_t buffer[STRING_BUFFER_LENGTH];
189     register Name true_member = NULL;
190     Ar ar;
191     char *long_names_table = NULL; /* Table of long
192                                     member names */

194     member = get_prop(target->prop, member_prop);
195     /*
196     * Check if the member has directory component.
197     * If so, remove the dir and see if we know the date.
198     */
199     if (member->body.member.member != NULL) {
200         Wstring member_string(member->body.member.member);
201         wchar_t * wcb = member_string.get_string();
202         if((slash = (wchar_t *) wcsrchr(wcb, (int) slash_char)) != NULL)
203             if((slash = (wchar_t *) wsrchr(wcb, (int) slash_char)) != NULL)
204                 INIT_STRING_FROM_STACK(true_member_name, buffer);
205         append_string(member->body.member.library->string_mb,
206                     &true_member_name,
207                     FIND_LENGTH);
208         append_char((int) parenleft_char, &true_member_name);

```

```

208     append_string(slash + 1, &true_member_name, FIND_LENGTH);
209     append_char((int) parenright_char, &true_member_name);
210     true_member = GETNAME(true_member_name.buffer.start,
211                          FIND_LENGTH);
212     if (true_member->stat.time != file_no_time) {
213         target->stat.time = true_member->stat.time;
214         return target->stat.time;
215     }
216 }
217 }
218 if (open_archive(member->body.member.library->string_mb, &ar) == failed)
219     if (errno == ENOENT) {
220         target->stat.stat_errno = ENOENT;
221         close_archive(&ar);
222         if (member->body.member.member == NULL) {
223             member->body.member.member = empty_name;
224         }
225         return target->stat.time = file_doesnt_exist;
226     } else {
227         fatal(gettext("Can't access archive '%s': %s"),
228             member->body.member.library->string_mb,
229             errmsg(errno));
230     }
231 }
232 if (target->stat.time == file_no_time) {
233     if (read_archive_dir(&ar, member->body.member.library,
234                        &long_names_table)
235         == failed){
236         fatal(gettext("Can't access archive '%s': %s"),
237             member->body.member.library->string_mb,
238             errmsg(errno));
239     }
240 }
241 if (member->body.member.entry != NULL) {
242     translate_entry(&ar, target, member,&long_names_table);
243 }
244 close_archive(&ar);
245 if (long_names_table) {
246     retmem_mb(long_names_table);
247 }
248 if (true_member != NULL) {
249     target->stat.time = true_member->stat.time;
250 }
251 if (target->stat.time == file_no_time) {
252     target->stat.time = file_doesnt_exist;
253 }
254 return target->stat.time;
255 }
_____unchanged_portion_omitted_____

604 /*
605  * translate_entry(arp, target, member)
606  *
607  * Finds the member for one lib.a((entry))
608  *
609  * Parameters:
610  *   arp           Pointer to ar file description block
611  *   target        Target to find member name for
612  *   member        Property to fill in with info
613  *
614  * Global variables used:
615  */
616 static void
617 translate_entry(register Ar *arp, Name target, register Property member, char **
618 {
619     register int len;

```

```

620     register int      i;
621     wchar_t          *member_string;
622     ar_port_word     *offs;
623     int              strtablen;
624     char             *syms;          /* string table */
625     char             *csym;         /* string table */
626     ar_port_word     *offend;      /* end of offsets table */
627     int              date;
628     register wchar_t *ap;
629     register char    *hp;
630     int              maxs;
631     int              offset;
632     char             buffer[4];

634     if (arp->sym_begin == 0L || arp->num_symbols == 0L) {
635         fatal(gettext("Cannot find symbol '%s' in archive '%s'"),
636             member->body.member.entry->string_mb,
637             member->body.member.library->string_mb);
638     }

640     if (fseek(arp->fd, arp->sym_begin, 0) != 0) {
641         goto read_error;
642     }
643     member_string = ALLOC_WC((int) ((int) ar_member_name_len * 2));

645     switch (arp->type) {
646     case AR_5:
647         if ((len = member->body.member.entry->hash.length) > 8) {
648             len = 8;
649         }
650         for (i = 0; i < arp->num_symbols; i++) {
651             if (fread((char *) &arp->ars_5,
652                 sizeof arp->ars_5,
653                 1,
654                 arp->fd) != 1) {
655                 goto read_error;
656             }
657             if (IS_EQUALN(arp->ars_5.sym_name,
658                 member->body.member.entry->string_mb,
659                 len)) {
660                 if ((fseek(arp->fd,
661                     sgetl(arp->ars_5.sym_ptr),
662                     0) != 0) ||
663                     (fread((char *) &arp->arf_5,
664                         sizeof arp->arf_5,
665                         1,
666                         arp->fd) != 1)) {
667                     goto read_error;
668                 }
669                 MBSTOWCS(wcs_buffer, arp->arf_5.arf_name);
670                 (void) wcsncpy(member_string,
671                     (void) wcsncpy(member_string,
672                         wcs_buffer,
673                         wcslen(wcs_buffer));
674                 wslen(wcs_buffer));
675                 member_string[sizeof(arp->arf_5.arf_name)] =
676                     (int) nul_char;
677                 member->body.member.member =
678                     GETNAME(member_string, FIND_LENGTH);
679                 target->stat.time.tv_sec = sgetl(arp->arf_5.arf_
680                 target->stat.time.tv_nsec = LONG_MAX;
681                 return;
682             }
683         }
684     case AR_PORT:

```

```

684     offs = (ar_port_word *) alloca((int) (arp->num_symbols * AR_PORT
685     if (fread((char *) offs,
686         AR_PORT_WORD,
687         (int) arp->num_symbols,
688         arp->fd) != arp->num_symbols) {
689         goto read_error;
690     }

692     for(i=0;i<arp->num_symbols;i++) {
693         *((int*)buffer)=offs[i];
694         offs[i]=(ar_port_word)sgetl(buffer);
695     }

697     strtablen=arp->sym_size-4-(int) (arp->num_symbols * AR_PORT_WORD
698     syms = (char *) alloca(strtablen);
699     if (fread(syms,
700         sizeof (char),
701         strtablen,
702         arp->fd) != strtablen) {
703         goto read_error;
704     }
705     offend = &offs[arp->num_symbols];
706     while (offs < offend) {
707         maxs = strlen(member->body.member.entry->string_mb);
708         if (strlen(syms) > maxs)
709             maxs = strlen(syms);
710         if (IS_EQUALN(syms,
711             member->body.member.entry->string_mb,
712             maxs)) {
713             if (fseek(arp->fd,
714                 (long) *offs,
715                 0) != 0) {
716                 goto read_error;
717             }
718             if ((fread((char *) &arp->ar_port,
719                 sizeof arp->ar_port,
720                 1,
721                 arp->fd) != 1) ||
722                 !IS_EQUALN(arp->ar_port.ar_fmags,
723                     AR_PORT_END_MAGIC,
724                     sizeof arp->ar_port.ar_fmags)) {
725                 goto read_error;
726             }
727             if (sscanf(arp->ar_port.ar_date,
728                 "%ld",
729                 &date) != 1) {
730                 fatal(gettext("Bad date field for member
731                 arp->ar_port.ar_name,
732                 target->string_mb);
733             }
734             /* If it's a long name, retrieve it from long name table */
735             if (arp->ar_port.ar_name[0] == '/') {
736                 sscanf(arp->ar_port.ar_name + 1,
737                     "%ld",
738                     &offset);
739                 len = ar_member_name_len;
740                 hp = *long_names_table + offset;
741             } else {
742                 len = sizeof arp->ar_port.ar_name;
743                 hp = arp->ar_port.ar_name;
744             }
745             ap = member_string;
746             while (*hp &&
747                 (*hp != (int) slash_char) &&
748                 (ap < &member_string[len])) {
749                 MBTOWC(ap, hp);

```

```
750             ap++;
751             hp++;
752         }
753         *ap = (int) nul_char;
754         member->body.member.member =
755             GETNAME(member_string, FIND_LENGTH);
756         target->stat.time.tv_sec = date;
757         target->stat.time.tv_nsec = LONG_MAX;
758         return;
759     }
760     ofs++;
761     while(*syms!='\0') syms++;
762     syms++;
763 }
764 }
765 fatal(gettext("Cannot find symbol '%s' in archive '%s'"),
766        member->body.member.entry->string_mb,
767        member->body.member.library->string_mb);
768 /*NOTREACHED*/

770 read_error:
771     if (ferror(arp->fd)) {
772         fatal(gettext("Read error in archive '%s': %s"),
773              member->body.member.library->string_mb,
774              errmsg(errno));
775     } else {
776         fatal(gettext("Read error in archive '%s': Premature EOF"),
777              member->body.member.library->string_mb);
778     }
779 }
```

unchanged portion omitted

```

*****
91165 Fri May 22 11:19:40 2015
new/usr/src/cmd/make/bin/doname.cc
make: use the more modern wchar routines, not widec.h
*****
_____unchanged_portion_omitted_____

1276 /*
1277 *      dynamic_dependencies(target)
1278 *
1279 *      Checks if any dependency contains a macro ref
1280 *      If so, it replaces the dependency with the expanded version.
1281 *      Here, "$@" gets translated to target->string. That is
1282 *      the current name on the left of the colon in the
1283 *      makefile. Thus,
1284 *          xyz: s.$@.c
1285 *      translates into
1286 *          xyz: s.xyz.c
1287 *
1288 *      Also, "$@F" translates to the same thing without a preceding
1289 *      directory path (if one exists).
1290 *      Note, to enter "$@" on a dependency line in a makefile
1291 *      "$$@" must be typed. This is because make expands
1292 *      macros in dependency lists upon reading them.
1293 *      dynamic_dependencies() also expands file wildcards.
1294 *      If there are any Shell meta characters in the name,
1295 *      search the directory, and replace the dependency
1296 *      with the set of files the pattern matches
1297 *
1298 *      Parameters:
1299 *          target      Target to sanitize dependencies for
1300 *
1301 *      Global variables used:
1302 *          c_at        The Name "@", used to set macro value
1303 *          debug_level Should we trace actions?
1304 *          dot         The Name ".", used to read directory
1305 *          recursion_level Used for tracing
1306 */
1307 void
1308 dynamic_dependencies(Name target)
1309 {
1310     wchar_t      pattern[MAXPATHLEN];
1311     register wchar_t *p;
1312     Property     line;
1313     register Dependency dependency;
1314     register Dependency *remove;
1315     String_rec   string;
1316     wchar_t      buffer[MAXPATHLEN];
1317     register Boolean set_at = false;
1318     register wchar_t *start;
1319     Dependency   new_depe;
1320     register Boolean reuse_cell;
1321     Dependency   first_member;
1322     Name         directory;
1323     Name         lib;
1324     Name         member;
1325     Property     prop;
1326     Name         true_target = target;
1327     wchar_t      *library;

1329     if ((line = get_prop(target->prop, line_prop)) == NULL) {
1330         return;
1331     }
1332     /* If the target is constructed from a "::" target we consider that */
1333     if (target->has_target_prop) {
1334         true_target = get_prop(target->prop,

```

```

1335                                     target_prop->body.target.target;
1336     }
1337     /* Scan all dependencies and process the ones that contain "$" chars */
1338     for (dependency = line->body.line.dependencies;
1339          dependency != NULL;
1340          dependency = dependency->next) {
1341         if (!dependency->name->dollar) {
1342             continue;
1343         }
1344         target->has_depe_list_expanded = true;

1346         /* The make macro $@ is bound to the target name once per */
1347         /* invocation of dynamic_dependencies() */
1348         if (!set_at) {
1349             (void) SETVAR(c_at, true_target, false);
1350             set_at = true;
1351         }
1352         /* Expand this dependency string */
1353         INIT_STRING_FROM_STACK(string, buffer);
1354         expand_value(dependency->name, &string, false);
1355         /* Scan the expanded string. It could contain whitespace */
1356         /* which mean it expands to several dependencies */
1357         start = string.buffer.start;
1358         while (iswspace(*start)) {
1359             start++;
1360         }
1361         /* Remove the cell (later) if the macro was empty */
1362         if (start[0] == (int) nul_char) {
1363             dependency->name = NULL;
1364         }

1366     /* azv 10/26/95 to fix bug BID_1170218 */
1367     if ((start[0] == (int) period_char) &&
1368         (start[1] == (int) slash_char)) {
1369         start += 2;
1370     }
1371     /* azv */

1373     first_member = NULL;
1374     /* We use the original dependency cell for the first */
1375     /* dependency from the expansion */
1376     reuse_cell = true;
1377     /* We also have to deal with dependencies that expand to */
1378     /* lib.a(members) notation */
1379     for (p = start; *p != (int) nul_char; p++) {
1380         if ((*p == (int) parenleft_char)) {
1381             lib = GETNAME(start, p - start);
1382             lib->is_member = true;
1383             first_member = dependency;
1384             start = p + 1;
1385             while (iswspace(*start)) {
1386                 start++;
1387             }
1388             break;
1389         }
1390     }
1391     do {
1392         /* First skip whitespace */
1393         for (p = start; *p != (int) nul_char; p++) {
1394             if ((*p == (int) nul_char) ||
1395                 iswspace(*p) ||
1396                 (*p == (int) parenright_char)) {
1397                 break;
1398             }
1399         }
1400         /* Enter dependency from expansion */

```

```

1401     if (p != start) {
1402         /* Create new dependency cell if */
1403         /* this is not the first dependency */
1404         /* picked from the expansion */
1405         if (!reuse_cell) {
1406             new_depe = ALLOC(Dependency);
1407             new_depe->next = dependency->next;
1408             new_depe->automatic = false;
1409             new_depe->stale = false;
1410             new_depe->built = false;
1411             dependency->next = new_depe;
1412             dependency = new_depe;
1413         }
1414         reuse_cell = false;
1415         /* Internalize the dependency name */
1416         // tolik. Fix for bug 4110429: inconsistent expa
1417         // include "/" and "/"
1418         //dependency->name = GETNAME(start, p - start);
1419         dependency->name = normalize_name(start, p - sta
1420         if ((debug_level > 0) &&
1421             (first_member == NULL)) {
1422             (void) printf(gettext("%*sDynamic depend
1423                 recursion_level,
1424                 dependency->name->string_m
1425                 true_target->string_mb);
1426         }
1427         for (start = p; iswspace(*start); start++);
1428         p = start;
1429     }
1430 } while ((*p != (int) nul_char) &&
1431         (*p != (int) parenright_char));
1432 /* If the expansion was of lib.a(members) format we now */
1433 /* enter the proper member cells */
1434 if (first_member != NULL) {
1435     /* Scan the new dependencies and transform them from */
1436     /* "foo" to "lib.a(foo)" */
1437     for (; first_member = first_member->next) {
1438         /* Build "lib.a(foo)" name */
1439         INIT_STRING_FROM_STACK(string, buffer);
1440         APPEND_NAME(lib,
1441                     &string,
1442                     (int) lib->hash.length);
1443         append_char((int) parenleft_char, &string);
1444         APPEND_NAME(first_member->name,
1445                     &string,
1446                     FIND_LENGTH);
1447         append_char((int) parenright_char, &string);
1448         member = first_member->name;
1449         /* Replace "foo" with "lib.a(foo)" */
1450         first_member->name =
1451             GETNAME(string.buffer.start, FIND_LENGTH);
1452         if (string.free_after_use) {
1453             retmem(string.buffer.start);
1454         }
1455     }
1456     if (debug_level > 0) {
1457         (void) printf(gettext("%*sDynamic depend
1458             recursion_level,
1459             ",
1460             first_member->name->
1461             string_mb,
1462             true_target->string_mb);
1463     }
1464     first_member->name->is_member = lib->is_member;
1465     /* Add member property to member */
1466     prop = maybe_append_prop(first_member->name,

```

```

1467         member_prop);
1468     prop->body.member.library = lib;
1469     prop->body.member.entry = NULL;
1470     prop->body.member.member = member;
1471     if (first_member == dependency) {
1472         break;
1473     }
1474 }
1475 }
1476 }
1477 Wstring wcb;
1478 /* Then scan all the dependencies again. This time we want to expand */
1479 /* shell file wildcards */
1480 for (remove = &line->body.line.dependencies, dependency = *remove;
1481     dependency != NULL;
1482     dependency = *remove) {
1483     if (dependency->name == NULL) {
1484         dependency = *remove = (*remove)->next;
1485         continue;
1486     }
1487     /* If dependency name string contains shell wildcards */
1488     /* replace the name with the expansion */
1489     if (dependency->name->wildcard) {
1490         wcb.init(dependency->name);
1491         if ((start = (wchar_t *) wcschr(wcb.get_string(),
1492             if (start = (wchar_t *) wcschr(wcb.get_string(),
1493                 (int) parenleft_char) != NULL) {
1494             /* lib(*) type pattern */
1495             library = buffer;
1496             (void) wcsncpy(buffer,
1497                 (void) wcsncpy(buffer,
1498                     wcb.get_string(),
1499                     start - wcb.get_string());
1500             buffer[start-wcb.get_string()] =
1501                 (int) nul_char;
1502             (void) wcsncpy(pattern,
1503                 (void) wcsncpy(pattern,
1504                     start + 1,
1505                     (int) (dependency->name->hash.length - (start-wcb.get_string() - 2));
1506                     pattern[dependency->name->hash.length -
1507                         (start-wcb.get_string() - 2)] =
1508                         (int) nul_char;
1509                 } else {
1510                     library = NULL;
1511                     (void) wcsncpy(pattern,
1512                         (void) wcsncpy(pattern,
1513                             wcb.get_string(),
1514                             (int) dependency->name->hash.length
1515                             (int) nul_char;
1516                 }
1517             start = (wchar_t *) wcsrchr(pattern, (int) slash_char);
1518             start = (wchar_t *) wschr(pattern, (int) slash_char);
1519             if (start == NULL) {
1520                 directory = dot;
1521                 p = pattern;
1522             } else {
1523                 directory = GETNAME(pattern, start-pattern);
1524                 p = start+1;
1525             }
1526             /* The expansion is handled by the read_dir() routine*/
1527             if (read_dir(directory, p, line, library)) {
1528                 *remove = (*remove)->next;
1529             } else {
1530                 remove = &dependency->next;
1531             }
1532         }

```

```

1528     } else {
1529         remove = &dependency->next;
1530     }
1531 }

1533 /* Then unbind $@ */
1534 (void) SETVAR(c_at, (Name) NULL, false);
1535 }

    unchanged_portion_omitted

2651 /*
2652 *   sccs_get(target, command)
2653 *
2654 *   Figures out if it possible to sccs get a file
2655 *   and builds the command to do it if it is.
2656 *
2657 *   Return value:
2658 *               Indicates if sccs get failed or not
2659 *
2660 *   Parameters:
2661 *       target   Target to get
2662 *       command  Where to deposit command to use
2663 *
2664 *   Global variables used:
2665 *       debug_level  Should we trace activities?
2666 *       recursion_level Used for tracing
2667 *       sccs_get_rule The rule to used for sccs getting
2668 */
2669 static Doname
2670 sccs_get(register Name target, register Property *command)
2671 {
2672     register int     result;
2673     char             link[MAXPATHLEN];
2674     String_rec       string;
2675     wchar_t          name[MAXPATHLEN];
2676     register wchar_t *p;
2677     timestruc_t      sccs_time;
2678     register Property line;
2679     int              sym_link_depth = 0;

2681     /* For sccs, we need to chase symlinks. */
2682     while (target->stat.is_sym_link) {
2683         if (sym_link_depth++ > 90) {
2684             fatal(gettext("Can't read symbolic link '%s': Number of
2685             target->string_mb);
2686         }
2687         /* Read the value of the link. */
2688         result = readlink_vroot(target->string_mb,
2689                                link,
2690                                sizeof(link),
2691                                NULL,
2692                                VROOT_DEFAULT);
2693         if (result == -1) {
2694             fatal(gettext("Can't read symbolic link '%s': %s"),
2695                  target->string_mb, errmsg(errno));
2696         }
2697         link[result] = 0;
2698         /* Use the value to build the proper filename. */
2699         INIT_STRING_FROM_STACK(string, name);

2701         Wstring wcb(target);
2702         if ((link[0] != slash_char) &&
2703             ((p = (wchar_t *) wcsrchr(wcb.get_string(), slash_char)) !=
2704              (p = (wchar_t *) wsrchr(wcb.get_string(), slash_char)) != N
2705             append_string(wcb.get_string(), &string, p - wcb.get_str

```

```

2706         append_string(link, &string, result);
2707         /* Replace the old name with the translated name. */
2708         target = normalize_name(string.buffer.start, string.text.p - str
2709         (void) exists(target);
2710         if (string.free_after_use) {
2711             retmem(string.buffer.start);
2712         }
2713     }

2715     /*
2716     * read_dir() also reads the ?/SCCS dir and saves information
2717     * about which files have SCSC/s. files.
2718     */
2719     if (target->stat.has_sccs == DONT_KNOW_SCCS) {
2720         read_directory_of_file(target);
2721     }
2722     switch (target->stat.has_sccs) {
2723     case DONT_KNOW_SCCS:
2724         /* We dont know by now there is no SCCS/s.* */
2725         target->stat.has_sccs = NO_SCCS;
2726     case NO_SCCS:
2727         /*
2728         * If there is no SCCS/s.* but the plain file exists,
2729         * we say things are OK.
2730         */
2731         if (target->stat.time > file_doesnt_exist) {
2732             return build_ok;
2733         }
2734         /* If we cant find the plain file, we give up. */
2735         return build_dont_know;
2736     case HAS_SCCS:
2737         /*
2738         * Pay dirt. We now need to figure out if the plain file
2739         * is out of date relative to the SCCS/s.* file.
2740         */
2741         sccs_time = exists(get_prop(target->prop,
2742                                   sccs_prop)->body.sccs.file);
2743         break;
2744     }

2746     if ((!target->has_complained &&
2747         (sccs_time != file_doesnt_exist) &&
2748         (sccs_get_rule != NULL)) {
2749         /* only checking */
2750         if (command == NULL) {
2751             return build_ok;
2752         }
2753         /*
2754         * We provide a command line for the target. The line is a
2755         * "sccs get" command from default.mk.
2756         */
2757         line = maybe_append_prop(target, line_prop);
2758         *command = line;
2759         if (sccs_time > target->stat.time) {
2760             /*
2761             * And only if the plain file is out of date do we
2762             * request execution of the command.
2763             */
2764             line->body.line.is_out_of_date = true;
2765             if (debug_level > 0) {
2766                 (void) printf(gettext("%sScCs getting %s because
2767                 recursion_level,
2768                 "",
2769                 target->string_mb);
2770             }
2771         }

```

```

2772     line->body.line.sccs_command = true;
2773     line->body.line.command_template = sccs_get_rule;
2774     if(!svr4 && (!allrules_read || posix)) {
2775         if((target->prop) &&
2776             (target->prop->body.sccs.file) &&
2777             (target->prop->body.sccs.file->string_mb)) {
2778             if((strlen(target->prop->body.sccs.file->string_mb) ==
2779                 strlen(target->string_mb) + 2) &&
2780                 (target->prop->body.sccs.file->string_mb[0] == 's') &&
2781                 (target->prop->body.sccs.file->string_mb[1] == '.')) {
2783                 line->body.line.command_template = get_posix_rule;
2784             }
2785         }
2786     }
2787     line->body.line.target = target;
2788     /*
2789     * Also make sure the rule is build with $* and $<
2790     * bound properly.
2791     */
2792     line->body.line.star = NULL;
2793     line->body.line.less = NULL;
2794     line->body.line.percent = NULL;
2795     return build_ok;
2796 }
2797 return build_dont_know;
2798 }

2800 /*
2801 * read_directory_of_file(file)
2802 *
2803 * Reads the directory the specified file lives in.
2804 *
2805 * Parameters:
2806 *     file           The file we need to read dir for
2807 *
2808 * Global variables used:
2809 *     dot           The Name ".", used as the default dir
2810 */
2811 void
2812 read_directory_of_file(register Name file)
2813 {
2815     Wstring file_string(file);
2816     wchar_t * wcb = file_string.get_string();
2817     wchar_t usr_include_buf[MAXPATHLEN];
2818     wchar_t usr_include_sys_buf[MAXPATHLEN];

2820     register Name     directory = dot;
2821     register wchar_t *p = (wchar_t *) wcsrchr(wcb,
2822     register wchar_t *p = (wchar_t *) wsrchr(wcb,
2823         (int) slash_char);
2824     register int     length = p - wcb;
2825     static Name     usr_include;
2826     static Name     usr_include_sys;

2827     if (usr_include == NULL) {
2828         MBSTOWCS(usr_include_buf, "/usr/include");
2829         usr_include = GETNAME(usr_include_buf, FIND_LENGTH);
2830         MBSTOWCS(usr_include_sys_buf, "/usr/include/sys");
2831         usr_include_sys = GETNAME(usr_include_sys_buf, FIND_LENGTH);
2832     }

2834     /*
2835     * If the filename contains a "/" we have to extract the path
2836     * Else the path defaults to ".".

```

```

2837     /*
2838     if (p != NULL) {
2839         /*
2840         * Check some popular directories first to possibly
2841         * save time. Compare string length first to gain speed.
2842         */
2843         if ((usr_include->hash.length == length) &&
2844             IS_WEQUALN(usr_include_buf,
2845                 wcb,
2846                 length)) {
2847             directory = usr_include;
2848         } else if ((usr_include_sys->hash.length == length) &&
2849             IS_WEQUALN(usr_include_sys_buf,
2850                 wcb,
2851                 length)) {
2852             directory = usr_include_sys;
2853         } else {
2854             directory = GETNAME(wcb, length);
2855         }
2856     }
2857     (void) read_dir(directory,
2858         (wchar_t *) NULL,
2859         (Property) NULL,
2860         (wchar_t *) NULL);
2861 }

2863 /*
2864 * add_pattern_conditionals(target)
2865 *
2866 * Scan the list of conditionals defined for pattern targets and add any
2867 * that match this target to its list of conditionals.
2868 *
2869 * Parameters:
2870 *     target           The target we should add conditionals for
2871 *
2872 * Global variables used:
2873 *     conditionals    The list of pattern conditionals
2874 */
2875 static void
2876 add_pattern_conditionals(register Name target)
2877 {
2878     register Property     conditional;
2879     register Property     new_prop;
2880     register Property     *previous;
2881     register Name_rec     dummy;
2882     register wchar_t     *pattern;
2883     register wchar_t     *percent;
2884     register int         length;

2886     Wstring wcb(target);
2887     Wstring wcb1;

2889     for (conditional = get_prop(conditionals->prop, conditional_prop);
2890         conditional != NULL;
2891         conditional = get_prop(conditional->next, conditional_prop)) {
2892         wcb1.init(conditional->body.conditional.target);
2893         pattern = wcb1.get_string();
2894         if (pattern[1] != 0) {
2895             percent = (wchar_t *) wcschr(pattern, (int) percent_char
2896             percent = (wchar_t *) wcschr(pattern, (int) percent_char)
2897             if (!wcb.equaln(pattern, percent-pattern) ||
2898                 !IS_WEQUAL(wcb.get_string(wcb.length()-wcslen(percent
2899                 !IS_WEQUAL(wcb.get_string(wcb.length()-wslen(percent
2900             continue;

```

```
2901         for (previous = &target->prop;
2902              *previous != NULL;
2903              previous = &(*previous)->next) {
2904             if (((*previous)->type == conditional_prop) &&
2905                 ((*previous)->body.conditional.sequence >
2906                  conditional->body.conditional.sequence)) {
2907                 break;
2908             }
2909         }
2910         if (*previous == NULL) {
2911             new_prop = append_prop(target, conditional_prop);
2912         } else {
2913             dummy.prop = NULL;
2914             new_prop = append_prop(&dummy, conditional_prop);
2915             new_prop->next = *previous;
2916             *previous = new_prop;
2917         }
2918         target->conditional_cnt++;
2919         new_prop->body.conditional = conditional->body.conditional;
2920     }
2921 }
_____unchanged_portion_omitted_____
```



```

*****
4095 Fri May 22 11:19:41 2015
new/usr/src/cmd/make/bin/dosys.cc
make: use the more modern wchar routines, not widec.h
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  *      dosys.cc
28  *
29  *      Execute one commandline
30  */

32 /*
33  * Included files
34  */
35 #include <fcntl.h>          /* open() */
36 #include <mk/defs.h>
37 #include <mksh/dosys.h>    /* doshell(), doexec() */
38 #include <mksh/misc.h>    /* getmem() */
39 #include <sys/stat.h>     /* open() */
40 #include <unistd.h>       /* getpid() */

42 /*
43  * Defined macros
44  */

46 /*
47  * typedefs & structs
48  */

50 /*
51  * Static variables
52  */
53 static int      filter_file;
54 static char     *filter_file_name;

56 /*
57  * File table of contents
58  */
59 static void     redirect_stderr(void);

61 /*

```

```

62 *      dosys(command, ignore_error, call_make, silent_error, target)
63 *
64 *      Check if command string contains meta chars and dispatch to
65 *      the proper routine for executing one command line.
66 *
67 *      Return value:
68 *
69 *
70 *      Parameters:
71 *      command      The command to run
72 *      ignore_error Should make abort when an error is seen?
73 *      call_make    Did command reference $(MAKE) ?
74 *      silent_error Should error messages be suppressed for pmake?
75 *      target       Target we are building
76 *
77 *      Global variables used:
78 *      do_not_exec_rule Is -n on?
79 *      working_on_targets We started processing real targets
80 */
81 Doname
82 dosys(register Name command, register Boolean ignore_error, register Boolean cal
83 {
84     timestruc_t      before;
85     register int     length = command->hash.length;
86     Wstring          wcb(command);
87     register wchar_t *p = wcb.get_string();
88     register wchar_t *q;
89     Doname           result;

91     /* Strip spaces from head of command string */
92     while (iswspace(*p)) {
93         p++, length--;
94     }
95     if (*p == (int) nul_char) {
96         return build_failed;
97     }
98     /* If we are faking it we just return */
99     if (do_not_exec_rule &&
100         working_on_targets &&
101         !call_make &&
102         !always_exec) {
103         return build_ok;
104     }
105     /* no_action_was_taken is used to print special message */
106     no_action_was_taken = false;

108     /* Copy string to make it OK to write it. */
109     q = ALLOC_WC(length + 1);
110     (void) wcscopy(q, p);
111     /* Write the state file iff this command uses make. */
112     if (call_make && command_changed) {
113         write_state_file(0, false);
114     }
115     make_state->stat.time = file_no_time;
116     (void)exists(make_state);
117     before = make_state->stat.time;
118     /*
119     * Run command directly if it contains no shell meta chars,
120     * else run it using the shell.
121     */
122     if (await(ignore_error,
123             silent_error,
124             target,
125             wcb.get_string(),
126             command->meta ?

```

```
127         doshell(q, ignore_error,
128                 stdout_file, stderr_file, 0) :
129         doexec(q, ignore_error,
130               stdout_file, stderr_file,
131               vroot_path, 0),
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

```

*****
18370 Fri May 22 11:19:41 2015
new/usr/src/cmd/make/bin/files.cc
make: use the more modern wchar routines, not widec.h
*****
_____unchanged_portion_omitted_____

194 /*
195 *      vpath_exists(target)
196 *
197 *      Called if exists() discovers that there is a VPATH defined.
198 *      This function stats the VPATH translation of the target.
199 *
200 *      Return value:
201 *
202 *          The time the target was created
203 *
204 *      Parameters:
205 *          target      The target to check
206 *
207 *      Global variables used:
208 *          vpath_name  The Name "VPATH", used to get macro value
209
210 static timestruc_t&
211 vpath_exists(register Name target)
212 {
213     wchar_t      *vpath;
214     wchar_t      file_name[MAXPATHLEN];
215     wchar_t      *name_p;
216     Name         alias;
217
218     /*
219     *      To avoid recursive search through VPATH when exists(alias) is called
220     */
221     vpath_defined = false;
222
223     Wstring wcb(getvar(vpath_name));
224     Wstring wcb1(target);
225
226     vpath = wcb.get_string();
227
228     while (*vpath != (int) nul_char) {
229         name_p = file_name;
230         while ((*vpath != (int) colon_char) &&
231             (*vpath != (int) nul_char)) {
232             *name_p++ = *vpath++;
233         }
234         *name_p++ = (int) slash_char;
235         (void) wcsncpy(name_p, wcb1.get_string());
236         (void) wcsncpy(name_p, wcb1.get_string());
237         alias = GETNAME(file_name, FIND_LENGTH);
238         if (exists(alias) != file_doesnt_exist) {
239             target->stat.is_file = true;
240             target->stat.mode = alias->stat.mode;
241             target->stat.size = alias->stat.size;
242             target->stat.is_dir = alias->stat.is_dir;
243             target->stat.time = alias->stat.time;
244             maybe_append_prop(target, vpath_alias_prop)->
245                 body.vpath_alias.alias = alias;
246             target->has_vpath_alias_prop = true;
247             vpath_defined = true;
248             return alias->stat.time;
249         }
250     }
251     while ((*vpath != (int) nul_char) &&
252         ((*vpath == (int) colon_char) || iswspace(*vpath))) {
253         vpath++;

```

```

251     }
252     }
253     /*
254     *      Restore vpath_defined
255     */
256     vpath_defined = true;
257     return target->stat.time;
258 }

260 /*
261 *      read_dir(dir, pattern, line, library)
262 *
263 *      Used to enter the contents of directories into makes namespace.
264 *      Presence of a file is important when scanning for implicit rules.
265 *      read_dir() is also used to expand wildcards in dependency lists.
266 *
267 *      Return value:
268 *
269 *          Non-0 if we found files to match the pattern
270 *
271 *      Parameters:
272 *          dir      Path to the directory to read
273 *          pattern  Pattern for that files should match or NULL
274 *          line     When we scan using a pattern we enter files
275 *          library  we find as dependencies for this line
276 *                  If we scan for "lib.a(<wildcard-member>)"
277 *
278 *      Global variables used:
279 *          debug_level  Should we trace the dir reading?
280 *          dot          The Name ".", compared against
281 *          sccs_dir_path  The path to the SCCS dir (from PROJECTDIR)
282 *          vpath_defined  Was the variable VPATH defined in environment?
283 *          vpath_name    The Name "VPATH", use to get macro value
284
285 int
286 read_dir(Name dir, wchar_t *pattern, Property line, wchar_t *library)
287 {
288     wchar_t      file_name[MAXPATHLEN];
289     wchar_t      *file_name_p = file_name;
290     Name         file;
291     wchar_t      plain_file_name[MAXPATHLEN];
292     wchar_t      *plain_file_name_p;
293     Name         plain_file;
294     wchar_t      tmp_wcs_buffer[MAXPATHLEN];
295     DIR          *dir_fd;
296     int          m_local_dependency=0;
297 #define d_fileno d_ino
298     register struct dirent *dp;
299     wchar_t      *vpath = NULL;
300     wchar_t      *p;
301     int          result = 0;
302
303     if(dir->hash.length >= MAXPATHLEN) {
304         return 0;
305     }
306
307     Wstring wcb(dir);
308     Wstring vps;
309
310     /* A directory is only read once unless we need to expand wildcards. */
311     if (pattern == NULL) {
312         if (dir->has_read_dir) {
313             return 0;
314         }
315         dir->has_read_dir = true;
316     }
317     /* Check if VPATH is active and setup list if it is. */

```

```

317     if (vpath_defined && (dir == dot)) {
318         vps.init(getvar(vpath_name));
319         vpath = vps.get_string();
320     }
321
322     /*
323     * Prepare the string where we build the full name of the
324     * files in the directory.
325     */
326     if ((dir->hash.length > 1) || (wcb.get_string()[0] != (int) period_char)
327         (void) wscpy(file_name, wcb.get_string());
328         (void) wscpy(file_name, wcb.get_string());
329         MBSTOWCS(wcs_buffer, "/");
330         (void) wscat(file_name, wcs_buffer);
331         file_name_p = file_name + wcslen(file_name);
332         (void) wscat(file_name, wcs_buffer);
333         file_name_p = file_name + wslen(file_name);
334     }
335
336     /* Open the directory. */
337     vpath_loop:
338     dir_fd = opendir(dir->string_mb);
339     if (dir_fd == NULL) {
340         return 0;
341     }
342
343     /* Read all the directory entries. */
344     while ((dp = readdir(dir_fd)) != NULL) {
345         /* We ignore "." and ".." */
346         if ((dp->d_fileno == 0) ||
347             ((dp->d_name[0] == (int) period_char) &&
348              (dp->d_name[1] == 0) ||
349              ((dp->d_name[1] == (int) period_char) &&
350               (dp->d_name[2] == 0)))) {
351             continue;
352         }
353
354         /*
355         * Build the full name of the file using whatever
356         * path supplied to the function.
357         */
358         MBSTOWCS(tmp_wcs_buffer, dp->d_name);
359         (void) wscpy(file_name_p, tmp_wcs_buffer);
360         (void) wscpy(file_name_p, tmp_wcs_buffer);
361         file = enter_file_name(file_name, library);
362         if ((pattern != NULL) && amatch(tmp_wcs_buffer, pattern)) {
363             /*
364             * If we are expanding a wildcard pattern, we
365             * enter the file as a dependency for the target.
366             */
367             if (debug_level > 0) {
368                 WCSTOMBS(mbs_buffer, pattern);
369                 (void) printf(gettext("'%s: %s' due to %s expans
370 line->body.line.target->string_mb,
371 file->string_mb,
372 mbs_buffer);
373             }
374             enter_dependency(line, file, false);
375             result++;
376         } else {
377             /*
378             * If the file has an SCCS/s. file,
379             * we will detect that later on.
380             */
381             file->stat.has_sccs = NO_SCCS;
382         }
383     }
384
385     /*
386     * If this is an s. file, we also enter it as if it

```

```

379     * existed in the plain directory.
380     */
381     if ((dp->d_name[0] == 's') &&
382         (dp->d_name[1] == (int) period_char)) {
383
384         MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
385         plain_file_name_p = plain_file_name;
386         (void) wscpy(plain_file_name_p, tmp_wcs_buffer);
387         (void) wscpy(plain_file_name_p, tmp_wcs_buffer);
388         plain_file = GETNAME(plain_file_name, FIND_LENGTH);
389         plain_file->stat.is_file = true;
390         plain_file->stat.has_sccs = HAS_SCCS;
391         /*
392         * Enter the s. file as a dependency for the
393         * plain file.
394         */
395         maybe_append_prop(plain_file, sccs_prop)->
396             body.sccs.file = file;
397         MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
398         if ((pattern != NULL) &&
399             amatch(tmp_wcs_buffer, pattern)) {
400             if (debug_level > 0) {
401                 WCSTOMBS(mbs_buffer, pattern);
402                 (void) printf(gettext("'%s: %s' due to %
403 line->body.line.target->
404 string_mb,
405 plain_file->string_mb,
406 mbs_buffer);
407             }
408             enter_dependency(line, plain_file, false);
409             result++;
410         }
411     }
412 }
413
414 (void) closedir(dir_fd);
415 if ((vpath != NULL) && (*vpath != (int) nul_char)) {
416     while ((*vpath != (int) nul_char) &&
417           (iswspace(*vpath) || (*vpath == (int) colon_char))) {
418         vpath++;
419     }
420     p = vpath;
421     while ((*vpath != (int) colon_char) &&
422           (*vpath != (int) nul_char)) {
423         vpath++;
424     }
425     if (vpath > p) {
426         dir = GETNAME(p, vpath - p);
427         goto vpath_loop;
428     }
429 }
430 /*
431 * look into SCCS directory only if it's not svr4. For svr4 dont do that.
432 */
433
434 /*
435 * Now read the SCCS directory.
436 * Files in the SCSC directory are considered to be part of the set of
437 * files in the plain directory. They are also entered in their own right.
438 * Prepare the string where we build the true name of the SCCS files.
439 */
440 (void) wscncpy(plain_file_name,
441               (void) wscncpy(plain_file_name,
442                             file_name,
443                             file_name_p - file_name);
444 plain_file_name[file_name_p - file_name] = 0;

```

```

443 plain_file_name_p = plain_file_name + wcslen(plain_file_name);
443 plain_file_name_p = plain_file_name + wslen(plain_file_name);
445 if(!svr4) {
447     if (sccs_dir_path != NULL) {
448         wchar_t      tmp_wchar;
449         wchar_t      path[MAXPATHLEN];
450         char          mb_path[MAXPATHLEN];
452         if (file_name_p - file_name > 0) {
453             tmp_wchar = *file_name_p;
454             *file_name_p = 0;
455             WCSTOMBS(mbs_buffer, file_name);
456             (void) sprintf(mb_path, "%s/%s/SCCS",
457                 sccs_dir_path,
458                 mbs_buffer);
459             *file_name_p = tmp_wchar;
460         } else {
461             (void) sprintf(mb_path, "%s/SCCS", sccs_dir_path);
462         }
463         MBSTOWCS(path, mb_path);
464         (void) wcsncpy(file_name, path);
464         (void) wscpy(file_name, path);
465     } else {
466         MBSTOWCS(wcs_buffer, "SCCS");
467         (void) wcsncpy(file_name_p, wcs_buffer);
467         (void) wscpy(file_name_p, wcs_buffer);
468     }
469 } else {
470     MBSTOWCS(wcs_buffer, ".");
471     (void) wcsncpy(file_name_p, wcs_buffer);
471     (void) wscpy(file_name_p, wcs_buffer);
472 }
473 /* Internalize the constructed SCCS dir name. */
474 (void) exists(dir = GETNAME(file_name, FIND_LENGTH));
475 /* Just give up if the directory file doesnt exist. */
476 if (!dir->stat.is_file) {
477     return result;
478 }
479 /* Open the directory. */
480 dir_fd = opendir(dir->string_mb);
481 if (dir_fd == NULL) {
482     return result;
483 }
484 MBSTOWCS(wcs_buffer, "/");
485 (void) wscat(file_name, wcs_buffer);
486 file_name_p = file_name + wcslen(file_name);
485 (void) wscat(file_name, wcs_buffer);
486 file_name_p = file_name + wslen(file_name);
488 while ((dp = readdir(dir_fd)) != NULL) {
489     if ((dp->d_fileno == 0) ||
490         ((dp->d_name[0] == (int) period_char) &&
491          ((dp->d_name[1] == 0) ||
492           ((dp->d_name[1] == (int) period_char) &&
493            (dp->d_name[2] == 0)))) {
494         continue;
495     }
496     /* Construct and internalize the true name of the SCCS file. */
497     MBSTOWCS(wcs_buffer, dp->d_name);
498     (void) wcsncpy(file_name_p, wcs_buffer);
498     (void) wscpy(file_name_p, wcs_buffer);
499     file = GETNAME(file_name, FIND_LENGTH);
500     file->stat.is_file = true;
501     file->stat.has_sccs = NO_SCCS;

```

```

502     /*
503     * If this is an s. file, we also enter it as if it
504     * existed in the plain directory.
505     */
506     if ((dp->d_name[0] == 's') &&
507         (dp->d_name[1] == (int) period_char)) {
508
509         MBSTOWCS(wcs_buffer, dp->d_name + 2);
510         (void) wcsncpy(plain_file_name_p, wcs_buffer);
510         (void) wscpy(plain_file_name_p, wcs_buffer);
511         plain_file = GETNAME(plain_file_name, FIND_LENGTH);
512         plain_file->stat.is_file = true;
513         plain_file->stat.has_sccs = HAS_SCCS;
514         /* if sccs dependency is already set, skip */
515         if(plain_file->prop) {
516             Property sprop = get_prop(plain_file->prop, sccs_
517                 if(sprop != NULL) {
518                     if (sprop->body.sccs.file) {
519                         goto try_pattern;
520                     }
521                 }
522         }
524     /*
525     * Enter the s. file as a dependency for the
526     * plain file.
527     */
528     maybe_append_prop(plain_file, sccs_prop)->
529         body.sccs.file = file;
530 try_pattern:
531     MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
532     if ((pattern != NULL) &&
533         amatch(tmp_wcs_buffer, pattern)) {
534         if (debug_level > 0) {
535             WCSTOMBS(mbs_buffer, pattern);
536             (void) printf(gettext("'%s: %s' due to %
537                 line->body.line.target->
538                 string_mb,
539                 plain_file->string_mb,
540                 mbs_buffer);
541         }
542         enter_dependency(line, plain_file, false);
543         result++;
544     }
545 }
546 }
547 (void) closedir(dir_fd);
549     return result;
550 }
_____unchanged_portion_omitted_____

```

```

*****
43023 Fri May 22 11:19:42 2015
new/usr/src/cmd/make/bin/implicit.cc
make: use the more modern wchar routines, not widec.h
*****
_____unchanged_portion_omitted_____

771 /*
772 *      find_percent_rule(target, command, rechecking)
773 *
774 *      Tries to find a rule from the list of wildcard matched rules.
775 *      It scans the list attempting to match the target.
776 *      For each target match it checks if the corresponding source exists.
777 *      If it does the match is returned.
778 *      The percent_list is built at makefile read time.
779 *      Each percent rule get one entry on the list.
780 *
781 *      Return value:
782 *
783 *          Indicates if the scan failed or not
784 *
785 *      Parameters:
786 *          target      The target we need a rule for
787 *          command     Pointer to slot where we stuff cmd, if found
788 *          rechecking  true if we are rechecking target which depends
789 *                    on conditional macro and keep_state is set
790 *
791 *      Global variables used:
792 *          debug_level  Indicates how much tracing to do
793 *          percent_list List of all percent rules
794 *          recursion_level Used for tracing
795 *          empty_name
796 Doname
797 find_percent_rule(register Name target, Property *command, Boolean rechecking)
798 {
799     register Percent      pat_rule, pat_depe;
800     register Name        depe_to_check;
801     register Dependency  depe;
802     register Property   line;
803     String_rec          string;
804     wchar_t             string_buf[STRING_BUFFER_LENGTH];
805     String_rec          percent;
806     wchar_t             percent_buf[STRING_BUFFER_LENGTH];
807     Name                true_target = target;
808     Name                less;
809     Boolean             nonpattern_less;
810     Boolean             dep_name_found = false;
811     Doname             result = build_dont_know;
812     Percent            rule_candidate = NULL;
813     Boolean            rule_maybe_ok;
814     Boolean            is_pattern;

816     /* If the target is constructed for a ":" target we consider that */
817     if (target->has_target_prop) {
818         true_target = get_prop(target->prop,
819                               target_prop->body.target.target;
820     }
821     if (target->has_long_member_name) {
822         true_target = get_prop(target->prop,
823                               long_member_name_prop->body.long_member_
824     }
825     if (debug_level > 1) {
826         (void) printf(gettext("%sLooking for %% rule for %s\n"),
827                     recursion_level,
828                     "",
829                     true_target->string_mb);

```

```

830     }
831     for (pat_rule = percent_list;
832          pat_rule != NULL;
833          pat_rule = pat_rule->next) {
834         /* Avoid infinite recursion when expanding patterns */
835         if (pat_rule->being_expanded == true) {
836             continue;
837         }

839         /* Mark this pat_rule as "maybe ok". If no % rule is found
840         make will use this rule. The following algorithm is used:
841         1) make scans all pattern rules in order to find the rule
842            where ALL dependencies, including nonpattern ones, exist o
843            can be built (GNU behaviour). If such rule is found make
844            will apply it.
845         2) During this check make also remembers the first pattern ru
846            where all PATTERN dependencies can be build (no matter wha
847            happens with nonpattern dependencies).
848         3) If no rule satisfying 1) is found, make will apply the rul
849            remembered in 2) if there is one.
850         */
851         rule_maybe_ok = true;

853         /* used to track first percent dependency */
854         less = NULL;
855         nonpattern_less = true;

857         /* check whether pattern matches.
858         if it matches, percent string will contain matched percent pa
859         if (!match_found_with_pattern(true_target, pat_rule, &percent, p
860             continue;
861         }
862         if (pat_rule->dependencies != NULL) {
863             for (pat_depe = pat_rule->dependencies;
864                  pat_depe != NULL;
865                  pat_depe = pat_depe->next) {
866                 /* checking result for dependency */
867                 result = build_dont_know;

869                 dep_name_found = true;
870                 if (pat_depe->name->percent) {
871                     is_pattern = true;
872                     /* build dependency name */
873                     INIT_STRING_FROM_STACK(string, string_bu
874                     construct_string_from_pattern(pat_depe,
875                     depe_to_check = getname_fn(string.buffer
876                     FIND_LENGTH,
877                     false,
878                     &dep_name_found
879                 );

881                 if ((less == NULL) || nonpattern_less) {
882                     less = depe_to_check;
883                     nonpattern_less = false;
884                 }
885             } else {
886                 /* nonpattern dependency */
887                 is_pattern = false;
888                 depe_to_check = pat_depe->name;
889                 if (depe_to_check->dollar) {
890                     INIT_STRING_FROM_STACK(string, s
891                     expand_value(depe_to_check, &str
892                     depe_to_check = getname_fn(strin
893                     FIND_LENGTH,
894                     false,
895                     &dep_name_found

```

```

896         );
897     }
898     if (less == NULL) {
899         less = depe_to_check;
900     }
901 }
902
903 if (depe_to_check == empty_name) {
904     result = build_ok;
905 } else {
906     if (debug_level > 1) {
907         (void) printf(gettext("%sTrying
908             recursion_level,
909             ",
910             depe_to_check->str
911         );
912     }
913     pat_rule->being_expanded = true;
914
915     /* suppress message output */
916     int save_debug_level = debug_level;
917     debug_level = 0;
918
919     /* check whether dependency can be built
920     if (dependency_exists(depe_to_check,
921         get_prop(target->prop,
922             line_prop)))
923     {
924         result = (Doname) depe_to_check-
925     } else {
926         if(actual_doname) {
927             result = doname(depe_to_
928         } else {
929             result = target_can_be_b
930         }
931         if(!dep_name_found) {
932             if(result != build_ok &&
933                 free_name(depe_t
934             } else {
935                 store_name(depe_
936             }
937         }
938     }
939     if(result != build_ok && is_pattern) {
940         rule_maybe_ok = false;
941     }
942
943     /* restore debug_level */
944     debug_level = save_debug_level;
945 }
946
947 if (pat_depe->name->percent) {
948     if (string.free_after_use) {
949         retmem(string.buffer.start);
950     }
951 }
952 /* make can't figure out how to make this depend
953 if (result != build_ok && result != build_runnin
954     pat_rule->being_expanded = false;
955     break;
956 }
957 }
958 } else {
959     result = build_ok;
960 }

```

```

962     /* this pattern rule is the needed one since all dependencies co
963     if (result == build_ok || result == build_running) {
964         break;
965     }
966
967     /* Make does not know how to build some of dependencies from thi
968     But if all "pattern" dependencies can be built, we remember t
969     as a candidate for the case if no other pattern rule found.
970     */
971     if(rule_maybe_ok && rule_candidate == NULL) {
972         rule_candidate = pat_rule;
973     }
974 }
975
976 /* if no pattern matching rule was found, use the remembered candidate
977 or return build_dont_know if there is no candidate.
978 */
979 if (result != build_ok && result != build_running) {
980     if(rule_candidate) {
981         pat_rule = rule_candidate;
982     } else {
983         return build_dont_know;
984     }
985 }
986
987 /* if we are performing only check whether dependency could be built wit
988 return success */
989 if (command == NULL) {
990     if(pat_rule != NULL) {
991         pat_rule->being_expanded = false;
992     }
993     return result;
994 }
995
996 if (debug_level > 1) {
997     (void) printf(gettext("%sMatched %s:"),
998         recursion_level,
999         ",
1000         target->string_mb);
1001
1002     for (pat_depe = pat_rule->dependencies;
1003         pat_depe != NULL;
1004         pat_depe = pat_depe->next) {
1005         if (pat_depe->name->percent) {
1006             INIT_STRING_FROM_STACK(string, string_buf);
1007             construct_string_from_pattern(pat_depe, &percent
1008             depe_to_check = GETNAME(string.buffer.start, FIN
1009         } else {
1010             depe_to_check = pat_depe->name;
1011             if(depe_to_check->dollar) {
1012                 INIT_STRING_FROM_STACK(string, string_bu
1013                 expand_value(depe_to_check, &string, fal
1014                 depe_to_check = GETNAME(string.buffer.st
1015             }
1016         }
1017     }
1018     if (depe_to_check != empty_name) {
1019         (void) printf(" %s", depe_to_check->string_mb);
1020     }
1021 }
1022
1023 (void) printf(gettext(" from: %s:"),
1024     pat_rule->name->string_mb);
1025
1026 for (pat_depe = pat_rule->dependencies;
1027     pat_depe != NULL;

```

```

1028         pat_depe = pat_depe->next) {
1029             (void) printf(" %s", pat_depe->name->string_mb);
1030         }
1032         (void) printf("\n");
1033     }
1035     if (true_target->colons == no_colon) {
1036         true_target->colons = one_colon;
1037     }
1039     /* create dependency list and target group from matched pattern rule */
1040     create_target_group_and_dependencies_list(target, pat_rule, &percent);
1042     /* save command */
1043     line = get_prop(target->prop, line_prop);
1044     *command = line;
1046     /* free query chain if one exist */
1047     while(line->body.line.query != NULL) {
1048         Chain to_free = line->body.line.query;
1049         line->body.line.query = line->body.line.query->next;
1050         retmem_mb((char *) to_free);
1051     }
1053     if (line->body.line.dependencies != NULL) {
1054         /* build all collected dependencies */
1055         for (depe = line->body.line.dependencies;
1056             depe != NULL;
1057             depe = depe->next) {
1058             actual_doname = true;
1059             result = doname_check(depe->name, true, true, depe->auto
1061             actual_doname = false;
1062             if (result == build_failed) {
1063                 pat_rule->being_expanded = false;
1064                 return build_failed;
1065             }
1066             if (result == build_running) {
1067                 pat_rule->being_expanded = false;
1068                 return build_running;
1069             }
1071             if ((depe->name->stat.time > line->body.line.dependency_
1072                 (debug_level > 1)) {
1073                 (void) printf(gettext("%sDate(%s)=%s Date-depen
1074                     recursion_level,
1075                     "",
1076                     depe->name->string_mb,
1077                     time_to_string(depe->name->stat.ti
1078                     true_target->string_mb,
1079                     time_to_string(line->body.line.dep
1080                 )
1082         line->body.line.dependency_time =
1083             MAX(line->body.line.dependency_time, depe->name->stat.
1085         /* determine whether this dependency made target out of
1086         Boolean out_of_date;
1087         if (target->is_member || depe->name->is_member) {
1088             out_of_date = (Boolean) OUT_OF_DATE_SEC(target->
1089         } else {
1090             out_of_date = (Boolean) OUT_OF_DATE(target->stat
1091         }
1092         if (build_unconditional || out_of_date) {
1093             if(!rechecking) {

```

```

1094         line->body.line.is_out_of_date = true;
1095     }
1096     add_target_to_chain(depe->name, &(line->body.lin
1098     if (debug_level > 0) {
1099         (void) printf(gettext("%sBuilding %s us
1100             recursion_level,
1101             "",
1102             true_target->string_mb,
1103             pat_rule->name->string_mb)
1105         for (pat_depe = pat_rule->dependencies;
1106             pat_depe != NULL;
1107             pat_depe = pat_depe->next) {
1108             (void) printf(" %s", pat_depe->n
1109         }
1111         (void) printf(gettext(" because it is ou
1112             depe->name->string_mb);
1113     }
1114 } else {
1115     if ((true_target->stat.time <= file_doesnt_exist) ||
1116         (true_target->stat.time < line->body.line.dependency_time))
1117         if(!rechecking) {
1118             line->body.line.is_out_of_date = true;
1119         }
1120         if (debug_level > 0) {
1121             (void) printf(gettext("%sBuilding %s using patt
1122                 recursion_level,
1123                 "",
1124                 true_target->string_mb,
1125                 pat_rule->name->string_mb,
1126                 (target->stat.time > file_doesnt_e
1127                 gettext("because it is out of date
1128                 gettext("because it does not exist
1129         }
1130     }
1131 }
1132 }
1133 }
1135     /* enter explicit rule from percent rule */
1136     Name lmn_target = true_target;
1137     if (true_target->has_long_member_name) {
1138         lmn_target = get_prop(true_target->prop, long_member_name_prop)-
1139     }
1140     line->body.line.sccs_command = false;
1141     line->body.line.target = true_target;
1142     line->body.line.command_template = pat_rule->command_template;
1143     line->body.line.star = GETNAME(percent.buffer.start, FIND_LENGTH);
1144     line->body.line.less = less;
1146     if (lmn_target->parenleft) {
1147         Wstring lmn_string(lmn_target);
1149         wchar_t *left = (wchar_t *) wcschr(lmn_string.get_string(), (int
1150         wchar_t *right = (wchar_t *) wcschr(lmn_string.get_string(), (in
1151         wchar_t *left = (wchar_t *) wcschr(lmn_string.get_string(), (int
1152         wchar_t *right = (wchar_t *) wcschr(lmn_string.get_string(), (int
1153         if ((left == NULL) || (right == NULL)) {
1154             line->body.line.percent = NULL;
1155         } else {
1156             line->body.line.percent = GETNAME(left + 1, right - left
1157         } else {

```



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

7

```
1158         line->body.line.percent = NULL;
1159     }
1160     pat_rule->being_expanded = false;

1162     return result;
1163 }
unchanged_portion_omitted
```

```

*****
4001 Fri May 22 11:19:42 2015
new/usr/src/cmd/make/bin/macro.cc
make: use the more modern wchar routines, not widec.h
*****
_____unchanged_portion_omitted_____

100 /*
101 *      setvar_envvar()
102 *
103 *      This function scans the list of environment variables that have
104 *      dynamic values and sets them.
105 *
106 *      Parameters:
107 *
108 *      Global variables used:
109 *          envvar      A list of environment vars with $ in value
110 */
111 void
112 setvar_envvar(void)
113 {
114     wchar_t      buffer[STRING_BUFFER_LENGTH];
115     int          length;
116     register     char  *mbs, *tmp_mbs_buffer = NULL;
117     register     char  *env, *tmp_mbs_buffer2 = NULL;
118     Envvar      p;
119     String_rec   value;

121     for (p = envvar; p != NULL; p = p->next) {
122         if (p->already_put
123             ) {
124             continue;
125         }
126         INIT_STRING_FROM_STACK(value, buffer);
127         expand_value(p->value, &value, false);
128         if ((length = wcslen(value.buffer.start)) >= MAXPATHLEN) {
129             if ((length = wslen(value.buffer.start)) >= MAXPATHLEN) {
130                 mbs = tmp_mbs_buffer = getmem((length + 1) * MB_LEN_MAX)
131                 (void) wcstombs(mbs,
132                                 value.buffer.start,
133                                 (length + 1) * MB_LEN_MAX);
134             } else {
135                 mbs = mbs_buffer;
136                 WCSTOMBS(mbs, value.buffer.start);
137             }
138             length = 2 + strlen(p->name->string_mb) + strlen(mbs);
139             if (!p->already_put || length > (MAXPATHLEN * MB_LEN_MAX)) {
140                 env = tmp_mbs_buffer2 = getmem(length);
141             } else {
142                 env = mbs_buffer2;
143             }
144             (void) sprintf(env,
145                            "%s=%s",
146                            p->name->string_mb,
147                            mbs);
148             if (!p->already_put) {
149                 (void) putenv(env);
150                 p->already_put = true;
151                 if (p->env_string) {
152                     retmem_mb(p->env_string);
153                 }
154                 p->env_string = env;
155                 tmp_mbs_buffer2 = NULL; // We should not return this mem
156             }
157             if (tmp_mbs_buffer2) {
158                 retmem_mb(tmp_mbs_buffer2);

```

```

158             tmp_mbs_buffer2 = NULL;
159         }
160         if (tmp_mbs_buffer) {
161             retmem_mb(tmp_mbs_buffer);
162             tmp_mbs_buffer = NULL;
163         }
164     }
165 }
_____unchanged_portion_omitted_____

```

```
*****
```

```
86753 Fri May 22 11:19:43 2015
```

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

```
make: use the more modern wchar routines, not widec.h
```

```
*****
```

```
_____unchanged_portion_omitted_____
```

```
1709 /*
1710 *      read_files_and_state(argc, argv)
1711 *
1712 *      Read the makefiles we care about and the environment
1713 *      Also read the = style command line options
1714 *
1715 *      Parameters:
1716 *          argc          You know what this is
1717 *          argv          You know what this is
1718 *
1719 *      Static variables used:
1720 *          env_wins      make -e, determines if env vars are RO
1721 *          ignore_default_mk make -r, determines if make.rules is read
1722 *          not_auto_depen dwight
1723 *
1724 *      Global variables used:
1725 *          default_target_to_build Set to first proper target from file
1726 *          do_not_exec_rule Set to false when makfile is made
1727 *          dot            The Name ".", used to read current dir
1728 *          empty_name    The Name "", use as macro value
1729 *          keep_state    Set if KEEP_STATE is in environment
1730 *          make_state    The Name ".make.state", used to read file
1731 *          makefile_type Set to type of file being read
1732 *          makeflags     The Name "MAKEFLAGS", used to set macro value
1733 *          not_auto      dwight
1734 *          read_trace_level Checked to see if the reader should trace
1735 *          report_dependencies If -P is on we do not read .make.state
1736 *          trace_reader  Set if reader should trace
1737 *          virtual_root  The Name "VIRTUAL_ROOT", used to check value
1738 */
1739 static void
1740 read_files_and_state(int argc, char **argv)
1741 {
1742     wchar_t      buffer[1000];
1743     wchar_t      buffer_posix[1000];
1744     register char ch;
1745     register char *cp;
1746     Property     def_make_macro = NULL;
1747     Name         def_make_name;
1748     Name         default_makefile;
1749     String_rec   dest;
1750     wchar_t      destbuffer[STRING_BUFFER_LENGTH];
1751     register int i;
1752     register int j;
1753     Name         keep_state_name;
1754     int          length;
1755     Name         Makefile;
1756     register Property macro;
1757     struct stat  make_state_stat;
1758     Name         makefile_name;
1759     register int makefile_next = 0;
1760     register Boolean makefile_read = false;
1761     String_rec   makeflags_string;
1762     String_rec   makeflags_string_posix;
1763     String_rec * makeflags_string_current;
1764     Name         makeflags_value_saved;
1765     register Name name;
1766     Name         new_make_value;
1767     Boolean      save_do_not_exec_rule;
```

```
1768     Name         sdotMakefile;
1769     Name         sdotmakefile_name;
1770     static wchar_t state_file_str;
1771     static char   state_file_str_mb[MAXPATHLEN];
1772     static struct _Name state_filename;
1773     Boolean      temp;
1774     char         tmp_char;
1775     wchar_t      *tmp_wcs_buffer;
1776     register Name value;
1777     ASCII_Dyn_Array makeflags_and_macro;
1778     Boolean      is_xpg4;

1780 /*
1781 *      Remember current mode. It may be changed after reading makefile
1782 *      and we will have to correct MAKEFLAGS variable.
1783 */
1784     is_xpg4 = posix;

1786     MBSTOWCS(wcs_buffer, "KEEP_STATE");
1787     keep_state_name = GETNAME(wcs_buffer, FIND_LENGTH);
1788     MBSTOWCS(wcs_buffer, "Makefile");
1789     Makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1790     MBSTOWCS(wcs_buffer, "makefile");
1791     makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
1792     MBSTOWCS(wcs_buffer, "s.makefile");
1793     sdotmakefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
1794     MBSTOWCS(wcs_buffer, "s.Makefile");
1795     sdotMakefile = GETNAME(wcs_buffer, FIND_LENGTH);

1797 /*
1798 *      initialize global dependency entry for .NOT_AUTO
1799 */
1800     not_auto_depen->next = NULL;
1801     not_auto_depen->name = not_auto;
1802     not_auto_depen->automatic = not_auto_depen->stale = false;

1804 /*
1805 *      Read internal definitions and rules.
1806 */
1807     if (read_trace_level > 1) {
1808         trace_reader = true;
1809     }
1810     if (!ignore_default_mk) {
1811         if (svr4) {
1812             MBSTOWCS(wcs_buffer, "svr4.make.rules");
1813             default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1814         } else {
1815             MBSTOWCS(wcs_buffer, "make.rules");
1816             default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1817         }
1818         default_makefile->stat.is_file = true;
1819     }

1820     (void) read_makefile(default_makefile,
1821                          true,
1822                          false,
1823                          true);
1824 }

1826 /*
1827 *      If the user did not redefine the MAKE macro in the
1828 *      default makefile (make.rules), then we'd like to
1829 *      change the macro value of MAKE to be some form
1830 *      of argv[0] for recursive MAKE builds.
1831 */
1832     MBSTOWCS(wcs_buffer, "MAKE");
1833     def_make_name = GETNAME(wcs_buffer, wcslen(wcs_buffer));
```

```

1833     def_make_name = GETNAME(wcs_buffer, wslen(wcs_buffer));
1834     def_make_macro = get_prop(def_make_name->prop, macro_prop);
1835     if ((def_make_macro != NULL) &&
1836         (IS_EQUAL(def_make_macro->body.macro.value->string_mb,
1837                  "make"))) {
1838         MBSTOWCS(wcs_buffer, argv_zero_string);
1839         new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
1840         new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
1841         (void) SETVAR(def_make_name,
1842                      new_make_value,
1843                      false);
1844     }
1845     default_target_to_build = NULL;
1846     trace_reader = false;
1847
1848 /*
1849 *   Read environment args. Let file args which follow override unless
1850 *   -e option seen. If -e option is not mentioned.
1851 */
1852 read_environment(env_wins);
1853 if (getvar(virtual_root)->hash.length == 0) {
1854     maybe_append_prop(virtual_root, macro_prop)
1855     ->body.macro.exported = true;
1856     MBSTOWCS(wcs_buffer, "/");
1857     (void) SETVAR(virtual_root,
1858                  GETNAME(wcs_buffer, FIND_LENGTH),
1859                  false);
1860 }
1861
1862 /*
1863 * We now scan mf_argv and argv to see if we need to set
1864 * any of the DMake-added options/variables in MAKEFLAGS.
1865 */
1866
1867     makeflags_and_macro.start = 0;
1868     makeflags_and_macro.size = 0;
1869     enter_argv_values(mf_argc, mf_argv, &makeflags_and_macro);
1870     enter_argv_values(argc, argv, &makeflags_and_macro);
1871
1872 /*
1873 *   Set MFLAGS and MAKEFLAGS
1874 *
1875 *   Before reading makefile we do not know exactly which mode
1876 *   (posix or not) is used. So prepare two MAKEFLAGS strings
1877 *   for both posix and solaris modes because they are different.
1878 */
1879     INIT_STRING_FROM_STACK(makeflags_string, buffer);
1880     INIT_STRING_FROM_STACK(makeflags_string_posix, buffer_posix);
1881     append_char((int) hyphen_char, &makeflags_string);
1882     append_char((int) hyphen_char, &makeflags_string_posix);
1883
1884     switch (read_trace_level) {
1885     case 2:
1886         append_char('D', &makeflags_string);
1887         append_char('D', &makeflags_string_posix);
1888     case 1:
1889         append_char('D', &makeflags_string);
1890         append_char('D', &makeflags_string_posix);
1891     }
1892     switch (debug_level) {
1893     case 2:
1894         append_char('d', &makeflags_string);
1895         append_char('d', &makeflags_string_posix);
1896     case 1:
1897         append_char('d', &makeflags_string);

```

```

1898         append_char('d', &makeflags_string_posix);
1899     }
1900     if (env_wins) {
1901         append_char('e', &makeflags_string);
1902         append_char('e', &makeflags_string_posix);
1903     }
1904     if (ignore_errors_all) {
1905         append_char('i', &makeflags_string);
1906         append_char('i', &makeflags_string_posix);
1907     }
1908     if (continue_after_error) {
1909         if (stop_after_error_ever_seen) {
1910             append_char('S', &makeflags_string_posix);
1911             append_char((int) space_char, &makeflags_string_posix);
1912             append_char((int) hyphen_char, &makeflags_string_posix);
1913         }
1914         append_char('k', &makeflags_string);
1915         append_char('k', &makeflags_string_posix);
1916     } else {
1917         if (stop_after_error_ever_seen
1918             && continue_after_error_ever_seen) {
1919             append_char('k', &makeflags_string_posix);
1920             append_char((int) space_char, &makeflags_string_posix);
1921             append_char((int) hyphen_char, &makeflags_string_posix);
1922             append_char('S', &makeflags_string_posix);
1923         }
1924     }
1925     if (do_not_exec_rule) {
1926         append_char('n', &makeflags_string);
1927         append_char('n', &makeflags_string_posix);
1928     }
1929     switch (report_dependencies_level) {
1930     case 4:
1931         append_char('P', &makeflags_string);
1932         append_char('P', &makeflags_string_posix);
1933     case 3:
1934         append_char('P', &makeflags_string);
1935         append_char('P', &makeflags_string_posix);
1936     case 2:
1937         append_char('P', &makeflags_string);
1938         append_char('P', &makeflags_string_posix);
1939     case 1:
1940         append_char('P', &makeflags_string);
1941         append_char('P', &makeflags_string_posix);
1942     }
1943     if (trace_status) {
1944         append_char('p', &makeflags_string);
1945         append_char('p', &makeflags_string_posix);
1946     }
1947     if (quest) {
1948         append_char('q', &makeflags_string);
1949         append_char('q', &makeflags_string_posix);
1950     }
1951     if (silent_all) {
1952         append_char('s', &makeflags_string);
1953         append_char('s', &makeflags_string_posix);
1954     }
1955     if (touch) {
1956         append_char('t', &makeflags_string);
1957         append_char('t', &makeflags_string_posix);
1958     }
1959     if (build_unconditional) {
1960         append_char('u', &makeflags_string);
1961         append_char('u', &makeflags_string_posix);
1962     }
1963     if (report_cwd) {

```

```

1964         append_char('w', &makeflags_string);
1965         append_char('w', &makeflags_string_posix);
1966     }
1967     /* -c dmake_rcfile */
1968     if (dmake_rcfile_specified) {
1969         MBSTOWCS(wcs_buffer, "DMAKE_RCFILE");
1970         dmake_rcfile = GETNAME(wcs_buffer, FIND_LENGTH);
1971         append_makeflags_string(dmake_rcfile, &makeflags_string);
1972         append_makeflags_string(dmake_rcfile, &makeflags_string_posix);
1973     }
1974     /* -g dmake_group */
1975     if (dmake_group_specified) {
1976         MBSTOWCS(wcs_buffer, "DMAKE_GROUP");
1977         dmake_group = GETNAME(wcs_buffer, FIND_LENGTH);
1978         append_makeflags_string(dmake_group, &makeflags_string);
1979         append_makeflags_string(dmake_group, &makeflags_string_posix);
1980     }
1981     /* -j dmake_max_jobs */
1982     if (dmake_max_jobs_specified) {
1983         MBSTOWCS(wcs_buffer, "DMAKE_MAX_JOBS");
1984         dmake_max_jobs = GETNAME(wcs_buffer, FIND_LENGTH);
1985         append_makeflags_string(dmake_max_jobs, &makeflags_string);
1986         append_makeflags_string(dmake_max_jobs, &makeflags_string_posix);
1987     }
1988     /* -m dmake_mode */
1989     if (dmake_mode_specified) {
1990         MBSTOWCS(wcs_buffer, "DMAKE_MODE");
1991         dmake_mode = GETNAME(wcs_buffer, FIND_LENGTH);
1992         append_makeflags_string(dmake_mode, &makeflags_string);
1993         append_makeflags_string(dmake_mode, &makeflags_string_posix);
1994     }
1995     /* -x dmake_compat_mode */
1996     if (dmake_compat_mode_specified) {
1997         MBSTOWCS(wcs_buffer, "SUN_MAKE_COMPAT_MODE");
1998         dmake_compat_mode = GETNAME(wcs_buffer, FIND_LENGTH);
1999         append_makeflags_string(dmake_compat_mode, &makeflags_string);
2000         append_makeflags_string(dmake_compat_mode, &makeflags_string_pos);
2001     }
2002     /* -x dmake_output_mode */
2003     if (dmake_output_mode_specified) {
2004         MBSTOWCS(wcs_buffer, "DMAKE_OUTPUT_MODE");
2005         dmake_output_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2006         append_makeflags_string(dmake_output_mode, &makeflags_string);
2007         append_makeflags_string(dmake_output_mode, &makeflags_string_pos);
2008     }
2009     /* -o dmake_odir */
2010     if (dmake_odir_specified) {
2011         MBSTOWCS(wcs_buffer, "DMAKE_ODIR");
2012         dmake_odir = GETNAME(wcs_buffer, FIND_LENGTH);
2013         append_makeflags_string(dmake_odir, &makeflags_string);
2014         append_makeflags_string(dmake_odir, &makeflags_string_posix);
2015     }
2016     /* -M pmake_machinesfile */
2017     if (pmake_machinesfile_specified) {
2018         MBSTOWCS(wcs_buffer, "PMAKE_MACHINESFILE");
2019         pmake_machinesfile = GETNAME(wcs_buffer, FIND_LENGTH);
2020         append_makeflags_string(pmake_machinesfile, &makeflags_string);
2021         append_makeflags_string(pmake_machinesfile, &makeflags_string_pos);
2022     }
2023     /* -R */
2024     if (pmake_cap_r_specified) {
2025         append_char((int) space_char, &makeflags_string);
2026         append_char((int) hyphen_char, &makeflags_string);
2027         append_char('R', &makeflags_string);
2028         append_char((int) space_char, &makeflags_string_posix);
2029         append_char((int) hyphen_char, &makeflags_string_posix);

```

```

2030         append_char('R', &makeflags_string_posix);
2031     }
2032
2033     /*
2034     * Make sure MAKEFLAGS is exported
2035     */
2036     maybe_append_prop(makeflags, macro_prop)->
2037         body.macro.exported = true;
2038
2039     if (makeflags_string.buffer.start[1] != (int) nul_char) {
2040         if (makeflags_string.buffer.start[1] != (int) space_char) {
2041             MBSTOWCS(wcs_buffer, "MFLAGS");
2042             (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2043                 GETNAME(makeflags_string.buffer.start,
2044                     FIND_LENGTH),
2045                 false);
2046         } else {
2047             MBSTOWCS(wcs_buffer, "MFLAGS");
2048             (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2049                 GETNAME(makeflags_string.buffer.start + 2,
2050                     FIND_LENGTH),
2051                 false);
2052         }
2053     }
2054
2055     /*
2056     * Add command line macro to POSIX makeflags_string
2057     */
2058     if (makeflags_and_macro.start) {
2059         tmp_char = (char) space_char;
2060         cp = makeflags_and_macro.start;
2061         do {
2062             append_char(tmp_char, &makeflags_string_posix);
2063         } while (tmp_char = *cp++);
2064         retmem_mb(makeflags_and_macro.start);
2065     }
2066
2067     /*
2068     * Now set the value of MAKEFLAGS macro in accordance
2069     * with current mode.
2070     */
2071     macro = maybe_append_prop(makeflags, macro_prop);
2072     temp = (Boolean) macro->body.macro.read_only;
2073     macro->body.macro.read_only = false;
2074     if (posix || gnu_style) {
2075         makeflags_string_current = &makeflags_string_posix;
2076     } else {
2077         makeflags_string_current = &makeflags_string;
2078     }
2079     if (makeflags_string_current->buffer.start[1] == (int) nul_char) {
2080         makeflags_value_saved =
2081             GETNAME( makeflags_string_current->buffer.start + 1
2082                 , FIND_LENGTH
2083                 );
2084     } else {
2085         if (makeflags_string_current->buffer.start[1] != (int) space_cha
2086             makeflags_value_saved =
2087                 GETNAME( makeflags_string_current->buffer.start
2088                     , FIND_LENGTH
2089                     );
2090     } else {
2091         makeflags_value_saved =
2092             GETNAME( makeflags_string_current->buffer.start
2093                 , FIND_LENGTH
2094                 );
2095     }

```

```

2096     }
2097     (void) SETVAR( makeflags
2098                 , makeflags_value_saved
2099                 , false
2100                 );
2101     macro->body.macro.read_only = temp;

2103 /*
2104 *   Read command line "-f" arguments and ignore -c, g, j, K, M, m, O and o a
2105 */
2106     save_do_not_exec_rule = do_not_exec_rule;
2107     do_not_exec_rule = false;
2108     if (read_trace_level > 0) {
2109         trace_reader = true;
2110     }

2112     for (i = 1; i < argc; i++) {
2113         if (argv[i] &&
2114             (argv[i][0] == (int) hyphen_char) &&
2115             (argv[i][1] == 'f') &&
2116             (argv[i][2] == (int) nul_char)) {
2117             argv[i] = NULL; /* Remove -f */
2118             if (i >= argc - 1) {
2119                 fatal(gettext("No filename argument after -f fla
2120             });
2121             MBSTOWCS(wcs_buffer, argv[+i]);
2122             primary_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2123             (void) read_makefile(primary_makefile, true, true, true)
2124             argv[i] = NULL; /* Remove filename */
2125             makefile_read = true;
2126         } else if (argv[i] &&
2127                 (argv[i][0] == (int) hyphen_char) &&
2128                 (argv[i][1] == 'c' ||
2129                 argv[i][1] == 'g' ||
2130                 argv[i][1] == 'j' ||
2131                 argv[i][1] == 'K' ||
2132                 argv[i][1] == 'M' ||
2133                 argv[i][1] == 'm' ||
2134                 argv[i][1] == 'O' ||
2135                 argv[i][1] == 'o') &&
2136                 (argv[i][2] == (int) nul_char)) {
2137             argv[i] = NULL;
2138             argv[+i] = NULL;
2139         }
2140     }

2142 /*
2143 *   If no command line "-f" args then look for "makefile", and then for
2144 *   "Makefile" if "makefile" isn't found.
2145 */
2146     if (!makefile_read) {
2147         (void) read_dir(dot,
2148                       (wchar_t *) NULL,
2149                       (Property) NULL,
2150                       (wchar_t *) NULL);
2151         if (!posix) {
2152             if (makefile_name->stat.is_file) {
2153                 if (Makefile->stat.is_file) {
2154                     warning(gettext("Both 'makefile' and 'Makefile'
2155                 });
2156                 primary_makefile = makefile_name;
2157                 makefile_read = read_makefile(makefile_name,
2158                                             false,
2159                                             false,
2160                                             true);
2161             }

```

```

2162         if (!makefile_read &&
2163             Makefile->stat.is_file) {
2164             primary_makefile = Makefile;
2165             makefile_read = read_makefile(Makefile,
2166                                         false,
2167                                         false,
2168                                         true);
2169         }
2170     } else {

2172         enum sccs_stat save_m_has_sccs = NO_SCCS;
2173         enum sccs_stat save_M_has_sccs = NO_SCCS;

2175         if (makefile_name->stat.is_file) {
2176             if (Makefile->stat.is_file) {
2177                 warning(gettext("Both 'makefile' and 'Makefile'
2178             });
2179         }
2180         if (makefile_name->stat.is_file) {
2181             if (makefile_name->stat.has_sccs == NO_SCCS) {
2182                 primary_makefile = makefile_name;
2183                 makefile_read = read_makefile(makefile_name,
2184                                             false,
2185                                             false,
2186                                             true);
2187             } else {
2188                 save_m_has_sccs = makefile_name->stat.has_sccs;
2189                 makefile_name->stat.has_sccs = NO_SCCS;
2190                 primary_makefile = makefile_name;
2191                 makefile_read = read_makefile(makefile_name,
2192                                             false,
2193                                             false,
2194                                             true);
2195             }
2196         }
2197         if (!makefile_read &&
2198             Makefile->stat.is_file) {
2199             if (Makefile->stat.has_sccs == NO_SCCS) {
2200                 primary_makefile = Makefile;
2201                 makefile_read = read_makefile(Makefile,
2202                                             false,
2203                                             false,
2204                                             true);
2205             } else {
2206                 save_M_has_sccs = Makefile->stat.has_sccs;
2207                 Makefile->stat.has_sccs = NO_SCCS;
2208                 primary_makefile = Makefile;
2209                 makefile_read = read_makefile(Makefile,
2210                                             false,
2211                                             false,
2212                                             true);
2213             }
2214         }
2215         if (!makefile_read &&
2216             makefile_name->stat.is_file) {
2217             makefile_name->stat.has_sccs = save_m_has_sccs;
2218             primary_makefile = makefile_name;
2219             makefile_read = read_makefile(makefile_name,
2220                                         false,
2221                                         false,
2222                                         true);
2223         }
2224         if (!makefile_read &&
2225             Makefile->stat.is_file) {
2226             Makefile->stat.has_sccs = save_M_has_sccs;
2227             primary_makefile = Makefile;

```

```

2228     makefile_read = read_makefile(Makefile,
2229                                   false,
2230                                   false,
2231                                   true);
2232     }
2233 }
2234 }
2235 do_not_exec_rule = save_do_not_exec_rule;
2236 allrules_read = makefile_read;
2237 trace_reader = false;

2239 /*
2240 * Now get current value of MAKEFLAGS and compare it with
2241 * the saved value we set before reading makefile.
2242 * If they are different then MAKEFLAGS is subsequently set by
2243 * makefile, just leave it there. Otherwise, if make mode
2244 * is changed by using .POSIX target in makefile we need
2245 * to correct MAKEFLAGS value.
2246 */
2247 Name mf_val = getvar(makeflags);
2248 if( (posix != is_xpg4)
2249     && (!strcmp(mf_val->string_mb, makeflags_value_saved->string_mb)))
2250 {
2251     if (makeflags_string_posix.buffer.start[1] == (int) nul_char) {
2252         (void) SETVAR(makeflags,
2253                       GETNAME(makeflags_string_posix.buffer.star
2254                               FIND_LENGTH),
2255                       false);
2256     } else {
2257         if (makeflags_string_posix.buffer.start[1] != (int) spac
2258             (void) SETVAR(makeflags,
2259                           GETNAME(makeflags_string_posix.buf
2260                                   FIND_LENGTH),
2261                           false);
2262     } else {
2263         (void) SETVAR(makeflags,
2264                       GETNAME(makeflags_string_posix.buf
2265                               FIND_LENGTH),
2266                       false);
2267     }
2268 }
2269 }

2271 if (makeflags_string.free_after_use) {
2272     retmem(makeflags_string.buffer.start);
2273 }
2274 if (makeflags_string_posix.free_after_use) {
2275     retmem(makeflags_string_posix.buffer.start);
2276 }
2277 makeflags_string.buffer.start = NULL;
2278 makeflags_string_posix.buffer.start = NULL;

2280 if (posix) {
2281     /*
2282     * If the user did not redefine the ARFLAGS macro in the
2283     * default makefile (make.rules), then we'd like to
2284     * change the macro value of ARFLAGS to be in accordance
2285     * with "POSIX" requirements.
2286     */
2287     MBSTOWCS(wcs_buffer, "ARFLAGS");
2288     name = GETNAME(wcs_buffer, wcslen(wcs_buffer));
2289     value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2290     macro = get_prop(name->prop, macro_prop);
2291     if ((macro != NULL) && /* Maybe (macro == NULL) || ? */
2292         (IS_EQUAL(macro->body.macro.value->string_mb,
2293                  "rv"))) {

```

```

2293     MBSTOWCS(wcs_buffer, "-rv");
2294     value = GETNAME(wcs_buffer, wcslen(wcs_buffer));
2295     value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2296     (void) SETVAR(name,
2297                   value,
2298                   false);
2299 }

2301 if (!posix && !svr4) {
2302     set_sgs_support();
2303 }

2306 /*
2307 * Make sure KEEP_STATE is in the environment if KEEP_STATE is on.
2308 */
2309 macro = get_prop(keep_state_name->prop, macro_prop);
2310 if ((macro != NULL) &&
2311     macro->body.macro.exported) {
2312     keep_state = true;
2313 }
2314 if (keep_state) {
2315     if (macro == NULL) {
2316         macro = maybe_append_prop(keep_state_name,
2317                                   macro_prop);
2318     }
2319     macro->body.macro.exported = true;
2320     (void) SETVAR(keep_state_name,
2321                   empty_name,
2322                   false);

2324     /*
2325     * Read state file
2326     */

2328     /* Before we read state, let's make sure we have
2329     ** right state file.
2330     */
2331     /* just in case macro references are used in make_state file
2332     ** name, we better expand them at this stage using expand_value.
2333     */
2334     INIT_STRING_FROM_STACK(dest, destbuffer);
2335     expand_value(make_state, &dest, false);

2337     make_state = GETNAME(dest.buffer.start, FIND_LENGTH);

2339     if (!stat(make_state->string_mb, &make_state_stat)) {
2340         if (!(make_state_stat.st_mode & S_IFREG)) {
2341             /* copy the make_state structure to the other
2342             ** and then let make_state point to the new
2343             ** one.
2344             */
2345             memcpy(&state_filename, make_state, sizeof(state_filename));
2346             state_filename.string_mb = state_file_str_mb;
2347             /* Just a kludge to avoid two slashes back to back */
2348             if ((make_state->hash.length == 1) &&
2349                 (make_state->string_mb[0] == '/')) {
2350                 make_state->hash.length = 0;
2351                 make_state->string_mb[0] = '\0';
2352             }
2353             sprintf(state_file_str_mb, "%s%s",
2354                   make_state->string_mb, "/.make.state");
2355             make_state = &state_filename;
2356             /* adjust the length to reflect the appended string */
2357             make_state->hash.length += 12;

```

```

2358     }
2359     } else { /* the file doesn't exist or no permission */
2360     char tmp_path[MAXPATHLEN];
2361     char *slashp;

2363     if (slashp = strrchr(make_state->string_mb, '/')) {
2364     strncpy(tmp_path, make_state->string_mb,
2365             (slashp - make_state->string_mb));
2366     tmp_path[slashp - make_state->string_mb]=0;
2367     if (strlen(tmp_path)) {
2368     if (stat(tmp_path, &make_state_stat)) {
2369     warning(gettext("directory %s for .KEEP_STATE_FILE doe
2370     });
2371     if (access(tmp_path, F_OK) != 0) {
2372     warning(gettext("can't access dir %s"), tmp_path);
2373     }
2374     }
2375     }
2376     }
2377     if (report_dependencies_level != 1) {
2378     Makefile_type makefile_type_temp = makefile_type;
2379     makefile_type = reading_statefile;
2380     if (read_trace_level > 1) {
2381     trace_reader = true;
2382     }
2383     (void) read_simple_file(make_state,
2384                             false,
2385                             false,
2386                             false,
2387                             false,
2388                             false,
2389                             true);
2390     trace_reader = false;
2391     makefile_type = makefile_type_temp;
2392     }
2393     }
2394 }

2396 /*
2397  * Scan the argv for options and "=" type args and make them readonly.
2398  */
2399 static void
2400 enter_argv_values(int argc, char *argv[], ASCII_Dyn_Array *makeflags_and_macro)
2401 {
2402     register char *cp;
2403     register int i;
2404     int length;
2405     register Name name;
2406     int opt_separator = argc;
2407     char tmp_char;
2408     wchar_t *tmp_wcs_buffer;
2409     register Name value;
2410     Boolean append = false;
2411     Property macro;
2412     struct stat statbuf;

2415     /* Read argv options and "=" type args and make them readonly. */
2416     makefile_type = reading_nothing;
2417     for (i = 1; i < argc; ++i) {
2418     append = false;
2419     if (argv[i] == NULL) {
2420     continue;
2421     } else if (((argv[i][0] == '-') && (argv[i][1] == '-')) ||
2422                ((argv[i][0] == (int) ' ') &&
2423                 (argv[i][1] == (int) '-') &&

```

```

2424     (argv[i][2] == (int) ' ') &&
2425     (argv[i][3] == (int) '-')) {
2426     argv[i] = NULL;
2427     opt_separator = i;
2428     continue;
2429     } else if ((i < opt_separator) && (argv[i][0] == (int) hyphen_ch
2430     switch (parse_command_option(argv[i][1])) {
2431     case 1: /* -f seen */
2432     ++i;
2433     continue;
2434     case 2: /* -c seen */
2435     if (argv[i+1] == NULL) {
2436     fatal(gettext("No dmake rcfile argument
2437     });
2438     MBSTOWCS(wcs_buffer, "DMAKE_RCFILE");
2439     name = GETNAME(wcs_buffer, FIND_LENGTH);
2440     break;
2441     case 4: /* -g seen */
2442     if (argv[i+1] == NULL) {
2443     fatal(gettext("No dmake group argument a
2444     });
2445     MBSTOWCS(wcs_buffer, "DMAKE_GROUP");
2446     name = GETNAME(wcs_buffer, FIND_LENGTH);
2447     break;
2448     case 8: /* -j seen */
2449     if (argv[i+1] == NULL) {
2450     fatal(gettext("No dmake max jobs argumen
2451     });
2452     MBSTOWCS(wcs_buffer, "DMAKE_MAX_JOBS");
2453     name = GETNAME(wcs_buffer, FIND_LENGTH);
2454     break;
2455     case 16: /* -M seen */
2456     if (argv[i+1] == NULL) {
2457     fatal(gettext("No pmake machinesfile arg
2458     });
2459     MBSTOWCS(wcs_buffer, "PMAKE_MACHINESFILE");
2460     name = GETNAME(wcs_buffer, FIND_LENGTH);
2461     break;
2462     case 32: /* -m seen */
2463     if (argv[i+1] == NULL) {
2464     fatal(gettext("No dmake mode argument af
2465     });
2466     MBSTOWCS(wcs_buffer, "DMAKE_MODE");
2467     name = GETNAME(wcs_buffer, FIND_LENGTH);
2468     break;
2469     case 256: /* -K seen */
2470     if (argv[i+1] == NULL) {
2471     fatal(gettext("No makestate filename arg
2472     });
2473     MBSTOWCS(wcs_buffer, argv[i+1]);
2474     make_state = GETNAME(wcs_buffer, FIND_LENGTH);
2475     keep_state = true;
2476     argv[i] = NULL;
2477     argv[i+1] = NULL;
2478     continue;
2479     case 512: /* -o seen */
2480     if (argv[i+1] == NULL) {
2481     fatal(gettext("No dmake output dir argum
2482     });
2483     MBSTOWCS(wcs_buffer, "DMAKE_ODIR");
2484     name = GETNAME(wcs_buffer, FIND_LENGTH);
2485     break;
2486     case 1024: /* -x seen */
2487     if (argv[i+1] == NULL) {
2488     fatal(gettext("No argument after -x flag
2489     });

```



```

2490 length = strlen( "SUN_MAKE_COMPAT_MODE=");
2491 if (strncmp(argv[i+1], "SUN_MAKE_COMPAT_MODE=",
2492             argv[i+1] + &argv[i+1][length];
2493     MBSTOWCS(wcs_buffer, "SUN_MAKE_COMPAT_MO
2494     name = GETNAME(wcs_buffer, FIND_LENGTH);
2495     dmake_compat_mode_specified = dmake_add_
2496     break;
2497 }
2498 length = strlen( "DMAKE_OUTPUT_MODE=");
2499 if (strncmp(argv[i+1], "DMAKE_OUTPUT_MODE=", len
2500     argv[i+1] + &argv[i+1][length];
2501     MBSTOWCS(wcs_buffer, "DMAKE_OUTPUT_MODE"
2502     name = GETNAME(wcs_buffer, FIND_LENGTH);
2503     dmake_output_mode_specified = dmake_add_
2504 } else {
2505     warning(gettext("Unknown argument '%s' a
2506             argv[i+1]));
2507     argv[i] = argv[i + 1] = NULL;
2508     continue;
2509 }
2510 break;
2511 default: /* Shouldn't reach here */
2512     argv[i] = NULL;
2513     continue;
2514 }
2515 argv[i] = NULL;
2516 if (i == (argc - 1)) {
2517     break;
2518 }
2519 if ((length = strlen(argv[i+1])) >= MAXPATHLEN) {
2520     tmp_wcs_buffer = ALLOC_WC(length + 1);
2521     (void) mbstowcs(tmp_wcs_buffer, argv[i+1], lengt
2522     value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2523     retmem(tmp_wcs_buffer);
2524 } else {
2525     MBSTOWCS(wcs_buffer, argv[i+1]);
2526     value = GETNAME(wcs_buffer, FIND_LENGTH);
2527 }
2528     argv[i+1] = NULL;
2529 } else if ((cp = strchr(argv[i], (int) equal_char)) != NULL) {
2530 /*
2531  * Combine all macro in dynamic array
2532  */
2533     if(*(cp-1) == (int) plus_char)
2534     {
2535         if(isspace(*(cp-2))) {
2536             append = true;
2537             cp--;
2538         }
2539     }
2540     if(!append)
2541         append_or_replace_macro_in_dyn_array(makeflags_a

2543 while (isspace(*(cp-1))) {
2544     cp--;
2545 }
2546 tmp_char = *cp;
2547 *cp = (int) nul_char;
2548 MBSTOWCS(wcs_buffer, argv[i]);
2549 *cp = tmp_char;
2550 name = GETNAME(wcs_buffer, wcslen(wcs_buffer));
2551 name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2552 while (*cp != (int) equal_char) {
2553     cp++;
2554 }
2555 }
2556 cp++;

```

```

2555 while (isspace(*cp) && (*cp != (int) nul_char)) {
2556     cp++;
2557 }
2558 if ((length = strlen(cp)) >= MAXPATHLEN) {
2559     tmp_wcs_buffer = ALLOC_WC(length + 1);
2560     (void) mbstowcs(tmp_wcs_buffer, cp, length + 1);
2561     value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2562     retmem(tmp_wcs_buffer);
2563 } else {
2564     MBSTOWCS(wcs_buffer, cp);
2565     value = GETNAME(wcs_buffer, FIND_LENGTH);
2566 }
2567     argv[i] = NULL;
2568 } else {
2569     /* Illegal MAKEFLAGS argument */
2570     continue;
2571 }
2572 if(append) {
2573     setvar_append(name, value);
2574     append = false;
2575 } else {
2576     macro = maybe_append_prop(name, macro_prop);
2577     macro->body.macro.exported = true;
2578     SETVAR(name, value, false)->body.macro.read_only = true;
2579 }
2580 }
2581 }

```

unchanged\_portion\_omitted

```

2622 /*
2623  * read_environment(read_only)
2624  *
2625  * This routine reads the process environment when make starts and enters
2626  * it as make macros. The environment variable SHELL is ignored.
2627  *
2628  * Parameters:
2629  *     read_only      Should we make env vars read only?
2630  *
2631  * Global variables used:
2632  *     report_pwd     Set if this make was started by other make
2633  */
2634 static void
2635 read_environment(Boolean read_only)
2636 {
2637     register char      **environment;
2638     int                length;
2639     wchar_t            *tmp_wcs_buffer;
2640     Boolean            allocated_tmp_wcs_buffer = false;
2641     register wchar_t   *name;
2642     register wchar_t   *value;
2643     register Name      macro;
2644     Property           val;
2645     Boolean            read_only_saved;

2647     reading_environment = true;
2648     environment = environ;
2649     for (; *environment; environment++) {
2650         read_only_saved = read_only;
2651         if ((length = strlen(*environment)) >= MAXPATHLEN) {
2652             tmp_wcs_buffer = ALLOC_WC(length + 1);
2653             allocated_tmp_wcs_buffer = true;
2654             (void) mbstowcs(tmp_wcs_buffer, *environment, length + 1
2655             name = tmp_wcs_buffer;
2656         } else {
2657             MBSTOWCS(wcs_buffer, *environment);
2658             name = wcs_buffer;

```

```

2659     }
2660     value = (wchar_t *) wcschr(name, (int) equal_char);
2661     value = (wchar_t *) wschr(name, (int) equal_char);
2662
2663     /*
2664     * Looks like there's a bug in the system, but sometimes
2665     * you can get blank lines in *environment.
2666     */
2667     if (!value) {
2668         continue;
2669     }
2670     MBSTOWCS(wcs_buffer2, "SHELL=");
2671     if (IS_WEQUALN(name, wcs_buffer2, wcslen(wcs_buffer2))) {
2672         if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2673             continue;
2674         }
2675         MBSTOWCS(wcs_buffer2, "MAKEFLAGS=");
2676         if (IS_WEQUALN(name, wcs_buffer2, wcslen(wcs_buffer2))) {
2677             if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2678                 report_pwd = true;
2679                 /*
2680                 * In POSIX mode we do not want MAKEFLAGS to be readonly
2681                 * If the MAKEFLAGS macro is subsequently set by the mak
2682                 * it replaces the MAKEFLAGS variable currently found in
2683                 * environment.
2684                 * See Assertion 50 in section 6.2.5.3 of standard P1003
2685                 */
2686                 if (posix) {
2687                     read_only_saved = false;
2688                 }
2689             }
2690         }
2691     }
2692     /*
2693     * We ignore SUNPRO_DEPENDENCIES. This environment variable is
2694     * set by make and read by cpp which then writes info to
2695     * .make.dependency.xxx. When make is invoked by another make
2696     * (recursive make), we don't want to read this because then
2697     * the child make will end up writing to the parent
2698     * directory's .make.state and clobbering them.
2699     */
2700     MBSTOWCS(wcs_buffer2, "SUNPRO_DEPENDENCIES");
2701     if (IS_WEQUALN(name, wcs_buffer2, wcslen(wcs_buffer2))) {
2702         if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2703             continue;
2704         }
2705     }
2706     macro = GETNAME(name, value - name);
2707     maybe_append_prop(macro, macro_prop)->body.macro.exported =
2708     true;
2709     if ((value == NULL) || ((value + 1)[0] == (int) nul_char)) {
2710         val = setvar_daemon(macro,
2711             (Name) NULL,
2712             false, no_daemon, false, debug_level
2713         )
2714     } else {
2715         val = setvar_daemon(macro,
2716             GETNAME(value + 1, FIND_LENGTH),
2717             false, no_daemon, false, debug_level
2718         )
2719     }
2720     val->body.macro.read_only = read_only_saved;
2721     if (allocated_tmp_wcs_buffer) {
2722         retmem(tmp_wcs_buffer);
2723         allocated_tmp_wcs_buffer = false;
2724     }
2725     reading_environment = false;
2726 }

```

unchanged\_portion\_omitted

```

2754 /*
2755 * make_targets(argc, argv, parallel_flag)
2756 *
2757 * Call doname on the specified targets
2758 *
2759 * Parameters:
2760 *     argc          You know what this is
2761 *     argv          You know what this is
2762 *     parallel_flag True if building in parallel
2763 *
2764 * Global variables used:
2765 *     build_failed_seen Used to generate message after failed -k
2766 *     commands_done     Used to generate message "Up to date"
2767 *     default_target_to_build First proper target in makefile
2768 *     init              The Name ".INIT", use to run command
2769 *     parallel          Global parallel building flag
2770 *     quest             make -q, suppresses messages
2771 *     recursion_level   Initialized, used for tracing
2772 *     report_dependencies make -P, regresses whole process
2773 */
2774 static void
2775 make_targets(int argc, char **argv, Boolean parallel_flag)
2776 {
2777     int i;
2778     char *cp;
2779     Boolean Doname result;
2780     Boolean register target_to_make_found = false;
2781
2782     (void) doname(init, true, true);
2783     recursion_level = 1;
2784     parallel = parallel_flag;
2785
2786     /*
2787     * make remaining args
2788     */
2789     if ((report_dependencies_level == 0) && parallel) {
2790         /*
2791         * If building targets in parallel, start all of the
2792         * remaining args to build in parallel.
2793         */
2794         for (i = 1; i < argc; i++) {
2795             if ((cp = argv[i]) != NULL) {
2796                 commands_done = false;
2797                 if ((cp[0] == (int) period_char) &&
2798                     (cp[1] == (int) slash_char)) {
2799                     cp += 2;
2800                 }
2801                 if ((cp[0] == (int) ' ') &&
2802                     (cp[1] == (int) '-') &&
2803                     (cp[2] == (int) ' ') &&
2804                     (cp[3] == (int) '-')) {
2805                     argv[i] = NULL;
2806                     continue;
2807                 }
2808             }
2809             MBSTOWCS(wcs_buffer, cp);
2810             //default_target_to_build = GETNAME(wcs_buffer,
2811             //                                FIND_LENGTH);
2812             default_target_to_build = normalize_name(wcs_buf
2813             wcslen(wcs_buf
2814             wslen(wcs_buff
2815             if (default_target_to_build == wait_name) {
2816                 if (parallel_process_cnt > 0) {
2817                     finish_running();

```

```

2818         }
2819         continue;
2820     }
2821     top_level_target = get_wstring(default_target_to
2822     /*
2823     * If we can't execute the current target in
2824     * parallel, hold off the target processing
2825     * to preserve the order of the targets as they
2826     * in command line.
2827     */
2828     if (!parallel_ok(default_target_to_build, false)
2829         && parallel_process_cnt > 0) {
2830         finish_running();
2831     }
2832     result = doname_check(default_target_to_build,
2833         true,
2834         false,
2835         false);
2836     gather_recursive_deps();
2837     if (/* !commands_done && */
2838         (result == build_ok) &&
2839         !quest &&
2840         (report_dependencies_level == 0) /* &&
2841         (exists(default_target_to_build) > file_does
2842         if (posix) {
2843             if (!commands_done) {
2844                 (void) printf(gettext("`
2845                 default_ta
2846             } else {
2847                 if (no_action_was_taken)
2848                     (void) printf(get
2849                     de
2850             }
2851         } else {
2852             default_target_to_build->stat.ti
2853             if (!commands_done &&
2854                 (exists(default_target_to_bu
2855                 (void) printf(gettext("`
2856                 default_ta
2857             }
2858         }
2859     }
2860 }
2861 }
2862 /* Now wait for all of the targets to finish running */
2863 finish_running();
2864 // setjmp(jmpbuffer);
2865 }
2866 }
2867 for (i = 1; i < argc; i++) {
2868     if ((cp = argv[i]) != NULL) {
2869         target_to_make_found = true;
2870         if ((cp[0] == (int) period_char) &&
2871             (cp[1] == (int) slash_char)) {
2872             cp += 2;
2873         }
2874         if ((cp[0] == (int) ' ') &&
2875             (cp[1] == (int) '-') &&
2876             (cp[2] == (int) ' ') &&
2877             (cp[3] == (int) '-')) {
2878             argv[i] = NULL;
2879             continue;
2880         }
2881     }
2882     MBSTOWCS(wcs_buffer, cp);
2883     default_target_to_build = normalize_name(wcs_buffer, wcs

```

```

2883     default_target_to_build = normalize_name(wcs_buffer, wcs
2884     top_level_target = get_wstring(default_target_to_build->
2885     report_recursion(default_target_to_build);
2886     commands_done = false;
2887     if (parallel) {
2888         result = (Doname) default_target_to_build->state
2889     } else {
2890         result = doname_check(default_target_to_build,
2891             true,
2892             false,
2893             false);
2894     }
2895     gather_recursive_deps();
2896     if (build_failed_seen) {
2897         build_failed_ever_seen = true;
2898         warning(gettext("Target '%s' not remade because
2899         default_target_to_build->string_mb);
2900     }
2901     build_failed_seen = false;
2902     if (report_dependencies_level > 0) {
2903         print_dependencies(default_target_to_build,
2904             get_prop(default_target_to_bu
2905             line_prop));
2906     }
2907     default_target_to_build->stat.time =
2908     file_no_time;
2909     if (default_target_to_build->colon_splits > 0) {
2910         default_target_to_build->state =
2911         build_dont_know;
2912     }
2913     if (!parallel &&
2914         /* !commands_done && */
2915         (result == build_ok) &&
2916         !quest &&
2917         (report_dependencies_level == 0) /* &&
2918         (exists(default_target_to_build) > file_doesnt_exist
2919         if (posix) {
2920             if (!commands_done) {
2921                 (void) printf(gettext("`%s' is u
2922                 default_target_to_
2923             } else {
2924                 if (no_action_was_taken) {
2925                     (void) printf(gettext("`
2926                     default_ta
2927             }
2928         } else {
2929             if (!commands_done &&
2930                 (exists(default_target_to_build) > f
2931                 (void) printf(gettext("`%s' is u
2932                 default_target_to_
2933             }
2934         }
2935     }
2936     }
2937     }
2938 }
2939 }
2940 /*
2941 * If no file arguments have been encountered,
2942 * make the first name encountered that doesnt start with a dot
2943 */
2944 if (!target_to_make_found) {
2945     if (default_target_to_build == NULL) {
2946         fatal(gettext("No arguments to build"));
2947     }
2948     commands_done = false;

```

```

2949         top_level_target = get_wstring(default_target_to_build->string_m
2950         report_recursion(default_target_to_build);

2953         if (getenv("SPRO_EXPAND_ERRORS")){
2954             (void) printf("::(%s)\n",
2955                 default_target_to_build->string_mb);
2956         }

2959         result = doname_parallel(default_target_to_build, true, false);
2960         gather_recursive_deps();
2961         if (build_failed_seen) {
2962             build_failed_ever_seen = true;
2963             warning(gettext("Target '%s' not remade because of error
2964                 default_target_to_build->string_mb);
2965         }
2966         build_failed_seen = false;
2967         if (report_dependencies_level > 0) {
2968             print_dependencies(default_target_to_build,
2969                 get_prop(default_target_to_build->
2970                     prop,
2971                     line_prop));
2972         }
2973         default_target_to_build->stat.time = file_no_time;
2974         if (default_target_to_build->colon_splits > 0) {
2975             default_target_to_build->state = build_dont_know;
2976         }
2977         if (/* !commands_done && */
2978             (result == build_ok) &&
2979             !quest &&
2980             (report_dependencies_level == 0) /* &&
2981             (exists(default_target_to_build) > file_doesnt_exist) */) {
2982             if (posix) {
2983                 if (!commands_done) {
2984                     (void) printf(gettext("'%s' is updated.\n
2985                         default_target_to_build->s
2986                 } else {
2987                     if (no_action_was_taken) {
2988                         (void) printf(gettext("'%s': no
2989                             default_target_to_
2990                     }
2991                 }
2992             } else {
2993                 if (!commands_done &&
2994                     (exists(default_target_to_build) > file_does
2995                     (void) printf(gettext("'%s' is up to dat
2996                         default_target_to_build->s
2997                 }
2998             }
2999         }
3000     }
3001 }

```

unchanged portion omitted

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

1

\*\*\*\*\*

19334 Fri May 22 11:19:43 2015

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

make: use the more modern wchar routines, not widec.h

\*\*\*\*\*

unchanged\_portion\_omitted\_

```
488 void
489 dump_target_list(void)
490 {
491     Name_set::iterator    p, e;
492     Wstring str;
493
494     for (p = hashtab.begin(), e = hashtab.end(); p != e; p++) {
495         str.init(p);
496         wchar_t * wcb = str.get_string();
497         if ((p->colons != no_colon) &&
498             ((wcb[0] != (int) period_char ||
499              (wcb[0] == (int) period_char) &&
500              (wcschr(wcb, (int) slash_char)))) {
501             (wschr(wcb, (int) slash_char))) {
502                 print_target_n_deps(p);
503             }
504     }
```

unchanged\_portion\_omitted\_

new/usr/src/cmd/make/bin/nse\_printdep.cc

1

\*\*\*\*\*

8765 Fri May 22 11:19:44 2015

new/usr/src/cmd/make/bin/nse\_printdep.cc

make: use the more modern wchar routines, not widec.h

\*\*\*\*\*

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_

```
345 /*
346 * If target is recursive, print the following to standard out:
347 * .RECURSIVE subdir targ Makefile
348 */
349 static void
350 print_rec_info(Name target)
351 {
352     Recursive_make rp;
353     wchar_t *colon;
354
355     report_recursive_init();
356
357     rp = find_recursive_target(target);
358
359     if (rp) {
360         /*
361          * if found, print starting with the space after the ':'
362          */
363         colon = (wchar_t *) wcschr(rp->oldline, (int) colon_char);
364         colon = (wchar_t *) wcschr(rp->oldline, (int) colon_char);
365         (void) printf("%s", colon + 1);
366     }
367 }
```

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_

```

*****
11055 Fri May 22 11:19:44 2015
new/usr/src/cmd/make/bin/pmake.cc
make: use the more modern wchar routines, not widec.h
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

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

42 /*
43 * Defined macros
44 */

46 /*
47 * typedefs & structs
48 */

50 /*
51 * Static variables
52 */

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

```

```

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

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

```

```

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

```

```

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

```



```

256         */
257         pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
258         /* Compare current hostname with full_host. */
259         } else if (wcslen(mp) == full_host_wslen &&
260                 } else if (wslen(mp) == full_host_wslen &&
261                        IS_WEQUALN(mp, full_host, full_host_w
262                                /*
263                                * Bingo, full_host is in .make.machines
264                                * Continue reading.
265                                */
266                                pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
267                                } else {
268                                if (c != '\n') {
269                                    SKIPTOEND(ms);
270                                    if (*ms) {
271                                        ms++;
272                                    }
273                                }
274                                continue;
275                                }
276                                /* If we get here, local_host is in .make.machin
277                                if (c != '\n') {
278                                    /* Now look for keyword 'max'. */
279                                    MBSTOWCS(wcs_buffer, "max");
280                                    SKIPSPACE(ms);
281                                    while ((*ms != '\n') && (*ms)) {
282                                        if (*ms == '#') {
283                                            pskip_comment(&ms);
284                                        } else if (IS_WEQUALN(ms, wcs_bu
285                                                /* Skip "max". */
286                                                ms += 3;
287                                                pmake_max_jobs = get_max
288                                                SKIPSPACE(ms);
289                                        } else {
290                                            warning(gettext("unknown
291                                            SKIPTOEND(ms);
292                                            break;
293                                        }
294                                    }
295                                }
296                                break; /* out of outermost for() loop. */
297                            }
298                        }
299                    retmem(make_machines_list);
300                    return(pmake_max_jobs);
301                }

```

unchanged\_portion\_omitted

```

*****
56715 Fri May 22 11:19:45 2015
new/usr/src/cmd/make/bin/read.cc
make: use the more modern wchar routines, not widec.h
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  *      read.c
28  *
29  *      This file contains the makefile reader.
30  */

32 /*
33  * Included files
34  */
35 #include <alloca.h>          /* alloca() */
36 #include <errno.h>          /* errno */
37 #include <fcntl.h>          /* fcntl() */
38 #include <mk/defs.h>
39 #include <mksh/macro.h>     /* expand_value(), expand_macro() */
40 #include <mksh/misc.h>     /* getmem() */
41 #include <mksh/read.h>     /* get_next_block_fn() */
42 #include <sys/uio.h>        /* read() */
43 #include <unistd.h>         /* read(), unlink() */
44 #include <libintl.h>

47 /*
48  * typedefs & structs
49  */

51 /*
52  * Static variables
53  */

55 static int line_started_with_space=0; // Used to diagnose spaces instead of tabs

57 /*
58  * File table of contents
59  */
60 static void      parse_makefile(register Name true_makefile_name, register
61 static Source    push_macro_value(register Source bp, register wchar_t *b

```

```

62 extern void      enter_target_groups_and_dependencies(Name_vector target,
63 extern Name      normalize_name(register wchar_t *name_string, register i

65 /*
66  *      read_simple_file(makefile_name, chase_path, doname_it,
67  *                      complain, must_exist, report_file, lock_makefile)
68  *
69  *      Make the makefile and setup to read it. Actually read it if it is stdio
70  *
71  *      Return value:
72  *                      false if the read failed
73  *
74  *      Parameters:
75  *      makefile_name   Name of the file to read
76  *      chase_path      Use the makefile path when opening file
77  *      doname_it       Call doname() to build the file first
78  *      complain        Print message if doname/open fails
79  *      must_exist      Generate fatal if file is missing
80  *      report_file     Report file when running -P
81  *      lock_makefile   Lock the makefile when reading
82  *
83  *      Static variables used:
84  *
85  *      Global variables used:
86  *      do_not_exec_rule Is -n on?
87  *      file_being_read  Set to the name of the new file
88  *      line_number      The number of the current makefile line
89  *      makefiles_used   A list of all makefiles used, appended to
90  */

93 Boolean
94 read_simple_file(register Name makefile_name, register Boolean chase_path, regis
95 {
96     static short      max_include_depth;
97     register Property makefile = maybe_append_prop(makefile_name,
98                                                     makefile_prop);
99     Boolean           forget_after_parse = false;
100     static pathpt    makefile_path;
101     register int      n;
102     char              *path;
103     register Source   source = ALLOC(Source);
104     Property          orig_makefile = makefile;
105     Dependency        *dpp;
106     Dependency        dp;
107     register int      length;
108     wchar_t           *previous_file_being_read = file_being_read;
109     int               previous_line_number = line_number;
110     wchar_t           previous_current_makefile[MAXPATHLEN];
111     Makefile_type     save_makefile_type;
112     Name              normalized_makefile_name;
113     register wchar_t  *string_start;
114     register wchar_t  *string_end;

118     wchar_t * wcb = get_wstring(makefile_name->string_mb);

120     if (max_include_depth++ >= 40) {
121         fatal(gettext("Too many nested include statements"));
122     }
123     if (makefile->body.makefile.contents != NULL) {
124         retmem(makefile->body.makefile.contents);
125     }
126     source->inp_buf =
127         source->inp_buf_ptr =

```

```

128     source->inp_buf_end = NULL;
129     source->error_converting = false;
130     makefile->body.makefile.contents = NULL;
131     makefile->body.makefile.size = 0;
132     if ((makefile_name->hash.length != 1) ||
133         (wcb[0] != (int) hyphen_char)) {
134         if ((makefile->body.makefile.contents == NULL) &&
135             (doname_it)) {
136             if (makefile_path == NULL) {
137                 char *pfx = make_install_prefix();
138                 char *path;

140                 add_dir_to_path(".",
141                               &makefile_path,
142                               -1);

144                 // As regularly installed
145                 asprintf(&path, "%s/../../share/lib/make", pfx);
146                 add_dir_to_path(path, &makefile_path, -1);
147                 free(path);

149                 // Tools build
150                 asprintf(&path, "%s/../../share/", pfx);
151                 add_dir_to_path(path, &makefile_path, -1);
152                 free(path);

154                 add_dir_to_path("/usr/share/lib/make",
155                               &makefile_path,
156                               -1);
157                 add_dir_to_path("/etc/default",
158                               &makefile_path,
159                               -1);

161                 free(pfx);
162             }
163             save_makefile_type = makefile_type;
164             makefile_type = reading_nothing;
165             if (doname(makefile_name, true, false) == build_dont_kno
166                 /* Try normalized filename */
167                 string_start=get_wstring(makefile_name->string_m
168                 for (string_end=string_start+1; *string_end != L
169                 normalized_makefile_name=normalize_name(string_s
170                 if ((strcmp(makefile_name->string_mb, normalized
171                     (doname(normalized_makefile_name, true,
172                         n = access_vroot(makefile_name->string_m
173                             4,
174                             chase_path ?
175                             makefile_path : NULL,
176                             VROOT_DEFAULT);
177                 if (n == 0) {
178                     get_vroot_path((char **) NULL,
179                                   &path,
180                                   (char **) NULL);
181                     if ((path[0] == (int) period_cha
182                         (path[1] == (int) slash_char
183                             path += 2;
184                     }
185                     MBSTOWCS(wcs_buffer, path);
186                     makefile_name = GETNAME(wcs_buff
187                                 FIND_LENGTH);
188                 }
189             }
190             retmem(string_start);
191             /*
192             * Commented out: retmem_mb(normalized_makefile_
193             * We have to return this memory, but it seems t

```

```

194         * in dmake or in Sun C++ 5.7 compiler (it works
195         * is compiled using Sun C++ 5.6).
196         */
197         // retmem_mb(normalized_makefile_name->string_mb
198     }
199     makefile_type = save_makefile_type;
200 }
201 source->string.free_after_use = false;
202 source->previous = NULL;
203 source->already_expanded = false;
204 /* Lock the file for read, but not when -n. */
205 if (lock_makefile &&
206     !do_not_exec_rule) {

208     make_state_lockfile = getmem(strlen(make_state->string_
209                                   (void) sprintf(make_state_lockfile,
210                                                 "%s.lock",
211                                                 make_state->string_mb);
212     (void) file_lock(make_state->string_mb,
213                     make_state_lockfile,
214                     (int *) &make_state_locked,
215                     0);
216     if(!make_state_locked) {
217         printf("-- NO LOCKING for read\n");
218         retmem_mb(make_state_lockfile);
219         make_state_lockfile = 0;
220         return failed;
221     }
222 }
223 if (makefile->body.makefile.contents == NULL) {
224     save_makefile_type = makefile_type;
225     makefile_type = reading_nothing;
226     if ((doname_it) &&
227         (doname(makefile_name, true, false) == build_failed)
228         if (complain) {
229             (void) fprintf(stderr,
230                             gettext("%s: Couldn't mak
231                                     getprogname(),
232                                     makefile_name->string_mb)
233             }
234             max_include_depth--;
235             makefile_type = save_makefile_type;
236             return failed;
237         }
238     makefile_type = save_makefile_type;
239     //
240     // Before calling exists() make sure that we have the ri
241     //
242     makefile_name->stat.time = file_no_time;

244     if (exists(makefile_name) == file_doesnt_exist) {
245         if (complain ||
246             (makefile_name->stat.stat_errno != ENOENT))
247             if (must_exist) {
248                 fatal(gettext("Can't find '%s':
249                         makefile_name->string_mb,
250                         errmsg(makefile_name->
251                                 stat.stat_errno));
252             } else {
253                 warning(gettext("Can't find '%s'
254                         makefile_name->string_mb
255                         errmsg(makefile_name->
256                                 stat.stat_errno))
257             }
258         }
259         max_include_depth--;

```

```

260     if(make_state_locked && (make_state_lockfile !=
261         (void) unlink(make_state_lockfile);
262         retmem_mb(make_state_lockfile);
263         make_state_lockfile = NULL;
264         make_state_locked = false;
265     }
266     retmem(wcb);
267     retmem_mb((char *)source);
268     return failed;
269 }
270 /*
271  * These values are the size and bytes of
272  * the MULTI-BYTE makefile.
273  */
274 orig_makefile->body.makefile.size =
275 makefile->body.makefile.size =
276 source->bytes_left_in_file =
277 makefile_name->stat.size;
278 if (report_file) {
279     for (dpp = &makefiles_used;
280          *dpp != NULL;
281          dpp = &(*dpp)->next);
282     dp = ALLOC(Dependency);
283     dp->next = NULL;
284     dp->name = makefile_name;
285     dp->automatic = false;
286     dp->stale = false;
287     dp->built = false;
288     *dpp = dp;
289 }
290 source->fd = open_vroot(makefile_name->string_mb,
291                        O_RDONLY,
292                        0,
293                        NULL,
294                        VROOT_DEFAULT);
295 if (source->fd < 0) {
296     if (complain || (errno != ENOENT)) {
297         if (must_exist) {
298             fatal(gettext("Can't open '%s':
299                 makefile_name->string_mb,
300                 errmsg(errno));
301         } else {
302             warning(gettext("Can't open '%s'
303                 makefile_name->string_mb
304                 errmsg(errno));
305         }
306     }
307     max_include_depth--;
308     return failed;
309 }
310 (void) fcntl(source->fd, F_SETFD, 1);
311 orig_makefile->body.makefile.contents =
312 makefile->body.makefile.contents =
313 source->string.text.p =
314 source->string.buffer.start =
315     ALLOC_WC((int) (makefile_name->stat.size + 2));
316 if (makefile_type == reading_cpp_file) {
317     forget_after_parse = true;
318 }
319 source->string.text.end = source->string.text.p;
320 source->string.buffer.end =
321     source->string.text.p + makefile_name->stat.size;
322 } else {
323     /* Do we ever reach here? */
324     source->fd = -1;
325     source->string.text.p =

```

```

326     source->string.buffer.start =
327     makefile->body.makefile.contents;
328     source->string.text.end =
329     source->string.buffer.end =
330     source->string.text.p + makefile->body.makefile.size
331     source->bytes_left_in_file =
332     makefile->body.makefile.size;
333 }
334 file_being_read = wcb;
335 } else {
336     char          *stdin_text_p;
337     char          *stdin_text_end;
338     char          *stdin_buffer_start;
339     char          *stdin_buffer_end;
340     char          *p_mb;
341     int           num_mb_chars;
342     size_t        num_wc_chars;
343
344     MBSTOWCS(wcs_buffer, "Standard in");
345     makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
346     /*
347      * Memory to read standard in, then convert it
348      * to wide char strings.
349      */
350     stdin_buffer_start =
351         stdin_text_p = getmem(length = 1024);
352     stdin_buffer_end = stdin_text_p + length;
353     MBSTOWCS(wcs_buffer, "standard input");
354     file_being_read = (wchar_t *) wcsdup(wcs_buffer);
355     file_being_read = (wchar_t *) wsdup(wcs_buffer);
356     line_number = 0;
357     while ((n = read(fileno(stdin),
358                    stdin_text_p,
359                    length)) > 0) {
360         length -= n;
361         stdin_text_p += n;
362         if (length == 0) {
363             p_mb = getmem(length = 1024 +
364                          (stdin_buffer_end -
365                           stdin_buffer_start));
366             (void) strncpy(p_mb,
367                          stdin_buffer_start,
368                          (stdin_buffer_end -
369                           stdin_buffer_start));
369             retmem_mb(stdin_buffer_start);
370             stdin_text_p = p_mb +
371                 (stdin_buffer_end - stdin_buffer_start);
372             stdin_buffer_start = p_mb;
373             stdin_buffer_end =
374                 stdin_buffer_start + length;
375             length = 1024;
376         }
377     }
378     if (n < 0) {
379         fatal(gettext("Error reading standard input: %s"),
380              errmsg(errno));
381     }
382     stdin_text_p = stdin_buffer_start;
383     stdin_text_end = stdin_buffer_end - length;
384     num_mb_chars = stdin_text_end - stdin_text_p;
385
386     /*
387      * Now, convert the sequence of multibyte chars into
388      * a sequence of corresponding wide character codes.
389      */
390     source->string.free_after_use = false;

```



```

523     for (WCTOMB(mb_buffer, GET_CHAR());
524          1;
525          source_p++, WCTOMB(mb_buffer, GET_CHAR()))
526         switch (mb_buffer[0]) {
527     */
528     for (char_number=0; 1; source_p++,char_number++) switch (GET_CHAR()) {
529     case nul_char:
530         /* End of this string. Pop it and return to the previous one */
531         GET_NEXT_BLOCK(source);
532         source_p--;
533         if (source == NULL) {
534             GOTO_STATE(on_eoln_state);
535         }
536         break;
537     case newline_char:
538     end_of_line:
539         source_p++;
540         if (source->fd >= 0) {
541             line_number++;
542         }
543         switch (GET_CHAR()) {
544     case nul_char:
545             GET_NEXT_BLOCK(source);
546             if (source == NULL) {
547                 GOTO_STATE(on_eoln_state);
548             }
549             /* Go back to the top of this loop */
550             goto start_new_line;
551     case newline_char:
552     case numbersign_char:
553     case dollar_char:
554     case space_char:
555     case tab_char:
556             /*
557              * Go back to the top of this loop since the
558              * new line does not start with a regular char.
559              */
560             goto start_new_line;
561     default:
562             /* We found the first proper char on the new line */
563             goto start_new_line_no_skip;
564         }
565     case space_char:
566         if (char_number == 0)
567             line_started_with_space=line_number;
568     case tab_char:
569         /* Whitespace. Just keep going in this loop */
570         break;
571     case numbersign_char:
572         /* Comment. Skip over it */
573         for (; 1; source_p++) {
574             switch (GET_CHAR()) {
575     case nul_char:
576                 GET_NEXT_BLOCK_NOCHK(source);
577                 if (source == NULL) {
578                     GOTO_STATE(on_eoln_state);
579                 }
580                 if (source->error_converting) {
581                     /* Illegal byte sequence - skip its first byte
582                      source->inp_buf_ptr++;
583                     */
584                     source_p--;
585                     break;
586                 case backslash_char:
587                     /* Comments can be continued */
588                     if (++source_p == (int) nul_char) {

```

```

589             GET_NEXT_BLOCK_NOCHK(source);
590             if (source == NULL) {
591                 GOTO_STATE(on_eoln_state);
592             }
593             if (source->error_converting) {
594                 /* Illegal byte sequence - skip its first
595                  source->inp_buf_ptr++;
596                  source_p--;
597                  break;
598             */
599             }
600             if (*source_p == (int) newline_char) {
601                 if (source->fd >= 0) {
602                     line_number++;
603                 }
604             }
605             break;
606     case newline_char:
607         /*
608          * After we skip the comment we go to
609          * the end of line handler since end of
610          * line terminates comments.
611          */
612         goto end_of_line;
613     }
614     }
615     case dollar_char:
616         /* Macro reference */
617         if (source->already_expanded) {
618             /*
619              * If we are reading from the expansion of a
620              * macro we already expanded everything enough.
621              */
622             goto start_new_line_no_skip;
623         }
624         /*
625          * Expand the value and push the Source on the stack of
626          * things being read.
627          */
628         source_p++;
629         UNCACHE_SOURCE();
630         {
631             Source t = (Source) alloca((int) sizeof (Source_rec));
632             source = push_macro_value(t,
633                                     buffer,
634                                     sizeof buffer,
635                                     source);
636         }
637         CACHE_SOURCE(1);
638         break;
639     default:
640         /* We found the first proper char on the new line */
641         goto start_new_line_no_skip;
642     }
643     }
644     /*
645     * We found the first normal char (one that starts an identifier)
646     * on the newline.
647     */
648     start_new_line_no_skip:
649     /* Inspect that first char to see if it maybe is special anyway */
650     switch (GET_CHAR()) {
651     case nul_char:
652         GET_NEXT_BLOCK(source);
653         if (source == NULL) {
654             GOTO_STATE(on_eoln_state);

```

```

655     }
656     goto start_new_line_no_skip;
657 case newline_char:
658     /* Just in case */
659     goto start_new_line;
660 case exclam_char:
661     /* Evaluate the line before it is read */
662     string_start = source_p + 1;
663     macro_seen_in_string = false;
664     /* Stuff the line in a string so we can eval it. */
665     for (; l; source_p++) {
666         switch (GET_CHAR()) {
667             case newline_char:
668                 goto eoln_l;
669             case nul_char:
670                 if (source->fd > 0) {
671                     if (!macro_seen_in_string) {
672                         macro_seen_in_string = true;
673                         INIT_STRING_FROM_STACK(
674                             name_string, name_buffer);
675                     }
676                     append_string(string_start,
677                                 &name_string,
678                                 source_p - string_start);
679                     GET_NEXT_BLOCK(source);
680                     string_start = source_p;
681                     source_p--;
682                     break;
683                 }
684             eoln_l:
685                 if (!macro_seen_in_string) {
686                     INIT_STRING_FROM_STACK(name_string,
687                                           name_buffer);
688                 }
689                 append_string(string_start,
690                             &name_string,
691                             source_p - string_start);
692                 extrap = (Source)
693                     alloca((int) sizeof (Source_rec));
694                 extrap->string.buffer.start = NULL;
695                 extrap->inp_buf =
696                     extrap->inp_buf_ptr =
697                     extrap->inp_buf_end = NULL;
698                 extrap->error_converting = false;
699                 if (*source_p == (int) nul_char) {
700                     source_p++;
701                 }
702                 /* Eval the macro */
703                 expand_value(GETNAME(name_string.buffer.start,
704                                   FIND_LENGTH),
705                             &extrap->string,
706                             false);
707                 if (name_string.free_after_use) {
708                     retmem(name_string.buffer.start);
709                 }
710                 UNCACHE_SOURCE();
711                 extrap->string.text.p =
712                     extrap->string.buffer.start;
713                 extrap->fd = -1;
714                 /* And push the value */
715                 extrap->previous = source;
716                 source = extrap;
717                 CACHE_SOURCE(0);
718                 goto line_evald;
719             }
720     }

```

```

721     default:
722         goto line_evald;
723     }
724
725     /* We now have a line we can start reading */
726 line_evald:
727     if (source == NULL) {
728         GOTO_STATE(exit_state);
729     }
730     /* Check if this is an include command */
731     if ((makefile_type == reading_makefile) &&
732         !source->already_expanded) {
733         if (include_space[0] == (int) nul_char) {
734             MBSTOWCS(include_space, "include ");
735             MBSTOWCS(include_tab, "include\t");
736         }
737         if ((IS_WEQUALN(source_p, include_space, 8) ||
738             (IS_WEQUALN(source_p, include_tab, 8)))) {
739             source_p += 7;
740             if (iswspace(*source_p)) {
741                 Makefile_type save_makefile_type;
742                 wchar_t *name_start;
743                 int name_length;
744
745                 /*
746                  * Yes, this is an include.
747                  * Skip spaces to get to the filename.
748                  */
749                 while (iswspace(*source_p) ||
750                       (*source_p == (int) nul_char)) {
751                     switch (GET_CHAR()) {
752                         case nul_char:
753                             GET_NEXT_BLOCK(source);
754                             if (source == NULL) {
755                                 GOTO_STATE(on_eoln_state);
756                             }
757                             break;
758
759                         default:
760                             source_p++;
761                             break;
762                     }
763                 }
764
765                 string_start = source_p;
766                 /* Find the end of the filename */
767                 macro_seen_in_string = false;
768                 while (!iswspace(*source_p) ||
769                       (*source_p == (int) nul_char)) {
770                     switch (GET_CHAR()) {
771                         case nul_char:
772                             if (!macro_seen_in_string) {
773                                 INIT_STRING_FROM_STACK(name_stri
774                                                         name_buff
775                                                         )
776                             }
777                             append_string(string_start,
778                                         &name_string,
779                                         source_p - string_start);
780                             macro_seen_in_string = true;
781                             GET_NEXT_BLOCK(source);
782                             string_start = source_p;
783                             if (source == NULL) {
784                                 GOTO_STATE(on_eoln_state);
785                             }
786                             break;

```

```

787         default:
788             source_p++;
789             break;
790     }
791 }
792
793 source->string.text.p = source_p;
794 if (macro_seen_in_string) {
795     append_string(string_start,
796                 &name_string,
797                 source_p - string_start);
798     name_start = name_string.buffer.start;
799     name_length = name_string.text.p - name_start;
800 } else {
801     name_start = string_start;
802     name_length = source_p - string_start;
803 }
804
805 /* Strip "/" from the head of the name */
806 if ((name_start[0] == (int) period_char) &&
807     (name_start[1] == (int) slash_char)) {
808     name_start += 2;
809     name_length -= 2;
810 }
811 /* if include file name is surrounded by double quotes */
812 if ((name_start[0] == (int) doublequote_char) &&
813     (name_start[name_length - 1] == (int) doublequote_ch
814     name_start += 1;
815     name_length -= 2;
816
817 /* if name does not begin with a slash char */
818 if (name_start[0] != (int) slash_char) {
819     if ((name_start[0] == (int) period_char)
820         (name_start[1] == (int) slash_char))
821         name_start += 2;
822     name_length -= 2;
823 }
824
825 INIT_STRING_FROM_STACK(include_name, inc
826 APPEND_NAME(true_makefile_name,
827             &include_name,
828             true_makefile_name->hash.l
829
830 wchar_t *slash = wcsrchr(include_name.bu
831 wchar_t *slash = wsrchr(include_name.buf
832 if (slash != NULL) {
833     include_name.text.p = slash + 1;
834     append_string(name_start,
835                 &include_name,
836                 name_length);
837
838     name_start = include_name.buffer
839     name_length = include_name.text.
840 }
841 }
842
843 /* Even when we run -n we want to create makefiles */
844 do_not_exec_rule = false;
845 makefile_name = GETNAME(name_start, name_length);
846 if (makefile_name->dollar) {
847     String_rec destination;
848     wchar_t buffer[STRING_BUFFER_LENGTH];
849     wchar_t *p;
850     wchar_t *q;

```

```

852     INIT_STRING_FROM_STACK(destination, buffer);
853     expand_value(makefile_name,
854                 &destination,
855                 false);
856     for (p = destination.buffer.start;
857          (*p != (int) nul_char) && iswspace(*p);
858          p++);
859     for (q = p;
860          (*q != (int) nul_char) && !iswspace(*q);
861          q++);
862     makefile_name = GETNAME(p, q-p);
863     if (destination.free_after_use) {
864         retmem(destination.buffer.start);
865     }
866 }
867 source_p++;
868 UNCACHE_SOURCE();
869 /* Read the file */
870 save_makefile_type = makefile_type;
871 if (read_simple_file(makefile_name,
872                     true,
873                     true,
874                     true,
875                     false,
876                     true,
877                     false) == failed) {
878     fatal_reader(gettext("Read of include file '%s'
879                     makefile_name->string_mb);
880 }
881 makefile_type = save_makefile_type;
882 do_not_exec_rule = save_do_not_exec_rule;
883 CACHE_SOURCE(0);
884 goto start_new_line;
885 } else {
886     source_p -= 7;
887 }
888 } else {
889     /* Check if the word include was split across 8K boundary. */
890
891     tmp_bytes_left_in_string = source->string.text.end - source_p;
892     if (tmp_bytes_left_in_string < 8) {
893         tmp_maybe_include = false;
894         if (IS_WEQUALN(source_p,
895                       include_space,
896                       tmp_bytes_left_in_string)) {
897             tmp_maybe_include = true;
898         }
899         if (tmp_maybe_include) {
900             GET_NEXT_BLOCK(source);
901             tmp_maybe_include = false;
902             goto line_evald;
903         }
904     }
905 }
906 }
907
908 /* Reset the status in preparation for the new line */
909 for (nvp = &target; nvp != NULL; nvp = nvp->next) {
910     nvp->used = 0;
911 }
912 for (nvp = &depes; nvp != NULL; nvp = nvp->next) {
913     nvp->used = 0;
914 }
915 target_group_seen = false;
916 command = command_tail = NULL;
917 macro_value = NULL;

```



```

918     append = false;
919     current_names = &target;
920     SET_STATE(scan_name_state);
921     on_eoln_state = illegal_eoln_state;
922     separator = none_seen;

924     /* The state machine starts here */
925     enter_state:
926     while (1) switch (state) {

928 /******
929 *     Scan name state
930 */
931 case scan_name_state:
932     /* Scan an identifier. We skip over chars until we find a break char */
933     /* First skip white space. */
934     for (; 1; source_p++) switch (GET_CHAR()) {
935     case nul_char:
936         GET_NEXT_BLOCK(source);
937         source_p--;
938         if (source == NULL) {
939             GOTO_STATE(on_eoln_state);
940         }
941         break;
942     case newline_char:
943         /* We found the end of the line. */
944         /* Do postprocessing or return error */
945         source_p++;
946         if (source->fd >= 0) {
947             line_number++;
948         }
949         GOTO_STATE(on_eoln_state);
950     case backslash_char:
951         /* Continuation */
952         if (++source_p == (int) nul_char) {
953             GET_NEXT_BLOCK(source);
954             if (source == NULL) {
955                 GOTO_STATE(on_eoln_state);
956             }
957         }
958         if (*source_p == (int) newline_char) {
959             if (source->fd >= 0) {
960                 line_number++;
961             }
962         } else {
963             source_p--;
964         }
965         break;
966     case tab_char:
967     case space_char:
968         /* Whitespace is skipped */
969         break;
970     case numbersign_char:
971         /* Comment. Skip over it */
972         for (; 1; source_p++) {
973             switch (GET_CHAR()) {
974             case nul_char:
975                 GET_NEXT_BLOCK_NOCHK(source);
976                 if (source == NULL) {
977                     GOTO_STATE(on_eoln_state);
978                 }
979                 if (source->error_converting) {
980                     /* Illegal byte sequence - skip its first byte
981                     source->inp_buf_ptr++;
982                     }
983                     source_p--;

```

```

984         break;
985     case backslash_char:
986         if (++source_p == (int) nul_char) {
987             GET_NEXT_BLOCK_NOCHK(source);
988             if (source == NULL) {
989                 GOTO_STATE(on_eoln_state);
990             }
991             if (source->error_converting) {
992                 /* Illegal byte sequence - skip its first
993                 source->inp_buf_ptr++;
994                 source_p--;
995                 break;
996             }
997         }
998         if (*source_p == (int) newline_char) {
999             if (source->fd >= 0) {
1000                 line_number++;
1001             }
1002         }
1003         break;
1004     case newline_char:
1005         source_p++;
1006         if (source->fd >= 0) {
1007             line_number++;
1008         }
1009         GOTO_STATE(on_eoln_state);
1010     }
1011 }
1012 case dollar_char:
1013     /* Macro reference. Expand and push value */
1014     if (source->already_expanded) {
1015         goto scan_name;
1016     }
1017     source_p++;
1018     UNCACHE_SOURCE();
1019     {
1020         Source t = (Source) alloca((int) sizeof (Source_rec));
1021         source = push_macro_value(t,
1022                                 buffer,
1023                                 sizeof buffer,
1024                                 source);
1025     }
1026     CACHE_SOURCE(1);
1027     break;
1028 default:
1029     /* End of white space */
1030     goto scan_name;
1031 }

1033     /* First proper identifier character */
1034     scan_name:

1036     string_start = source_p;
1037     paren_count = brace_count = 0;
1038     macro_seen_in_string = false;
1039     resume_name_scan:
1040     for (; 1; source_p++) {
1041         switch (GET_CHAR()) {
1042         case nul_char:
1043             /* Save what we have seen so far of the identifier */
1044             if (source_p != string_start) {
1045                 if (!macro_seen_in_string) {
1046                     INIT_STRING_FROM_STACK(name_string,
1047                                             name_buffer);
1048                 }
1049                 append_string(string_start,

```

```

1050             &name_string,
1051             source_p - string_start);
1052             macro_seen_in_string = true;
1053         }
1054         /* Get more text to read */
1055         GET_NEXT_BLOCK(source);
1056         string_start = source_p;
1057         source_p--;
1058         if (source == NULL) {
1059             GOTO_STATE(on_eoln_state);
1060         }
1061         break;
1062     case newline_char:
1063         if (paren_count > 0) {
1064             fatal_reader(gettext("Unmatched '(' on line"));
1065         }
1066         if (brace_count > 0) {
1067             fatal_reader(gettext("Unmatched '{' on line"));
1068         }
1069         source_p++;
1070         /* Enter name */
1071         current_names = enter_name(&name_string,
1072                                 macro_seen_in_string,
1073                                 string_start,
1074                                 source_p - 1,
1075                                 current_names,
1076                                 &extra_names,
1077                                 &target_group_seen);
1078         first_target = false;
1079         if (extra_names == NULL) {
1080             extra_names = (Name_vector)
1081                 alloca((int) sizeof (Name_vector_rec));
1082         }
1083         /* Do postprocessing or return error */
1084         if (source->fd >= 0) {
1085             line_number++;
1086         }
1087         GOTO_STATE(on_eoln_state);
1088     case backslash_char:
1089         /* Check if this is a quoting backslash */
1090         if (!macro_seen_in_string) {
1091             INIT_STRING_FROM_STACK(name_string,
1092                                 name_buffer);
1093             macro_seen_in_string = true;
1094         }
1095         append_string(string_start,
1096                     &name_string,
1097                     source_p - string_start);
1098         if (++source_p == (int) nul_char) {
1099             GET_NEXT_BLOCK(source);
1100             if (source == NULL) {
1101                 GOTO_STATE(on_eoln_state);
1102             }
1103         }
1104         if (*source_p == (int) newline_char) {
1105             if (source->fd >= 0) {
1106                 line_number++;
1107             }
1108             *source_p = (int) space_char;
1109             string_start = source_p;
1110             goto resume_name_scan;
1111         } else {
1112             string_start = source_p;
1113             break;
1114         }
1115         break;

```

```

1116     case numbersign_char:
1117         if (paren_count + brace_count > 0) {
1118             break;
1119         }
1120         fatal_reader(gettext("Unexpected comment seen"));
1121     case dollar_char:
1122         if (source->already_expanded) {
1123             break;
1124         }
1125         /* Save the identifier so far */
1126         if (source_p != string_start) {
1127             if (!macro_seen_in_string) {
1128                 INIT_STRING_FROM_STACK(name_string,
1129                                     name_buffer);
1130             }
1131             append_string(string_start,
1132                         &name_string,
1133                         source_p - string_start);
1134             macro_seen_in_string = true;
1135         }
1136         /* Eval and push the macro */
1137         source_p++;
1138         UNCACHE_SOURCE();
1139         {
1140             Source t =
1141                 (Source) alloca((int) sizeof (Source_rec));
1142             source = push_macro_value(t,
1143                                     buffer,
1144                                     sizeof buffer,
1145                                     source);
1146         }
1147         CACHE_SOURCE(1);
1148         string_start = source_p + 1;
1149         break;
1150     case parenleft_char:
1151         paren_count++;
1152         break;
1153     case parenright_char:
1154         if (--paren_count < 0) {
1155             fatal_reader(gettext("Unmatched ')' on line"));
1156         }
1157         break;
1158     case braceleft_char:
1159         brace_count++;
1160         break;
1161     case braceright_char:
1162         if (--brace_count < 0) {
1163             fatal_reader(gettext("Unmatched '}' on line"));
1164         }
1165         break;
1166     case ampersand_char:
1167     case greater_char:
1168     case bar_char:
1169         if (paren_count + brace_count == 0) {
1170             source_p++;
1171         }
1172         /* Fall into */
1173     case tab_char:
1174     case space_char:
1175         if (paren_count + brace_count > 0) {
1176             break;
1177         }
1178         current_names = enter_name(&name_string,
1179                                 macro_seen_in_string,
1180                                 string_start,
1181                                 source_p,

```

```

1182         current_names,
1183         &extra_names,
1184         &target_group_seen);
1185     first_target = false;
1186     if (extra_names == NULL) {
1187         extra_names = (Name_vector)
1188             alloca((int) sizeof (Name_vector_rec));
1189     }
1190     goto enter_state;
1191 case colon_char:
1192     if (paren_count + brace_count > 0) {
1193         break;
1194     }
1195     if (separator == conditional_seen) {
1196         break;
1197     }
1198     /** POSIX **/
1199     #if 0
1200     if(posix) {
1201         emptycount = 0;
1202     }
1203     #endif
1204     /** END POSIX **/
1205     /* End of the target list. We now start reading */
1206     /* dependencies or a conditional assignment */
1207     if (separator != none_seen) {
1208         fatal_reader(gettext("Extra ':', '::', or ':=' o
1209     })
1210     /* Enter the last target */
1211     if ((string_start != source_p) ||
1212         macro_seen_in_string) {
1213         current_names =
1214             enter_name(&name_string,
1215                     macro_seen_in_string,
1216                     string_start,
1217                     source_p,
1218                     current_names,
1219                     &extra_names,
1220                     &target_group_seen);
1221         first_target = false;
1222         if (extra_names == NULL) {
1223             extra_names = (Name_vector)
1224                 alloca((int)
1225                     sizeof (Name_vector_rec));
1226         }
1227     }
1228     /* Check if it is ":" "::" or "!=" */
1229     scan_colon_label:
1230     switch (++source_p) {
1231     case nul_char:
1232         GET_NEXT_BLOCK(source);
1233         source_p--;
1234         if (source == NULL) {
1235             GOTO_STATE(enter_dependencies_state);
1236         }
1237         goto scan_colon_label;
1238     case equal_char:
1239         if(svr4) {
1240             fatal_reader(gettext("syntax error"));
1241         }
1242         separator = conditional_seen;
1243         source_p++;
1244         current_names = &depes;
1245         GOTO_STATE(scan_name_state);
1246     case colon_char:
1247         separator = two_colon;

```

```

1248         source_p++;
1249         break;
1250     default:
1251         separator = one_colon;
1252     }
1253     current_names = &depes;
1254     on_eoln_state = enter_dependencies_state;
1255     GOTO_STATE(scan_name_state);
1256 case semicolon_char:
1257     if (paren_count + brace_count > 0) {
1258         break;
1259     }
1260     /* End of reading names. Start reading the rule */
1261     if ((separator != one_colon) &&
1262         (separator != two_colon)) {
1263         fatal_reader(gettext("Unexpected command seen"));
1264     }
1265     /* Enter the last dependency */
1266     if ((string_start != source_p) ||
1267         macro_seen_in_string) {
1268         current_names =
1269             enter_name(&name_string,
1270                     macro_seen_in_string,
1271                     string_start,
1272                     source_p,
1273                     current_names,
1274                     &extra_names,
1275                     &target_group_seen);
1276     }
1277     first_target = false;
1278     if (extra_names == NULL) {
1279         extra_names = (Name_vector)
1280             alloca((int)
1281                 sizeof (Name_vector_rec));
1282     }
1283     }
1284     source_p++;
1285     /* Make sure to enter a rule even if the is */
1286     /* no text here */
1287     command = command_tail = ALLOC(Cmd_line);
1288     command->next = NULL;
1289     command->command_line = empty_name;
1290     command->make_refd = false;
1291     command->ignore_command_dependency = false;
1292     command->assign = false;
1293     command->ignore_error = false;
1294     command->silent = false;
1295     GOTO_STATE(scan_command_state);
1296 case plus_char:
1297     /*
1298     ** following code drops the target separator plus char i
1299     ** a line.
1300     */
1301     if(first_target && !macro_seen_in_string &&
1302         source_p == string_start) {
1303         for (; l; source_p++)
1304             switch (GET_CHAR()) {
1305             case nul_char:
1306                 if (source_p != string_start) {
1307                     if (!macro_seen_in_string) {
1308                         INIT_STRING_FROM_STACK(n
1309                             n
1310                         )
1311                     }
1312                     append_string(string_start,
1313                                 &name_string,
1314                                 source_p - string_

```

```

1314         macro_seen_in_string = true;
1315     }
1316     GET_NEXT_BLOCK(source);
1317     string_start = source_p;
1318     source_p--;
1319     if (source == NULL) {
1320         GOTO_STATE(on_eoln_state);
1321     }
1322     break;
1323 case plus_char:
1324     source_p++;
1325     while (*source_p == (int) nul_char) {
1326         if (source_p != string_start) {
1327             if (!macro_seen_in_string)
1328                 INIT_STRING_FROM
1329                     n
1330             }
1331             append_string(string_start,
1332                 &name_string,
1333                 source_p -
1334                 macro_seen_in_string = true;
1335             }
1336             GET_NEXT_BLOCK(source);
1337             string_start = source_p;
1338             if (source == NULL) {
1339                 GOTO_STATE(on_eoln_state);
1340             }
1341         }
1342         if (*source_p == (int) tab_char ||
1343             *source_p == (int) space)
1344             macro_seen_in_string = false;
1345         string_start = source_p + 1;
1346     } else {
1347         goto resume_name_scan;
1348     }
1349     break;
1350 case tab_char:
1351 case space_char:
1352     string_start = source_p + 1;
1353     break;
1354 default:
1355     goto resume_name_scan;
1356 }
1357 }
1358 if (paren_count + brace_count > 0) {
1359     break;
1360 }
1361 /* We found "+" construct */
1362 if (source_p != string_start) {
1363     /* "+" is not a break char. */
1364     /* Ignore it if it is part of an identifier */
1365     source_p++;
1366     goto resume_name_scan;
1367 }
1368 /* Make sure the "+" is followed by a "=" */
1369 scan_append:
1370 switch (*++source_p) {
1371 case nul_char:
1372     if (!macro_seen_in_string) {
1373         INIT_STRING_FROM_STACK(name_string,
1374             name_buffer);
1375     }
1376     append_string(string_start,
1377         &name_string,
1378         source_p - string_start);
1379     GET_NEXT_BLOCK(source);

```

```

1380     source_p--;
1381     string_start = source_p;
1382     if (source == NULL) {
1383         GOTO_STATE(illegal_eoln_state);
1384     }
1385     goto scan_append;
1386 case equal_char:
1387     if (!svr4) {
1388         append = true;
1389     } else {
1390         fatal_reader(gettext("Must be a separator on r
1391     });
1392     break;
1393 default:
1394     /* The "+" just starts a regular name. */
1395     /* Start reading that name */
1396     goto resume_name_scan;
1397 }
1398 /* Fall into */
1399 case equal_char:
1400     if (paren_count + brace_count > 0) {
1401         break;
1402     }
1403     /* We found macro assignment. */
1404     /* Check if it is legal and if it is appending */
1405     switch (separator) {
1406     case none_seen:
1407         separator = equal_seen;
1408         on_eoln_state = enter_equal_state;
1409         break;
1410     case conditional_seen:
1411         on_eoln_state = enter_conditional_state;
1412         break;
1413     default:
1414         /* Reader must special check for "MACRO:sh=" */
1415         /* notation */
1416         if (sh_name == NULL) {
1417             MBSTOWCS(wcs_buffer, "sh");
1418             sh_name = GETNAME(wcs_buffer, FIND_LENGTH);
1419             MBSTOWCS(wcs_buffer, "shell");
1420             shell_name = GETNAME(wcs_buffer, FIND_LENGTH);
1421         }
1422     }
1423     if (!macro_seen_in_string) {
1424         INIT_STRING_FROM_STACK(name_string,
1425             name_buffer);
1426     }
1427     append_string(string_start,
1428         &name_string,
1429         source_p - string_start);
1430 }
1431 }
1432 if ( (((target.used == 1) &&
1433     (depes.used == 1) &&
1434     (depes.names[0] == sh_name)) ||
1435     ((target.used == 1) &&
1436     (depes.used == 0) &&
1437     (separator == one_colon) &&
1438     (GETNAME(name_string.buffer.start, FIND_LENGTH)
1439     (!svr4)) {
1440     String_rec      macro_name;
1441     wchar_t         buffer[100];
1442 }
1443     INIT_STRING_FROM_STACK(macro_name,
1444         buffer);
1445     APPEND_NAME(target.names[0],

```

```

1446             &macro_name,
1447             FIND_LENGTH);
1448     append_char((int) colon_char,
1449                &macro_name);
1450     APPEND_NAME(sh_name,
1451                &macro_name,
1452                FIND_LENGTH);
1453     target.names[0] =
1454         GETNAME(macro_name.buffer.start,
1455                FIND_LENGTH);
1456     separator = equal_seen;
1457     on_eoln_state = enter_equal_state;
1458     break;
1459 } else if ( ((target.used == 1) &&
1460            (depes.used == 1) &&
1461            (depes.names[0] == shell_name)) ||
1462            ((target.used == 1) &&
1463            (depes.used == 0) &&
1464            (separator == one_colon) &&
1465            (GETNAME(name_string.buffer.start,FI
1466            (!svr4)) {
1467     String_rec    macro_name;
1468     wchar_t      buffer[100];

1470     INIT_STRING_FROM_STACK(macro_name,
1471                            buffer);
1472     APPEND_NAME(target.names[0],
1473                &macro_name,
1474                FIND_LENGTH);
1475     append_char((int) colon_char,
1476                &macro_name);
1477     APPEND_NAME(shell_name,
1478                &macro_name,
1479                FIND_LENGTH);
1480     target.names[0] =
1481         GETNAME(macro_name.buffer.start,
1482                FIND_LENGTH);
1483     separator = equal_seen;
1484     on_eoln_state = enter_equal_state;
1485     break;
1486 }
1487 if(svr4) {
1488     fatal_reader(gettext("syntax error"));
1489 }
1490 else {
1491     fatal_reader(gettext("Macro assignment on depe
1492     });
1493 }
1494 if (append) {
1495     source_p--;
1496 }
1497 /* Enter the macro name */
1498 if ((string_start != source_p) ||
1499     macro_seen_in_string) {
1500     current_names =
1501         enter_name(&name_string,
1502                  macro_seen_in_string,
1503                  string_start,
1504                  source_p,
1505                  current_names,
1506                  &extra_names,
1507                  &target_group_seen);
1508     first_target = false;
1509     if (extra_names == NULL) {
1510         extra_names = (Name_vector)
1511             alloca((int)

```

```

1512             sizeof (Name_vector_rec));
1513     }
1514     }
1515     if (append) {
1516         source_p++;
1517     }
1518     macro_value = NULL;
1519     source_p++;
1520     distance = 0;
1521     /* Skip whitespace to the start of the value */
1522     macro_seen_in_string = false;
1523     for (; 1; source_p++) {
1524         switch (GET_CHAR()) {
1525             case nul_char:
1526                 GET_NEXT_BLOCK(source);
1527                 source_p--;
1528                 if (source == NULL) {
1529                     GOTO_STATE(on_eoln_state);
1530                 }
1531                 break;
1532             case backslash_char:
1533                 if (++source_p == (int) nul_char) {
1534                     GET_NEXT_BLOCK(source);
1535                     if (source == NULL) {
1536                         GOTO_STATE(on_eoln_state)
1537                     }
1538                 }
1539                 if (*source_p != (int) newline_char) {
1540                     if (!macro_seen_in_string) {
1541                         macro_seen_in_string =
1542                             true;
1543                         INIT_STRING_FROM_STACK(n
1544                             n
1545                         )
1546                     }
1547                     append_char((int)
1548                                 backslash_char,
1549                                 &name_string);
1550                     append_char(*source_p,
1551                                 &name_string);
1552                     string_start = source_p+1;
1553                     goto macro_value_start;
1554                 } else {
1555                     if (source->fd >= 0) {
1556                         line_number++;
1557                     }
1558                     break;
1559                 }
1560             case newline_char:
1561             case numbersign_char:
1562                 string_start = source_p;
1563                 goto macro_value_end;
1564             case tab_char:
1565             case space_char:
1566                 break;
1567             default:
1568                 string_start = source_p;
1569                 goto macro_value_start;
1570         }
1571     }
1572     macro_value_start:
1573     /* Find the end of the value */
1574     for (; 1; source_p++) {
1575         if (distance != 0) {
1576             *source_p = *(source_p + distance);
1577         }
1578         switch (GET_CHAR()) {

```





```

1842     GET_NEXT_BLOCK(source);
1843     if (source == NULL) {
1844         string_start = source_p;
1845         goto command_newline;
1846     }
1847 }
1848 append_char((int) backslash_char, &name_string);
1849 append_char(*source_p, &name_string);
1850 if (*source_p == (int) newline_char) {
1851     if (source->fd >= 0) {
1852         line_number++;
1853     }
1854     if (++source_p == (int) nul_char) {
1855         GET_NEXT_BLOCK(source);
1856         if (source == NULL) {
1857             string_start = source_p;
1858             goto command_newline;
1859         }
1860     }
1861     if (*source_p == (int) tab_char) {
1862         source_p++;
1863     }
1864 } else {
1865     if (++source_p == (int) nul_char) {
1866         GET_NEXT_BLOCK(source);
1867         if (source == NULL) {
1868             string_start = source_p;
1869             goto command_newline;
1870         }
1871     }
1872 }
1873 string_start = source_p;
1874 if ((*source_p == (int) newline_char) ||
1875     (*source_p == (int) backslash_char) ||
1876     (*source_p == (int) nul_char)) {
1877     source_p--;
1878 }
1879 break;
1880 case newline_char:
1881 command_newline:
1882     if ((string_start != source_p) ||
1883         macro_seen_in_string) {
1884         if (macro_seen_in_string) {
1885             append_string(string_start,
1886                          &name_string,
1887                          source_p - string_start);
1888             string_start =
1889                 name_string.buffer.start;
1890             string_end = name_string.text.p;
1891         } else {
1892             string_end = source_p;
1893         }
1894         while ((*string_start != (int) newline_char) &&
1895             iswspace(*string_start)){
1896             string_start++;
1897         }
1898         if ((string_end > string_start) ||
1899             (makefile_type == reading_statefile)) {
1900             if (command_tail == NULL) {
1901                 command =
1902                     command_tail =
1903                         ALLOC(Cmd_line);
1904             } else {
1905                 command_tail->next =
1906                     ALLOC(Cmd_line);
1907                 command_tail =

```

```

1908         command_tail->next;
1909     }
1910     command_tail->next = NULL;
1911     command_tail->make_refd = false;
1912     command_tail->ignore_command_dependency
1913     command_tail->assign = false;
1914     command_tail->ignore_error = false;
1915     command_tail->silent = false;
1916     command_tail->command_line =
1917         GETNAME(string_start,
1918                string_end - string_start);
1919     if (macro_seen_in_string &&
1920         name_string.free_after_use) {
1921         retmem(name_string.
1922               buffer.start);
1923     }
1924 }
1925 }
1926 do {
1927     if ((source != NULL) && (source->fd >= 0)) {
1928         line_number++;
1929     }
1930     if ((source != NULL) &&
1931         (++source_p == (int) nul_char)) {
1932         GET_NEXT_BLOCK(source);
1933         if (source == NULL) {
1934             GOTO_STATE(on_eoln_state);
1935         }
1936     }
1937 } while (*source_p == (int) newline_char);
1938 }
1939 GOTO_STATE(enter_dependencies_state);
1940 case nul_char:
1941     if (!macro_seen_in_string) {
1942         INIT_STRING_FROM_STACK(name_string,
1943                                name_buffer);
1944     }
1945     append_string(string_start,
1946                  &name_string,
1947                  source_p - string_start);
1948     macro_seen_in_string = true;
1949     GET_NEXT_BLOCK(source);
1950     string_start = source_p;
1951     source_p--;
1952     if (source == NULL) {
1953         GOTO_STATE(enter_dependencies_state);
1954     }
1955     break;
1956 }
1957 }
1958
1959 /*****
1960 *   enter equal state
1961 */
1962 case enter_equal_state:
1963     if (target.used != 1) {
1964         GOTO_STATE(poorly_formed_macro_state);
1965     }
1966     enter_equal(target.names[0], macro_value, append);
1967     goto start_new_line;
1968
1969 /*****
1970 *   enter conditional state
1971 */
1972 case enter_conditional_state:
1973     if (depes.used != 1) {

```



```
1974         GOTO_STATE(poorly_formed_macro_state);
1975     }
1976     for (nvp = &target; nvp != NULL; nvp = nvp->next) {
1977         for (i = 0; i < nvp->used; i++) {
1978             enter_conditional(nvp->names[i],
1979                             depes.names[0],
1980                             macro_value,
1981                             append);
1982         }
1983     }
1984     goto start_new_line;

1986 /*****
1987  *      Error states
1988  */
1989 case illegal_bytes_state:
1990     fatal_reader(gettext("Invalid byte sequence"));
1991 case illegal_eoln_state:
1992     if (line_number > 1) {
1993         if (line_started_with_space == (line_number - 1)) {
1994             line_number--;
1995             fatal_reader(gettext("Unexpected end of line seen\n\t***
1996
1997         }
1998         fatal_reader(gettext("Unexpected end of line seen"));
1999 case poorly_formed_macro_state:
2000     fatal_reader(gettext("Badly formed macro assignment"));
2001 case exit_state:
2002     return;
2003 default:
2004     fatal_reader(gettext("Internal error. Unknown reader state"));
2005 }
_____unchanged_portion_omitted_
```

```

*****
51209 Fri May 22 11:19:46 2015
new/usr/src/cmd/make/bin/read2.cc
make: use the more modern wchar routines, not widec.h
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  *      read.c
28  *
29  *      This file contains the makefile reader.
30  */

32 /*
33  * Included files
34  */
35 #include <mk/defs.h>
36 #include <mksh/dosys.h>          /* sh_command2string() */
37 #include <mksh/macro.h>         /* expand_value() */
38 #include <mksh/misc.h>         /* retmem() */
39 #include <stdarg.h>             /* va_list, va_start(), va_end() */
40 #include <libintl.h>

42 /*
43  * Defined macros
44  */

46 /*
47  * typedefs & structs
48  */

50 /*
51  * Static variables
52  */
53 static Boolean      built_last_make_run_seen;

55 /*
56  * File table of contents
57  */
58 static Name_vector  enter_member_name(register wchar_t *lib_start, register
59 extern Name         normalize_name(register wchar_t *name_string, register i
60 static void         read_suffixes_list(register Name_vector depes);
61 static void         make_relative(wchar_t *to, wchar_t *result);

```

```

62 static void         print_rule(register Cmd_line command);
63 static void         sh_transform(Name *name, Name *value);

66 /*
67  *      enter_name(string, tail_present, string_start, string_end,
68  *      current_names, extra_names, target_group_seen)
69  *
70  *      Take one string and enter it as a name. The string is passed in
71  *      two parts. A make string and possibly a C string to append to it.
72  *      The result is stuffed in the vector current_names.
73  *      extra_names points to a vector that is used if current_names overflows.
74  *      This is allocad in the calling routine.
75  *      Here we handle the "lib.a[members]" notation.
76  *
77  *      Return value:
78  *
79  *          The name vector that was used
80  *
81  *      Parameters:
82  *          tail_present    Indicates if both C and make string was passed
83  *          string_start    C string
84  *          string_end      Pointer to char after last in C string
85  *          string          make style string with head of name
86  *          current_names   Vector to deposit the name in
87  *          extra_names     Where to get next name vector if we run out
88  *          target_group_seen Pointer to boolean that is set if "+" is seen
89  *
90  *      Global variables used:
91  *          makefile_type   When we read a report file we normalize paths
92  *          plus            Points to the Name "+"
93  */

94 Name_vector
95 enter_name(String string, Boolean tail_present, register wchar_t *string_start,
96 {
97     Name          name;
98     register wchar_t *cp;
99     wchar_t       ch;

101     /* If we were passed a separate tail of the name we append it to the */
102     /* make string with the rest of it */
103     if (tail_present) {
104         append_string(string_start, string, string_end - string_start);
105         string_start = string->buffer.start;
106         string_end = string->text.p;
107     }
108     ch = *string_end;
109     *string_end = (int) nul_char;
110     /*
111     * Check if there are any ( or [ that are not prefixed with $.
112     * If there are, we have to deal with the lib.a(members) format.
113     */
114     for (cp = (wchar_t *) wcschr(string_start, (int) parenleft_char);
115          cp != NULL;
116          cp = (wchar_t *) wcschr(cp + 1, (int) parenleft_char)) {
117         cp = (wchar_t *) wschr(cp + 1, (int) parenleft_char) {
118             if (*(cp - 1) != (int) dollar_char) {
119                 *string_end = ch;
120                 return enter_member_name(string_start,
121                                         cp,
122                                         string_end,
123                                         current_names,
124                                         extra_names);
125             }
126         }

```

```

126     *string_end = ch;
128     if (makefile_type == reading_cpp_file) {
129         /* Remove extra ../ constructs if we are reading from a report f
130         name = normalize_name(string_start, string_end - string_start);
131     } else {
132         /*
133         * /tolik, fix bug 1197477/
134         * Normalize every target name before entering.
135         * ../obj/a.o and ../obj//a.o are not two different targets.
136         * There is only one target ../obj/a.o
137         */
138         /*name = GETNAME(string_start, string_end - string_start);*/
139         name = normalize_name(string_start, string_end - string_start);
140     }
142     /* Internalize the name. Detect the name "+" (target group here) */
143     if(current_names->used != 0 && current_names->names[current_names->used-1] == pl
144         if(name == plus) {
145             return current_names;
146         }
147 }

```

unchanged portion omitted

```

172 /*
173 *     enter_member_name(lib_start, member_start, string_end,
174 *     current_names, extra_names)
175 *
176 *     A string has been found to contain member names.
177 *     (The "lib.a[members]" and "lib.a(members)" notation)
178 *     Handle it pretty much as enter_name() does for simple names.
179 *
180 *     Return value:
181 *
182 *         The name vector that was used
183 *
184 *     Parameters:
185 *         lib_start     Points to the of start of "lib.a(member.o)"
186 *         member_start Points to "member.o" from above string.
187 *         string_end   Points to char after last of above string.
188 *         current_names Vector to deposit the name in
189 *         extra_names  Where to get next name vector if we run out
190 *
191 *     Global variables used:
192 */
192 static Name_vector
193 enter_member_name(register wchar_t *lib_start, register wchar_t *member_start, r
194 {
195     register Boolean     entry = false;
196     wchar_t             buffer[STRING_BUFFER_LENGTH];
197     Name                lib;
198     Name                member;
199     Name                name;
200     Property            prop;
201     wchar_t             *memberp;
202     wchar_t             *q;
203     register int        paren_count;
204     register Boolean    has_dollar;
205     register wchar_t   *cq;
206     Name                long_member_name = NULL;
208     /* Internalize the name of the library */
209     lib = GETNAME(lib_start, member_start - lib_start);
210     lib->is_member = true;
211     member_start++;
212     if (*member_start == (int) parenleft_char) {
213         /* This is really the "lib.a((entries))" format */

```

```

214         entry = true;
215         member_start++;
216     }
217     /* Move the library name to the buffer where we intend to build the */
218     /* "lib.a(member)" for each member */
219     (void) wcsncpy(buffer, lib_start, member_start - lib_start);
219     (void) wsncpy(buffer, lib_start, member_start - lib_start);
220     memberp = buffer + (member_start-lib_start);
221     while (1) {
222         long_member_name = NULL;
223         /* Skip leading spaces */
224         for (;
225             (member_start < string_end) && iswspace(*member_start);
226             member_start++);
227         /* Find the end of the member name. Allow nested (). Detect $*/
228         for (cq = memberp, has_dollar = false, paren_count = 0;
229             (member_start < string_end) &&
230             ((*member_start != (int) parenright_char) ||
231              (paren_count > 0)) &&
232             !iswspace(*member_start);
233             *cq++ = *member_start++) {
234             switch (*member_start) {
235                 case parenleft_char:
236                     paren_count++;
237                     break;
238                 case parenright_char:
239                     paren_count--;
240                     break;
241                 case dollar_char:
242                     has_dollar = true;
243             }
244         }
245         /* Internalize the member name */
246         member = GETNAME(memberp, cq - memberp);
247         *cq = 0;
248         if ((q = (wchar_t *) wcsrchr(memberp, (int) slash_char)) == NULL
249             if ((q = (wchar_t *) wsrchr(memberp, (int) slash_char)) == NULL)
250             q = memberp;
251         if ((cq - q > (int) ar_member_name_len) &&
252             !has_dollar) {
253             *cq++ = (int) parenright_char;
254             if (entry) {
255                 *cq++ = (int) parenright_char;
256             }
257             long_member_name = GETNAME(buffer, cq - buffer);
258             cq = q + (int) ar_member_name_len;
259         }
260         *cq++ = (int) parenright_char;
261         if (entry) {
262             *cq++ = (int) parenright_char;
263         }
264         /* Internalize the "lib.a(member)" notation for this member */
265         name = GETNAME(buffer, cq - buffer);
266         name->is_member = lib->is_member;
267         if (long_member_name != NULL) {
268             prop = append_prop(name, long_member_name_prop);
269             name->has_long_member_name = true;
270             prop->body.long_member_name.member_name =
271                 long_member_name;
272         }
273         /* And add the member prop */
274         prop = append_prop(name, member_prop);
275         prop->body.member.library = lib;
276         if (entry) {
277             /* "lib.a((entry))" notation */

```

```

278         prop->body.member.entry = member;
279         prop->body.member.member = NULL;
280     } else {
281         /* "lib.a(member)" Notation */
282         prop->body.member.entry = NULL;
283         prop->body.member.member = member;
284     }
285     /* Handle overflow of current_names */
286     if (current_names->used == VSIZEOF(current_names->names)) {
287         if (current_names->next != NULL) {
288             current_names = current_names->next;
289         } else {
290             if (*extra_names == NULL) {
291                 current_names =
292                     current_names->next =
293                         ALLOC(Name_vector);
294             } else {
295                 current_names =
296                     current_names->next =
297                         *extra_names;
298                 *extra_names = NULL;
299             }
300             current_names->used = 0;
301             current_names->next = NULL;
302         }
303     }
304     current_names->target_group[current_names->used] = NULL;
305     current_names->names[current_names->used++] = name;
306     while (iswspace(*member_start)) {
307         member_start++;
308     }
309     /* Check if there are more members */
310     if ((*member_start == (int) parenright_char) ||
311         (member_start >= string_end)) {
312         return current_names;
313     }
314 }
315 /* NOTREACHED */
316 }

318 /*
319 * normalize_name(name_string, length)
320 *
321 * Take a namestring and remove redundant ../, // and ./ constructs
322 *
323 * Return value:
324 *             The normalized name
325 *
326 * Parameters:
327 *   name_string   Path string to normalize
328 *   length        Length of that string
329 *
330 * Global variables used:
331 *   dot           The Name ".", compared against
332 *   dotdot       The Name "..", compared against
333 */
334 Name
335 normalize_name(register wchar_t *name_string, register int length)
336 {
337     static Name dotdot;
338     register wchar_t *string = ALLOC_WC(length + 1);
339     register wchar_t *string2;
340     register wchar_t *cdp;
341     wchar_t *current_component;
342     Name name;
343     register int count;

```

```

345     if (dotdot == NULL) {
346         MBSTOWCS(wcs_buffer, "..");
347         dotdot = GETNAME(wcs_buffer, FIND_LENGTH);
348     }
349
350     /*
351     * Copy string removing ./ and //.
352     * First strip leading ./
353     */
354     while ((length > 1) &&
355           (name_string[0] == (int) period_char) &&
356           (name_string[1] == (int) slash_char)) {
357         name_string += 2;
358         length -= 2;
359         while ((length > 0) && (name_string[0] == (int) slash_char)) {
360             name_string++;
361             length--;
362         }
363     }
364     /* Then copy the rest of the string removing ../ & // */
365     cdp = string;
366     while (length > 0) {
367         if ((length > 2) &&
368             (name_string[0] == (int) slash_char) &&
369             (name_string[1] == (int) period_char) &&
370             (name_string[2] == (int) slash_char)) ||
371             ((length == 2) &&
372             (name_string[0] == (int) slash_char) &&
373             (name_string[1] == (int) period_char))) {
374             name_string += 2;
375             length -= 2;
376             continue;
377         }
378         if ((length > 1) &&
379             (name_string[0] == (int) slash_char) &&
380             (name_string[1] == (int) slash_char)) {
381             name_string++;
382             length--;
383             continue;
384         }
385         *cdp++ = *name_string++;
386         length--;
387     }
388     *cdp = (int) nul_char;
389     /*
390     * Now scan for <name>/../ and remove such combinations iff <name>
391     * is not another ..
392     * Each time something is removed, the whole process is restarted.
393     */
394     removed_one:
395     name_string = string;
396     string2 = name_string;          /*save for free*/
397     current_component =
398         cdp =
399         string =
400         ALLOC_WC((length = wcslen(name_string)) + 1);
401     ALLOC_WC((length = wslen(name_string)) + 1);
402     while (length > 0) {
403         if ((length > 3) &&
404             (name_string[0] == (int) slash_char) &&
405             (name_string[1] == (int) period_char) &&
406             (name_string[2] == (int) period_char) &&
407             (name_string[3] == (int) slash_char)) ||
408             ((length == 3) &&
409             (name_string[0] == (int) slash_char) &&

```

```

409     (name_string[1] == (int) period_char) &&
410     (name_string[2] == (int) period_char)) {
411     /* Positioned on the / that starts a ../ sequence */
412     if (((count = cdp - current_component) != 0) &&
413         (exists(name = GETNAME(string, cdp - string)) > file
414          (!name->stat.is_sym_link)) {
415         name = GETNAME(current_component, count);
416         if(name != dotdot) {
417             cdp = current_component;
418             name_string += 3;
419             length -= 3;
420             if (length > 0) {
421                 name_string++; /* skip slash */
422                 length--;
423                 while (length > 0) {
424                     *cdp++ = *name_string++;
425                     length--;
426                 }
427             }
428             *cdp = (int) nul_char;
429             retmem(string2);
430             goto removed_one;
431         }
432     }
433     }
434     if ((*cdp++ = *name_string++) == (int) slash_char) {
435         current_component = cdp;
436     }
437     length--;
438 }
439 *cdp = (int) nul_char;
440 if (string[0] == (int) nul_char) {
441     name = dot;
442 } else {
443     name = GETNAME(string, FIND_LENGTH);
444 }
445 retmem(string);
446 retmem(string2);
447 return name;
448 }

```

unchanged\_portion\_omitted

```

538 /*
539 *   enter_dependencies(target, target_group, depes, command, separator)
540 *
541 *   Take one target and a list of dependencies and process the whole thing.
542 *   The target might be special in some sense in which case that is handled
543 *
544 *   Parameters:
545 *       target           The target we want to enter
546 *       target_group    Non-NULL if target is part of a group this time
547 *       depes           A list of dependencies for the target
548 *       command         The command the target should be entered with
549 *       separator       Indicates if this is a ":" or a "::" rule
550 *
551 *   Static variables used:
552 *       built_last_make_run_seen If the previous target was
553 *                               .BUILT_LAST_MAKE_RUN we say to rewrite
554 *                               the state file later on
555 *
556 *   Global variables used:
557 *       command_changed Set to indicate if .make.state needs rewriting
558 *       default_target_to_build Set to the target if reading makefile
559 *                               and this is the first regular target
560 *       force           The Name " FORCE", used with "::" targets
561 *       makefile_type   We do different things for makefile vs. report

```

```

562 *           not_auto       The Name ".NOT_AUTO", compared against
563 *           recursive_name The Name ".RECURSIVE", compared against
564 *           temp_file_number Used to figure out when to clear stale
565 *                           automatic dependencies
566 *           trace_reader   Indicates that we should echo stuff we read
567 */
568 void
569 enter_dependencies(register Name target, Chain target_group, register Name_vecto
570 {
571     register int         i;
572     register Property    line;
573     Name                 name;
574     Name                 directory;
575     wchar_t              *namep;
576     char                 *mb_namep;
577     Dependency           dp;
578     Dependency           *dpp;
579     Property             line2;
580     wchar_t              relative[MAXPATHLEN];
581     register int         recursive_state;
582     Boolean               register_as_auto;
583     Boolean               not_auto_found;
584     char                 *slash;
585     Wstring               depstr;
586
587     /* Check if this is a .RECURSIVE line */
588     if ((depes->used >= 3) &&
589         (depes->names[0] == recursive_name)) {
590         target->has_recursive_dependency = true;
591         depes->names[0] = NULL;
592         recursive_state = 0;
593         dp = NULL;
594         dpp = &dp;
595         /* Read the dependencies. They are "<directory> <target-made>*/
596         /* <makefile>*" */
597         for (; depes != NULL; depes = depes->next) {
598             for (i = 0; i < depes->used; i++) {
599                 if (depes->names[i] != NULL) {
600                     switch (recursive_state++) {
601                         case 0: /* Directory */
602                             {
603                                 depstr.init(depes->names[i]);
604                                 make_relative(depstr.get_string(
605                                     relative);
606
607                                 directory =
608                                     GETNAME(relative,
609                                         FIND_LENGTH);
610                             }
611                             break;
612                         case 1: /* Target */
613                             name = depes->names[i];
614                             break;
615                         default: /* Makefiles */
616                             *dpp = ALLOC(Dependency);
617                             (*dpp)->next = NULL;
618                             (*dpp)->name = depes->names[i];
619                             (*dpp)->automatic = false;
620                             (*dpp)->stale = false;
621                             (*dpp)->built = false;
622                             dpp = &((*dpp)->next);
623                             break;
624                     }
625                 }
626             }
627         }
628     }
629     /* Check if this recursion already has been reported else */

```

```

628     /* enter the recursive prop for the target */
629     /* The has_built flag is used to tell if this .RECURSIVE */
630     /* was discovered from this run (read from a tmp file) */
631     /* or was from discovered from the original .make.state */
632     /* file */
633     for (line = get_prop(target->prop, recursive_prop);
634          line != NULL;
635          line = get_prop(line->next, recursive_prop)) {
636         if ((line->body.recursive.directory == directory) &&
637             (line->body.recursive.target == name)) {
638             line->body.recursive.makefiles = dp;
639             line->body.recursive.has_built =
640                 (Boolean)
641                 (makefile_type == reading_cpp_file);
642             return;
643         }
644     }
645     line2 = append_prop(target, recursive_prop);
646     line2->body.recursive.directory = directory;
647     line2->body.recursive.target = name;
648     line2->body.recursive.makefiles = dp;
649     line2->body.recursive.has_built =
650         (Boolean) (makefile_type == reading_cpp_file);
651     line2->body.recursive.in_depinfo = false;
652     return;
653 }
654 /* If this is the first target that doesnt start with a "." in the */
655 /* makefile we remember that */
656 Wstring tstr(target);
657 wchar_t * wcb = tstr.get_string();
658 if ((makefile_type == reading_makefile) &&
659     (default_target_to_build == NULL) &&
660     ((wcb[0] != (int) period_char) ||
661      wcsrchr(wcb, (int) slash_char))) {
662     wcsrchr(wcb, (int) slash_char)) {

```

```

663 /* BID 1181577: $(EMPTY_MACRO) + $(EMPTY_MACRO):
664 ** The target with empty name cannot be default_target_to_build
665 */
666     if (target->hash.length != 0)
667         default_target_to_build = target;
668 }
669 /* Check if the line is ":" or "::" */
670 if (makefile_type == reading_makefile) {
671     if (target->colons == no_colon) {
672         target->colons = separator;
673     } else {
674         if (target->colons != separator) {
675             fatal_reader(gettext("/::: conflict for target `
676                 target->string_mb);
677         }
678     }
679     if (target->colons == two_colon) {
680         if (depes->used == 0) {
681             /* If this is a "::" type line with no */
682             /* dependencies we add one "FRC" type */
683             /* dependency for free */
684             depes->used = 1; /* Force :: targets with no
685                 * force to always run */
686             depes->names[0] = depes;
687         }
688         /* Do not delete "::" type targets when interrupted */
689         target->stat.is_precious = true;
690         /*
691         * Build a synthetic target "<number>%target"
692         * for "target".

```

```

693     */
694     mb_namep = getmem((int) (strlen(target->string_mb) + 10)
695     namep = ALLOC_WC((int) (target->hash.length + 10));
696     slash = strrchr(target->string_mb, (int) slash_char);
697     if (slash == NULL) {
698         (void) sprintf(mb_namep,
699             "%d%s",
700             target->colon_splits++,
701             target->string_mb);
702     } else {
703         *slash = 0;
704         (void) sprintf(mb_namep,
705             "%s/%d%s",
706             target->string_mb,
707             target->colon_splits++,
708             slash + 1);
709         *slash = (int) slash_char;
710     }
711     MBSTOWCS(namep, mb_namep);
712     retmem_mb(mb_namep);
713     name = GETNAME(namep, FIND_LENGTH);
714     retmem(namep);
715     if (trace_reader) {
716         (void) printf("%s:\t", target->string_mb);
717     }
718     /* Make "target" depend on "<number>%target */
719     line2 = maybe_append_prop(target, line_prop);
720     enter_dependency(line2, name, true);
721     line2->body.line.target = target;
722     /* Put a prop on "<number>%target that makes */
723     /* appear as "target" */
724     /* when it is processed */
725     maybe_append_prop(name, target_prop->
726         body.target.target = target;
727         target->is_double_colon_parent = true;
728         name->is_double_colon = true;
729         name->has_target_prop = true;
730         if (trace_reader) {
731             (void) printf("\n");
732         }
733         (target = name)->stat.is_file = true;
734     }
735 }
736 /* This really is a regular dependency line. Just enter it */
737 line = maybe_append_prop(target, line_prop);
738 line->body.line.target = target;
739 /* Depending on what kind of makefile we are reading we have to */
740 /* treat things differently */
741 switch (makefile_type) {
742     case reading_makefile:
743         /* Reading regular makefile. Just notice whether this */
744         /* redefines the rule for the target */
745         if (command != NULL) {
746             if (line->body.line.command_template != NULL) {
747                 line->body.line.command_template_redefined =
748                     true;
749                 if ((wcb[0] == (int) period_char) &&
750                     !wcsrchr(wcb, (int) slash_char)) {
751                     !wcsrchr(wcb, (int) slash_char)) {
752                         line->body.line.command_template =
753                             command;
754                     }
755                 } else {
756                     line->body.line.command_template = command;
757                 }
758             } else {

```

```

758         if ((wcb[0] == (int) period_char) &&
759             !wcschr(wcb, (int) slash_char)) {
759             !wchr(wcb, (int) slash_char)) {
760             line->body.line.command_template = command;
761         }
762     }
763     break;
764 case rereading_statefile:
765     /* Rereading the statefile. We only enter thing that changed */
766     /* since the previous time we read it */
767     if (!built_last_make_run_seen) {
768         for (Cmd_line next, cmd = command; cmd != NULL; cmd = ne
769             next = cmd->next;
770             free(cmd);
771         }
772         return;
773     }
774     built_last_make_run_seen = false;
775     command_changed = true;
776     target->ran_command = true;
777 case reading_statefile:
778     /* Reading the statefile for the first time. Enter the rules */
779     /* as "Commands used" not "templates to use" */
780     if (command != NULL) {
781         for (Cmd_line next, cmd = line->body.line.command_used;
782             cmd != NULL; cmd = next) {
783             next = cmd->next;
784             free(cmd);
785         }
786         line->body.line.command_used = command;
787     }
788 case reading_cpp_file:
789     /* Reading report file from programs that reports */
790     /* dependencies. If this is the first time the target is */
791     /* read from this reportfile we clear all old */
792     /* automatic depes */
793     if (target->temp_file_number == temp_file_number) {
794         break;
795     }
796     target->temp_file_number = temp_file_number;
797     command_changed = true;
798     if (line != NULL) {
799         for (dp = line->body.line.dependencies;
800             dp != NULL;
801             dp = dp->next) {
802             if (dp->automatic) {
803                 dp->stale = true;
804             }
805         }
806     }
807     break;
808 default:
809     fatal_reader(gettext("Internal error. Unknown makefile type %d")
810                 makefile_type);
811 }
812 /* A target may only be involved in one target group */
813 if (line->body.line.target_group != NULL) {
814     if (target_group != NULL) {
815         fatal_reader(gettext("Too many target groups for target
816             target->string_mb);
817     }
818 } else {
819     line->body.line.target_group = target_group;
820 }
822 if (trace_reader) {

```

```

823         (void) printf("%s:\t", target->string_mb);
824     }
825     /* Enter the dependencies */
826     register_as_auto = BOOLEAN(makefile_type != reading_makefile);
827     not_auto_found = false;
828     for (
829         (depes != NULL) && !not_auto_found;
830         depes = depes->next) {
831         for (i = 0; i < depes->used; i++) {
832             /* the dependency .NOT_AUTO signals beginning of
833              * explicit dependencies which were put at end of
834              * list in .make.state file - we stop entering
835              * dependencies at this point
836              */
837             if (depes->names[i] == not_auto) {
838                 not_auto_found = true;
839                 break;
840             }
841             enter_dependency(line,
842                             depes->names[i],
843                             register_as_auto);
844         }
845     }
846     if (trace_reader) {
847         (void) printf("\n");
848         print_rule(command);
849     }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

```

946     /* get patterns count */
947     Wstring wcb(target);
948     cp = wcb.get_string();
949     while (true) {
950         cp = (wchar_t *) wcschr(cp, (int) percent_char);
950         cp = (wchar_t *) wschr(cp, (int) percent_char);
951         if (cp != NULL) {
952             result->patterns_total++;
953             cp++;
954         } else {
955             break;
956         }
957     }
958     result->patterns_total++;

960     /* allocate storage for patterns */
961     result->patterns = (Name *) getmem(sizeof(Name) * result->patterns_total

963     /* then create patterns */
964     cp = wcb.get_string();
965     pattern = 0;
966     while (true) {
967         cpl = (wchar_t *) wcschr(cp, (int) percent_char);
967         cpl = (wchar_t *) wschr(cp, (int) percent_char);
968         if (cpl != NULL) {
969             result->patterns[pattern] = GETNAME(cp, cpl - cp);
970             cp = cpl + 1;
971             pattern++;
972         } else {
973             result->patterns[pattern] = GETNAME(cp, (int) target->ha
974             break;
975         }
976     }

978     Wstring wcb1;

980     /* build dependencies list */
981     for (nvp = depes; nvp != NULL; nvp = nvp->next) {
982         for (i = 0; i < nvp->used; i++) {
983             depe = ALLOC(Percent);
984             depe->next = NULL;
985             depe->patterns = NULL;
986             depe->patterns_total = 0;
987             depe->name = nvp->names[i];
988             depe->dependencies = NULL;
989             depe->command_template = NULL;
990             depe->being_expanded = false;
991             depe->target_group = NULL;

993             *depe_tail = depe;
994             depe_tail = &depe->next;

996             if (depe->name->percent) {
997                 /* get patterns count */
998                 wcb1.init(depe->name);
999                 cp = wcb1.get_string();
1000                 while (true) {
1001                     cp = (wchar_t *) wcschr(cp, (int) percen
1001                     cp = (wchar_t *) wschr(cp, (int) percent
1002                     if (cp != NULL) {
1003                         depe->patterns_total++;
1004                         cp++;
1005                     } else {
1006                         break;
1007                 }

```

```

1008     }
1009     depe->patterns_total++;

1011     /* allocate storage for patterns */
1012     depe->patterns = (Name *) getmem(sizeof(Name) *

1014     /* then create patterns */
1015     cp = wcb1.get_string();
1016     pattern = 0;
1017     while (true) {
1018         cpl = (wchar_t *) wcschr(cp, (int) perce
1018         cpl = (wchar_t *) wschr(cp, (int) percen
1019         if (cpl != NULL) {
1020             depe->patterns[pattern] = GETNAM
1021             cp = cpl + 1;
1022             pattern++;
1023         } else {
1024             depe->patterns[pattern] = GETNAM
1025             break;
1026         }
1027     }
1028 }
1029 }
1030 }

1032     /* Find the end of the percent list and append the new pattern */
1033     for (insert = &percent_list; (*insert) != NULL; insert = &(*insert)->nex
1034         *insert = result;

1036     if (trace_reader) {
1037         (void) printf("%s:", result->name->string_mb);

1039         for (depe = result->dependencies; depe != NULL; depe = depe->nex
1040             (void) printf(" %s", depe->name->string_mb);
1041         }

1043         (void) printf("\n");

1045         print_rule(command);
1046     }

1048     return result;
1049 }

unchanged_portion_omitted

1546 /*
1547 *     make_relative(to, result)
1548 *
1549 *     Given a file name compose a relative path name from it to the
1550 *     current directory.
1551 *
1552 *     Parameters:
1553 *         to           The path we want to make relative
1554 *         result      Where to put the resulting relative path
1555 *
1556 *     Global variables used:
1557 */
1558 static void
1559 make_relative(wchar_t *to, wchar_t *result)
1560 {
1561     wchar_t     *from;
1562     wchar_t     *allocated;
1563     wchar_t     *cp;
1564     wchar_t     *tocomp;
1565     int         ncomps;
1566     int         i;

```



```

1567     int                len;

1569     /* Check if the path is already relative. */
1570     if (to[0] != (int) slash_char) {
1571         (void) wscopy(result, to);
1571         (void) wscopy(result, to);
1572         return;
1573     }

1575     MBSTOWCS(wcs_buffer, get_current_path());
1576     from = allocated = (wchar_t *) wsdup(wcs_buffer);
1576     from = allocated = (wchar_t *) wsdup(wcs_buffer);

1578     /*
1579      * Find the number of components in the from name.
1580      * ncomp = number of slashes + 1.
1581      */
1582     ncomps = 1;
1583     for (cp = from; *cp != (int) nul_char; cp++) {
1584         if (*cp == (int) slash_char) {
1585             ncomps++;
1586         }
1587     }

1589     /*
1590      * See how many components match to determine how many "..",
1591      * if any, will be needed.
1592      */
1593     result[0] = (int) nul_char;
1594     tocomp = to;
1595     while ((*from != (int) nul_char) && (*from == *to)) {
1596         if (*from == (int) slash_char) {
1597             ncomps--;
1598             tocomp = &to[1];
1599         }
1600         from++;
1601         to++;
1602     }

1604     /*
1605      * Now for some special cases. Check for exact matches and
1606      * for either name terminating exactly.
1607      */
1608     if (*from == (int) nul_char) {
1609         if (*to == (int) nul_char) {
1610             MBSTOWCS(wcs_buffer, ".");
1611             (void) wscopy(result, wcs_buffer);
1611             (void) wscopy(result, wcs_buffer);
1612             retmem(allocated);
1613             return;
1614         }
1615         if (*to == (int) slash_char) {
1616             ncomps--;
1617             tocomp = &to[1];
1618         }
1619     } else if ((*from == (int) slash_char) && (*to == (int) nul_char)) {
1620         ncomps--;
1621         tocomp = to;
1622     }
1623     /* Add on the ".."s. */
1624     for (i = 0; i < ncomps; i++) {
1625         MBSTOWCS(wcs_buffer, "../");
1626         (void) wscat(result, wcs_buffer);
1626         (void) wscat(result, wcs_buffer);
1627     }

```

```

1629     /* Add on the remainder of the to name, if any. */
1630     if (*tocomp == (int) nul_char) {
1631         len = wcslen(result);
1631         len = wslen(result);
1632         result[len - 1] = (int) nul_char;
1633     } else {
1634         (void) wscat(result, tocomp);
1634         (void) wscat(result, tocomp);
1635     }
1636     retmem(allocated);
1637     return;
1638 }

unchanged_portion_omitted

1781 /*
1782 *      sh_transform(name, value)
1783 *
1784 *      Parameters:
1785 *          name      The name of the macro we might transform
1786 *          value     The value to transform
1787 *
1788 */
1789 static void
1790 sh_transform(Name *name, Name *value)
1791 {
1792     /* Check if we need :sh transform */
1793     wchar_t      *colon;
1794     String_rec    command;
1795     String_rec    destination;
1796     wchar_t      buffer[1000];
1797     wchar_t      buffer1[1000];

1799     static wchar_t  colon_sh[4];
1800     static wchar_t  colon_shell[7];

1802     if (colon_sh[0] == (int) nul_char) {
1803         MBSTOWCS(colon_sh, ":sh");
1804         MBSTOWCS(colon_shell, ":shell");
1805     }
1806     Wstring nms((*name));
1807     wchar_t *wcb = nms.get_string();

1809     colon = (wchar_t *) wcsrchr(wcb, (int) colon_char);
1809     colon = (wchar_t *) wsrchr(wcb, (int) colon_char);
1810     if ((colon != NULL) && (IS_WEQUAL(colon, colon_sh) || IS_WEQUAL(colon, c
1811         INIT_STRING_FROM_STACK(destination, buffer);

1813     if (*value == NULL) {
1814         buffer[0] = 0;
1815     } else {
1816         Wstring wcb1((*value));
1817         if (IS_WEQUAL(colon, colon_shell)) {
1818             INIT_STRING_FROM_STACK(command, buffer1);
1819             expand_value(*value, &command, false);
1820         } else {
1821             command.text.p = wcb1.get_string() + (*value)->h
1822             command.text.end = command.text.p;
1823             command.buffer.start = wcb1.get_string();
1824             command.buffer.end = command.text.p;
1825         }
1826         sh_command2string(&command, &destination);
1827     }

1829     (*value) = GETNAME(destination.buffer.start, FIND_LENGTH);
1830     *colon = (int) nul_char;
1831     (*name) = GETNAME(wcb, FIND_LENGTH);

```

new/usr/src/cmd/make/bin/read2.cc

17

```
1832         *colon = (int) colon_char;
1833     }
1834 }
_____unchanged_portion_omitted_
```

```

*****
9647 Fri May 22 11:19:47 2015
new/usr/src/cmd/make/bin/rep.cc
make: use the more modern wchar routines, not widec.h
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2003 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  *      rep.c
28  *
29  *      This file handles the .nse_depinfo file
30  */

32 /*
33  * Included files
34  */
35 #include <mk/defs.h>
36 #include <mksh/misc.h>          /* retmem() */
37 #include <vroot/report.h>      /* NSE_DEPINFO */

39 /*
40  * Static variables
41  */
42 static Recursive_make recursive_list;
43 static Recursive_make *bpatch = &recursive_list;
44 static Boolean changed;

46 /*
47  * File table of contents
48  */

51 /*
52  *      report_recursive_init()
53  *
54  *      Read the .nse_depinfo file and make a list of all the
55  *      .RECURSIVE entries.
56  *
57  *      Parameters:
58  *
59  *      Static variables used:
60  *          bpatch      Points to slot where next cell should be added
61  */

```

```

62 *      Global variables used:
63 *          recursive_name  The Name ".RECURSIVE", compared against
64 */

66 void
67 report_recursive_init(void)
68 {
69     char          *search_dir;
70     char          nse_depinfo[MAXPATHLEN];
71     FILE          *fp;
72     int           line_size, line_index;
73     wchar_t       *line;
74     wchar_t       *bigger_line;
75     wchar_t       *colon;
76     wchar_t       *dollar;
77     Recursive_make rp;

79     /*
80      * This routine can be called more than once, don't do
81      * anything after the first time.
82      */
83     if (depinfo_already_read) {
84         return;
85     } else {
86         depinfo_already_read = true;
87     }

88     search_dir = getenv("NSE_DEP");
89     if (search_dir == NULL) {
90         return;
91     }
92     (void) sprintf(nse_depinfo, "%s/%s", search_dir, NSE_DEPINFO);
93     fp = fopen(nse_depinfo, "r");
94     if (fp == NULL) {
95         return;
96     }
97     line_size = MAXPATHLEN;
98     line_index = line_size - 1;
99     line = ALLOC_WC(line_size);
100    Wstring rns(recursive_name);
101    wchar_t *wcb = rns.get_string();
102    while (fgetws(line, line_size, fp) != NULL) {
103        while (wcslen(line) == line_index) {
104            if (line[wcslen(line) - 1] == '\n') {
105                while (wslen(line) == line_index) {
106                    if (line[wslen(line) - 1] == '\n') {
107                        continue;
108                    }
109                    bigger_line = ALLOC_WC(2 * line_size);
110                    wscopy(bigger_line, line);
111                    wscopy(bigger_line, line);
112                    retmem(line);
113                    line = bigger_line;
114                    if (fgetws(&line[line_index], line_size, fp) == NULL)
115                        continue;
116                    line_index = 2 * line_index;
117                    line_size = 2 * line_size;
118                }
119            }
120            colon = (wchar_t *) wcschr(line, (int) colon_char);
121            if (colon == NULL) {
122                continue;
123            }
124            dollar = (wchar_t *) wcschr(line, (int) dollar_char);
125            line[wcslen(line) - 1] = (int) nul_char;

```

```

122     dollar = (wchar_t *) wschr(line, (int) dollar_char);
123     line[wslen(line) - 1] = (int) nul_char;
124     if (IS_WEQUALN(&colon[2], wcb,
125         (int) recursive_name->hash.length)) {
126         /*
127          * If this entry is an old entry, ignore it
128          */
129         MBSTOWCS(wcs_buffer, DEPINFO_FMT_VERSION);
130         if (dollar == NULL ||
131             !IS_WEQUALN(wcs_buffer, (dollar+1) - VER_LEN, VER_LE
132                 continue;
133         }
134         rp = ALLOC(Recursive_make);
135         (void) memset((char *) rp, 0, sizeof (Recursive_make_rec
136         /*
137          * set conditional_macro_string if string is present
138          */
139         rp->oldline = (wchar_t *) wcsdup(line);
140         rp->oldline = (wchar_t *) wsdup(line);
141         if ( dollar != NULL ){
142             rp->cond_macrostring =
143                 (wchar_t *) wcsdup(dollar - VER_LEN + 1);
144                 (wchar_t *) wsdup(dollar - VER_LEN + 1);
145         }
146         /*
147          * get target name into recursive struct
148          */
149         *colon = (int) nul_char;
150         rp->target = (wchar_t *) wcsdup(line);
151         rp->target = (wchar_t *) wsdup(line);
152         *bpatch = rp;
153         bpatch = &rp->next;
154     }
155 }
156 /*
157 *   report_recursive_dep(target, line)
158 *
159 *   Report a target as recursive.
160 *
161 *   Parameters:
162 *       line           Dependency line reported
163 *
164 *   Static variables used:
165 *       bpatch         Points to slot where next cell should be added
166 *       changed        Written if report set changed
167 */
168 void
169 report_recursive_dep(Name target, wchar_t *line)
170 {
171     Recursive_make  rp;
172     wchar_t        rec_buf[STRING_BUFFER_LENGTH];
173     String_rec     string;
174
175     INIT_STRING_FROM_STACK(string, rec_buf);
176     cond_macros_into_string(target, &string);
177     /*
178      * find an applicable recursive entry, if there isn't one, create it
179      */
180     rp = find_recursive_target(target);
181     if (rp == NULL) {
182         rp = ALLOC(Recursive_make);
183         (void) memset((char *) rp, 0, sizeof (Recursive_make_rec));
184         wchar_t * wcb = get_wstring(target->string_mb); // XXX Tolik: ne

```

```

185         rp->target = wcb;
186         rp->newline = (wchar_t *) wcsdup(line);
187         rp->cond_macrostring = (wchar_t *) wcsdup(rec_buf);
188         rp->newline = (wchar_t *) wsdup(line);
189         rp->cond_macrostring = (wchar_t *) wsdup(rec_buf);
190         *bpatch = rp;
191         bpatch = &rp->next;
192         changed = true;
193     } else {
194         if ((rp->oldline != NULL) && !IS_WEQUAL(rp->oldline, line)) {
195             rp->newline = (wchar_t *) wcsdup(line);
196             rp->newline = (wchar_t *) wsdup(line);
197             changed = true;
198         }
199         rp->removed = false;
200     }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

```

*****
22490 Fri May 22 11:19:47 2015
new/usr/src/cmd/make/include/mksh/defs.h
make: use the more modern wchar routines, not widec.h
*****
1 #ifndef _MKSH_DEFS_H
2 #define _MKSH_DEFS_H
3 /*
4  * CDDL HEADER START
5  *
6  * The contents of this file are subject to the terms of the
7  * Common Development and Distribution License (the "License").
8  * You may not use this file except in compliance with the License.
9  *
10 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11 * or http://www.opensolaris.org/os/licensing.
12 * See the License for the specific language governing permissions
13 * and limitations under the License.
14 *
15 * When distributing Covered Code, include this CDDL HEADER in each
16 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17 * If applicable, add the following below this CDDL HEADER, with the
18 * fields enclosed by brackets "[]" replaced with your own identifying
19 * information: Portions Copyright [yyyy] [name of copyright owner]
20 *
21 * CDDL HEADER END
22 */
23 /*
24 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
25 * Use is subject to license terms.
26 */

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

38 #include <wctype.h>
39 #include <widec.h>

40 /*
41  * A type and some utilities for boolean values
42  */

44 #define false    BOOLEAN_false
45 #define true     BOOLEAN_true

47 typedef enum {
48     false =    0,
49     true  =    1,
50     failed =    0,
51     succeeded = 1
52 } Boolean;
    unchanged_portion_omitted

141 /*
142  * CHAR_SEMANTICS_ENTRIES should be the number of entries above.
143  * The last entry in char_semantics[] should be blank.
144  */
145 #define CHAR_SEMANTICS_ENTRIES 27

```

```

146 /*
147 #define CHAR_SEMANTICS_STRING  "%*@\[\]!>-\n#(){}?;^<'\"
148 */

150 /*
151  * Some utility macros
152  */
153 #define ALLOC(x)                ((struct _##x *)getmem(sizeof (struct _##x)))
154 #define ALLOC_WC(x)            ((wchar_t *)getmem(x) * SIZEOFWCHAR_T)
155 #define FIND_LENGTH            -1
156 #define GETNAME(a,b)           getname_fn((a), (b), false)
157 #define IS_EQUAL(a,b)          (!strcmp((a), (b)))
158 #define IS_EQUALN(a,b,n)      (!strncmp((a), (b), (n)))
159 #define IS_WEQUAL(a,b)        (!wcsncmp((a), (b)))
160 #define IS_WEQUALN(a,b,n)     (!wscncmp((a), (b), (n)))
161 #define IS_WEQUAL(a,b)        (!wscmp((a), (b)))
162 #define IS_WEQUALN(a,b,n)     (!wsncmp((a), (b), (n)))
163 #define MBLEN(a)              mblen((a), MB_LEN_MAX)
164 #define MBSTOWCS(a,b)         (void) mbstowcs_with_check((a), (b), MAXPATHLEN)
165 #define MBTOWC(a,b)           mbtowc((a), (b), MB_LEN_MAX)
166 #define SIZEOFWCHAR_T        (sizeof (wchar_t))
167 #define VSIZEOF(v)            (sizeof (v) / sizeof ((v)[0]))
168 #define WCSTOMBS(a,b)         (void) wcstombs((a), (b), (MAXPATHLEN * MB_LEN_M
169 #define WCTOMB(a,b)           (void) wctomb((a), (b))
170 #define HASH(v, c)            (v = (v)*31 + (unsigned int)(c))

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

172 /*
173  * Bits stored in funny vector to classify chars
174  */
175 enum {
176     dollar_sem =          0001,
177     meta_sem =           0002,
178     percent_sem =        0004,
179     wildcard_sem =       0010,
180     command_prefix_sem = 0020,
181     special_macro_sem =  0040,
182     colon_sem =           0100,
183     parenleft_sem =      0200
184 };
    unchanged_portion_omitted

220 #define STRING_BUFFER_LENGTH 1024
221 #define INIT_STRING_FROM_STACK(str, buf) { \
222     str.buffer.start = (buf); \
223     str.text.p = (buf); \
224     str.text.end = NULL; \
225     str.buffer.end = (buf) \
226     + (sizeof (buf)/SIZEOFWCHAR_T); \
227     str.free_after_use = false; \
228 }

230 #define APPEND_NAME(np, dest, len)    append_string((np)->string_mb, (dest), (

232 class Wstring {
233     public:
234         struct _String  string;
235         wchar_t         string_buf[STRING_BUFFER_LENGTH];

237     public:
238         Wstring();
239         Wstring(struct _Name * name);
240         ~Wstring();

242         void init(struct _Name * name);

```

```
243 void init(wchar_t * name, unsigned length);
244 unsigned length() {
245     return wcslen(string.buffer.start);
247     return wslen(string.buffer.start);
246 };
247 void append_to_str(struct _String * str, unsigned off, unsigned
249 wchar_t * get_string() {
250     return string.buffer.start;
251 };
253 wchar_t * get_string(unsigned off) {
254     return string.buffer.start + off;
255 };
257 Boolean equaln(wchar_t * str, unsigned length);
258 Boolean equal(wchar_t * str);
259 Boolean equal(wchar_t * str, unsigned off);
260 Boolean equal(wchar_t * str, unsigned off, unsigned length);
262 Boolean equaln(Wstring * str, unsigned length);
263 Boolean equal(Wstring * str);
264 Boolean equal(Wstring * str, unsigned off);
265 Boolean equal(Wstring * str, unsigned off, unsigned length);
266 };
unchanged_portion_omitted
```

```

*****
15031 Fri May 22 11:19:48 2015
new/usr/src/cmd/make/lib/mksh/dosys.cc
make: use the more modern wchar routines, not widec.h
*****
_____unchanged_portion_omitted_____

131 /*
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 *
139 *              The pid of the process we started
140 *
141 *      Parameters:
142 *          command      The command to run
143 *          ignore_error  Should we abort on error?
144 *
145 *      Global variables used:
146 *          filter_stderr  If -X is on we redirect stderr
147 *          shell_name     The Name "SHELL", used to get the path to shell
148 */
149 doshell(wchar_t *command, register Boolean ignore_error, char *stdout_file, char
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 *) "nice";
177         (void) sprintf(nice_prio_buf, "-%d", nice_prio);
178         argv[argv_index++] = strdup(nice_prio_buf);
179     }
180     argv[argv_index++] = shellname;
181     argv[argv_index++] = (char *) (ignore_error ? "-c" : "-ce");
182     if ((length = wcslen(command)) >= MAXPATHLEN) {
183         if ((length = wslen(command)) >= MAXPATHLEN) {
184             tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
185             (void) wcstombs(tmp_mbs_buffer, command, (length * MB_LEN_MAX) +
186             cmd_argv_index = argv_index;
187             argv[argv_index++] = strdup(tmp_mbs_buffer);
188             retmem_mb(tmp_mbs_buffer);
189         } 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);
197     #if 0
198         if (filter_stderr) {
199             redirect_stderr();
200         }
201     #endif
202     if (nice_prio != 0) {
203         (void) execve("/usr/bin/nice", argv, environ);
204         fatal_mksh(gettext("Could not load '/usr/bin/nice': %s"),
205         errmsg(errno));
206     } else {
207         (void) execve(shell->string_mb, argv, environ);
208         fatal_mksh(gettext("Could not load Shell from '%s': %s"),
209         shell->string_mb,
210         errmsg(errno));
211     }
212 }
213 if (childPid == -1) {
214     fatal_mksh(gettext("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 *
305 *              The pid of the process started here
306 *
307 *      Parameters:
308 *          command      The command to run
309 *          ignore_error  Should we abort on error?
310 *
311 *      Global variables used:
312 *          filter_stderr  If -X is on we redirect stderr
313 */
314 doexec(register wchar_t *command, register Boolean ignore_error, char *stdout_fi
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 = wcslen(command)) >= MAXPATHLEN) {
346     if ((length = wslen(command)) >= MAXPATHLEN) {
347         tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
348         (void) wcstombs(tmp_mbs_buffer, command, (length * MB_LEN_MAX) +
349         argv[0] = strdup(tmp_mbs_buffer);
350         retmem_mb(tmp_mbs_buffer);
351     } else {
352         WCSTOMBS(mbs_buffer, command);
353         argv[0] = strdup(mbs_buffer);
354     }
355     }
356     if (nice_prio != 0) {
357         *p++ = strdup("/usr/bin/nice");
358         (void) sprintf(nice_prio_buf, "-%d", nice_prio);
359         *p++ = strdup(nice_prio_buf);
360     }
361     /* Build list of argument words. */
362     for (t = command; *t;) {
363         if (p >= &argv[arg_count]) {
364             /* This should never happen, right? */
365             WCSTOMBS(mbs_buffer, command);
366             fatal_mksh(gettext("Command '%s' has more than %d argume
367             mbs_buffer,
368             arg_count);
369         }
370         q = t;
371         while (!iswspace(*t) && (*t != (int) nul_char)) {
372             t++;
373         }
374         if (*t) {
375             for (*t++ = (int) nul_char; iswspace(*t); t++);
376         }
377         if ((length = wcslen(q)) >= MAXPATHLEN) {
378         if ((length = wslen(q)) >= MAXPATHLEN) {
379             tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
380             (void) wcstombs(tmp_mbs_buffer, q, (length * MB_LEN_MAX)
381             *p++ = strdup(tmp_mbs_buffer);
382             retmem_mb(tmp_mbs_buffer);
383         } else {
384             WCSTOMBS(mbs_buffer, q);
385             *p++ = strdup(mbs_buffer);
386         }
387     }
388     }
389     *p = NULL;
390     /* Then exec the command with that argument list. */
391     (void) fflush(stdout);
392     if ((childPid = fork()) == 0) {
393         enable_interrupt((void) (*) (int)) SIG_DFL);
394     }
395     #if 0
396     if (filter_stderr) {

```

```

394         redirect_stderr();
395     }
396     #endif
397     (void) exec_vp(argv[1], argv, environ, ignore_error, vroot_path)
398     fatal_mksh(gettext("Cannot load command '%s': %s"), argv[1], err
399     }
400     if (childPid == -1) {
401         fatal_mksh(gettext("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 }
_____unchanged_portion_omitted_

```



new/usr/src/cmd/make/lib/mksh/i18n.cc

1

\*\*\*\*\*

2182 Fri May 22 11:19:48 2015

new/usr/src/cmd/make/lib/mksh/i18n.cc

make: use the more modern wchar routines, not widec.h

\*\*\*\*\*

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_

```
61 /*
62 *   get_char_semantics_entry(ch)
63 *
64 *   Return value:
65 *       The slot number in the array for special make chars,
66 *       else the slot number of the last array entry.
67 *
68 *   Parameters:
69 *       ch           The wide character
70 *
71 *   Global variables used:
72 *       char_semantics_char[]  array of special wchar_t chars
73 *                               "&*@'\\" |[]:=${!>-\\n#()%;^<'\""
74 */
75 int
76 get_char_semantics_entry(wchar_t ch)
77 {
78     wchar_t      *char_sem_char;
79
80     char_sem_char = (wchar_t *) wcschr(char_semantics_char, ch);
80     char_sem_char = (wchar_t *) wschr(char_semantics_char, ch);
81     if (char_sem_char == NULL) {
82         /*
83          * Return the integer entry for the last slot,
84          * whose content is empty.
85          */
86         return (CHAR_SEMANTICS_ENTRIES - 1);
87     } else {
88         return (char_sem_char - char_semantics_char);
89     }
90 }
_____unchanged_portion_omitted_____
```

```

*****
36552 Fri May 22 11:19:49 2015
new/usr/src/cmd/make/lib/mksh/macro.cc
make: use the more modern wchar routines, not widec.h
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

27 /*
28 *      macro.cc
29 *
30 *      Handle expansion of make macros
31 */

33 /*
34 * Included files
35 */
36 #include <mksh/dosys.h>      /* sh_command2string() */
37 #include <mksh/il8n.h>      /* get_char_semantics_value() */
38 #include <mksh/macro.h>
39 #include <mksh/misc.h>      /* retmem() */
40 #include <mksh/read.h>      /* get_next_block_fn() */

42 #include <widec.h>
42 #include <libintl.h>

44 /*
45 * File table of contents
46 */
47 static void      add_macro_to_global_list(Name macro_to_add);
48 static void      expand_value_with_daemon(Name, register Property macro, register

50 static void      init_arch_macros(void);
51 static void      init_mach_macros(void);
52 static Boolean   init_arch_done = false;
53 static Boolean   init_mach_done = false;

56 long env_alloc_num = 0;
57 long env_alloc_bytes = 0;

59 /*
60 *      getvar(name)

```

```

61 *
62 *      Return expanded value of macro.
63 *
64 *      Return value:
65 *
66 *          The expanded value of the macro
67 *
68 *      Parameters:
69 *          name
70 *          The name of the macro we want the value for
71 */
72 Name
73 getvar(register Name name)
74 {
75     String_rec      destination;
76     wchar_t         buffer[STRING_BUFFER_LENGTH];
77     register Name   result;

79     if ((name == host_arch) || (name == target_arch)) {
80         if (!init_arch_done) {
81             init_arch_done = true;
82             init_arch_macros();
83         }
84     }
85     if ((name == host_mach) || (name == target_mach)) {
86         if (!init_mach_done) {
87             init_mach_done = true;
88             init_mach_macros();
89         }
90     }

92     INIT_STRING_FROM_STACK(destination, buffer);
93     expand_value(maybe_append_prop(name, macro_prop)->body.macro.value,
94                 &destination,
95                 false);
96     result = GETNAME(destination.buffer.start, FIND_LENGTH);
97     if (destination.free_after_use) {
98         retmem(destination.buffer.start);
99     }
100    return result;
101 }

103 /*
104 *      expand_value(value, destination, cmd)
105 *
106 *      Recursively expands all macros in the string value.
107 *      destination is where the expanded value should be appended.
108 *
109 *      Parameters:
110 *          value
111 *          The value we are expanding
112 *          destination
113 *          Where to deposit the expansion
114 *          cmd
115 *          If we are evaluating a command line we
116 *          turn \ quoting off
117 */
118 void
119 expand_value(Name value, register String destination, Boolean cmd)
120 {
121     Source_rec      sourcecb;
122     register Source source = &sourcecb;
123     register wchar_t *source_p = NULL;
124     register wchar_t *source_end = NULL;
125     wchar_t         *block_start = NULL;
126     int             quote_seen = 0;

```

```

127     if (value == NULL) {
128         /*
129          * Make sure to get a string allocated even if it
130          * will be empty.
131          */
132         MBSTOWCS(wcs_buffer, "");
133         append_string(wcs_buffer, destination, FIND_LENGTH);
134         destination->text.end = destination->text.p;
135         return;
136     }
137     if (!value->dollar) {
138         /*
139          * If the value we are expanding does not contain
140          * any $, we don't have to parse it.
141          */
142         APPEND_NAME(value,
143                     destination,
144                     (int) value->hash.length
145                 );
146         destination->text.end = destination->text.p;
147         return;
148     }
149
150     if (value->being_expanded) {
151         fatal_reader_mksh(gettext("Loop detected when expanding macro va
152                                value->string_mb);
153     }
154     value->being_expanded = true;
155     /* Setup the structure we read from */
156     Wstring vals(value);
157     sourcecb.string.text.p = sourcecb.string.buffer.start = wcsdup(vals.get_st
158     sourcecb.string.text.p = sourcecb.string.buffer.start = wsdup(vals.get_str
159     sourcecb.string.free_after_use = true;
160     sourcecb.string.text.end =
161         sourcecb.string.buffer.end =
162         sourcecb.string.text.p + value->hash.length;
163     sourcecb.previous = NULL;
164     sourcecb.fd = -1;
165     sourcecb.inp_buf =
166         sourcecb.inp_buf_ptr =
167         sourcecb.inp_buf_end = NULL;
168     sourcecb.error_converting = false;
169     /* Lift some pointers from the struct to local register variables */
170     CACHE_SOURCE(0);
171     /* We parse the string in segments */
172     /* We read chars until we find a $, then we append what we have read so far */
173     /* (since last $ processing) to the destination. When we find a $ we call */
174     /* expand_macro() and let it expand that particular $ reference into dest */
175     block_start = source_p;
176     quote_seen = 0;
177     for (; l; source_p++) {
178         switch (GET_CHAR()) {
179             case backslash_char:
180                 /* Quote $ in macro value */
181                 if (!cmd) {
182                     quote_seen = ~quote_seen;
183                 }
184                 continue;
185             case dollar_char:
186                 /* Save the plain string we found since */
187                 /* start of string or previous $ */
188                 if (quote_seen) {
189                     append_string(block_start,
190                                 destination,
191                                 source_p - block_start - 1);
192                 }
193                 block_start = source_p;

```

```

192         break;
193     }
194     append_string(block_start,
195                 destination,
196                 source_p - block_start);
197     source->string.text.p = ++source_p;
198     UNCACHE_SOURCE();
199     /* Go expand the macro reference */
200     expand_macro(source, destination, sourcecb.string.buffer.
201                CACHE_SOURCE(1);
202                block_start = source_p + 1;
203                break;
204     case nul_char:
205         /* The string ran out. Get some more */
206         append_string(block_start,
207                     destination,
208                     source_p - block_start);
209         GET_NEXT_BLOCK_NOCHK(source);
210         if (source == NULL) {
211             destination->text.end = destination->text.p;
212             value->being_expanded = false;
213             return;
214         }
215         if (source->error_converting) {
216             fatal_reader_mksh("Internal error: Invalid byte
217                                ");
218             block_start = source_p;
219             source_p--;
220             continue;
221         }
222         quote_seen = 0;
223     }
224     retmem(sourcecb.string.buffer.start);
225 }
226
227 /*
228 *
229 *
230 *
231 * Should be called with source->string.text.p pointing to
232 * the first char after the $ that starts a macro reference.
233 * source->string.text.p is returned pointing to the first char after
234 * the macro name.
235 * It will read the macro name, expanding any macros in it,
236 * and get the value. The value is then expanded.
237 * destination is a String that is filled in with the expanded macro.
238 * It may be passed in referencing a buffer to expand the macro into.
239 * Note that most expansions are done on demand, e.g. right
240 * before the command is executed and not while the file is
241 * being parsed.
242 *
243 * Parameters:
244 *     source           The source block that references the string
245 *                     to expand
246 *     destination     Where to put the result
247 *     current_string   The string we are expanding, for error msg
248 *     cmd             If we are evaluating a command line we
249 *                     turn \ quoting off
250 *
251 * Global variables used:
252 *     funny           Vector of semantic tags for characters
253 *     is_conditional Set if a conditional macro is refd
254 *     make_word_mentioned Set if the word "MAKE" is mentioned
255 *     makefile_type   We deliver extra msg when reading makefiles
256 *     query           The Name "?", compared against
257 *     query_mentioned Set if the word "?" is mentioned
258 */

```

```

258 void
259 expand_macro(register Source source, register String destination, wchar_t *curre
260 {
261     static Name          make = (Name)NULL;
262     static wchar_t      colon_sh[4];
263     static wchar_t      colon_shell[7];
264     String_rec          string;
265     wchar_t             buffer[STRING_BUFFER_LENGTH];
266     register wchar_t     *source_p = source->string.text.p;
267     register wchar_t     *source_end = source->string.text.end;
268     register int         closer = 0;
269     wchar_t             *block_start = (wchar_t *)NULL;
270     int                 quote_seen = 0;
271     register int         closer_level = 1;
272     Name                name = (Name)NULL;
273     wchar_t             *colon = (wchar_t *)NULL;
274     wchar_t             *percent = (wchar_t *)NULL;
275     wchar_t             *eq = (wchar_t *) NULL;
276     Property            macro = NULL;
277     wchar_t             *p = (wchar_t*)NULL;
278     String_rec          extracted;
279     wchar_t             extracted_string[MAXPATHLEN];
280     wchar_t             *left_head = NULL;
281     wchar_t             *left_tail = NULL;
282     wchar_t             *right_tail = NULL;
283     int                 left_head_len = 0;
284     int                 left_tail_len = 0;
285     int                 tmp_len = 0;
286     wchar_t             *right_hand[128];
287     int                 i = 0;
288     enum {
289         no_extract,
290         dir_extract,
291         file_extract
292     } extraction = no_extract;
293     enum {
294         no_replace,
295         suffix_replace,
296         pattern_replace,
297         sh_replace
298     } replacement = no_replace;
299
300     if (make == NULL) {
301         MBSTOWCS(wcs_buffer, "MAKE");
302         make = GETNAME(wcs_buffer, FIND_LENGTH);
303
304         MBSTOWCS(colon_sh, ":sh");
305         MBSTOWCS(colon_shell, ":shell");
306     }
307
308     right_hand[0] = NULL;
309
310     /* First copy the (macro-expanded) macro name into string. */
311     INIT_STRING_FROM_STACK(string, buffer);
312     recheck_first_char:
313     /* Check the first char of the macro name to figure out what to do. */
314     switch (GET_CHAR()) {
315     case nul_char:
316         GET_NEXT_BLOCK_NOCHK(source);
317         if (source == NULL) {
318             WCSTOMBS(mbs_buffer, current_string);
319             fatal_reader_mksh(gettext("' '$' at end of string '%s'"),
320                             mbs_buffer);
321         }
322         if (source->error_converting) {
323             fatal_reader_mksh("Internal error: Invalid byte sequence

```

```

324     }
325     goto recheck_first_char;
326     case parenleft_char:
327         /* Multi char name. */
328         closer = (int) parenright_char;
329         break;
330     case braceleft_char:
331         /* Multi char name. */
332         closer = (int) braceright_char;
333         break;
334     case newline_char:
335         fatal_reader_mksh(gettext("' '$' at end of line"));
336     default:
337         /* Single char macro name. Just suck it up */
338         append_char(*source_p, &string);
339         source->string.text.p = source_p + 1;
340         goto get_macro_value;
341     }
342
343     /* Handle multi-char macro names */
344     block_start = ++source_p;
345     quote_seen = 0;
346     for (; 1; source_p++) {
347         switch (GET_CHAR()) {
348         case nul_char:
349             append_string(block_start,
350                          &string,
351                          source_p - block_start);
352             GET_NEXT_BLOCK_NOCHK(source);
353             if (source == NULL) {
354                 if (current_string != NULL) {
355                     WCSTOMBS(mbs_buffer, current_string);
356                     fatal_reader_mksh(gettext("Unmatched '%c
357                                     closer ==
358                                     (int) braceright_char ?
359                                     (int) braceleft_char :
360                                     (int) parenleft_char,
361                                     mbs_buffer);
362                 } else {
363                     fatal_reader_mksh(gettext("Premature EOF
364                                     ));
365                 }
366             }
367             if (source->error_converting) {
368                 fatal_reader_mksh("Internal error: Invalid byte
369             );
370             block_start = source_p;
371             source_p--;
372             continue;
373         case newline_char:
374             fatal_reader_mksh(gettext("Unmatched '%c' on line"),
375                             closer == (int) braceright_char ?
376                             (int) braceleft_char :
377                             (int) parenleft_char);
378         case backslash_char:
379             /* Quote dollar in macro value. */
380             if (!cmd) {
381                 quote_seen = ~quote_seen;
382             }
383             continue;
384         case dollar_char:
385             /*
386              * Macro names may reference macros.
387              * This expands the value of such macros into the
388              * macro name string.
389              */
390             if (quote_seen) {

```

```

390     append_string(block_start,
391                  &string,
392                  source_p - block_start - 1);
393     block_start = source_p;
394     break;
395 }
396 append_string(block_start,
397              &string,
398              source_p - block_start);
399 source->string.text.p = ++source_p;
400 UNCACHE_SOURCE();
401 expand_macro(source, &string, current_string, cmd);
402 CACHE_SOURCE(0);
403 block_start = source_p;
404 source_p--;
405 break;
406 case parenleft_char:
407     /* Allow nested pairs of () in the macro name. */
408     if (closer == (int) parenright_char) {
409         closer_level++;
410     }
411     break;
412 case braceleft_char:
413     /* Allow nested pairs of {} in the macro name. */
414     if (closer == (int) braceright_char) {
415         closer_level++;
416     }
417     break;
418 case parenright_char:
419 case braceright_char:
420     /*
421      * End of the name. Save the string in the macro
422      * name string.
423      */
424     if ((*source_p == closer) && (--closer_level <= 0)) {
425         source->string.text.p = source_p + 1;
426         append_string(block_start,
427                     &string,
428                     source_p - block_start);
429         goto get_macro_value;
430     }
431     break;
432 }
433 quote_seen = 0;
434 }
435 /*
436 * We got the macro name. We now inspect it to see if it
437 * specifies any translations of the value.
438 */
439 get_macro_value:
440     name = NULL;
441     /* First check if we have a $(@D) type translation. */
442     if ((get_char_semantics_value(string.buffer.start[0]) &
443         (int) special_macro_sem) &&
444         (string.text.p - string.buffer.start >= 2) &&
445         ((string.buffer.start[1] == 'D') ||
446          (string.buffer.start[1] == 'F'))) {
447         switch (string.buffer.start[1]) {
448             case 'D':
449                 extraction = dir_extract;
450                 break;
451             case 'F':
452                 extraction = file_extract;
453                 break;
454             default:
455                 WCSTOMBS(mbs_buffer, string.buffer.start);

```

```

456         fatal_reader_mksh(gettext("Illegal macro reference '%s'"
457                                 mbs_buffer));
458     }
459     /* Internalize the macro name using the first char only. */
460     name = GETNAME(string.buffer.start, 1);
461     (void) wcsncpy(string.buffer.start, string.buffer.start + 2);
462     (void) wscopy(string.buffer.start, string.buffer.start + 2);
463 }
464 /* Check for other kinds of translations. */
465 if ((colon = (wchar_t *) wcschr(string.buffer.start,
466                                if ((colon = (wchar_t *) wschr(string.buffer.start,
467                                                                (int) colon_char)) != NULL) {
468     /*
469      * We have a $(FOO:.c=.o) type translation.
470      * Get the name of the macro proper.
471      */
472     if (name == NULL) {
473         name = GETNAME(string.buffer.start,
474                       colon - string.buffer.start);
475     }
476     /* Pickup all the translations. */
477     if (IS_WEQUAL(colon, colon_sh) || IS_WEQUAL(colon, colon_shell))
478         replacement = sh_replace;
479     } else if ((svr4) ||
480                ((percent = (wchar_t *) wcschr(colon + 1,
481                                                ((percent = (wchar_t *) wschr(colon + 1,
482                                                                (int) percent_char)) ==
483     while (colon != NULL) {
484         if ((eq = (wchar_t *) wcschr(colon + 1,
485                                     if ((eq = (wchar_t *) wschr(colon + 1,
486                                                                (int) equal_char)) =
487             fatal_reader_mksh(gettext("= missing fro
488         left_tail_len = eq - colon - 1;
489         if(left_tail) {
490             retmem(left_tail);
491         }
492         left_tail = ALLOC_WC(left_tail_len + 1);
493         (void) wcsncpy(left_tail,
494                       (void) wscopy(left_tail,
495                                     colon + 1,
496                                     eq - colon - 1);
497         left_tail[eq - colon - 1] = (int) nul_char;
498         replacement = suffix_replace;
499         if ((colon = (wchar_t *) wcschr(eq + 1,
500                                         if ((colon = (wchar_t *) wschr(eq + 1,
501                                                                (int) colon_char)
502             tmp_len = colon - eq;
503             if(right_tail) {
504                 retmem(right_tail);
505             }
506             right_tail = ALLOC_WC(tmp_len);
507             (void) wcsncpy(right_tail,
508                           (void) wscopy(right_tail,
509                                     eq + 1,
510                                     colon - eq - 1);
511             right_tail[colon - eq - 1] =
512                 (int) nul_char;
513         } else {
514             if(right_tail) {
515                 retmem(right_tail);
516             }
517             right_tail = ALLOC_WC(wcslen(eq) + 1);
518             (void) wcsncpy(right_tail, eq + 1);
519             right_tail = ALLOC_WC(wslen(eq) + 1);
520             (void) wscopy(right_tail, eq + 1);

```

```

513     }
514     } else {
515         if ((eq = (wchar_t *) wcschr(colon + 1,
516             if ((eq = (wchar_t *) wschr(colon + 1,
517                 (int) equal_char)) == NULL)
518                 fatal_reader_mksh(gettext("= missing from replac
519             })
520             if ((percent = (wchar_t *) wcschr(colon + 1,
521                 if ((percent = (wchar_t *) wschr(colon + 1,
522                     (int) percent_char)) ==
523                     fatal_reader_mksh(gettext("%% missing from repla
524             })
525             if (eq < percent) {
526                 fatal_reader_mksh(gettext("%% missing from repla
527             })
528             if (percent > (colon + 1)) {
529                 tmp_len = percent - colon;
530                 if(left_head) {
531                     retmem(left_head);
532                 }
533                 left_head = ALLOC_WC(tmp_len);
534                 (void) wcsncpy(left_head,
535                     (void) wcsncpy(left_head,
536                         colon + 1,
537                         percent - colon - 1);
538                 left_head[percent-colon-1] = (int) nul_char;
539                 left_head_len = percent-colon-1;
540             } else {
541                 left_head = NULL;
542                 left_head_len = 0;
543             }
544             if (eq > percent+1) {
545                 tmp_len = eq - percent;
546                 if(left_tail) {
547                     retmem(left_tail);
548                 }
549                 left_tail = ALLOC_WC(tmp_len);
550                 (void) wcsncpy(left_tail,
551                     (void) wcsncpy(left_tail,
552                         percent + 1,
553                         eq - percent - 1);
554                 left_tail[eq-percent-1] = (int) nul_char;
555                 left_tail_len = eq-percent-1;
556             } else {
557                 left_tail = NULL;
558                 left_tail_len = 0;
559             }
560             if ((percent = (wchar_t *) wcschr(++eq,
561                 if ((percent = (wchar_t *) wschr(++eq,
562                     (int) percent_char)) ==
563                 right_hand[0] = ALLOC_WC(wcslen(eq) + 1);
564                 right_hand[0] = ALLOC_WC(wslen(eq) + 1);
565                 right_hand[1] = NULL;
566                 (void) wcsncpy(right_hand[0], eq);
567                 (void) wscpy(right_hand[0], eq);
568             } else {
569                 i = 0;
570                 do {
571                     right_hand[i] = ALLOC_WC(percent-eq+1);
572                     (void) wcsncpy(right_hand[i],
573                     (void) wscpy(right_hand[i],

```

```

571         eq,
572         percent - eq);
573         right_hand[i][percent - eq] =
574             (int) nul_char;
575         if (i++ >= VSZEOF(right_hand)) {
576             fatal_mksh(gettext("Too many %%
577         })
578         eq = percent + 1;
579         if (eq[0] == (int) nul_char) {
580             MBSTOWCS(wcs_buffer, "");
581             right_hand[i] = (wchar_t *) wcsd
582             right_hand[i] = (wchar_t *) wsdu
583             i++;
584             break;
585         }
586         while ((percent = (wchar_t *) wcschr(eq, (int)
587             while ((percent = (wchar_t *) wschr(eq, (int)
588             if (eq[0] != (int) nul_char) {
589                 right_hand[i] = ALLOC_WC(wcslen(eq) + 1)
590                 (void) wcsncpy(right_hand[i], eq);
591                 right_hand[i] = ALLOC_WC(wslen(eq) + 1);
592                 (void) wscpy(right_hand[i], eq);
593                 i++;
594             }
595             right_hand[i] = NULL;
596         }
597         replacement = pattern_replace;
598     }
599     if (name == NULL) {
600         /*
601         * No translations found.
602         * Use the whole string as the macro name.
603         */
604         name = GETNAME(string.buffer.start,
605             string.text.p - string.buffer.start);
606     }
607     if (string.free_after_use) {
608         retmem(string.buffer.start);
609     }
610     if (name == make) {
611         make_word_mentioned = true;
612     }
613     if (name == query) {
614         query_mentioned = true;
615     }
616     if ((name == host_arch) || (name == target_arch)) {
617         if (!init_arch_done) {
618             init_arch_done = true;
619             init_arch_macros();
620         }
621     }
622     if ((name == host_mach) || (name == target_mach)) {
623         if (!init_mach_done) {
624             init_mach_done = true;
625             init_mach_macros();
626         }
627     }
628     /* Get the macro value. */
629     macro = get_prop(name->prop, macro_prop);
630     if ((macro != NULL) && macro->body.macro.is_conditional) {
631         conditional_macro_used = true;
632         /*
633         * Add this conditional macro to the beginning of the
634         * global list.
635         */

```

```

633     add_macro_to_global_list(name);
634     if (makefile_type == reading_makefile) {
635         warning_mksh(gettext("Conditional macro '%s' referenced
636                             name->string_mb, file_being_read, line_n
637     })
638     }
639     /* Macro name read and parsed. Expand the value. */
640     if ((macro == NULL) || (macro->body.macro.value == NULL)) {
641         /* If the value is empty, we just get out of here. */
642         goto exit;
643     }
644     if (replacement == sh_replace) {
645         /* If we should do a :sh transform, we expand the command
646         * and process it.
647         */
648         INIT_STRING_FROM_STACK(string, buffer);
649         /* Expand the value into a local string buffer and run cmd. */
650         expand_value_with_daemon(name, macro, &string, cmd);
651         sh_command2string(&string, destination);
652     } else if ((replacement != no_replace) || (extraction != no_extract)) {
653         /*
654         * If there were any transforms specified in the macro
655         * name, we deal with them here.
656         */
657         INIT_STRING_FROM_STACK(string, buffer);
658         /* Expand the value into a local string buffer. */
659         expand_value_with_daemon(name, macro, &string, cmd);
660         /* Scan the expanded string. */
661         p = string.buffer.start;
662         while (*p != (int) nul_char) {
663             wchar_t chr;
664
665             /*
666             * First skip over any white space and append
667             * that to the destination string.
668             */
669             block_start = p;
670             while ((*p != (int) nul_char) && iswspace(*p)) {
671                 p++;
672             }
673             append_string(block_start,
674                          destination,
675                          p - block_start);
676             /* Then find the end of the next word. */
677             block_start = p;
678             while ((*p != (int) nul_char) && !iswspace(*p)) {
679                 p++;
680             }
681             /* If we cant find another word we are done */
682             if (block_start == p) {
683                 break;
684             }
685             /* Then apply the transforms to the word */
686             INIT_STRING_FROM_STACK(extracted, extracted_string);
687             switch (extraction) {
688             case dir_extract:
689                 /*
690                 * $(@D) type transform. Extract the
691                 * path from the word. Deliver "." if
692                 * none is found.
693                 */
694                 if (p != NULL) {
695                     chr = *p;
696                     *p = (int) nul_char;
697                 }
698                 eq = (wchar_t *) wcsrchr(block_start, (int) slash

```

```

699                 eq = (wchar_t *) wsrchr(block_start, (int) slash
699                 if (p != NULL) {
700                     *p = chr;
701                 }
702                 if ((eq == NULL) || (eq > p)) {
703                     MBSTOWCS(wcs_buffer, ".");
704                     append_string(wcs_buffer, &extracted, 1)
705                 } else {
706                     append_string(block_start,
707                                  &extracted,
708                                  eq - block_start);
709                 }
710                 break;
711             case file_extract:
712                 /*
713                 * $(@F) type transform. Remove the path
714                 * from the word if any.
715                 */
716                 if (p != NULL) {
717                     chr = *p;
718                     *p = (int) nul_char;
719                 }
720                 eq = (wchar_t *) wcsrchr(block_start, (int) slash
721                 eq = (wchar_t *) wsrchr(block_start, (int) slash
722                 if (p != NULL) {
723                     *p = chr;
724                 }
725                 if ((eq == NULL) || (eq > p)) {
726                     append_string(block_start,
727                                  &extracted,
728                                  p - block_start);
729                 } else {
730                     append_string(eq + 1,
731                                  &extracted,
732                                  p - eq - 1);
733                 }
734                 break;
735             case no_extract:
736                 append_string(block_start,
737                              &extracted,
738                              p - block_start);
739                 break;
740             }
741             switch (replacement) {
742             case suffix_replace:
743                 /*
744                 * $(FOO:.o=.c) type transform.
745                 * Maybe replace the tail of the word.
746                 */
747                 if (((extracted.text.p -
748                     extracted.buffer.start) >=
749                     left_tail_len) &&
750                     IS_WEQUALN(extracted.text.p - left_tail_len,
751                                 left_tail,
752                                 left_tail_len)) {
753                     append_string(extracted.buffer.start,
754                                  destination,
755                                  (extracted.text.p -
756                                   extracted.buffer.start)
757                                   - left_tail_len);
758                     append_string(right_tail,
759                                  destination,
760                                  FIND_LENGTH);
761                 } else {
762                     append_string(extracted.buffer.start,
763                                  destination,

```

```

763             FIND_LENGTH);
764         }
765         break;
766     case pattern_replace:
767         /* $(X:a%b=c%d) type transform. */
768         if ((extracted.text.p -
769             extracted.buffer.start) >=
770             left_head_len+left_tail_len) &&
771             IS_WEQUALN(left_head,
772                 extracted.buffer.start,
773                 left_head_len) &&
774             IS_WEQUALN(left_tail,
775                 extracted.text.p - left_tail_len,
776                 left_tail_len)) {
777             i = 0;
778             while (right_hand[i] != NULL) {
779                 append_string(right_hand[i],
780                     destination,
781                     FIND_LENGTH);
782                 i++;
783                 if (right_hand[i] != NULL) {
784                     append_string(extracted.
785                         start +
786                             left_head_
787                             destinatio
788                             (extracted
789                             )
790                             )
791             } else {
792                 append_string(extracted.buffer.start,
793                     destination,
794                     FIND_LENGTH);
795             }
796             break;
797     case no_replace:
798         append_string(extracted.buffer.start,
799             destination,
800             FIND_LENGTH);
801         break;
802     case sh_replace:
803         break;
804     }
805 }
806 if (string.free_after_use) {
807     retmem(string.buffer.start);
808 }
809 } else {
810     /*
811     * This is for the case when the macro name did not
812     * specify transforms.
813     */
814     if (!strncmp(name->string_mb, "GET", 3)) {
815         dollarget_seen = true;
816     }
817     dollarless_flag = false;
818     if (!strncmp(name->string_mb, "<", 1) &&
819         dollarget_seen) {
820         dollarless_flag = true;
821         dollarget_seen = false;
822     }
823     expand_value_with_daemon(name, macro, destination, cmd);
824 }
825 exit:
826 if(left_tail) {
827     retmem(left_tail);
828 }

```

```

829     if(right_tail) {
830         retmem(right_tail);
831     }
832     if(left_head) {
833         retmem(left_head);
834     }
835     i = 0;
836     while (right_hand[i] != NULL) {
837         retmem(right_hand[i]);
838         i++;
839     }
840     *destination->text.p = (int) nul_char;
841     destination->text.end = destination->text.p;
842 }

```

unchanged\_portion\_omitted

```

883 /*
884 *   init_arch_macros(void)
885 *
886 *   Set the magic macros TARGET_ARCH, HOST_ARCH,
887 *
888 *   Parameters:
889 *
890 *   Global variables used:
891 *       host_arch   Property for magic macro HOST_ARCH
892 *       target_arch Property for magic macro TARGET_ARCH
893 *
894 *   Return value:
895 *       The function does not return a value, but can
896 *       call fatal() in case of error.
897 */
898 static void
899 init_arch_macros(void)
900 {
901     String_rec   result_string;
902     wchar_t      wc_buf[STRING_BUFFER_LENGTH];
903     char         mb_buf[STRING_BUFFER_LENGTH];
904     FILE         *pipe;
905     Name         value;
906     int          set_host, set_target;
907     const char   *mach_command = "/bin/mach";
908
909     set_host = (get_prop(host_arch->prop, macro_prop) == NULL);
910     set_target = (get_prop(target_arch->prop, macro_prop) == NULL);
911
912     if (set_host || set_target) {
913         INIT_STRING_FROM_STACK(result_string, wc_buf);
914         append_char((int) hyphen_char, &result_string);
915
916         if ((pipe = popen(mach_command, "r")) == NULL) {
917             fatal_mksh(gettext("Execute of %s failed"), mach_command);
918         }
919         while (fgets(mb_buf, sizeof(mb_buf), pipe) != NULL) {
920             MBSTOWCS(wcs_buffer, mb_buf);
921             append_string(wcs_buffer, &result_string, wcslen(wcs_buf
922             append_string(wcs_buffer, &result_string, wslen(wcs_buff
923         }
924         if (pclose(pipe) != 0) {
925             fatal_mksh(gettext("Execute of %s failed"), mach_command);
926         }
927
928         value = GETNAME(result_string.buffer.start, wcslen(result_string
929         value = GETNAME(result_string.buffer.start, wslen(result_string.
929
930         if (set_host) {
931             (void) setvar_daemon(host_arch, value, false, no_daemon,

```



```

931     }
932     if (set_target) {
933         (void) setvar_daemon(target_arch, value, false, no_daemo
934     }
935 }
936 }

938 /*
939 *   init_mach_macros(void)
940 *
941 *   Set the magic macros TARGET_MACH, HOST_MACH,
942 *
943 *   Parameters:
944 *
945 *   Global variables used:
946 *       host_mach  Property for magic macro HOST_MACH
947 *       target_mach Property for magic macro TARGET_MACH
948 *
949 *   Return value:
950 *       The function does not return a value, but can
951 *       call fatal() in case of error.
952 */
953 static void
954 init_mach_macros(void)
955 {
956     String_rec    result_string;
957     wchar_t      wc_buf[STRING_BUFFER_LENGTH];
958     char          mb_buf[STRING_BUFFER_LENGTH];
959     FILE          *pipe;
960     Name          value;
961     int           set_host, set_target;
962     const char   *arch_command = "/bin/arch";

964     set_host = (get_prop(host_mach->prop, macro_prop) == NULL);
965     set_target = (get_prop(target_mach->prop, macro_prop) == NULL);

967     if (set_host || set_target) {
968         INIT_STRING_FROM_STACK(result_string, wc_buf);
969         append_char((int) hyphen_char, &result_string);

971         if ((pipe = popen(arch_command, "r")) == NULL) {
972             fatal_mksh(gettext("Execute of %s failed"), arch_command
973         }
974         while (fgets(mb_buf, sizeof(mb_buf), pipe) != NULL) {
975             MBSTOWCS(wcs_buffer, mb_buf);
976             append_string(wcs_buffer, &result_string, wcslen(wcs_buf
977             append_string(wcs_buffer, &result_string, wslen(wcs_buff
978         }
979         if (pclose(pipe) != 0) {
980             fatal_mksh(gettext("Execute of %s failed"), arch_command

982     value = GETNAME(result_string.buffer.start, wcslen(result_string
983     value = GETNAME(result_string.buffer.start, wslen(result_string

984     if (set_host) {
985         (void) setvar_daemon(host_mach, value, false, no_daemon,
986     }
987     if (set_target) {
988         (void) setvar_daemon(target_mach, value, false, no_daemo
989     }
990 }
991 }

```

unchanged portion omitted

1044 /\*

```

1045 * We use a permanent buffer to reset SUNPRO_DEPENDENCIES value.
1046 */
1047 char    *sunpro_dependencies_buf = NULL;
1048 char    *sunpro_dependencies_oldbuf = NULL;
1049 int     sunpro_dependencies_buf_size = 0;

1051 /*
1052 *   setvar_daemon(name, value, append, daemon, strip_trailing_spaces)
1053 *
1054 *   Set a macro value, possibly supplying a daemon to be used
1055 *   when referencing the value.
1056 *
1057 *   Return value:
1058 *       The property block with the new value
1059 *
1060 *   Parameters:
1061 *       name      Name of the macro to set
1062 *       value     The value to set
1063 *       append    Should we reset or append to the current value?
1064 *       daemon    Special treatment when reading the value
1065 *       strip_trailing_spaces from the end of value->string
1066 *       debug_level Indicates how much tracing we should do
1067 *
1068 *   Global variables used:
1069 *       makefile_type Used to check if we should enforce read only
1070 *       path_name     The Name "PATH", compared against
1071 *       virtual_root  The Name "VIRTUAL_ROOT", compared against
1072 *       vpath_defined Set if the macro VPATH is set
1073 *       vpath_name    The Name "VPATH", compared against
1074 *       envvar        A list of environment vars with $ in value
1075 */
1076 Property
1077 setvar_daemon(register Name name, register Name value, Boolean append, Daemon da
1078 {
1079     register Property    macro = maybe_append_prop(name, macro_prop);
1080     register Property    macro_apx = get_prop(name->prop, macro_append_pr
1081     int                  length = 0;
1082     String_rec           destination;
1083     wchar_t              buffer[STRING_BUFFER_LENGTH];
1084     register Chain       chain;
1085     Name                 val;
1086     wchar_t              *val_string = (wchar_t*)NULL;
1087     Wstring              wcb;

1090     if ((makefile_type != reading_nothing) &&
1091         macro->body.macro.read_only) {
1092         return macro;
1093     }
1094     /* Strip spaces from the end of the value */
1095     if (daemon == no_daemon) {
1096         if (value != NULL) {
1097             wcb.init(value);
1098             length = wcb.length();
1099             val_string = wcb.get_string();
1100         }
1101         if ((length > 0) && iswspace(val_string[length-1])) {
1102             INIT_STRING_FROM_STACK(destination, buffer);
1103             buffer[0] = 0;
1104             append_string(val_string, &destination, length);
1105             if (strip_trailing_spaces) {
1106                 while ((length > 0) &&
1107                     iswspace(destination.buffer.start[length-
1108                         destination.buffer.start[--length] = 0;
1109             }
1110     }

```

```

1111         value = GETNAME(destination.buffer.start, FIND_LENGTH);
1112     }
1113 }
1114
1115 if(macro_apx != NULL) {
1116     val = macro_apx->body.macro_appendix.value;
1117 } else {
1118     val = macro->body.macro.value;
1119 }
1120
1121 if (append) {
1122     /*
1123     * If we are appending, we just tack the new value after
1124     * the old one with a space in between.
1125     */
1126     INIT_STRING_FROM_STACK(destination, buffer);
1127     buffer[0] = 0;
1128     if ((macro != NULL) && (val != NULL)) {
1129         APPEND_NAME(val,
1130                   &destination,
1131                   (int) val->hash.length);
1132         if (value != NULL) {
1133             wcb.init(value);
1134             if(wcb.length() > 0) {
1135                 MBTOWC(wcs_buffer, " ");
1136                 append_char(wcs_buffer[0], &destination)
1137             }
1138         }
1139     }
1140     if (value != NULL) {
1141         APPEND_NAME(value,
1142                   &destination,
1143                   (int) value->hash.length);
1144     }
1145     value = GETNAME(destination.buffer.start, FIND_LENGTH);
1146     wcb.init(value);
1147     if (destination.free_after_use) {
1148         retmem(destination.buffer.start);
1149     }
1150 }
1151
1152 /* Debugging trace */
1153 if (debug_level > 1) {
1154     if (value != NULL) {
1155         switch (daemon) {
1156             case chain_daemon:
1157                 (void) printf("%s =", name->string_mb);
1158                 for (chain = (Chain) value;
1159                     chain != NULL;
1160                     chain = chain->next) {
1161                     (void) printf(" %s", chain->name->string
1162                                 );
1163                 }
1164                 (void) printf("\n");
1165                 break;
1166             case no_daemon:
1167                 (void) printf("%s= %s\n",
1168                             name->string_mb,
1169                             value->string_mb);
1170                 break;
1171         }
1172     } else {
1173         (void) printf("%s =\n", name->string_mb);
1174     }
1175 }
1176 /**/

```

```

1177 if(macro_apx != NULL) {
1178     macro_apx->body.macro_appendix.value = value;
1179     INIT_STRING_FROM_STACK(destination, buffer);
1180     buffer[0] = 0;
1181     if (value != NULL) {
1182         APPEND_NAME(value,
1183                   &destination,
1184                   (int) value->hash.length);
1185         if (macro_apx->body.macro_appendix.value_to_append != NU
1186             MBTOWC(wcs_buffer, " ");
1187             append_char(wcs_buffer[0], &destination);
1188         }
1189     }
1190     if (macro_apx->body.macro_appendix.value_to_append != NULL) {
1191         APPEND_NAME(macro_apx->body.macro_appendix.value_to_appen
1192                   &destination,
1193                   (int) macro_apx->body.macro_appendix.value
1194                   );
1195     }
1196     value = GETNAME(destination.buffer.start, FIND_LENGTH);
1197     if (destination.free_after_use) {
1198         retmem(destination.buffer.start);
1199     }
1200 /**/
1201 macro->body.macro.value = value;
1202 macro->body.macro.daemon = daemon;
1203 /*
1204 * If the user changes the VIRTUAL_ROOT, we need to flush
1205 * the root package cache.
1206 */
1207 if (name == path_name) {
1208     flush_path_cache();
1209 }
1210 if (name == virtual_root) {
1211     flush_root_cache();
1212 }
1213 /* If this sets the VPATH we remember that */
1214 if ((name == vpath_name) &&
1215     (value != NULL) &&
1216     (value->hash.length > 0)) {
1217     vpath_defined = true;
1218 }
1219 /*
1220 * For environment variables we also set the
1221 * environment value each time.
1222 */
1223 if (macro->body.macro.exported) {
1224     static char *env;
1225
1226     if (!reading_environment && (value != NULL)) {
1227         Envvar p;
1228
1229         for (p = envvar; p != NULL; p = p->next) {
1230             if (p->name == name) {
1231                 p->value = value;
1232                 p->already_put = false;
1233                 goto found_it;
1234             }
1235         }
1236         p = ALLOC(Envvar);
1237         p->name = name;
1238         p->value = value;
1239         p->next = envvar;
1240         p->env_string = NULL;
1241         p->already_put = false;
1242         envvar = p;

```

```

1243 found_it::
1244     } if (reading_environment || (value == NULL) || !value->dollar)
1245         length = 2 + strlen(name->string_mb);
1246         if (value != NULL) {
1247             length += strlen(value->string_mb);
1248         }
1249         Property env_prop = maybe_append_prop(name, env_mem_prop
1250         /*
1251          * We use a permanent buffer to reset SUNPRO_DEPENDENCIE
1252          */
1253         if (!strcmp(name->string_mb, "SUNPRO_DEPENDENCIES", 19)
1254             if (length >= sunpro_dependencies_buf_size) {
1255                 sunpro_dependencies_buf_size=length*2;
1256                 if (sunpro_dependencies_buf_size < 4096)
1257                     sunpro_dependencies_buf_size = 4
1258                 if (sunpro_dependencies_buf)
1259                     sunpro_dependencies_oldbuf = sun
1260                     sunpro_dependencies_buf=getmem(sunpro_de
1261                 }
1262                 env = sunpro_dependencies_buf;
1263             } else {
1264                 env = getmem(length);
1265             }
1266             env_alloc_num++;
1267             env_alloc_bytes += length;
1268             (void) sprintf(env,
1269                 "%s=%s",
1270                 name->string_mb,
1271                 value == NULL ?
1272                 "" : value->string_mb);
1273             (void) putenv(env);
1274             env_prop->body.env_mem.value = env;
1275             if (sunpro_dependencies_oldbuf) {
1276                 /* Return old buffer */
1277                 retmem_mb(sunpro_dependencies_oldbuf);
1278                 sunpro_dependencies_oldbuf = NULL;
1279             }
1280         }
1281     }
1282     if (name == target_arch) {
1283         Name ha = getvar(host_arch);
1284         Name ta = getvar(target_arch);
1285         Name vr = getvar(virtual_root);
1286         int length;
1287         wchar_t *new_value;
1288         wchar_t *old_vr;
1289         Boolean new_value_allocated = false;
1290
1291         Wstring ha_str(ha);
1292         Wstring ta_str(ta);
1293         Wstring vr_str(vr);
1294
1295         wchar_t * wcb_ha = ha_str.get_string();
1296         wchar_t * wcb_ta = ta_str.get_string();
1297         wchar_t * wcb_vr = vr_str.get_string();
1298
1299         length = 32 +
1300             wcslen(wcb_ha) +
1301             wcslen(wcb_ta) +
1302             wcslen(wcb_vr);
1303         wslen(wcb_ha) +
1304         wslen(wcb_ta) +
1305         wslen(wcb_vr);
1306         old_vr = wcb_vr;
1307         MBSTOWCS(wcs_buffer, "/usr/arch/");
1308         if (IS_WEQUALN(old_vr,

```

```

1306             wcs_buffer,
1307             wcslen(wcs_buffer)) {
1308             old_vr = (wchar_t *) wcschr(old_vr, (int) colon_char) +
1309             wslen(wcs_buffer)) {
1310             old_vr = (wchar_t *) wcschr(old_vr, (int) colon_char) + 1
1311         }
1312         if ( (ha == ta) || (wcslen(wcb_ta) == 0) ) {
1313             if ( (ha == ta) || (wslen(wcb_ta) == 0) ) {
1314                 new_value = old_vr;
1315             } else {
1316                 new_value = ALLOC_WC(length);
1317                 new_value_allocated = true;
1318                 WCSTOMBS(mbs_buffer, old_vr);
1319                 (void) swprintf(new_value, length * SIZEOFWCHAR_T,
1320                     L"/usr/arch/%s/%s:%s",
1321                     "/usr/arch/%s/%s:%s",
1322                     ha->string_mb + 1,
1323                     ta->string_mb + 1,
1324                     mbs_buffer);
1325             }
1326             if (new_value[0] != 0) {
1327                 (void) setvar_daemon(virtual_root,
1328                     GETNAME(new_value, FIND_LENGTH),
1329                     false,
1330                     no_daemon,
1331                     true,
1332                     debug_level);
1333             }
1334             if (new_value_allocated) {
1335                 retmem(new_value);
1336             }
1337         }
1338         return macro;
1339     }
1340 }
1341 unchanged_portion_omitted

```

```

*****
24135 Fri May 22 11:19:49 2015
new/usr/src/cmd/make/lib/mksh/misc.cc
make: use the more modern wchar routines, not widec.h
*****
_____unchanged_portion_omitted_____

141 /*
142 *      getname_fn(name, len, dont_enter)
143 *
144 *      Hash a name string to the corresponding nameblock.
145 *
146 *      Return value:
147 *
148 *              The Name block for the string
149 *
150 *      Parameters:
151 *          name      The string we want to internalize
152 *          len       The length of that string
153 *          dont_enter Don't enter the name if it does not exist
154 *
155 *      Global variables used:
156 *          funny     The vector of semantic tags for characters
157 *          hashtab   The hashtable used for the nametable
158 *
159 Name
160 getname_fn(wchar_t *name, register int len, register Boolean dont_enter, register
161 {
162     register int      length;
163     register wchar_t *cap = name;
164     register Name     np;
165     static Name_rec   empty_Name;
166     char              *tmp_mbs_buffer = NULL;
167     char              *mbs_name = mbs_buffer;
168
169     /*
170      * First figure out how long the string is.
171      * If the len argument is -1 we count the chars here.
172      */
173     if (len == FIND_LENGTH) {
174         length = wcslen(name);
175     } else {
176         length = len;
177     }
178
179     Wstring ws;
180     ws.init(name, length);
181     if (length >= MAXPATHLEN) {
182         mbs_name = tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
183     }
184     (void) wcstombs(mbs_name, ws.get_string(), (length * MB_LEN_MAX) + 1);
185
186     /* Look for the string */
187     if (dont_enter || (foundp != 0)) {
188         np = hashtab.lookup(mbs_name);
189         if (foundp != 0) {
190             *foundp = (np != 0) ? true : false;
191         }
192         if ((np != 0) || dont_enter) {
193             if (tmp_mbs_buffer != NULL) {
194                 retmem_mb(tmp_mbs_buffer);
195             }
196             return np;
197         } else {
198             np = ALLOC(Name);
199         }

```

```

199     } else {
200         Boolean found;
201         np = hashtab.insert(mbs_name, found);
202         if (found) {
203             if (tmp_mbs_buffer != NULL) {
204                 retmem_mb(tmp_mbs_buffer);
205             }
206             return np;
207         }
208     }
209     getname_struct_count += sizeof(struct _Name);
210     *np = empty_Name;
211
212     np->string_mb = strdup(mbs_name);
213     if (tmp_mbs_buffer != NULL) {
214         retmem_mb(tmp_mbs_buffer);
215         mbs_name = tmp_mbs_buffer = NULL;
216     }
217     getname_bytes_count += strlen(np->string_mb) + 1;
218     /* Fill in the new Name */
219     np->stat.time = file_no_time;
220     np->hash.length = length;
221     /* Scan the namestring to classify it */
222     for (cap = name, len = 0; --length >= 0;) {
223         len |= get_char_semantics_value(*cap++);
224     }
225     np->dollar = BOOLEAN((len & (int) dollar_sem) != 0);
226     np->meta = BOOLEAN((len & (int) meta_sem) != 0);
227     np->percent = BOOLEAN((len & (int) percent_sem) != 0);
228     np->wildcard = BOOLEAN((len & (int) wildcard_sem) != 0);
229     np->colon = BOOLEAN((len & (int) colon_sem) != 0);
230     np->parenleft = BOOLEAN((len & (int) parenleft_sem) != 0);
231     getname_names_count++;
232     return np;
233 }
_____unchanged_portion_omitted_____

668 /*
669 *      append_string(from, to, length)
670 *
671 *      Append a C string to a make string expanding it if nessecary
672 *
673 *      Parameters:
674 *          from      The source (C style) string
675 *          to        The destination (make style) string
676 *          length    The length of the from string
677 *
678 *      Global variables used:
679 *
680 void
681 append_string(register wchar_t *from, register String to, register int length)
682 {
683     if (length == FIND_LENGTH) {
684         length = wcslen(from);
685     }
686     if (to->buffer.start == NULL) {
687         expand_string(to, 32 + length);
688     }
689     if (to->buffer.end - to->text.p <= length) {
690         expand_string(to,
691             (to->buffer.end - to->buffer.start) * 2 +
692             length);
693     }
694     if (length > 0) {
695         (void) wcsncpy(to->text.p, from, length);

```

```

695         (void) wsncpy(to->text.p, from, length);
696         to->text.p += length;
697     }
698     *(to->text.p) = (int) nul_char;
699 }
_____unchanged_portion_omitted_____

732 /*
733 *     expand_string(string, length)
734 *
735 *     Allocate more memory for strings that run out of space.
736 *
737 *     Parameters:
738 *         string          The make style string we want to expand
739 *         length          The new length we need
740 *
741 *     Global variables used:
742 */
743 static void
744 expand_string(register String string, register int length)
745 {
746     register wchar_t    *p;

748     if (string->buffer.start == NULL) {
749         /* For strings that have no memory allocated */
750         string->buffer.start =
751             string->text.p =
752                 string->text.end =
753                     ALLOC_WC(length);
754         string->buffer.end = string->buffer.start + length;
755         string->text.p[0] = (int) nul_char;
756         string->free_after_use = true;
757         expandstring_count++;
758         return;
759     }
760     if (string->buffer.end - string->buffer.start >= length) {
761         /* If we really don't need more memory. */
762         return;
763     }
764     /*
765     * Get more memory, copy the string and free the old buffer if
766     * it is was malloc()'ed.
767     */
768     expandstring_count++;
769     p = ALLOC_WC(length);
770     (void) wscpy(p, string->buffer.start);
770     (void) wscpy(p, string->buffer.start);
771     string->text.p = p + (string->text.p - string->buffer.start);
772     string->text.end = p + (string->text.end - string->buffer.start);
773     string->buffer.end = p + length;
774     if (string->free_after_use) {
775         retmem(string->buffer.start);
776     }
777     string->buffer.start = p;
778     string->free_after_use = true;
779 }
_____unchanged_portion_omitted_____

```