```
**********************************************************
   35019 Wed May 20 12:19:44 2015
new/usr/src/Makefile.master
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

```
   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 #

  22 #
  23 # Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
  24 # Copyright (c) 2012 by Delphix. All rights reserved.
  25 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
  26 # Copyright 2015, OmniTI Computer Consulting, Inc. All rights reserved.
  27 #

  29 #
  30 # Makefile.master, global definitions for system source
  31 #
  32 ROOT=           /proto

  34 #
  35 # Adjunct root, containing an additional proto area to be used for headers
  36 # and libraries.
  37 #
  38 ADJUNCT_PROTO=

  40 #
  41 # Adjunct for building things that run on the build machine.
  42 #
  43 NATIVE_ADJUNCT= /usr

  45 #
  46 # RELEASE_BUILD should be cleared for final release builds.
  47 # NOT_RELEASE_BUILD is exactly what the name implies.
  48 #
  49 # __GNUC toggles the building of ON components using gcc and related tools.
  50 # Normally set to '#', set it to '' to do gcc build.
  51 #
  52 # The declaration POUND_SIGN is always '#'. This is needed to get around the
  53 # make feature that '#' is always a comment delimiter, even when escaped or
  54 # quoted. We use this macro expansion method to get POUND_SIGN rather than
  55 # always breaking out a shell because the general case can cause a noticable
  56 # slowdown in build times when so many Makefiles include Makefile.master.
  57 #
  58 # While the majority of users are expected to override the setting below
  59 # with an env file (via nightly or bldenv), if you aren't building that way
  60 # (ie, you're using "ws" or some other bootstrapping method) then you need
  61 # this definition in order to avoid the subshell invocation mentioned above.
```



```
  62 #

  64 PRE_POUND=                              pre\#
  65 POUND_SIGN=                             $(PRE_POUND:pre\%=%)

  67 NOT_RELEASE_BUILD=
  68 RELEASE_BUILD=                          $(POUND_SIGN)
  69 $(RELEASE_BUILD)NOT_RELEASE_BUILD=      $(POUND_SIGN)
  70 PATCH_BUILD=                            $(POUND_SIGN)

  72 # SPARC_BLD is '#' for an Intel build.
  73 # INTEL_BLD is '#' for a Sparc build.
  74 SPARC_BLD_1=    $(MACH:i386=$(POUND_SIGN))
  75 SPARC_BLD=      $(SPARC_BLD_1:sparc=)
  76 INTEL_BLD_1=    $(MACH:sparc=$(POUND_SIGN))
  77 INTEL_BLD=      $(INTEL_BLD_1:i386=)

  79 # The variables below control the compilers used during the build.
  80 # There are a number of permutations.
  81 #
  82 # __GNUC and __SUNC control (and indicate) the primary compiler.  Whichever
  83 # one is not POUND_SIGN is the primary, with the other as the shadow.  They
  84 # may also be used to control entirely compiler-specific Makefile assignments.
  85 # __GNUC and GCC are the default.
  86 #
  87 # __GNUC64 indicates that the 64bit build should use the GNU C compiler.
  88 # There is no Sun C analogue.
  89 #
  90 # The following version-specific options are operative regardless of which
  91 # compiler is primary, and control the versions of the given compilers to be
  92 # used.  They also allow compiler-version specific Makefile fragments.
  93 #

  95 __SUNC=                 $(POUND_SIGN)
  96 $(__SUNC)__GNUC=        $(POUND_SIGN)
  97 __GNUC64=               $(__GNUC)

  99 # Allow build-time "configuration" to enable or disable some things.
 100 # The default is POUND_SIGN, meaning "not enabled". If the environment
 101 # passes in an override like ENABLE_SMB_PRINTING= (empty) that will
 102 # uncomment things in the lower Makefiles to enable the feature.
 103 ENABLE_IPP_PRINTING=    $(POUND_SIGN)
 104 ENABLE_SMB_PRINTING=    $(POUND_SIGN)

 106 # CLOSED is the root of the tree that contains source which isn't released
 107 # as open source
 108 CLOSED=         $(SRC)/../closed

 110 # BUILD_TOOLS is the root of all tools including compilers.
 111 # ONBLD_TOOLS is the root of all the tools that are part of SUNWonbld.

 113 BUILD_TOOLS=            /ws/onnv-tools
 114 ONBLD_TOOLS=            $(BUILD_TOOLS)/onbld

 116 JAVA_ROOT=      /usr/java

 118 SFW_ROOT=       /usr/sfw
 119 SFWINCDIR=      $(SFW_ROOT)/include
 120 SFWLIBDIR=      $(SFW_ROOT)/lib
 121 SFWLIBDIR64=    $(SFW_ROOT)/lib/$(MACH64)

 123 GCC_ROOT=       /opt/gcc/4.4.4
 124 GCCLIBDIR=      $(GCC_ROOT)/lib
 125 GCCLIBDIR64=    $(GCC_ROOT)/lib/$(MACH64)

 127 DOCBOOK_XSL_ROOT=       /usr/share/sgml/docbook/xsl-stylesheets
```

```
129 RPCGEN=         /usr/bin/rpcgen
130 STABS=          $(ONBLD_TOOLS)/bin/$(MACH)/stabs
131 ELFEXTRACT=     $(ONBLD_TOOLS)/bin/$(MACH)/elfextract
132 MBH_PATCH=      $(ONBLD_TOOLS)/bin/$(MACH)/mbh_patch
133 ECHO=           echo
134 INS=            install
135 TRUE=           true
136 SYMLINK=        /usr/bin/ln -s
137 LN=             /usr/bin/ln
138 CHMOD=          /usr/bin/chmod
139 MV=             /usr/bin/mv -f
140 RM=             /usr/bin/rm -f
141 CUT=            /usr/bin/cut
142 NM=             /usr/ccs/bin/nm
143 DIFF=           /usr/bin/diff
144 GREP=           /usr/bin/grep
145 EGREP=          /usr/bin/egrep
146 ELFWRAP=        /usr/bin/elfwrap
147 KSH93=          /usr/bin/ksh93
148 SED=            /usr/bin/sed
149 NAWK=           /usr/bin/nawk
150 CP=             /usr/bin/cp -f
151 MCS=            /usr/ccs/bin/mcs
152 CAT=            /usr/bin/cat
153 ELFDUMP=        /usr/ccs/bin/elfdump
154 M4=             /usr/ccs/bin/m4
155 STRIP=          /usr/ccs/bin/strip
156 LEX=            /usr/ccs/bin/lex
157 FLEX=           $(SFW_ROOT)/bin/flex
158 YACC=           /usr/ccs/bin/yacc
159 CPP=            /usr/lib/cpp
160 JAVAC=          $(JAVA_ROOT)/bin/javac
161 JAVAH=          $(JAVA_ROOT)/bin/javah
162 JAVADOC=        $(JAVA_ROOT)/bin/javadoc
163 RMIC=           $(JAVA_ROOT)/bin/rmic
164 JAR=            $(JAVA_ROOT)/bin/jar
165 CTFCONVERT=     $(ONBLD_TOOLS)/bin/$(MACH)/ctfconvert
166 CTFMERGE=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfmerge
167 CTFSTABS=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfstabs
168 CTFSTRIP=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfstrip
169 NDRGEN=         $(ONBLD_TOOLS)/bin/$(MACH)/ndrgen
170 GENOFFSETS=     $(ONBLD_TOOLS)/bin/genoffsets
171 XREF=           $(ONBLD_TOOLS)/bin/xref
172 FIND=           /usr/bin/find
173 PERL=           /usr/bin/perl
174 PERL_VERSION=   5.10.0
175 PERL_PKGVERS=   -510
176 PERL_ARCH =             i86pc-solaris-64int
177 $(SPARC_BLD)PERL_ARCH = sun4-solaris-64int
178 PYTHON_26=      /usr/bin/python2.6
179 PYTHON=         $(PYTHON_26)
180 SORT=           /usr/bin/sort
181 TOUCH=          /usr/bin/touch
182 WC=             /usr/bin/wc
183 XARGS=          /usr/bin/xargs
184 ELFEDIT=        /usr/bin/elfedit
185 ELFSIGN=        /usr/bin/elfsign
186 DTRACE=         /usr/sbin/dtrace -xnolibs
187 UNIQ=           /usr/bin/uniq
188 TAR=            /usr/bin/tar
189 ASTBINDIR=      /usr/ast/bin
190 MSGCC=          $(ASTBINDIR)/msgcc

192 FILEMODE=       644
193 DIRMODE=        755
```



```
195 # Declare that nothing should be built in parallel.
196 # Individual Makefiles can use the .PARALLEL target to declare otherwise.
197 .NO_PARALLEL:

199 # For stylistic checks
200 #
201 # Note that the X and C checks are not used at this time and may need
202 # modification when they are actually used.
203 #
204 CSTYLE=         $(ONBLD_TOOLS)/bin/cstyle
205 CSTYLE_TAIL=
206 HDRCHK=         $(ONBLD_TOOLS)/bin/hdrchk
207 HDRCHK_TAIL=
208 JSTYLE=         $(ONBLD_TOOLS)/bin/jstyle

210 DOT_H_CHECK=    \
211         @$(ECHO) "checking $<"; $(CSTYLE) $< $(CSTYLE_TAIL); \
212         $(HDRCHK) $< $(HDRCHK_TAIL)

214 DOT_X_CHECK=    \
215         @$(ECHO) "checking $<"; $(RPCGEN) -C -h $< | $(CSTYLE) $(CSTYLE_TAIL); \
216         $(RPCGEN) -C -h $< | $(HDRCHK) $< $(HDRCHK_TAIL)

218 DOT_C_CHECK=    \
219         @$(ECHO) "checking $<"; $(CSTYLE) $< $(CSTYLE_TAIL)

221 MANIFEST_CHECK= \
222         @$(ECHO) "checking $<"; \
223         SVCCFG_DTD=$(SRC)/cmd/svc/dtd/service_bundle.dtd.1 \
224         SVCCFG_REPOSITORY=$(SRC)/cmd/svc/seed/global.db \
225         SVCCFG_CONFIGD_PATH=$(SRC)/cmd/svc/configd/svc.configd-native \
226         $(SRC)/cmd/svc/svccfg/svccfg-native validate $<

228 INS.file=       $(RM) $@; $(INS) -s -m $(FILEMODE) -f $(@D) $<
229 INS.dir=        $(INS) -s -d -m $(DIRMODE) $@
230 # installs and renames at once
231 #
232 INS.rename=     $(INS.file); $(MV) $(@D)/$(<F) $@

234 # install a link
235 INSLINKTARGET=  $<
236 INS.link=       $(RM) $@; $(LN) $(INSLINKTARGET) $@
237 INS.symlink=    $(RM) $@; $(SYMLINK) $(INSLINKTARGET) $@

239 #
240 # Python bakes the mtime of the .py file into the compiled .pyc and
241 # rebuilds if the baked-in mtime != the mtime of the source file
242 # (rather than only if it's less than), thus when installing python
243 # files we must make certain to not adjust the mtime of the source
244 # (.py) file.
245 #
246 INS.pyfile=     $(INS.file); $(TOUCH) -r $< $@

248 # MACH must be set in the shell environment per uname -p on the build host
249 # More specific architecture variables should be set in lower makefiles.
250 #
251 # MACH64 is derived from MACH, and BUILD64 is set to '#' for
252 # architectures on which we do not build 64-bit versions.
253 # (There are no such architectures at the moment.)
254 #
255 # Set BUILD64=# in the environment to disable 64-bit amd64
256 # builds on i386 machines.

258 MACH64_1=       $(MACH:sparc=sparcv9)
259 MACH64=         $(MACH64_1:i386=amd64)
```

```
261 MACH32_1=       $(MACH:sparc=sparcv7)
262 MACH32=         $(MACH32_1:i386=i86)

264 sparc_BUILD64=
265 i386_BUILD64=
266 BUILD64=        $($(MACH)_BUILD64)

268 #
269 # C compiler mode. Future compilers may change the default on us,
270 # so force extended ANSI mode globally. Lower level makefiles can
271 # override this by setting CCMODE.
272 #
273 CCMODE=                 -Xa
274 CCMODE64=               -Xa

276 #
277 # C compiler verbose mode. This is so we can enable it globally,
278 # but turn it off in the lower level makefiles of things we cannot
279 # (or aren't going to) fix.
280 #
281 CCVERBOSE=              -v

283 # set this to the secret flag "-Wc,-Qiselect-v9abiwarn=1" to get warnings
284 # from the compiler about places the -xarch=v9 may differ from -xarch=v9c.
285 V9ABIWARN=

287 # set this to the secret flag "-Wc,-Qiselect-regsym=0" to disable register
288 # symbols (used to detect conflicts between objects that use global registers)
289 # we disable this now for safety, and because genunix doesn't link with
290 # this feature (the v9 default) enabled.
291 #
292 # REGSYM is separate since the C++ driver syntax is different.
293 CCREGSYM=               -Wc,-Qiselect-regsym=0
294 CCCREGSYM=              -Qoption cg -Qiselect-regsym=0

296 # Prevent the removal of static symbols by the SPARC code generator (cg).
297 # The x86 code generator (ube) does not remove such symbols and as such
298 # using this workaround is not applicable for x86.
299 #
300 CCSTATICSYM=            -Wc,-Qassembler-ounrefsym=0
301 #
302 # generate 32-bit addresses in the v9 kernel. Saves memory.
303 CCABS32=                -Wc,-xcode=abs32
304 #
305 # generate v9 code which tolerates callers using the v7 ABI, for the sake of
306 # system calls.
307 CC32BITCALLERS=         -_gcc=-massume-32bit-callers

309 # GCC, especially, is increasingly beginning to auto-inline functions and
310 # sadly does so separately not under the general -fno-inline-functions
311 # Additionally, we wish to prevent optimisations which cause GCC to clone
312 # functions -- in particular, these may cause unhelpful symbols to be
313 # emitted instead of function names
314 CCNOAUTOINLINE= -_gcc=-fno-inline-small-functions \
315         -_gcc=-fno-inline-functions-called-once \
316         -_gcc=-fno-ipa-cp

318 # One optimization the compiler might perform is to turn this:
319 #       #pragma weak foo
320 #       extern int foo;
321 #       if (&foo)
322 #               foo = 5;
323 # into
324 #       foo = 5;
325 # Since we do some of this (foo might be referenced in common kernel code
```



```
326 # but provided only for some cpu modules or platforms), we disable this
327 # optimization.
328 #
329 sparc_CCUNBOUND = -Wd,-xsafe=unboundsym
330 i386_CCUNBOUND  =
331 CCUNBOUND       = $($(MACH)_CCUNBOUND)

333 #
334 # compiler '-xarch' flag. This is here to centralize it and make it
335 # overridable for testing.
336 sparc_XARCH=    -m32
337 sparcv9_XARCH=  -m64
338 i386_XARCH=
339 amd64_XARCH=    -m64 -Ui386 -U__i386

341 # assembler '-xarch' flag.  Different from compiler '-xarch' flag.
342 sparc_AS_XARCH=         -xarch=v8plus
343 sparcv9_AS_XARCH=       -xarch=v9
344 i386_AS_XARCH=
345 amd64_AS_XARCH=         -xarch=amd64 -P -Ui386 -U__i386

347 #
348 # These flags define what we need to be 'standalone' i.e. -not- part
349 # of the rather more cosy userland environment.  This basically means
350 # the kernel.
351 #
352 # XX64  future versions of gcc will make -mcmodel=kernel imply -mno-red-zone
353 #
354 sparc_STAND_FLAGS=      -_gcc=-ffreestanding
355 sparcv9_STAND_FLAGS=    -_gcc=-ffreestanding
356 # Disabling MMX also disables 3DNow, disabling SSE also disables all later
357 # additions to SSE (SSE2, AVX ,etc.)
358 NO_SIMD=                -_gcc=-mno-mmx -_gcc=-mno-sse
359 i386_STAND_FLAGS=       -_gcc=-ffreestanding $(NO_SIMD)
360 amd64_STAND_FLAGS=      -xmodel=kernel $(NO_SIMD)

362 SAVEARGS=               -Wu,-save_args
363 amd64_STAND_FLAGS       += $(SAVEARGS)

365 STAND_FLAGS_32 = $($(MACH)_STAND_FLAGS)
366 STAND_FLAGS_64 = $($(MACH64)_STAND_FLAGS)

368 #
369 # disable the incremental linker
370 ILDOFF=                 -xildoff
371 #
372 XDEPEND=                -xdepend
373 XFFLAG=                 -xF=%all
374 XESS=                   -xs
375 XSTRCONST=              -xstrconst

377 #
378 # turn warnings into errors (C)
379 CERRWARN = -errtags=yes -errwarn=%all
380 CERRWARN += -erroff=E_EMPTY_TRANSLATION_UNIT
381 CERRWARN += -erroff=E_STATEMENT_NOT_REACHED

383 CERRWARN += -_gcc=-Wno-missing-braces
384 CERRWARN += -_gcc=-Wno-sign-compare
385 CERRWARN += -_gcc=-Wno-unknown-pragmas
386 CERRWARN += -_gcc=-Wno-unused-parameter
387 CERRWARN += -_gcc=-Wno-missing-field-initializers

389 # Unfortunately, this option can misfire very easily and unfixably.
390 CERRWARN +=     -_gcc=-Wno-array-bounds
```

```
392 # DEBUG v. -nd make for frequent unused variables, empty conditions, etc. in
393 # -nd builds
394 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-unused
395 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-empty-body

397 #
398 # turn warnings into errors (C++)
399 CCERRWARN=              -xwe

401 # C99 mode
402 C99_ENABLE=     -xc99=%all
403 C99_DISABLE=    -xc99=%none
404 C99MODE=        $(C99_DISABLE)
405 C99LMODE=       $(C99MODE:-xc99%=-Xc99%)

407 # In most places, assignments to these macros should be appended with +=
408 # (CPPFLAGS.master allows values to be prepended to CPPFLAGS).
409 sparc_CFLAGS=   $(sparc_XARCH) $(CCSTATICSYM)
410 sparcv9_CFLAGS= $(sparcv9_XARCH) -dalign $(CCVERBOSE) $(V9ABIWARN) $(CCREGSYM) \
411                 $(CCSTATICSYM)
412 i386_CFLAGS=    $(i386_XARCH)
413 amd64_CFLAGS=   $(amd64_XARCH)

415 sparc_ASFLAGS=  $(sparc_AS_XARCH)
416 sparcv9_ASFLAGS=$(sparcv9_AS_XARCH)
417 i386_ASFLAGS=   $(i386_AS_XARCH)
418 amd64_ASFLAGS=  $(amd64_AS_XARCH)

420 #
421 sparc_COPTFLAG=         -xO3
422 sparcv9_COPTFLAG=       -xO3
423 i386_COPTFLAG=          -O
424 amd64_COPTFLAG=         -xO3

426 COPTFLAG= $($(MACH)_COPTFLAG)
427 COPTFLAG64= $($(MACH64)_COPTFLAG)

429 # When -g is used, the compiler globalizes static objects
430 # (gives them a unique prefix). Disable that.
431 CNOGLOBAL= -W0,-noglobal

433 # Direct the Sun Studio compiler to use a static globalization prefix based on t
434 # name of the module rather than something unique. Otherwise, objects
435 # will not build deterministically, as subsequent compilations of identical
436 # source will yeild objects that always look different.
437 #
438 # In the same spirit, this will also remove the date from the N_OPT stab.
439 CGLOBALSTATIC= -W0,-xglobalstatic

441 # Sometimes we want all symbols and types in debugging information even
442 # if they aren't used.
443 CALLSYMS=       -W0,-xdbggen=no%usedonly

445 #
446 # Default debug format for Sun Studio 11 is dwarf, so force it to
447 # generate stabs.
448 #
449 DEBUGFORMAT=    -xdebugformat=stabs

451 #
452 # Flags used to build in debug mode for ctf generation.  Bugs in the Devpro
453 # compilers currently prevent us from building with cc-emitted DWARF.
454 #
455 CTF_FLAGS_sparc = -g -Wc,-Qiselect-T1 $(C99MODE) $(CNOGLOBAL) $(CDWARFSTR)
456 CTF_FLAGS_i386  = -g $(C99MODE) $(CNOGLOBAL) $(CDWARFSTR)
```



```
458 CTF_FLAGS_sparcv9       = $(CTF_FLAGS_sparc)
459 CTF_FLAGS_amd64         = $(CTF_FLAGS_i386)

461 # Sun Studio produces broken userland code when saving arguments.
462 $(__GNUC)CTF_FLAGS_amd64 += $(SAVEARGS)

464 CTF_FLAGS_32    = $(CTF_FLAGS_$(MACH)) $(DEBUGFORMAT)
465 CTF_FLAGS_64    = $(CTF_FLAGS_$(MACH64)) $(DEBUGFORMAT)
466 CTF_FLAGS       = $(CTF_FLAGS_32)

468 #
469 # Flags used with genoffsets
470 #
471 GOFLAGS = -_noecho \
472         $(CALLSYMS) \
473         $(CDWARFSTR)

475 OFFSETS_CREATE = $(GENOFFSETS) -s $(CTFSTABS) -r $(CTFCONVERT) \
476         $(CC) $(GOFLAGS) $(CFLAGS) $(CPPFLAGS)

478 OFFSETS_CREATE64 = $(GENOFFSETS) -s $(CTFSTABS) -r $(CTFCONVERT) \
479         $(CC) $(GOFLAGS) $(CFLAGS64) $(CPPFLAGS)

481 #
482 # tradeoff time for space (smaller is better)
483 #
484 sparc_SPACEFLAG         = -xspace -W0,-Lt
485 sparcv9_SPACEFLAG       = -xspace -W0,-Lt
486 i386_SPACEFLAG          = -xspace
487 amd64_SPACEFLAG         =

489 SPACEFLAG               = $($(MACH)_SPACEFLAG)
490 SPACEFLAG64             = $($(MACH64)_SPACEFLAG)

492 #
493 # The Sun Studio 11 compiler has changed the behaviour of integer
494 # wrap arounds and so a flag is needed to use the legacy behaviour
495 # (without this flag panics/hangs could be exposed within the source).
496 #
497 sparc_IROPTFLAG         = -W2,-xwrap_int
498 sparcv9_IROPTFLAG       = -W2,-xwrap_int
499 i386_IROPTFLAG          =
500 amd64_IROPTFLAG         =

502 IROPTFLAG               = $($(MACH)_IROPTFLAG)
503 IROPTFLAG64             = $($(MACH64)_IROPTFLAG)

505 sparc_XREGSFLAG         = -xregs=no%appl
506 sparcv9_XREGSFLAG       = -xregs=no%appl
507 i386_XREGSFLAG          =
508 amd64_XREGSFLAG         =

510 XREGSFLAG               = $($(MACH)_XREGSFLAG)
511 XREGSFLAG64             = $($(MACH64)_XREGSFLAG)

513 # dmake SOURCEDEBUG=yes ... enables source-level debugging information, and
514 # avoids stripping it.
515 SOURCEDEBUG     = $(POUND_SIGN)
516 SRCDBGBLD       = $(SOURCEDEBUG:yes=)

518 #
519 # These variables are intended ONLY for use by developers to safely pass extra
520 # flags to the compilers without unintentionally overriding Makefile-set
521 # flags.  They should NEVER be set to any value in a Makefile.
522 #
523 # They come last in the associated FLAGS variable such that they can
```

```
524 # explicitly override things if necessary, there are gaps in this, but it's
525 # the best we can manage.
526 #
527 CUSERFLAGS              =
528 CUSERFLAGS64            = $(CUSERFLAGS)
529 CCUSERFLAGS             =
530 CCUSERFLAGS64           = $(CCUSERFLAGS)

532 CSOURCEDEBUGFLAGS       =
533 CCSOURCEDEBUGFLAGS      =
534 $(SRCDBGBLD)CSOURCEDEBUGFLAGS   = -g -xs
535 $(SRCDBGBLD)CCSOURCEDEBUGFLAGS  = -g -xs

537 CFLAGS=         $(COPTFLAG) $($(MACH)_CFLAGS) $(SPACEFLAG) $(CCMODE) \
538                 $(ILDOFF) $(CERRWARN) $(C99MODE) $(CCUNBOUND) $(IROPTFLAG) \
539                 $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CSOURCEDEBUGFLAGS) \
540                 $(CUSERFLAGS)
541 CFLAGS64=       $(COPTFLAG64) $($(MACH64)_CFLAGS) $(SPACEFLAG64) $(CCMODE64) \
542                 $(ILDOFF) $(CERRWARN) $(C99MODE) $(CCUNBOUND) $(IROPTFLAG64) \
543                 $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CSOURCEDEBUGFLAGS) \
544                 $(CUSERFLAGS64)
545 #
546 # Flags that are used to build parts of the code that are subsequently
547 # run on the build machine (also known as the NATIVE_BUILD).
548 #
549 NATIVE_CFLAGS=  $(COPTFLAG) $($(NATIVE_MACH)_CFLAGS) $(CCMODE) \
550                 $(ILDOFF) $(CERRWARN) $(C99MODE) $($(NATIVE_MACH)_CCUNBOUND) \
551                 $(IROPTFLAG) $(CGLOBALSTATIC) $(CCNOAUTOINLINE) \
552                 $(CSOURCEDEBUGFLAGS) $(CUSERFLAGS)

554 DTEXTDOM=-DTEXT_DOMAIN=\"$(TEXT_DOMAIN)\"       # For messaging.
555 DTS_ERRNO=-D_TS_ERRNO
556 CPPFLAGS.master=$(DTEXTDOM) $(DTS_ERRNO) \
557         $(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) $(ENVCPPFLAGS4) \
558         $(ADJUNCT_PROTO:%=-I%/usr/include)
559 CPPFLAGS.native=$(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) \
560                 $(ENVCPPFLAGS4) -I$(NATIVE_ADJUNCT)/include
561 CPPFLAGS=       $(CPPFLAGS.master)
562 AS_CPPFLAGS=    $(CPPFLAGS.master)
563 JAVAFLAGS=      -source 1.6 -target 1.6 -Xlint:deprecation,-options

565 #
566 # For source message catalogue
567 #
568 .SUFFIXES: $(SUFFIXES) .i .po
569 MSGROOT= $(ROOT)/catalog
570 MSGDOMAIN= $(MSGROOT)/$(TEXT_DOMAIN)
571 MSGDOMAINPOFILE = $(MSGDOMAIN)/$(POFILE)
572 DCMSGDOMAIN= $(MSGROOT)/LC_TIME/$(TEXT_DOMAIN)
573 DCMSGDOMAINPOFILE = $(DCMSGDOMAIN)/$(DCFILE:.dc=.po)

575 CLOBBERFILES += $(POFILE) $(POFILES)
576 COMPILE.cpp= $(CC) -E -C $(CFLAGS) $(CPPFLAGS)
577 XGETTEXT= /usr/bin/xgettext
578 XGETFLAGS= -c TRANSLATION_NOTE
579 GNUXGETTEXT= /usr/gnu/bin/xgettext
580 GNUXGETFLAGS= --add-comments=TRANSLATION_NOTE --keyword=_ \
581         --strict --no-location --omit-header
582 BUILD.po= $(XGETTEXT) $(XGETFLAGS) -d $(<F) $<.i ;\
583         $(RM)   $@ ;\
584         $(SED) "/^domain/d" < $(<F).po > $@ ;\
585         $(RM) $(<F).po $<.i

587 #
588 # This is overwritten by local Makefile when PROG is a list.
589 #
```



```
590 POFILE= $(PROG).po

592 sparc_CCFLAGS=          -cg92 -compat=4 \
593                         -Qoption ccfe -messages=no%anachronism \
594                         $(CCERRWARN)
595 sparcv9_CCFLAGS=        $(sparcv9_XARCH) -dalign -compat=5 \
596                         -Qoption ccfe -messages=no%anachronism \
597                         -Qoption ccfe -features=no%conststrings \
598                         $(CCCREGSYM) \
599                         $(CCERRWARN)
600 i386_CCFLAGS=           -compat=4 \
601                         -Qoption ccfe -messages=no%anachronism \
602                         -Qoption ccfe -features=no%conststrings \
603                         $(CCERRWARN)
604 amd64_CCFLAGS=          $(amd64_XARCH) -compat=5 \
605                         -Qoption ccfe -messages=no%anachronism \
606                         -Qoption ccfe -features=no%conststrings \
607                         $(CCERRWARN)

609 sparc_CCOPTFLAG=        -O
610 sparcv9_CCOPTFLAG=      -O
611 i386_CCOPTFLAG=         -O
612 amd64_CCOPTFLAG=        -O

614 CCOPTFLAG=      $($(MACH)_CCOPTFLAG)
615 CCOPTFLAG64=    $($(MACH64)_CCOPTFLAG)
616 CCFLAGS=        $(CCOPTFLAG) $($(MACH)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
617                 $(CCUSERFLAGS)
618 CCFLAGS64=      $(CCOPTFLAG64) $($(MACH64)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
619                 $(CCUSERFLAGS64)

621 #
622 #
623 #
624 ELFWRAP_FLAGS   =
625 ELFWRAP_FLAGS64 =       -64

627 #
628 # Various mapfiles that are used throughout the build, and delivered to
629 # /usr/lib/ld.
630 #
631 MAPFILE.NED_i386 =      $(SRC)/common/mapfiles/common/map.noexdata
632 MAPFILE.NED_sparc =
633 MAPFILE.NED =           $(MAPFILE.NED_$(MACH))
634 MAPFILE.PGA =           $(SRC)/common/mapfiles/common/map.pagealign
635 MAPFILE.NES =           $(SRC)/common/mapfiles/common/map.noexstk
636 MAPFILE.FLT =           $(SRC)/common/mapfiles/common/map.filter
637 MAPFILE.LEX =           $(SRC)/common/mapfiles/common/map.lex.yy

639 #
640 # Generated mapfiles that are compiler specific, and used throughout the
641 # build.  These mapfiles are not delivered in /usr/lib/ld.
642 #
643 MAPFILE.NGB_sparc=      $(SRC)/common/mapfiles/gen/sparc_cc_map.noexeglobs
644 $(__GNUC64)MAPFILE.NGB_sparc= \
645                         $(SRC)/common/mapfiles/gen/sparc_gcc_map.noexeglobs
646 MAPFILE.NGB_sparcv9=    $(SRC)/common/mapfiles/gen/sparcv9_cc_map.noexeglobs
647 $(__GNUC64)MAPFILE.NGB_sparcv9= \
648                         $(SRC)/common/mapfiles/gen/sparcv9_gcc_map.noexeglobs
649 MAPFILE.NGB_i386=       $(SRC)/common/mapfiles/gen/i386_cc_map.noexeglobs
650 $(__GNUC64)MAPFILE.NGB_i386= \
651                         $(SRC)/common/mapfiles/gen/i386_gcc_map.noexeglobs
652 MAPFILE.NGB_amd64=      $(SRC)/common/mapfiles/gen/amd64_cc_map.noexeglobs
653 $(__GNUC64)MAPFILE.NGB_amd64= \
654                         $(SRC)/common/mapfiles/gen/amd64_gcc_map.noexeglobs
655 MAPFILE.NGB =           $(MAPFILE.NGB_$(MACH))
```

```
#
# A generic interface mapfile name, used by various dynamic objects to define
# the interfaces and interposers the object must export.
#
MAPFILE.INT =           mapfile-intf

#
# LDLIBS32 and LDLIBS64 can be set in the environment to override the following
# assignments.
#
# These environment settings make sure that no libraries are searched outside
# of the local workspace proto area:
#       LDLIBS32=-YP,$ROOT/lib:$ROOT/usr/lib
#       LDLIBS64=-YP,$ROOT/lib/$MACH64:$ROOT/usr/lib/$MACH64
#
LDLIBS32 =      $(ENVLDLIBS1) $(ENVLDLIBS2) $(ENVLDLIBS3)
LDLIBS32 +=     $(ADJUNCT_PROTO:%=-L%/usr/lib -L%/lib)
LDLIBS.cmd =    $(LDLIBS32)
LDLIBS.lib =    $(LDLIBS32)

LDLIBS64 =      $(ENVLDLIBS1:%=%/$(MACH64)) \
                $(ENVLDLIBS2:%=%/$(MACH64)) \
                $(ENVLDLIBS3:%=%/$(MACH64))
LDLIBS64 +=     $(ADJUNCT_PROTO:%=-L%/usr/lib/$(MACH64) -L%/lib/$(MACH64))

#
# Define compilation macros.
#
COMPILE.c=      $(CC) $(CFLAGS) $(CPPFLAGS) -c
COMPILE64.c=    $(CC) $(CFLAGS64) $(CPPFLAGS) -c
COMPILE.cc=     $(CCC) $(CCFLAGS) $(CPPFLAGS) -c
COMPILE64.cc=   $(CCC) $(CCFLAGS64) $(CPPFLAGS) -c
COMPILE.s=      $(AS) $(ASFLAGS) $(AS_CPPFLAGS)
COMPILE64.s=    $(AS) $(ASFLAGS) $($(MACH64)_AS_XARCH) $(AS_CPPFLAGS)
COMPILE.d=      $(DTRACE) -G -32
COMPILE64.d=    $(DTRACE) -G -64
COMPILE.b=      $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))
COMPILE64.b=    $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))

CLASSPATH=      .
COMPILE.java=   $(JAVAC) $(JAVAFLAGS) -classpath $(CLASSPATH)

#
# Link time macros
#
CCNEEDED                = -lC
CCEXTNEEDED             = -lCrun -lCstd
$(__GNUC)CCNEEDED       = -L$(GCCLIBDIR) -lstdc++ -lgcc_s
$(__GNUC)CCEXTNEEDED    = $(CCNEEDED)

LINK.c=         $(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS)
LINK64.c=       $(CC) $(CFLAGS64) $(CPPFLAGS) $(LDFLAGS)
NORUNPATH=      -norunpath -nolib
LINK.cc=        $(CCC) $(CCFLAGS) $(CPPFLAGS) $(NORUNPATH) \
                $(LDFLAGS) $(CCNEEDED)
LINK64.cc=      $(CCC) $(CCFLAGS64) $(CPPFLAGS) $(NORUNPATH) \
                $(LDFLAGS) $(CCNEEDED)

#
# lint macros
#
# Note that the undefine of __PRAGMA_REDEFINE_EXTNAME can be removed once
# ON is built with a version of lint that has the fix for 4484186.
#
ALWAYS_LINT_DEFS =      -errtags=yes -s
```



```
ALWAYS_LINT_DEFS +=     -erroff=E_PTRDIFF_OVERFLOW
ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_NARROW_CONV
ALWAYS_LINT_DEFS +=     -U__PRAGMA_REDEFINE_EXTNAME
ALWAYS_LINT_DEFS +=     $(C99LMODE)
ALWAYS_LINT_DEFS +=     -errsecurity=$(SECLEVEL)
ALWAYS_LINT_DEFS +=     -erroff=E_SEC_CREAT_WITHOUT_EXCL
ALWAYS_LINT_DEFS +=     -erroff=E_SEC_FORBIDDEN_WARN_CREAT
# XX64 -- really only needed for amd64 lint
ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_INT_TO_SMALL_INT
ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_CONST_TO_SMALL_INT
ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_TO_SMALL_INT
ALWAYS_LINT_DEFS +=     -erroff=E_CAST_TO_PTR_FROM_INT
ALWAYS_LINT_DEFS +=     -erroff=E_COMP_INT_WITH_LARGE_INT
ALWAYS_LINT_DEFS +=     -erroff=E_INTEGRAL_CONST_EXP_EXPECTED
ALWAYS_LINT_DEFS +=     -erroff=E_PASS_INT_TO_SMALL_INT
ALWAYS_LINT_DEFS +=     -erroff=E_PTR_CONV_LOSES_BITS

# This forces lint to pick up note.h and sys/note.h from Devpro rather than
# from the proto area.  The note.h that ON delivers would disable NOTE().
ONLY_LINT_DEFS =        -I$(SPRO_VROOT)/prod/include/lint

SECLEVEL=       core
LINT.c=         $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS) $(CPPFLAGS) \
                $(ALWAYS_LINT_DEFS)
LINT64.c=       $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS64) $(CPPFLAGS) \
                $(ALWAYS_LINT_DEFS)
LINT.s=         $(LINT.c)

# For some future builds, NATIVE_MACH and MACH might be different.
# Therefore, NATIVE_MACH needs to be redefined in the
# environment as ‘uname -p‘ to override this macro.
#
# For now at least, we cross-compile amd64 on i386 machines.
NATIVE_MACH=    $(MACH:amd64=i386)

# Define native compilation macros
#

# Base directory where compilers are loaded.
# Defined here so it can be overridden by developer.
#
SPRO_ROOT=              $(BUILD_TOOLS)/SUNWspro
SPRO_VROOT=             $(SPRO_ROOT)/SS12
GNU_ROOT=               $(SFW_ROOT)

# Till SS12u1 formally becomes the NV CBE, LINT is hard
# coded to be picked up from the $SPRO_ROOT/sunstudio12.1/
# location. Impacted variables are sparc_LINT, sparcv9_LINT,
# i386_LINT, amd64_LINT.
# Reset them when SS12u1 is rolled out.
#

# Specify platform compiler versions for languages
# that we use (currently only c and c++).
#
sparc_CC=               $(ONBLD_TOOLS)/bin/$(MACH)/cw -_cc
$(__GNUC)sparc_CC=      $(ONBLD_TOOLS)/bin/$(MACH)/cw -_gcc
sparc_CCC=              $(ONBLD_TOOLS)/bin/$(MACH)/cw -_CC
$(__GNUC)sparc_CCC=     $(ONBLD_TOOLS)/bin/$(MACH)/cw -_g++
sparc_CPP=              /usr/ccs/lib/cpp
sparc_AS=               /usr/ccs/bin/as -xregsym=no
sparc_LD=               /usr/ccs/bin/ld
sparc_LINT=             $(SPRO_ROOT)/sunstudio12.1/bin/lint

sparcv9_CC=             $(ONBLD_TOOLS)/bin/$(MACH)/cw -_cc
$(__GNUC64)sparcv9_CC=  $(ONBLD_TOOLS)/bin/$(MACH)/cw -_gcc
```

```
788 sparcv9_CCC=            $(ONBLD_TOOLS)/bin/$(MACH)/cw -_CC
789 $(__GNUC64)sparcv9_CCC= $(ONBLD_TOOLS)/bin/$(MACH)/cw -_g++
790 sparcv9_CPP=            /usr/ccs/lib/cpp
791 sparcv9_AS=             /usr/ccs/bin/as -xregsym=no
792 sparcv9_LD=             /usr/ccs/bin/ld
793 sparcv9_LINT=           $(SPRO_ROOT)/sunstudio12.1/bin/lint

795 i386_CC=               $(ONBLD_TOOLS)/bin/$(MACH)/cw -_cc
796 $(__GNUC)i386_CC=      $(ONBLD_TOOLS)/bin/$(MACH)/cw -_gcc
797 i386_CCC=              $(ONBLD_TOOLS)/bin/$(MACH)/cw -_CC
798 $(__GNUC)i386_CCC=     $(ONBLD_TOOLS)/bin/$(MACH)/cw -_g++
799 i386_CPP=              /usr/ccs/lib/cpp
800 i386_AS=               /usr/ccs/bin/as
801 $(__GNUC)i386_AS=      $(ONBLD_TOOLS)/bin/$(MACH)/aw
802 i386_LD=               /usr/ccs/bin/ld
803 i386_LINT=             $(SPRO_ROOT)/sunstudio12.1/bin/lint

805 amd64_CC=              $(ONBLD_TOOLS)/bin/$(MACH)/cw -_cc
806 $(__GNUC64)amd64_CC=   $(ONBLD_TOOLS)/bin/$(MACH)/cw -_gcc
807 amd64_CCC=             $(ONBLD_TOOLS)/bin/$(MACH)/cw -_CC
808 $(__GNUC64)amd64_CCC=  $(ONBLD_TOOLS)/bin/$(MACH)/cw -_g++
809 amd64_CPP=             /usr/ccs/lib/cpp
810 amd64_AS=              $(ONBLD_TOOLS)/bin/$(MACH)/aw
811 amd64_LD=              /usr/ccs/bin/ld
812 amd64_LINT=            $(SPRO_ROOT)/sunstudio12.1/bin/lint

814 NATIVECC=              $($(NATIVE_MACH)_CC)
815 NATIVECCC=             $($(NATIVE_MACH)_CCC)
816 NATIVECPP=             $($(NATIVE_MACH)_CPP)
817 NATIVEAS=              $($(NATIVE_MACH)_AS)
818 NATIVELD=              $($(NATIVE_MACH)_LD)
819 NATIVELINT=            $($(NATIVE_MACH)_LINT)

821 #
822 # Makefile.master.64 overrides these settings
823 #
824 CC=                    $(NATIVECC)
825 CCC=                   $(NATIVECCC)
826 CPP=                   $(NATIVECPP)
827 AS=                    $(NATIVEAS)
828 LD=                    $(NATIVELD)
829 LINT=                  $(NATIVELINT)

831 # The real compilers used for this build
832 CW_CC_CMD=             $(CC) -_compiler
833 CW_CCC_CMD=            $(CCC) -_compiler
834 REAL_CC=               $(CW_CC_CMD:sh)
835 REAL_CCC=              $(CW_CCC_CMD:sh)

837 # Pass -Y flag to cpp (method of which is release-dependent)
838 CCYFLAG=               -Y I,

840 BDIRECT=        -Bdirect
841 BDYNAMIC=       -Bdynamic
842 BLOCAL=         -Blocal
843 BNODIRECT=      -Bnodirect
844 BREDUCE=        -Breduce
845 BSTATIC=        -Bstatic

847 ZDEFS=          -zdefs
848 ZDIRECT=        -zdirect
849 ZIGNORE=        -zignore
850 ZINITFIRST=     -zinitfirst
851 ZINTERPOSE=     -zinterpose
852 ZLAZYLOAD=      -zlazyload
853 ZLOADFLTR=      -zloadfltr
```



```
854 ZMULDEFS=       -zmuldefs
855 ZNODEFAULTLIB=  -znodefaultlib
856 ZNODEFS=        -znodefs
857 ZNODELETE=      -znodelete
858 ZNODLOPEN=      -znodlopen
859 ZNODUMP=        -znodump
860 ZNOLAZYLOAD=    -znolazyload
861 ZNOLDYNSYM=     -znoldynsym
862 ZNORELOC=       -znoreloc
863 ZNOVERSION=     -znoversion
864 ZRECORD=        -zrecord
865 ZREDLOCSYM=     -zredlocsym
866 ZTEXT=          -ztext
867 ZVERBOSE=       -zverbose

869 GSHARED=        -G
870 CCMT=           -mt

872 # Handle different PIC models on different ISAs
873 # (May be overridden by lower-level Makefiles)

875 sparc_C_PICFLAGS =      -K pic
876 sparcv9_C_PICFLAGS =    -K pic
877 i386_C_PICFLAGS =       -K pic
878 amd64_C_PICFLAGS =      -K pic
879 C_PICFLAGS =            $($(MACH)_C_PICFLAGS)
880 C_PICFLAGS64 =          $($(MACH64)_C_PICFLAGS)

882 sparc_C_BIGPICFLAGS =   -K PIC
883 sparcv9_C_BIGPICFLAGS = -K PIC
884 i386_C_BIGPICFLAGS =    -K PIC
885 amd64_C_BIGPICFLAGS =   -K PIC
886 C_BIGPICFLAGS =         $($(MACH)_C_BIGPICFLAGS)
887 C_BIGPICFLAGS64 =       $($(MACH64)_C_BIGPICFLAGS)

889 # CC requires there to be no space between '-K' and 'pic' or 'PIC'.
890 sparc_CC_PICFLAGS =     -Kpic
891 sparcv9_CC_PICFLAGS =   -KPIC
892 i386_CC_PICFLAGS =      -Kpic
893 amd64_CC_PICFLAGS =     -Kpic
894 CC_PICFLAGS =           $($(MACH)_CC_PICFLAGS)
895 CC_PICFLAGS64 =         $($(MACH64)_CC_PICFLAGS)

897 AS_PICFLAGS=            $(C_PICFLAGS)
898 AS_BIGPICFLAGS=         $(C_BIGPICFLAGS)

900 #
901 # Default label for CTF sections
902 #
903 CTFCVTFLAGS=            -i -L VERSION

905 #
906 # Override to pass module-specific flags to ctfmerge.  Currently used only by
907 # krtld to turn on fuzzy matching, and source-level debugging to inhibit
908 # stripping.
909 #
910 CTFMRGFLAGS=

912 CTFCONVERT_O            = $(CTFCONVERT) $(CTFCVTFLAGS) $@

914 ELFSIGN_O=      $(TRUE)
915 ELFSIGN_CRYPTO= $(ELFSIGN_O)
916 ELFSIGN_OBJECT= $(ELFSIGN_O)

918 # Rules (normally from make.rules) and macros which are used for post
919 # processing files. Normally, these do stripping of the comment section
```

```
# automatically.
#    RELEASE_CM:        Should be editted to reflect the release.
#    POST_PROCESS_O:    Post-processing for '.o' files.
#    POST_PROCESS_A:    Post-processing for '.a' files (currently null).
#    POST_PROCESS_SO:   Post-processing for '.so' files.
#    POST_PROCESS:      Post-processing for executable files (no suffix).
# Note that these macros are not completely generalized as they are to be
# used with the file name to be processed following.
#
# It is left as an exercise to Release Engineering to embellish the generation
# of the release comment string.
#
#       If this is a standard development build:
#               compress the comment section (mcs -c)
#               add the standard comment (mcs -a $(RELEASE_CM))
#               add the development specific comment (mcs -a $(DEV_CM))
#
#       If this is an installation build:
#               delete the comment section (mcs -d)
#               add the standard comment (mcs -a $(RELEASE_CM))
#               add the development specific comment (mcs -a $(DEV_CM))
#
#       If this is an release build:
#               delete the comment section (mcs -d)
#               add the standard comment (mcs -a $(RELEASE_CM))
#
# The following list of macros are used in the definition of RELEASE_CM
# which is used to label all binaries in the build:
#
#       RELEASE         Specific release of the build, eg: 5.2
#       RELEASE_MAJOR   Major version number part of $(RELEASE)
#       RELEASE_MINOR   Minor version number part of $(RELEASE)
#       VERSION         Version of the build (alpha, beta, Generic)
#       PATCHID         If this is a patch this value should contain
#                       the patchid value (eg: "Generic 100832-01"), otherwise
#                       it will be set to $(VERSION)
#       RELEASE_DATE    Date of the Release Build
#       PATCH_DATE      Date the patch was created, if this is blank it
#                       will default to the RELEASE_DATE
#
RELEASE_MAJOR=  5
RELEASE_MINOR=  11
RELEASE=        $(RELEASE_MAJOR).$(RELEASE_MINOR)
VERSION=        SunOS Development
PATCHID=        $(VERSION)
RELEASE_DATE=   release date not set
PATCH_DATE=     $(RELEASE_DATE)
RELEASE_CM=     "@($(POUND_SIGN))SunOS $(RELEASE) $(PATCHID) $(PATCH_DATE)"
DEV_CM=         "@($(POUND_SIGN))SunOS Internal Development: non-nightly build"

PROCESS_COMMENT=  @?${MCS} -d -a $(RELEASE_CM) -a $(DEV_CM)
$(RELEASE_BUILD)PROCESS_COMMENT=   @?${MCS} -d -a $(RELEASE_CM)

STRIP_STABS=                    $(STRIP) -x $@
$(SRCDBGBLD)STRIP_STABS=        :

POST_PROCESS_O=
POST_PROCESS_A=
POST_PROCESS_SO=        $(PROCESS_COMMENT) $@ ; $(STRIP_STABS) ; \
                        $(ELFSIGN_OBJECT)
POST_PROCESS=           $(PROCESS_COMMENT) $@ ; $(STRIP_STABS) ; \
                        $(ELFSIGN_OBJECT)

#
# chk4ubin is a tool that inspects a module for a symbol table
# ELF section size which can trigger an OBP bug on older platforms.
```



```
# This problem affects only specific sun4u bootable modules.
#
CHK4UBIN=       $(ONBLD_TOOLS)/bin/$(MACH)/chk4ubin
CHK4UBINFLAGS=
CHK4UBINARY=    $(CHK4UBIN) $(CHK4UBINFLAGS) $@

#
# PKGARCHIVE specifies the default location where packages should be
# placed if built.
#
$(RELEASE_BUILD)PKGARCHIVESUFFIX=       -nd
PKGARCHIVE=$(SRC)/../../packages/$(MACH)/nightly$(PKGARCHIVESUFFIX)

#
# The repositories will be created with these publisher settings.  To
# update an image to the resulting repositories, this must match the
# publisher name provided to "pkg set-publisher."
#
PKGPUBLISHER_REDIST=    on-nightly
PKGPUBLISHER_NONREDIST= on-extra

#       Default build rules which perform comment section post-processing.
#
.c:
        $(LINK.c) -o $@ $< $(LDLIBS)
        $(POST_PROCESS)
.c.o:
        $(COMPILE.c) $(OUTPUT_OPTION) $< $(CTFCONVERT_HOOK)
        $(POST_PROCESS_O)
.c.a:
        $(COMPILE.c) -o $% $<
        $(PROCESS_COMMENT) $%
        $(AR) $(ARFLAGS) $@ $%
        $(RM) $%
.s.o:
        $(COMPILE.s) -o $@ $<
        $(POST_PROCESS_O)
.s.a:
        $(COMPILE.s) -o $% $<
        $(PROCESS_COMMENT) $%
        $(AR) $(ARFLAGS) $@ $%
        $(RM) $%
.cc:
        $(LINK.cc) -o $@ $< $(LDLIBS)
        $(POST_PROCESS)
.cc.o:
        $(COMPILE.cc) $(OUTPUT_OPTION) $<
        $(POST_PROCESS_O)
.cc.a:
        $(COMPILE.cc) -o $% $<
        $(AR) $(ARFLAGS) $@ $%
        $(PROCESS_COMMENT) $%
        $(RM) $%
.y:
        $(YACC.y) $<
        $(LINK.c) -o $@ y.tab.c $(LDLIBS)
        $(POST_PROCESS)
        $(RM) y.tab.c
.y.o:
        $(YACC.y) $<
        $(COMPILE.c) -o $@ y.tab.c $(CTFCONVERT_HOOK)
        $(POST_PROCESS_O)
        $(RM) y.tab.c
.l:
        $(RM) $*.c
        $(LEX.l) $< > $*.c
```

```
1052         $(LINK.c) -o $@ $*.c -ll $(LDLIBS)
1053         $(POST_PROCESS)
1054         $(RM) $*.c
1055 .l.o:
1056         $(RM) $*.c
1057         $(LEX.l) $< > $*.c
1058         $(COMPILE.c) -o $@ $*.c $(CTFCONVERT_HOOK)
1059         $(POST_PROCESS_O)
1060         $(RM) $*.c

1062 .bin.o:
1063         $(COMPILE.b) -o $@ $<
1064         $(POST_PROCESS_O)

1066 .java.class:
1067         $(COMPILE.java) $<

1069 # Bourne and Korn shell script message catalog build rules.
1070 # We extract all gettext strings with sed(1) (being careful to permit
1071 # multiple gettext strings on the same line), weed out the dups, and
1072 # build the catalogue with awk(1).

1074 .sh.po .ksh.po:
1075         $(SED) -n -e ":a"                                       \
1076                   -e "h"                                        \
1077                   -e "s/.*gettext *\(\"[^\"]*\"\).*/\1/p"       \
1078                   -e "x"                                        \
1079                   -e "s/\(.*\)gettext *\"[^\"]*\"\(.*\)/\1\2/"  \
1080                   -e "t a"                                      \
1081                $< | sort -u | awk '{ print "msgid\t" $$0 "\nmsgstr" }' > $@

1083 #
1084 # Python and Perl executable and message catalog build rules.
1085 #
1086 .SUFFIXES: .pl .pm .py .pyc

1088 .pl:
1089         $(RM) $@;
1090         $(SED) -e "s@TEXT_DOMAIN@\"$(TEXT_DOMAIN)\"@" $< > $@;
1091         $(CHMOD) +x $@

1093 .py:
1094         $(RM) $@; $(CAT) $< > $@; $(CHMOD) +x $@

1096 .py.pyc:
1097         $(RM) $@
1098         $(PYTHON) -mpy_compile $<
1099         @[ $(<)c = $@ ] || $(MV) $(<)c $@

1101 .py.po:
1102         $(GNUXGETTEXT) $(GNUXGETFLAGS) -d $(<F:%.py=%) $< ;

1104 .pl.po .pm.po:
1105         $(XGETTEXT) $(XGETFLAGS) -d $(<F) $< ;
1106         $(RM)   $@ ;
1107         $(SED) "/^domain/d" < $(<F).po > $@ ;
1108         $(RM) $(<F).po

1110 #
1111 # When using xgettext, we want messages to go to the default domain,
1112 # rather than the specified one.  This special version of the
1113 # COMPILE.cpp macro effectively prevents expansion of TEXT_DOMAIN,
1114 # causing xgettext to put all messages into the default domain.
1115 #
1116 CPPFORPO=$(COMPILE.cpp:\"$(TEXT_DOMAIN)\"=TEXT_DOMAIN)
```



```
1118 .c.i:
1119         $(CPPFORPO) $< > $@

1121 .h.i:
1122         $(CPPFORPO) $< > $@

1124 .y.i:
1125         $(YACC) -d $<
1126         $(CPPFORPO) y.tab.c  > $@
1127         $(RM) y.tab.c

1129 .l.i:
1130         $(LEX) $<
1131         $(CPPFORPO) lex.yy.c  > $@
1132         $(RM) lex.yy.c

1134 .c.po:
1135         $(CPPFORPO) $< > $<.i
1136         $(BUILD.po)

1138 .cc.po:
1139         $(CPPFORPO) $< > $<.i
1140         $(BUILD.po)

1142 #endif /* ! codereview */
1143 .y.po:
1144         $(YACC) -d $<
1145         $(CPPFORPO) y.tab.c  > $<.i
1146         $(BUILD.po)
1147         $(RM) y.tab.c

1149 .l.po:
1150         $(LEX) $<
1151         $(CPPFORPO) lex.yy.c  > $<.i
1152         $(BUILD.po)
1153         $(RM) lex.yy.c

1155 #
1156 # Rules to perform stylistic checks
1157 #
1158 .SUFFIXES: .x .xml .check .xmlchk

1160 .h.check:
1161         $(DOT_H_CHECK)

1163 .x.check:
1164         $(DOT_X_CHECK)

1166 .xml.xmlchk:
1167         $(MANIFEST_CHECK)

1169 #
1170 # Include rules to render automated sccs get rules "safe".
1171 #
1172 include $(SRC)/Makefile.noget
```

```
**********************************************************
   10855 Wed May 20 12:19:45 2015
new/usr/src/cmd/Makefile
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

```
   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 #

  22 # Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
  23 # Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
  24 # Copyright (c) 2012 Joyent, Inc.  All rights reserved.
  25 # Copyright (c) 2012 by Delphix. All rights reserved.
  26 # Copyright (c) 2013 DEY Storage Systems, Inc. All rights reserved.
  27 # Copyright 2014 Garrett D'Amore <garrett@damore.org>

  29 include ../Makefile.master

  31 #
  32 #       Note that the commands 'lp', and 'perl' are first in
  33 #       the list, violating alphabetical order.  This is because they are very
  34 #       long-running and should be given the most wall-clock time for a
  35 #       parallel build.
  36 #
  37 #       Commands in the FIRST_SUBDIRS list are built before starting the build
  38 #       of other commands.  Currently this includes only 'isaexec' and
  39 #       'platexec'.  This is necessary because $(ROOT)/usr/lib/isaexec or
  40 #       $(ROOT)/usr/lib/platexec must exist when some other commands are built
  41 #       because their 'make install' creates a hard link to one of them.
  42 #
  43 #       Commands are listed one per line so that TeamWare can auto-merge most
  44 #       changes.
  45 #

  47 FIRST_SUBDIRS=          \
  48         isaexec         \
  49         platexec

  51 COMMON_SUBDIRS=         \
  52         allocate        \
  53         availdevs       \
  54         lp              \
  55         perl            \
  56         Adm             \
  57         abi             \
  58         adbgen          \
  59         acct            \
  60         acctadm         \
  61         arch            \
```



```
  62         asa             \
  63         ast             \
  64         audio           \
  65         auths           \
  66         autopush        \
  67         avs             \
  68         awk             \
  69         awk_xpg4        \
  70         backup          \
  71         banner          \
  72         bart            \
  73         basename        \
  74         bc              \
  75         bdiff           \
  76         beadm           \
  77         bnu             \
  78         boot            \
  79         busstat         \
  80         cal             \
  81         calendar        \
  82         captoinfo       \
  83         cat             \
  84         cdrw            \
  85         cfgadm          \
  86         checkeq         \
  87         checknr         \
  88         chgrp           \
  89         chmod           \
  90         chown           \
  91         chroot          \
  92         clear           \
  93         clinfo          \
  94         cmd-crypto      \
  95         cmd-inet        \
  96         col             \
  97         compress        \
  98         consadm         \
  99         coreadm         \
 100         cpio            \
 101         cpc             \
 102         cron            \
 103         crypt           \
 104         csh             \
 105         csplit          \
 106         ctrun           \
 107         ctstat          \
 108         ctwatch         \
 109         datadm          \
 110         date            \
 111         dc              \
 112         dd              \
 113         deroff          \
 114         devfsadm        \
 115         syseventd       \
 116         devctl          \
 117         devinfo         \
 118         devmgmt         \
 119         devprop         \
 120         dfs.cmds        \
 121         diff            \
 122         diff3           \
 123         diffmk          \
 124         dircmp          \
 125         dirname         \
 126         dis             \
 127         diskmgtd        \
```

```
128         dispadmin       \
129         dladm           \
130         dlstat          \
131         dmesg           \
132         dodatadm        \
133         dtrace          \
134         du              \
135         dumpadm         \
136         dumpcs          \
137         echo            \
138         ed              \
139         eeprom          \
140         egrep           \
141         eject           \
142         emul64ioctl     \
143         enhance         \
144         env             \
145         eqn             \
146         expand          \
147         expr            \
148         exstr           \
149         factor          \
150         false           \
151         fcinfo          \
152         fcoesvc         \
153         fdetach         \
154         fdformat        \
155         fdisk           \
156         filesync        \
157         fgrep           \
158         file            \
159         find            \
160         flowadm         \
161         flowstat        \
162         fm              \
163         fmt             \
164         fmthard         \
165         fmtmsg          \
166         fold            \
167         format          \
168         fs.d            \
169         fstyp           \
170         fuser           \
171         fwflash         \
172         gcore           \
173         gencat          \
174         geniconvtbl     \
175         genmsg          \
176         getconf         \
177         getdevpolicy    \
178         getent          \
179         getfacl         \
180         getmajor        \
181         getopt          \
182         gettext         \
183         gettxt          \
184         grep            \
185         grep_xpg4       \
186         groups          \
187         grpck           \
188         gss             \
189         hal             \
190         halt            \
191         head            \
192         hostid          \
193         hostname        \
```



```
194         hotplug         \
195         hotplugd        \
196         hwdata          \
197         ibd_upgrade     \
198         id              \
199         idmap           \
200         infocmp         \
201         init            \
202         initpkg         \
203         install.d       \
204         intrd           \
205         intrstat        \
206         ipcrm           \
207         ipcs            \
208         ipdadm          \
209         ipf             \
210         isainfo         \
211         isalist         \
212         itutools        \
213         iscsiadm        \
214         iscsid          \
215         iscsitsvc       \
216         isns            \
217         itadm           \
218         kbd             \
219         keyserv         \
220         killall         \
221         krb5            \
222         ksh             \
223         kvmstat         \
224         last            \
225         lastcomm        \
226         latencytop      \
227         ldap            \
228         ldapcachemgr    \
229         lgrpinfo        \
230         line            \
231         link            \
232         dlmgmtd         \
233         listen          \
234         loadkeys        \
235         locale          \
236         localedef       \
237         lockstat        \
238         locator         \
239         lofiadm         \
240         logadm          \
241         logger          \
242         login           \
243         logins          \
244         look            \
245         ls              \
246         luxadm          \
247         lvm             \
248         mach            \
249         mail            \
250         mailwrapper     \
251         mailx           \
252         make            \
253         makekey         \
254         man             \
255         mandoc          \
256         mdb             \
257         mesg            \
258         mkdir           \
259         mkfifo          \
```

```
260         mkfile          \
261         mkmsgs          \
262         mknod           \
263         mkpwdict        \
264         mktemp          \
265         modload         \
266         more            \
267         mpathadm        \
268         msgfmt          \
269         msgid           \
270         mt              \
271         mv              \
272         mvdir           \
273         ndmpadm         \
274         ndmpd           \
275         ndmpstat        \
276         netadm          \
277         netfiles        \
278         newform         \
279         newgrp          \
280         news            \
281         newtask         \
282         nice            \
283         nl              \
284         nlsadmin        \
285         nohup           \
286         nsadmin         \
287         nscd            \
288         oamuser         \
289         oawk            \
290         od              \
291         pack            \
292         pagesize        \
293         passmgmt        \
294         passwd          \
295         pathchk         \
296         pbind           \
297         pcidr           \
298         pcitool         \
299         pfexec          \
300         pfexecd         \
301         pginfo          \
302         pgstat          \
303         pgrep           \
304         picl            \
305         plimit          \
306         policykit       \
307         pools           \
308         power           \
309         powertop        \
310         ppgsz           \
311         pg              \
312         plockstat       \
313         pr              \
314         prctl           \
315         print           \
316         printf          \
317         priocntl        \
318         profiles        \
319         projadd         \
320         projects        \
321         prstat          \
322         prtconf         \
323         prtdiag         \
324         prtvtoc         \
325         ps              \
```



```
326         psradm          \
327         psrinfo         \
328         psrset          \
329         ptools          \
330         pwck            \
331         pwconv          \
332         pwd             \
333         pyzfs           \
334         raidctl         \
335         ramdiskadm      \
336         rcap            \
337         rcm_daemon      \
338         rctladm         \
339         refer           \
340         regcmp          \
341         renice          \
342         rexd            \
343         rm              \
344         rmdir           \
345         rmformat        \
346         rmmount         \
347         rmt             \
348         rmvolmgr        \
349         roles           \
350         rpcbind         \
351         rpcgen          \
352         rpcinfo         \
353         rpcsvc          \
354         runat           \
355         sa              \
356         saf             \
357         sasinfo         \
358         savecore        \
359         sbdadm          \
360         script          \
361         scsi            \
362         sdiff           \
363         sdpadm          \
364         sed             \
365         sendmail        \
366         setfacl         \
367         setmnt          \
368         setpgrp         \
369         setuname        \
370         sgs             \
371         sh              \
372         shcomp          \
373         smbios          \
374         smbsrv          \
375         smserverd       \
376         soelim          \
377         sort            \
378         spell           \
379         split           \
380         sqlite          \
381         srchtxt         \
382         srptadm         \
383         srptsvc         \
384         ssh             \
385         stat            \
386         stmfadm         \
387         stmfproxy       \
388         stmfsvc         \
389         stmsboot        \
390         streams         \
391         strings         \
```

```
392         su              \
393         sulogin         \
394         sunpc           \
395         svc             \
396         svr4pkg         \
397         swap            \
398         sync            \
399         sysdef          \
400         syseventadm     \
401         syslogd         \
402         tabs            \
403         tail            \
404         tar             \
405         tbl             \
406         tcopy           \
407         tcpd            \
408         terminfo        \
409         th_tools        \
410         tic             \
411         time            \
412         tip             \
413         tnf             \
414         touch           \
415         tput            \
416         tr              \
417         trapstat        \
418         troff           \
419         true            \
420         truss           \
421         tsol            \
422         tty             \
423         ttymon          \
424         tzreload        \
425         uadmin          \
426         ul              \
427         uname           \
428         units           \
429         unlink          \
430         unpack          \
431         userattr        \
432         users           \
433         utmp_update     \
434         utmpd           \
435         valtools        \
436         vgrind          \
437         vi              \
438         volcheck        \
439         volrmmount      \
440         vrrpadm         \
441         vscan           \
442         vt              \
443         w               \
444         wall            \
445         which           \
446         who             \
447         whodo           \
448         wracct          \
449         write           \
450         xargs           \
451         xstr            \
452         yes             \
453         ypcmd           \
454         yppasswd        \
455         zdb             \
456         zdump           \
457         zfs             \
```



```
458         zhack           \
459         zic             \
460         zinject         \
461         zlogin          \
462         zoneadm         \
463         zoneadmd        \
464         zonecfg         \
465         zonename        \
466         zpool           \
467         zlook           \
468         zonestat        \
469         zstreamdump     \
470         ztest

472 i386_SUBDIRS=           \
473         acpihpd         \
474         addbadsec       \
475         biosdev         \
476         diskscan        \
477         lms             \
478         rtc             \
479         ucodeadm        \
480         xvm

482 sparc_SUBDIRS=          \
483         cvcd            \
484         dcs             \
485         device_remap    \
486         drd             \
487         fruadm          \
488         ldmad           \
489         oplhpd          \
490         prtdscp         \
491         prtfru          \
492         scadm           \
493         sckmd           \
494         sf880drd        \
495         virtinfo        \
496         vntsd

498 #
499 # Commands that are messaged.  Note that 'lp' comes first
500 # (see previous comment about 'lp'.)
501 #
502 MSGSUBDIRS=             \
503         lp              \
504         abi             \
505         acctadm         \
506         allocate        \
507         asa             \
508         audio           \
509         audit           \
510         auditconfig     \
511         auditd          \
512         auditrecord     \
513         auditset        \
514         auths           \
515         autopush        \
516         avs             \
517         awk             \
518         awk_xpg4        \
519         backup          \
520         banner          \
521         bart            \
522         basename        \
523         beadm           \
```

```
524         bnu             \
525         busstat         \
526         cal             \
527         cat             \
528         cdrw            \
529         cfgadm          \
530         checkeq         \
531         checknr         \
532         chgrp           \
533         chmod           \
534         chown           \
535         cmd-crypto      \
536         cmd-inet        \
537         col             \
538         compress        \
539         consadm         \
540         coreadm         \
541         cpio            \
542         cpc             \
543         cron            \
544         csh             \
545         csplit          \
546         ctrun           \
547         ctstat          \
548         ctwatch         \
549         datadm          \
550         date            \
551         dc              \
552         dcs             \
553         dd              \
554         deroff          \
555         devfsadm        \
556         dfs.cmds        \
557         diff            \
558         diffmk          \
559         dladm           \
560         dlstat          \
561         du              \
562         dumpcs          \
563         ed              \
564         eject           \
565         env             \
566         eqn             \
567         expand          \
568         expr            \
569         fcinfo          \
570         fgrep           \
571         file            \
572         filesync        \
573         find            \
574         flowadm         \
575         flowstat        \
576         fm              \
577         fold            \
578         fs.d            \
579         fwflash         \
580         geniconvtbl     \
581         genmsg          \
582         getconf         \
583         getent          \
584         gettext         \
585         gettxt          \
586         grep            \
587         grep_xpg4       \
588         grpck           \
589         gss             \
```



```
590         halt            \
591         head            \
592         hostname        \
593         hotplug         \
594         id              \
595         idmap           \
596         isaexec         \
597         iscsiadm        \
598         iscsid          \
599         isns            \
600         itadm           \
601         kbd             \
602         krb5            \
603         ksh             \
604         last            \
605         ldap            \
606         ldapcachemgr    \
607         lgrpinfo        \
608         locale          \
609         lofiadm         \
610         logadm          \
611         logger          \
612         logins          \
613         ls              \
614         luxadm          \
615         lvm             \
616         mailx           \
617         make            \
618 #endif /* ! codereview */
619         man             \
620         mesg            \
621         mkdir           \
622         mkpwdict        \
623         mktemp          \
624         more            \
625         mpathadm        \
626         msgfmt          \
627         mv              \
628         ndmpadm         \
629         ndmpstat        \
630         newgrp          \
631         newtask         \
632         nice            \
633         nohup           \
634         oawk            \
635         pack            \
636         passwd          \
637         passmgmt        \
638         pathchk         \
639         pfexec          \
640         pg              \
641         pgrep           \
642         picl            \
643         pools           \
644         power           \
645         pr              \
646         praudit         \
647         print           \
648         profiles        \
649         projadd         \
650         projects        \
651         prstat          \
652         prtdiag         \
653         ps              \
654         psrinfo         \
655         ptools          \
```

```
656         pwconv          \
657         pwd             \
658         pyzfs           \
659         raidctl         \
660         ramdiskadm      \
661         rcap            \
662         rcm_daemon      \
663         refer           \
664         regcmp          \
665         renice          \
666         roles           \
667         rm              \
668         rmdir           \
669         rmformat        \
670         rmmount         \
671         rmvolmgr        \
672         sasinfo         \
673         sbdadm          \
674         scadm           \
675         script          \
676         scsi            \
677         sdiff           \
678         sdpadm          \
679         sgs             \
680         sh              \
681         shcomp          \
682         smbsrv          \
683         sort            \
684         split           \
685         srptadm         \
686         ssh             \
687         stat            \
688         stmfadm         \
689         stmsboot        \
690         strings         \
691         su              \
692         svc             \
693         svr4pkg         \
694         swap            \
695         syseventadm     \
696         syseventd       \
697         tabs            \
698         tar             \
699         tbl             \
700         time            \
701         tnf             \
702         touch           \
703         tput            \
704         troff           \
705         tsol            \
706         tty             \
707         ttymon          \
708         tzreload        \
709         ul              \
710         uname           \
711         units           \
712         unlink          \
713         unpack          \
714         userattr        \
715         valtools        \
716         vgrind          \
717         vi              \
718         volcheck        \
719         volrmmount      \
720         vrrpadm         \
721         vscan           \
```



```
722         w               \
723         who             \
724         whodo           \
725         wracct          \
726         write           \
727         xargs           \
728         yppasswd        \
729         zdump           \
730         zfs             \
731         zic             \
732         zlogin          \
733         zoneadm         \
734         zoneadmd        \
735         zonecfg         \
736         zonename        \
737         zpool           \
738         zonestat

740 sparc_MSGSUBDIRS=       \
741         fruadm          \
742         prtdscp         \
743         prtfru          \
744         virtinfo        \
745         vntsd

747 i386_MSGSUBDIRS=        \
748         ucodeadm

750 #
751 # commands that use dcgettext for localized time, LC_TIME
752 #
753 DCSUBDIRS=              \
754         cal             \
755         cfgadm          \
756         diff            \
757         ls              \
758         pr              \
759         ps              \
760         tar             \
761         w               \
762         who             \
763         whodo           \
764         write

766 #
767 # commands that belong only to audit.
768 #
769 AUDITSUBDIRS=           \
770         amt             \
771         audit           \
772         audit_warn      \
773         auditconfig     \
774         auditd          \
775         auditrecord     \
776         auditreduce     \
777         auditset        \
778         auditstat       \
779         praudit

781 #
782 # commands not owned by the systems group
783 #
784 BWOSDIRS=


787 all :=          TARGET = all
```

```
788 install :=      TARGET = install
789 clean :=        TARGET = clean
790 clobber :=      TARGET = clobber
791 lint :=         TARGET = lint
792 _msg :=         TARGET = _msg
793 _dc :=          TARGET = _dc

795 .KEEP_STATE:

797 SUBDIRS = $(COMMON_SUBDIRS) $($(MACH)_SUBDIRS)

799 .PARALLEL:      $(BWOSDIRS) $(SUBDIRS) $(MSGSUBDIRS) $(AUDITSUBDIRS)

801 all install clean clobber lint: $(FIRST_SUBDIRS) .WAIT $(SUBDIRS) \
802         $(AUDITSUBDIRS)

804 #
805 # Manifests cannot be checked in parallel, because we are using
806 # the global repository that is in $(SRC)/cmd/svc/seed/global.db.
807 # For this reason, to avoid .PARALLEL and .NO_PARALLEL conflicts,
808 # we spawn off a sub-make to perform the non-parallel 'make check'
809 #
810 check:
811         $(MAKE) -f Makefile.check check

813 #
814 # The .WAIT directive works around an apparent bug in parallel make.
815 # Evidently make was getting the target _msg vs. _dc confused under
816 # some level of parallelization, causing some of the _dc objects
817 # not to be built.
818 #
819 _msg: $(MSGSUBDIRS) $($(MACH)_MSGSUBDIRS) .WAIT _dc

821 _dc: $(DCSUBDIRS)

823 #
824 # Dependencies
825 #
826 fs.d: fstyp
827 ksh:    shcomp isaexec
828 mdb:    terminfo
829 print:  lp
830 fmt:    mailx

832 $(FIRST_SUBDIRS) $(BWOSDIRS) $(SUBDIRS) $(AUDITSUBDIRS): FRC
833         @if [ -f $@/Makefile  ]; then \
834                 cd $@; pwd; $(MAKE) $(TARGET); \
835         else \
836                 true; \
837         fi

839 FRC:
```

```
**********************************************************
     730 Wed May 20 12:19:45 2015
new/usr/src/cmd/make/Makefile
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

  12 # Copyright 2015, Richard Lowe.

  14 SUBDIRS= bin \
  14 SUBDIRS=bin \
  15          lib

  17 all     :=      TARGET = all
  18 install :=      TARGET = install
  19 clean   :=      TARGET = clean
  20 clobber :=      TARGET = clobber
  21 lint    :=      TARGET = lint
  22 _msg    :=      TARGET = _msg
  23 #endif /* ! codereview */

  25 .KEEP_STATE:

  27 bin: lib

  29 all clean clobber lint install _msg: $(SUBDIRS)
  22 all clean clobber lint install: $(SUBDIRS)

  31 $(SUBDIRS):     FRC
  32         @cd $@; pwd; $(MAKE) $(TARGET)

  34 FRC:
```

```
**********************************************************
     706 Wed May 20 12:19:46 2015
new/usr/src/cmd/make/Makefile.com
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

  12 # Copyright 2015, Richard Lowe.

  14 MAKE_INCLUDE= $(SRC)/cmd/make/include
  15 $(RELEASE_BUILD)MAKE_DEFS += -DNDEBUG
  16 CFLAGS += $(CCVERBOSE)
  17 CPPFLAGS += -I$(MAKE_INCLUDE) $(MAKE_DEFS)

  19 # So that it's set even for the libraries we build
  20 TEXT_DOMAIN = SUNW_OST_OSCMD

  22 $(POFILE): $(POFILES)
  23         $(CAT) $(POFILES) > $@
  24 #endif /* ! codereview */
```

```
**********************************************************
    1737 Wed May 20 12:19:46 2015
new/usr/src/cmd/make/bin/Makefile
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

# Copyright 2015, Richard Lowe.

PROG=   make
POFILE= make.po
#endif /* ! codereview */
OBJS=   ar.o            \
        depvar.o        \
        doname.o        \
        dosys.o         \
        files.o         \
        globals.o       \
        implicit.o      \
        macro.o         \
        main.o          \
        misc.o          \
        nse_printdep.o  \
        parallel.o      \
        pmake.o         \
        read.o          \
        read2.o         \
        rep.o           \
        state.o
POFILES= $(OBJS:%.o=%.po)
#endif /* ! codereview */

include ../../Makefile.cmd
include ../Makefile.com

LDLIBS += ../lib/mksh/libmksh.a ../lib/vroot/libvroot.a
LDLIBS += ../lib/mksh/libmksh.a ../lib/mksdmsi18n/libmksdmsi18n.a ../lib/vroot/l
LDLIBS += ../lib/bsd/libbsd.a -lc -lnsl -lumem

CPPFLAGS += -D_FILE_OFFSET_BITS=64

ROOTLINKS = $(ROOTCCSBIN)/make $(ROOTXPG4BIN)/make $(ROOTBIN)/dmake $(ROOTCCSLIB
        $(ROOTLIB)/svr4.make

ROOTRULES = $(ROOTSHLIB)/make/make.rules $(ROOTSHLIB)/make/svr4.make.rules

all:    $(PROG)

install: all $(ROOTPROG) $(ROOTLINKS) $(ROOTRULES)

$(PROG):        $(OBJS)
        $(LINK.cc) $(OBJS) -o $@ $(LDLIBS)
        $(POST_PROCESS)

$(ROOTCCSBIN)/make:
        -$(RM) $@; $(SYMLINK) ../../bin/make $@
```



```
$(ROOTCCSLIB)/svr4.make:
        -$(RM) $@; $(SYMLINK) ../../bin/make $@

$(ROOTLIB)/svr4.make:
        -$(RM) $@; $(SYMLINK) ../bin/make $@

$(ROOTXPG4BIN)/make:
        -$(RM) $@; $(SYMLINK) ../../bin/make $@

$(ROOTBIN)/dmake:
        -$(RM) $@; $(SYMLINK) ./make $@

$(ROOTRULES) := FILEMODE = 0444

$(ROOTRULES): $(ROOTSHLIB)/make

$(ROOTSHLIB)/make: FRC
        $(INS.dir)

$(ROOTSHLIB)/make/%: %.file
        $(INS.rename)

lint:

clean:
        $(RM) $(OBJS)

FRC:

include ../../Makefile.targ
```

```
**********************************************************
   23112 Wed May 20 12:19:46 2015
new/usr/src/cmd/make/bin/ar.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

/*
 *      ar.c
 *
 *      Deal with the lib.a(member.o) and lib.a((entry-point)) notations
 *
 * Look inside archives for notations a(b) and a((b))
 *      a(b)    is file member   b  in archive a
 *      a((b))  is entry point   b  in object archive a
 *
 * For 6.0, create a make which can understand all archive
 * formats.  This is kind of tricky, and <ar.h> isnt any help.
 */

/*
 * Included files
 */
#include <alloca.h>             /* alloca() */
#include <ar.h>
#include <errno.h>              /* errno */
#include <fcntl.h>              /* open() */
#include <libintl.h>
#endif /* ! codereview */
#include <mk/defs.h>
#include <mksh/misc.h>          /* retmem_mb() */

struct ranlib {
        union {
                off_t   ran_strx;       /* string table index of */
                char    *ran_name;      /* symbol defined by */
        }       ran_un;
        off_t   ran_off;                /* library member at this offset */
};

#include <unistd.h>             /* close() */
```



```
/*
 * Defined macros
 */
#ifndef S5EMUL
#undef BITSPERBYTE
#define BITSPERBYTE     8
#endif

/*
 * Defines for all the different archive formats.  See next comment
 * block for justification for not using <ar.h>s versions.
 */
#define AR_5_MAGIC              "<ar>"          /* 5.0 format magic string */
#define AR_5_MAGIC_LENGTH       4               /* 5.0 format string length */

#define AR_PORT_MAGIC           "!<arch>\n"     /* Port. (6.0) magic string */
#define AR_PORT_MAGIC_LENGTH    8               /* Port. (6.0) string length */
#define AR_PORT_END_MAGIC       "`\n"           /* Port. (6.0) end of header */
#define AR_PORT_WORD            4               /* Port. (6.0) 'word' length */

/*
 * typedefs & structs
 */
/*
 * These are the archive file headers for the formats.  Note
 * that it really doesnt matter if these structures are defined
 * here.  They are correct as of the respective archive format
 * releases.  If the archive format is changed, then since backwards
 * compatability is the desired behavior, a new structure is added
 * to the list.
 */
typedef struct {        /* 5.0 ar header format: vax family; 3b family */
        char                    ar_magic[AR_5_MAGIC_LENGTH];    /* AR_5_MAGIC*/
        char                    ar_name[16];    /* Space terminated */
        char                    ar_date[AR_PORT_WORD];  /* sgetl() accessed */
        char                    ar_syms[AR_PORT_WORD];  /* sgetl() accessed */
}                       Arh_5;

typedef struct {        /* 5.0 ar symbol format: vax family; 3b family */
        char                    sym_name[8];    /* Space terminated */
        char                    sym_ptr[AR_PORT_WORD];  /* sgetl() accessed */
}                       Ars_5;

typedef struct {        /* 5.0 ar member format: vax family; 3b family */
        char                    arf_name[16];   /* Space terminated */
        char                    arf_date[AR_PORT_WORD]; /* sgetl() accessed */
        char                    arf_uid[AR_PORT_WORD];  /* sgetl() accessed */
        char                    arf_gid[AR_PORT_WORD];  /* sgetl() accessed */
        char                    arf_mode[AR_PORT_WORD]; /* sgetl() accessed */
        char                    arf_size[AR_PORT_WORD]; /* sgetl() accessed */
}                       Arf_5;

typedef struct {        /* Portable (6.0) ar format: vax family; 3b family */
        char                    ar_name[16];    /* Space terminated */
        /* left-adjusted fields; decimal ascii; blank filled */
        char                    ar_date[12];
        char                    ar_uid[6];
        char                    ar_gid[6];
        char                    ar_mode[8];     /* octal ascii */
        char                    ar_size[10];
        /* special end-of-header string (AR_PORT_END_MAGIC) */
        char                    ar_fmag[2];
}                       Ar_port;

enum ar_type {
                AR_5,
```

```
128                 AR_PORT
129 };

131 typedef unsigned int ar_port_word; // must be 4-bytes long

133 typedef struct {
134         FILE                    *fd;
135         /* to distiguish ar format */
136         enum ar_type            type;
137         /* where first ar member header is at */
138         long                    first_ar_mem;
139         /* where the symbol lookup starts */
140         long                    sym_begin;
141         /* the number of symbols available */
142         long                    num_symbols;
143         /* length of symbol directory file */
144         long                    sym_size;
145         Arh_5                   arh_5;
146         Ars_5                   ars_5;
147         Arf_5                   arf_5;
148         Ar_port                 ar_port;
149 }                       Ar;

151 /*
152  * Static variables
153  */

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

165 /*
166  *      read_archive(target)
167  *
168  *      Read the contents of an ar file.
169  *
170  *      Return value:
171  *                              The time the member was created
172  *
173  *      Parameters:
174  *              target          The member to find time for
175  *
176  *      Global variables used:
177  *              empty_name      The Name ""
178  */

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

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



```
195         member = get_prop(target->prop, member_prop);
196         /*
197          * Check if the member has directory component.
198          * If so, remove the dir and see if we know the date.
199          */
200         if (member->body.member.member != NULL) {
201                 Wstring member_string(member->body.member.member);
202                 wchar_t * wcb = member_string.get_string();
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);
209                         append_string(slash + 1, &true_member_name, FIND_LENGTH)
210                         append_char((int) parenright_char, &true_member_name);
211                         true_member = GETNAME(true_member_name.buffer.start,
212                                               FIND_LENGTH);
213                         if (true_member->stat.time != file_no_time) {
214                                 target->stat.time = true_member->stat.time;
215                                 return target->stat.time;
216                         }
217                 }
218         }
219         if (open_archive(member->body.member.library->string_mb, &ar) == failed)
220                 if (errno == ENOENT) {
221                         target->stat.stat_errno = ENOENT;
222                         close_archive(&ar);
223                         if (member->body.member.member == NULL) {
224                                 member->body.member.member = empty_name;
225                         }
226                         return target->stat.time = file_doesnt_exist;
227                 } else {
228                         fatal(gettext("Can't access archive '%s': %s"),
 46                         fatal(catgets(catd, 1, 1, "Can't access archive '%s': %s
229                               member->body.member.library->string_mb,
230                               errmsg(errno));
231                 }
232         }
233         if (target->stat.time == file_no_time) {
234                 if (read_archive_dir(&ar, member->body.member.library,
235                                      &long_names_table)
236                     == failed){
237                         fatal(gettext("Can't access archive '%s': %s"),
 55                         fatal(catgets(catd, 1, 2, "Can't access archive '%s': %s
238                               member->body.member.library->string_mb,
239                               errmsg(errno));
240                 }
241         }
242         if (member->body.member.entry != NULL) {
243                 translate_entry(&ar, target, member,&long_names_table);
244         }
245         close_archive(&ar);
246         if (long_names_table) {
247                 retmem_mb(long_names_table);
248         }
249         if (true_member != NULL) {
250                 target->stat.time = true_member->stat.time;
251         }
252         if (target->stat.time == file_no_time) {
253                 target->stat.time = file_doesnt_exist;
254         }
255         return target->stat.time;
256 }
```

```
258 /*
259  *      open_archive(filename, arp)
260  *
261  *      Return value:
262  *                              Indicates if open failed or not
263  *
264  *      Parameters:
265  *              filename        The name of the archive we need to read
266  *              arp             Pointer to ar file description block
267  *
268  *      Global variables used:
269  */
270 static Boolean
271 open_archive(char *filename, register Ar *arp)
272 {
273         int                     fd;
274         char                    mag_5[AR_5_MAGIC_LENGTH];
275         char                    mag_port[AR_PORT_MAGIC_LENGTH];
276         char                    buffer[4];

278         arp->fd = NULL;
279         fd = open_vroot(filename, O_RDONLY, 0, NULL, VROOT_DEFAULT);
280         if ((fd < 0) || ((arp->fd = fdopen(fd, "r")) == NULL)) {
281                 return failed;
282         }
283         (void) fcntl(fileno(arp->fd), F_SETFD, 1);

285         if (fread(mag_port, AR_PORT_MAGIC_LENGTH, 1, arp->fd) != 1) {
286                 return failed;
287         }
288         if (IS_EQUALN(mag_port, AR_PORT_MAGIC, AR_PORT_MAGIC_LENGTH)) {
289                 arp->type = AR_PORT;
290                 /*
291                  * Read in first member header to find out if there is
292                  * a symbol definition table.
293                  */

295                 int ret = read_member_header(&arp->ar_port, arp->fd, filename);
296                 if (ret == failed) {
297                         return failed;
298                 } else if(ret == -1) {
299                         /* There is no member header - empty archive */
300                         arp->sym_size = arp->num_symbols = arp->sym_begin = 0L;
301                         arp->first_ar_mem = ftell(arp->fd);
302                         return succeeded;
303                 }
304                 /*
305                  * The following values are the default if there is
306                  * no symbol directory and long member names.
307                  */
308                 arp->sym_size = arp->num_symbols = arp->sym_begin = 0L;
309                 arp->first_ar_mem = ftell(arp->fd) - (long) sizeof (Ar_port);

311                 /*
312                  * Do we have a symbol table? A symbol table is always
313                  * the first member in an archive. In 4.1.x it has the
314                  * name __.SYMDEF, in SVr4, it has the name "/        "
315                  */
316 /*
317                 MBSTOWCS(wcs_buffer, "/               ");
318                 if (IS_WEQUALN(arp->ar_port.ar_name, wcs_buffer, 16)) {
319  */
320                 if (IS_EQUALN(arp->ar_port.ar_name,
321                                 "/               ",
139                                 NOCATGETS("/               "),
322                                 16)) {
```



```
323                         if (sscanf(arp->ar_port.ar_size,
324                                 "%ld",
325                                 &arp->sym_size) != 1) {
326                                 return failed;
327                         }
328                         arp->sym_size += (arp->sym_size & 1); /* round up */
329                         if (fread(buffer, sizeof buffer, 1, arp->fd) != 1) {
330                                 return failed;
331                         }
332                         arp->num_symbols = sgetl(buffer);
333                         arp->sym_begin = ftell(arp->fd);
334                         arp->first_ar_mem = arp->sym_begin +
335                                                 arp->sym_size - sizeof buffer;
336                 }
337                 return succeeded;
338         }
339         fatal(gettext("`%s' is not an archive"), filename);
157         fatal(catgets(catd, 1, 3, "`%s' is not an archive"), filename);
340         /* NOTREACHED */
341         return failed;
342 }
```

_____*unchanged_portion_omitted_*

```
361 /*
362  *      read_archive_dir(arp, library, long_names_table)
363  *
364  *      Reads the directory of an archive and enters all
365  *      the members into the make symboltable in lib(member) format
366  *      with their dates.
367  *
368  *      Parameters:
369  *              arp             Pointer to ar file description block
370  *              library         Name of lib to enter members for.
371  *                              Used to form "lib(member)" string.
372  *              long_names_table table that contains list of members
373  *                              with names > 15 characters long
374  *
375  *      Global variables used:
376  */
377 static Boolean
378 read_archive_dir(register Ar *arp, Name library, char **long_names_table)
379 {
380         wchar_t                 *name_string;
381         wchar_t                 *member_string;
382         register long           len;
383         register wchar_t        *p;
384         register char           *q;
385         register Name           name;
386         Property                member;
387         long                    ptr;
388         long                    date;

390         int                     offset;

392         /*
393          * If any of the members has a name > 15 chars,
394          * it will be found here.
395          */
396         if (process_long_names_member(arp, long_names_table, library->string_mb)
397                 return failed;
398         }
399         name_string = ALLOC_WC((int) (library->hash.length +
400                                 (int) ar_member_name_len * 2));
401         (void) mbstowcs(name_string, library->string_mb, (int) library->hash.len
402         member_string = name_string + library->hash.length;
403         *member_string++ = (int) parenleft_char;
```

```
405         if (fseek(arp->fd, arp->first_ar_mem, 0) != 0) {
406                 goto read_error;
407         }
408         /* Read the directory using the appropriate format */
409         switch (arp->type) {
410         case AR_5:
411             for (;;) {
412                 if (fread((char *) &arp->arf_5, sizeof arp->arf_5, 1, arp->fd)
413                     != 1) {
414                     if (feof(arp->fd)) {
415                             return succeeded;
416                     }
417                     break;
418                 }
419                 len = sizeof arp->arf_5.arf_name;
420                 for (p = member_string, q = arp->arf_5.arf_name;
421                      (len > 0) && (*q != (int) nul_char) && !isspace(*q);
422                      ) {
423                     MBTOWC(p, q);
424                     p++;
425                     q++;
426                 }
427                 *p++ = (int) parenright_char;
428                 *p = (int) nul_char;
429                 name = GETNAME(name_string, FIND_LENGTH);
430                 /*
431                  * [tolik] Fix for dmake bug 1234018.
432                  * If name->stat.time is already set, then it should not
433                  * be changed. (D)make propogates time stamp for one
434                  * member, and when it calls exists() for another member,
435                  * the first one may be changed.
436                  */
437                 if(name->stat.time == file_no_time) {
438                         name->stat.time.tv_sec = sgetl(arp->arf_5.arf_date);
439                         name->stat.time.tv_nsec = LONG_MAX;
440                 }
441                 name->is_member = library->is_member;
442                 member = maybe_append_prop(name, member_prop);
443                 member->body.member.library = library;
444                 *--p = (int) nul_char;
445                 if (member->body.member.member == NULL) {
446                         member->body.member.member =
447                           GETNAME(member_string, FIND_LENGTH);
448                 }
449                 ptr = sgetl(arp->arf_5.arf_size);
450                 ptr += (ptr & 1);
451                 if (fseek(arp->fd, ptr, 1) != 0) {
452                         goto read_error;
453                 }
454             }
455             break;
456         case AR_PORT:
457             for (;;) {
458                     if ((fread((char *) &arp->ar_port,
459                                sizeof arp->ar_port,
460                                1,
461                                arp->fd) != 1) ||
462                         !IS_EQUALN(arp->ar_port.ar_fmag,
463                                    AR_PORT_END_MAGIC,
464                                    sizeof arp->ar_port.ar_fmag)) {
465                             if (feof(arp->fd)) {
466                                  return succeeded;
467                             }
468                             fatal(
469                                 gettext("Read error in archive ‘%s’: invalid arc
```



```
287                                 catgets(catd, 1, 28, "Read error in archive ‘%s’
470                                 library->string_mb,
471                                 ftell(arp->fd)
472                         );
473                 }
474                 /* If it’s a long name, retrieve it from long name table */
475                 if (arp->ar_port.ar_name[0] == ’/’) {
476                         /*
477                          * "len" is used for hashing the string.
478                          * We’re using "ar_member_name_len" instead of
479                          * the actual name length since it’s the longest
480                          * string the "ar" command can handle at this
481                          * point.
482                          */
483                         len = ar_member_name_len;
484                         sscanf(arp->ar_port.ar_name + 1,
485                                "%ld",
486                                &offset);
487                         q = *long_names_table + offset;
488                 } else {
489                         q = arp->ar_port.ar_name;
490                         len = sizeof arp->ar_port.ar_name;
491                 }
492
493                 for (p = member_string;
494                      (len > 0) &&
495                      (*q != (int) nul_char) &&
496                      !isspace(*q) &&
497                      (*q != (int) slash_char);
498                      ) {
499                         MBTOWC(p, q);
500                         p++;
501                         q++;
502                 }
503                 *p++ = (int) parenright_char;
504                 *p = (int) nul_char;
505                 name = GETNAME(name_string, FIND_LENGTH);
506                 name->is_member = library->is_member;
507                 member = maybe_append_prop(name, member_prop);
508                 member->body.member.library = library;
509                 *--p = (int) nul_char;
510                 if (member->body.member.member == NULL) {
511                         member->body.member.member =
512                           GETNAME(member_string, FIND_LENGTH);
513                 }
514                 if (sscanf(arp->ar_port.ar_date, "%ld", &date) != 1) {
515                         WCSTOMBS(mbs_buffer, name_string);
516                         fatal(gettext("Bad date field for member ‘%s’ in arc
334                         fatal(catgets(catd, 1, 4, "Bad date field for member
517                               mbs_buffer,
518                               library->string_mb);
519                 }
520                 /*
521                  * [tolik] Fix for dmake bug 1234018.
522                  */
523                 if(name->stat.time == file_no_time) {
524                     name->stat.time.tv_sec = date;
525                     name->stat.time.tv_nsec = LONG_MAX;
526                 }
527                 if (sscanf(arp->ar_port.ar_size, "%ld", &ptr) != 1) {
528                         WCSTOMBS(mbs_buffer, name_string);
529                         fatal(gettext("Bad size field for member ‘%s’ in arc
347                         fatal(catgets(catd, 1, 5, "Bad size field for member
530                               mbs_buffer,
531                               library->string_mb);
532                 }
```

```
533                     ptr += (ptr & 1);
534                     if (fseek(arp->fd, ptr, 1) != 0) {
535                             goto read_error;
536                     }
537             }
538             break;
539         }

541         /* Only here if fread() [or IS_EQUALN()] failed and not at EOF */
542 read_error:
543         fatal(gettext("Read error in archive `%s': %s"),
361         fatal(catgets(catd, 1, 6, "Read error in archive `%s': %s"),
544               library->string_mb,
545               errmsg(errno));
546             /* NOTREACHED */
547 }


550 /*
551  *      process_long_names_member(arp)
552  *
553  *      If the archive contains members with names longer
554  *      than 15 characters, then it has a special member
555  *      with the name "//        " that contains a table
556  *      of null-terminated long names. This member
557  *      is always the first member, after the symbol table
558  *      if it exists.
559  *
560  *      Parameters:
561  *              arp             Pointer to ar file description block
562  *
563  *      Global variables used:
564  */
565 int
566 process_long_names_member(register Ar *arp, char **long_names_table, char *filen
567 {
568         Ar_port                 *ar_member_header;
569         int                     table_size;

571         if (fseek(arp->fd, arp->first_ar_mem, 0) != 0) {
572                 return failed;
573         }
574         if ((ar_member_header =
575              (Ar_port *) alloca((int) sizeof(Ar_port))) == NULL){
576                 perror(gettext("memory allocation failure"));
394                 perror(catgets(catd, 1, 7, "memory allocation failure"));
577                 return failed;
578         }
579         int ret = read_member_header(ar_member_header, arp->fd, filename);
580         if (ret == failed) {
581                 return failed;
582         } else if(ret == -1) {
583                 /* There is no member header - empty archive */
584                 return succeeded;
585         }
586         /* Do we have special member containing long names? */
587         if (IS_EQUALN(ar_member_header->ar_name,
588                       "//              ",
406                       NOCATGETS("//              "),
589                       16)){
590                 if (sscanf(ar_member_header->ar_size,
591                            "%ld",
592                            &table_size) != 1) {
593                   return failed;
594                 }
595                 *long_names_table = (char *) malloc(table_size);
```



```
596                 /* Read the list of long member names into the table */
597                 if (fread(*long_names_table, table_size, 1, arp->fd) != 1) {
598                         return failed;
599                 }
600                 arp->first_ar_mem = ftell(arp->fd);
601         }
602         return succeeded;
603 }

605 /*
606  *      translate_entry(arp, target, member)
607  *
608  *      Finds the member for one lib.a((entry))
609  *
610  *      Parameters:
611  *              arp             Pointer to ar file description block
612  *              target          Target to find member name for
613  *              member          Property to fill in with info
614  *
615  *      Global variables used:
616  */
617 static void
618 translate_entry(register Ar *arp, Name target, register Property member, char **
619 {
620         register int            len;
621         register int            i;
622         wchar_t                 *member_string;
623         ar_port_word            *offs;
624         int                     strtablen;
625         char                    *syms;          /* string table */
626         char                    *csym;          /* string table */
627         ar_port_word            *offend;        /* end of offsets table */
628         int                     date;
629         register wchar_t        *ap;
630         register char           *hp;
631         int                     maxs;
632         int                     offset;
633         char            buffer[4];

635         if (arp->sym_begin == 0L || arp->num_symbols == 0L) {
636                 fatal(gettext("Cannot find symbol `%s' in archive `%s'"),
454                 fatal(catgets(catd, 1, 8, "Cannot find symbol `%s' in archive `%
637                       member->body.member.entry->string_mb,
638                       member->body.member.library->string_mb);
639         }

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

646         switch (arp->type) {
647         case AR_5:
648                 if ((len = member->body.member.entry->hash.length) > 8) {
649                         len = 8;
650                 }
651                 for (i = 0; i < arp->num_symbols; i++) {
652                         if (fread((char *) &arp->ars_5,
653                                   sizeof arp->ars_5,
654                                   1,
655                                   arp->fd) != 1) {
656                                 goto read_error;
657                         }
658                         if (IS_EQUALN(arp->ars_5.sym_name,
659                                       member->body.member.entry->string_mb,
660                                       len)) {
```

```
661                                 if ((fseek(arp->fd,
662                                         sgetl(arp->ars_5.sym_ptr),
663                                         0) != 0) ||
664                                     (fread((char *) &arp->arf_5,
665                                         sizeof arp->arf_5,
666                                         1,
667                                         arp->fd) != 1)) {
668                                         goto read_error;
669                                 }
670                                 MBSTOWCS(wcs_buffer, arp->arf_5.arf_name);
671                                 (void) wsncpy(member_string,
672                                             wcs_buffer,
673                                             wslen(wcs_buffer));
674                                 member_string[sizeof(arp->arf_5.arf_name)] =
675                                                         (int) nul_char;
676                                 member->body.member.member =
677                                         GETNAME(member_string, FIND_LENGTH);
678                                 target->stat.time.tv_sec = sgetl(arp->arf_5.arf_
679                                 target->stat.time.tv_nsec = LONG_MAX;
680                                 return;
681                         }
682                 }
683                 break;
684         case AR_PORT:
685                 offs = (ar_port_word *) alloca((int) (arp->num_symbols * AR_PORT
686                 if (fread((char *) offs,
687                           AR_PORT_WORD,
688                           (int) arp->num_symbols,
689                           arp->fd) != arp->num_symbols) {
690                         goto read_error;
691                 }

693                 for(i=0;i<arp->num_symbols;i++) {
694                         *((int*)buffer)=offs[i];
695                         offs[i]=(ar_port_word)sgetl(buffer);
696                 }

698                 strtablen=arp->sym_size-4-(int) (arp->num_symbols * AR_PORT_WORD
699                 syms = (char *) alloca(strtablen);
700                 if (fread(syms,
701                           sizeof (char),
702                           strtablen,
703                           arp->fd) != strtablen) {
704                         goto read_error;
705                 }
706                 offend = &offs[arp->num_symbols];
707                 while (offs < offend) {
708                         maxs = strlen(member->body.member.entry->string_mb);
709                         if(strlen(syms) > maxs)
710                                 maxs = strlen(syms);
711                         if (IS_EQUALN(syms,
712                                       member->body.member.entry->string_mb,
713                                       maxs)) {
714                                 if (fseek(arp->fd,
715                                           (long) *offs,
716                                           0) != 0) {
717                                         goto read_error;
718                                 }
719                                 if ((fread((char *) &arp->ar_port,
720                                           sizeof arp->ar_port,
721                                           1,
722                                           arp->fd) != 1) ||
723                                     !IS_EQUALN(arp->ar_port.ar_fmag,
724                                                AR_PORT_END_MAGIC,
725                                                sizeof arp->ar_port.ar_fmag)) {
726                                         goto read_error;
```



```
727                                 }
728                                 if (sscanf(arp->ar_port.ar_date,
729                                            "%ld",
730                                            &date) != 1) {
731                                         fatal(gettext("Bad date field for member
549                                         fatal(catgets(catd, 1, 9, "Bad date fiel
732                                               arp->ar_port.ar_name,
733                                               target->string_mb);
734                                 }
735                         /* If it's a long name, retrieve it from long name table */
736                         if (arp->ar_port.ar_name[0] == '/') {
737                                 sscanf(arp->ar_port.ar_name + 1,
738                                        "%ld",
739                                        &offset);
740                                 len = ar_member_name_len;
741                                 hp = *long_names_table + offset;
742                         } else {
743                                 len = sizeof arp->ar_port.ar_name;
744                                 hp = arp->ar_port.ar_name;
745                         }
746                                 ap = member_string;
747                                 while (*hp &&
748                                        (*hp != (int) slash_char) &&
749                                        (ap < &member_string[len])) {
750                                         MBTOWC(ap, hp);
751                                         ap++;
752                                         hp++;
753                                 }
754                                 *ap = (int) nul_char;
755                                 member->body.member.member =
756                                         GETNAME(member_string, FIND_LENGTH);
757                                 target->stat.time.tv_sec = date;
758                                 target->stat.time.tv_nsec = LONG_MAX;
759                                 return;
760                         }
761                         offs++;
762                         while(*syms!='\0') syms++;
763                         syms++;
764                 }
765         }
766         fatal(gettext("Cannot find symbol `%s' in archive `%s'"),
584         fatal(catgets(catd, 1, 10, "Cannot find symbol `%s' in archive `%s'"),
767               member->body.member.entry->string_mb,
768               member->body.member.library->string_mb);
769         /*NOTREACHED*/

771 read_error:
772         if (ferror(arp->fd)) {
773                 fatal(gettext("Read error in archive `%s': %s"),
591                 fatal(catgets(catd, 1, 11, "Read error in archive `%s': %s"),
774                       member->body.member.library->string_mb,
775                       errmsg(errno));
776         } else {
777                 fatal(gettext("Read error in archive `%s': Premature EOF"),
595                 fatal(catgets(catd, 1, 12, "Read error in archive `%s': Prematur
778                       member->body.member.library->string_mb);
779         }
780 }
______unchanged_portion_omitted_


810 /*
811  *      read_member_header(header, fd, filename)
812  *
813  *      reads the member header for the 4.1.x and SVr4 archives.
814  *
```

```
 815  *      Return value:
 816  *                              fails if read error or member
 817  *                              header is not the right format
 818  *      Parameters:
 819  *              header          There's one before each archive member
 820  *              fd              file descriptor for the archive file.
 821  *
 822  *      Global variables used:
 823  */
 824 int
 825 read_member_header(Ar_port *header, FILE *fd, char* filename)
 826 {
 827         int num = fread((char *) header, sizeof (Ar_port), 1, fd);
 828         if (num != 1 && feof(fd)) {
 829                 /* There is no member header - empty archive */
 830                 return -1;
 831         }
 832         if ((num != 1) ||
 833             !IS_EQUALN(
 834                 AR_PORT_END_MAGIC,
 835                 header->ar_fmag,
 836                 sizeof (header->ar_fmag)
 837             )
 838         ) {
 839                 fatal(
 840                         gettext("Read error in archive '%s': invalid archive fil
 658                         catgets(catd, 1, 28, "Read error in archive '%s': invali
 841                         filename,
 842                         ftell(fd)
 843                 );
 844         }
 845         return succeeded;
 846 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    2679 Wed May 20 12:19:47 2015
new/usr/src/cmd/make/bin/depvar.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 1995 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Included files
 */
#include <libintl.h>

#endif /* ! codereview */
#include <mk/defs.h>
#include <mksh/misc.h>          /* getmem() */

/*
 * This file deals with "Dependency Variables".
 * The "-V var" command line option is used to indicate
 * that var is a dependency variable.  Used in conjunction with
 * the -P option the user is asking if the named variables affect
 * the dependencies of the given target.
 */

struct _Depvar {
        Name            name;           /* Name of variable */
        struct _Depvar  *next;          /* Linked list */
        Boolean         cmdline;        /* Macro defined on the cmdline? */
};

typedef struct _Depvar  *Depvar;

static  Depvar          depvar_list;
static  Depvar          *bpatch = &depvar_list;
static  Boolean         variant_deps;

/*
 * Add a name to the list.
 */

void
depvar_add_to_list(Name name, Boolean cmdline)
{
```



```
        Depvar          dv;

        dv = ALLOC(Depvar);
        dv->name = name;
        dv->next = NULL;
        dv->cmdline = cmdline;
        *bpatch = dv;
        bpatch = &dv->next;
}

/*
 * The macro 'name' has been used in either the left-hand or
 * right-hand side of a dependency.  See if it is in the
 * list.  Two things are looked for.  Names given as args
 * to the -V list are checked so as to set the same/differ
 * output for the -P option.  Names given as macro=value
 * command-line args are checked and, if found, an NSE
 * warning is produced.
 */
void
depvar_dep_macro_used(Name name)
{
        Depvar          dv;

        for (dv = depvar_list; dv != NULL; dv = dv->next) {
                if (name == dv->name) {
                        variant_deps = true;
                        break;
                }
        }
}


/*
 * Print the results.  If any of the Dependency Variables
 * affected the dependencies then the dependencies potentially
 * differ because of these variables.
 */
void
depvar_print_results(void)
{
        if (variant_deps) {
                printf(gettext("differ\n"));
                printf(catgets(catd, 1, 234, "differ\n"));
        } else {
                printf(gettext("same\n"));
                printf(catgets(catd, 1, 235, "same\n"));
        }
}
_____unchanged_portion_omitted_
```

```
**********************************************************
   91156 Wed May 20 12:19:47 2015
new/usr/src/cmd/make/bin/doname.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

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

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


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


#include <signal.h>
#include <stropts.h>

#       include <stropts.h>

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

/*
 * Defined macros
 */
```



```
#       define LOCALHOST "localhost"

#define MAXRULES 100

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

/*
 * typedefs & structs
 */

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


static int      second_pass = 0;

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

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



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

```
 *      Global variables used:
 *              build_failed_seen       Set if -k is on and error occurs
 *              continue_after_error    Indicates that -k is on
 *              report_dependencies     No error msg if -P is on
 */
Doname
doname_check(register Name target, register Boolean do_get, register Boolean imp
{
        int first_time = 1;
        (void) fflush(stdout);
try_again:
        switch (doname(target, do_get, implicit, automatic)) {
        case build_ok:
                second_pass = 0;
                return build_ok;
        case build_running:
                second_pass = 0;
                return build_running;
        case build_failed:
                if (!continue_after_error) {
                        fatal(gettext("Target '%s' not remade because of errors"
                        fatal(catgets(catd, 1, 13, "Target '%s' not remade becau
                                target->string_mb);
                }
                build_failed_seen = true;
                second_pass = 0;
                return build_failed;
        case build_dont_know:
                /*
                 * If we can't figure out how to build an automatic
                 * (hidden) dependency, we just ignore it.
                 * We later declare the target to be out of date just in
                 * case something changed.
                 * Also, don't complain if just reporting the dependencies
                 * and not building anything.
                 */
                if (automatic || (report_dependencies_level > 0)) {
                        second_pass = 0;
                        return build_dont_know;
                }
                if(first_time) {
                        first_time = 0;
                        second_pass = 1;
                        goto try_again;
                }
                second_pass = 0;
                if (continue_after_error && !svr4) {
                        warning(gettext("Don't know how to make target '%s'"),
                        warning(catgets(catd, 1, 14, "Don't know how to make tar
                                target->string_mb);
                        build_failed_seen = true;
                        return build_failed;
                }
                fatal(gettext("Don't know how to make target '%s'"), target->str
                fatal(catgets(catd, 1, 15, "Don't know how to make target '%s'")
                break;
        }
#ifdef lint
        return build_failed;
#endif
}
```

unchanged_portion_omitted_

```
/*
 * DONE.
 *
```



```
 *      doname(target, do_get, implicit)
 *
 *      Chases all files the target depends on and builds any that
 *      are out of date. If the target is out of date it is then rebuilt.
 *
 *      Return value:
 *                              Indiates if build failed or nt
 *
 *      Parameters:
 *              target          Target to build
 *              do_get          Run sccs get is nessecary
 *              implicit        doname is trying to find an implicit rule
 *
 *      Global variables used:
 *              assign_done     True if command line assgnment has happened
 *              commands_done   Preserved for the case that we need local value
 *              debug_level     Should we trace make's actions?
 *              default_rule    The rule for ".DEFAULT", used as last resort
 *              empty_name      The Name "", used when looking for single sfx
 *              keep_state      Indicates that .KEEP_STATE is on
 *              parallel        True if building in parallel
 *              recursion_level Used for tracing
 *              report_dependencies make -P is on
 */
Doname
doname(register Name target, register Boolean do_get, register Boolean implicit,
{
        Doname                  result = build_dont_know;
        Chain                   out_of_date_list = NULL;
        Chain                   target_group;
        Property                old_locals = NULL;
        register Property       line;
        Property                command = NULL;
        register Dependency     dependency;
        Name                    less = NULL;
        Name                    true_target = target;
        Name                    *automatics = NULL;
        register int            auto_count;
        Boolean                 rechecking_target = false;
        Boolean                 saved_commands_done;
        Boolean                 restart = false;
        Boolean                 save_parallel = parallel;
        Boolean                 doing_subtree = false;

        Boolean                 recheck_conditionals = false;

        if (target->state == build_running) {
                return build_running;
        }
        line = get_prop(target->prop, line_prop);
        if (line != NULL) {
                /*
                 * If this target is a member of target group and one of the
                 * other members of the group is running, mark this target
                 * as running.
                 */
                for (target_group = line->body.line.target_group;
                     target_group != NULL;
                     target_group = target_group->next) {
                        if (is_running(target_group->name)) {
                                target->state = build_running;
                                add_pending(target,
                                            recursion_level,
                                            do_get,
                                            implicit,
                                            false);
```

```
322                                 return build_running;
323                         }
324                 }
325         }
326         /*
327          * If the target is a constructed one for a "::" target,
328          * we need to consider that.
329          */
330         if (target->has_target_prop) {
331                 true_target = get_prop(target->prop,
332                                        target_prop)->body.target.target;
333                 if (true_target->colon_splits > 0) {
334                         /* Make sure we have a valid time for :: targets */
335                         Property        time;

337                         time = get_prop(true_target->prop, time_prop);
338                         if (time != NULL) {
339                                 true_target->stat.time = time->body.time.time;
340                         }
341                 }
342         }
343         (void) exists(true_target);
344         /*
345          * If the target has been processed, we don't need to do it again,
346          * unless it depends on conditional macros or a delayed assignment,
347          * or it has been done when KEEP_STATE is on.
348          */
349         if (target->state == build_ok) {
350                 if((!keep_state || (!target->depends_on_conditional && !assign_d
351                         return build_ok;
352                 } else {
353                         recheck_conditionals = true;
354                 }
355         }
356         if (target->state == build_subtree) {
357                 /* A dynamic macro subtree is being built */
358                 target->state = build_dont_know;
359                 doing_subtree = true;
360                 if (!target->checking_subtree) {
361                         /*
362                          * This target has been started before and therefore
363                          * not all dependencies have to be built.
364                          */
365                         restart = true;
366                 }
367         } else if (target->state == build_pending) {
368                 target->state = build_dont_know;
369                 restart = true;
370 /*
371         } else if (parallel &&
372                    keep_state &&
373                    (target->conditional_cnt > 0)) {
374             if (!parallel_ok(target, false)) {
375                 add_subtree(target, recursion_level, do_get, implicit);
376                 target->state = build_running;
377                 return build_running;
378             }
379  */
380         }
381         /*
382          * If KEEP_STATE is on, we have to rebuild the target if the
383          * building of it caused new automatic dependencies to be reported.
384          * This is where we restart the build.
385          */
386         if (line != NULL) {
387                 line->body.line.percent = NULL;
```



```
388         }
389 recheck_target:
390         /* Init all local variables */
391         result = build_dont_know;
392         out_of_date_list = NULL;
393         command = NULL;
394         less = NULL;
395         auto_count = 0;
396         if (!restart && line != NULL) {
397                 /*
398                  * If this target has never been built before, mark all
399                  * of the dependencies as never built.
400                  */
401                 for (dependency = line->body.line.dependencies;
402                      dependency != NULL;
403                      dependency = dependency->next) {
404                         dependency->built = false;
405                 }
406         }
407         /* Save the set of automatic depes defined for this target */
408         if (keep_state &&
409             (line != NULL) &&
410             (line->body.line.dependencies != NULL)) {
411                 Name *p;

413                 /*
414                  * First run thru the dependency list to see how many
415                  * autos there are.
416                  */
417                 for (dependency = line->body.line.dependencies;
418                      dependency != NULL;
419                      dependency = dependency->next) {
420                         if (dependency->automatic && !dependency->stale) {
421                                 auto_count++;
422                         }
423                 }
424                 /* Create vector to hold the current autos */
425                 automatics =
426                   (Name *) alloca((int) (auto_count * sizeof (Name)));
427                 /* Copy them */
428                 for (p = automatics, dependency = line->body.line.dependencies;
429                      dependency != NULL;
430                      dependency = dependency->next) {
431                         if (dependency->automatic && !dependency->stale) {
432                                 *p++ = dependency->name;
433                         }
434                 }
435         }
436         if (debug_level > 1) {
437                 (void) printf("%*sdoname(%s)\n",
442                 (void) printf(NOCATGETS("%*sdoname(%s)\n"),
438                               recursion_level,
439                               "",
440                               target->string_mb);
441         }
442         recursion_level++;
443         /* Avoid infinite loops */
444         if (target->state == build_in_progress) {
445                 warning(gettext("Infinite loop: Target `%s' depends on itself"),
450                 warning(catgets(catd, 1, 16, "Infinite loop: Target `%s' depends
446                         target->string_mb);
447                 return build_ok;
448         }
449         target->state = build_in_progress;

451         /* Activate conditional macros for the target */
```

```
452         if (!target->added_pattern_conditionals) {
453                 add_pattern_conditionals(target);
454                 target->added_pattern_conditionals = true;
455         }
456         if (target->conditional_cnt > 0) {
457                 old_locals = (Property) alloca(target->conditional_cnt *
458                                                sizeof (Property_rec));
459                 set_locals(target, old_locals);
460         }

462 /*
463  * after making the call to dynamic_dependecies unconditional we can handle
464  * target names that are same as file name. In this case $$@ in the
465  * dependencies did not mean anything. WIth this change it expands it
466  * as expected.
467  */
468         if (!target->has_depe_list_expanded)
469         {
470                 dynamic_dependencies(target);
471         }

473 /*
474  *      FIRST SECTION -- GO THROUGH DEPENDENCIES AND COLLECT EXPLICIT
475  *      COMMANDS TO RUN
476  */
477         if ((line = get_prop(target->prop, line_prop)) != NULL) {
478                 if (check_dependencies(&result,
479                                        line,
480                                        do_get,
481                                        target,
482                                        true_target,
483                                        doing_subtree,
484                                        &out_of_date_list,
485                                        old_locals,
486                                        implicit,
487                                        &command,
488                                        less,
489                                        rechecking_target,
490                                        recheck_conditionals)) {
491                         return build_running;
492                 }
493                 if (line->body.line.query != NULL) {
494                         delete_query_chain(line->body.line.query);
495                 }
496                 line->body.line.query = out_of_date_list;
497         }


500 /*
501  * If the target is a :: type, do not try to find the rule for the target,
502  * all actions will be taken by separate branches.
503  * Else, we try to find an implicit rule using various methods,
504  * we quit as soon as one is found.
505  *
506  * [tolik, 12 Sep 2002] Do not try to find implicit rule for the target
507  * being rechecked - the target is being rechecked means that it already
508  * has explicit dependencies derived from an implicit rule found
509  * in previous step.
510  */
511         if (target->colon_splits == 0 && !rechecking_target) {
512                 /* Look for percent matched rule */
513                 if ((result == build_dont_know) &&
514                     (command == NULL)) {
515                         switch (find_percent_rule(
516                                        target,
517                                        &command,
```



```
518                                        recheck_conditionals)) {
519                         case build_failed:
520                                 result = build_failed;
521                                 break;
522                         case build_running:
523                                 target->state = build_running;
524                                 add_pending(target,
525                                             --recursion_level,
526                                             do_get,
527                                             implicit,
528                                             false);
529                                 if (target->conditional_cnt > 0) {
530                                         reset_locals(target,
531                                                      old_locals,
532                                                      get_prop(target->prop,
533                                                               conditional_prop),
534                                                      0);
535                                 }
536                                 return build_running;
537                         case build_ok:
538                                 result = build_ok;
539                                 break;
540                         }
541                 }
542                 /* Look for double suffix rule */
543                 if (result == build_dont_know) {
544                         Property member;

546                         if (target->is_member &&
547                             ((member = get_prop(target->prop, member_prop)) !=
548                              NULL)) {
549                                 switch (find_ar_suffix_rule(target,
550                                                member->body.
551                                                member.member,
552                                                &command,
553                                                recheck_conditionals)) {
554                                 case build_failed:
555                                         result = build_failed;
556                                         break;
557                                 case build_running:
558                                         target->state = build_running;
559                                         add_pending(target,
560                                                     --recursion_level,
561                                                     do_get,
562                                                     implicit,
563                                                     false);
564                                     if (target->conditional_cnt > 0) {
565                                         reset_locals(target,
566                                                      old_locals,
567                                                      get_prop(target->prop,
568                                                        conditional_prop),
569                                                      0);
570                                     }
571                                         return build_running;
572                                 default:
573                                         /* ALWAYS bind $% for old style */
574                                         /* ar rules */
575                                         if (line == NULL) {
576                                                 line =
577                                                   maybe_append_prop(target,
578                                                                     line_prop);
579                                         }
580                                         line->body.line.percent =
581                                           member->body.member.member;
582                                         break;
583                                 }
```

```
584                         } else {
585                                 switch (find_double_suffix_rule(target,
586                                         &command,
587                                         recheck_conditionals)) {
588                                 case build_failed:
589                                         result = build_failed;
590                                         break;
591                                 case build_running:
592                                         target->state = build_running;
593                                         add_pending(target,
594                                                 --recursion_level,
595                                                 do_get,
596                                                 implicit,
597                                                 false);
598                                         if (target->conditional_cnt > 0) {
599                                                 reset_locals(target,
600                                                         old_locals,
601                                                         get_prop(target->
602                                                                 prop,
603                                                                 conditiona
604                                                         0);
605                                         }
606                                         return build_running;
607                                 }
608                         }
609                 }
610                 /* Look for single suffix rule */

612 /* /tolik/
613  * I commented !implicit to fix bug 1247448: Suffix Rules failed when combine wi
614  * This caused problem with SVR4 tilde rules (infinite recursion). So I made som
615  */
616 /* /tolik, 06.21.96/
617  * Regression! See BugId 1255360
618  * If more than one percent rules are defined for the same target then
619  * the behaviour of 'make' with my previous fix may be different from one
620  * of the 'old make'.
621  * The global variable second_pass (maybe it should be an argument to doname())
622  * is intended to avoid this regression. It is set in doname_check().
623  * First, 'make' will work as it worked before. Only when it is
624  * going to say "don't know how to make target" it sets second_pass to true and
625  * run 'doname' again but now trying to use Single Suffix Rules.
626  */
627                 if ((result == build_dont_know) && !automatic && (!implicit || s
628                     ((line == NULL) ||
629                      ((line->body.line.target != NULL) &&
630                       !line->body.line.target->has_regular_dependency))) {
631                         switch (find_suffix_rule(target,
632                                                  target,
633                                                  empty_name,
634                                                  &command,
635                                                  recheck_conditionals)) {
636                         case build_failed:
637                                 result = build_failed;
638                                 break;
639                         case build_running:
640                                 target->state = build_running;
641                                 add_pending(target,
642                                             --recursion_level,
643                                             do_get,
644                                             implicit,
645                                             false);
646                                 if (target->conditional_cnt > 0) {
647                                         reset_locals(target,
648                                                      old_locals,
649                                                      get_prop(target->prop,
```



```
650                                                                       conditional_prop),
651                                                              0);
652                                 }
653                                 return build_running;
654                         }
655                 }
656                 /* Try to sccs get */
657                 if ((command == NULL) &&
658                     (result == build_dont_know) &&
659                     do_get) {
660                         result = sccs_get(target, &command);
661                 }

663                 /* Use .DEFAULT rule if it is defined. */
664                 if ((command == NULL) &&
665                     (result == build_dont_know) &&
666                     (true_target->colons == no_colon) &&
667                     default_rule &&
668                     !implicit) {
669                         /* Make sure we have a line prop */
670                         line = maybe_append_prop(target, line_prop);
671                         command = line;
672                         Boolean out_of_date;
673                         if (true_target->is_member) {
674                                 out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
675                                                                         line->bo
676                         } else {
677                                 out_of_date = (Boolean) OUT_OF_DATE(true_target-
678                                                                     line->body.l
679                         }
680                         if (build_unconditional || out_of_date) {
681                                 line->body.line.is_out_of_date = true;
682                                 if (debug_level > 0) {
683                                         (void) printf(gettext("%*sBuilding %s us
684                                         (void) printf(catgets(catd, 1, 17, "%*sB
685                                                       recursion_level,
686                                                       "",
687                                                       true_target->string_mb);
688                                 }
689                         }
690                         line->body.line.sccs_command = false;
691                         line->body.line.command_template = default_rule;
692                         line->body.line.target = true_target;
693                         line->body.line.star = NULL;
694                         line->body.line.less = true_target;
695                         line->body.line.percent = NULL;
696                 }
697         }

699         /* We say "target up to date" if no cmd were executed for the target */
700         if (!target->is_double_colon_parent) {
701                 commands_done = false;
702         }

704         silent = silent_all;
705         ignore_errors = ignore_errors_all;
706         if  (posix)
707         {
708           if  (!silent)
709           {
710             silent = (Boolean) target->silent_mode;
711           }
712           if  (!ignore_errors)
713           {
714             ignore_errors = (Boolean) target->ignore_error_mode;
715           }
```

```
715         }
716
717         int doname_dyntarget = 0;
718 r_command:
719         /* Run commands if any. */
720         if ((command != NULL) &&
721             (command->body.line.command_template != NULL)) {
722                 if (result != build_failed) {
723                         result = run_command(command,
724                                         (Boolean) ((parallel || save_parall
725                 }
726                 switch (result) {
727                 case build_running:
728                         add_running(target,
729                                     true_target,
730                                     command,
731                                     --recursion_level,
732                                     auto_count,
733                                     automatics,
734                                     do_get,
735                                     implicit);
736                         target->state = build_running;
737                         if ((line = get_prop(target->prop,
738                                              line_prop)) != NULL) {
739                                 if (line->body.line.query != NULL) {
740                                         delete_query_chain(line->body.line.query
741                                 }
742                                 line->body.line.query = NULL;
743                         }
744                         if (target->conditional_cnt > 0) {
745                                 reset_locals(target,
746                                              old_locals,
747                                              get_prop(target->prop,
748                                                       conditional_prop),
749                                              0);
750                         }
751                         return build_running;
752                 case build_serial:
753                         add_serial(target,
754                                    --recursion_level,
755                                    do_get,
756                                    implicit);
757                         target->state = build_running;
758                         line = get_prop(target->prop, line_prop);
759                         if (line != NULL) {
760                                 if (line->body.line.query != NULL) {
761                                         delete_query_chain(line->body.line.query
762                                 }
763                                 line->body.line.query = NULL;
764                         }
765                         if (target->conditional_cnt > 0) {
766                                 reset_locals(target,
767                                              old_locals,
768                                              get_prop(target->prop,
769                                                       conditional_prop),
770                                              0);
771                         }
772                         return build_running;
773                 case build_ok:
774                         /* If all went OK set a nice timestamp */
775                         if (true_target->stat.time == file_doesnt_exist) {
776                                 true_target->stat.time = file_max_time;
777                         }
778                         break;
779                 }
780         } else {
```



```
781                 /*
782                  * If no command was found for the target, and it doesn't
783                  * exist, and it is mentioned as a target in the makefile,
784                  * we say it is extremely new and that it is OK.
785                  */
786                 if (target->colons != no_colon) {
787                         if (true_target->stat.time == file_doesnt_exist){
788                                 true_target->stat.time = file_max_time;
789                         }
790                         result = build_ok;
791                 }
792                 /*
793                  * Trying dynamic targets.
794                  */
795                 if(!doname_dyntarget) {
796                         doname_dyntarget = 1;
797                         Name dtarg = find_dyntarget(target);
798                         if(dtarg!=NULL) {
799                                 if (!target->has_depe_list_expanded) {
800                                         dynamic_dependencies(target);
801                                 }
802                                 if ((line = get_prop(target->prop, line_prop)) !
803                                         if (check_dependencies(&result,
804                                                                line,
805                                                                do_get,
806                                                                target,
807                                                                true_target,
808                                                                doing_subtree,
809                                                                &out_of_date_list
810                                                                old_locals,
811                                                                implicit,
812                                                                &command,
813                                                                less,
814                                                                rechecking_target
815                                                                recheck_condition
816                                         {
817                                                 return build_running;
818                                         }
819                                         if (line->body.line.query != NULL) {
820                                                 delete_query_chain(line->body.li
821                                         }
822                                         line->body.line.query = out_of_date_list
823                                 }
824                                 goto r_command;
825                         }
826                 }
827                 /*
828                  * If the file exists, it is OK that we couldnt figure
829                  * out how to build it.
830                  */
831                 (void) exists(target);
832                 if ((target->stat.time != file_doesnt_exist) &&
833                     (result == build_dont_know)) {
834                         result = build_ok;
835                 }
836         }
837
838         /*
839          * Some of the following is duplicated in the function finish_doname.
840          * If anything is changed here, check to see if it needs to be
841          * changed there.
842          */
843         if ((line = get_prop(target->prop, line_prop)) != NULL) {
844                 if (line->body.line.query != NULL) {
845                         delete_query_chain(line->body.line.query);
846                 }
```

```
847                 line->body.line.query = NULL;
848         }
849         target->state = result;
850         parallel = save_parallel;
851         if (target->conditional_cnt > 0) {
852                 reset_locals(target,
853                              old_locals,
854                              get_prop(target->prop, conditional_prop),
855                              0);
856         }
857         recursion_level--;
858         if (target->is_member) {
859                 Property member;

861                 /* Propagate the timestamp from the member file to the member*/
862                 if ((target->stat.time != file_max_time) &&
863                     ((member = get_prop(target->prop, member_prop)) != NULL) &&
864                     (exists(member->body.member.member) > file_doesnt_exist)) {
865                         target->stat.time =
866                           member->body.member.member->stat.time;
867                 }
868         }
869         /*
870          * Check if we found any new auto dependencies when we
871          * built the target.
872          */
873         if ((result == build_ok) && check_auto_dependencies(target,
874                                                             auto_count,
875                                                             automatics)) {
876                 if (debug_level > 0) {
877                         (void) printf(gettext("%*sTarget '%s' acquired new depen
882                         (void) printf(catgets(catd, 1, 18, "%*sTarget '%s' acqui
878                                       recursion_level,
879                                       "",
880                                       true_target->string_mb);
881                 }
882                 rechecking_target = true;
883                 saved_commands_done = commands_done;
884                 goto recheck_target;
885         }

887         if (rechecking_target && !commands_done) {
888                 commands_done = saved_commands_done;
889         }

891         return result;
892 }

894 /*
895  * DONE.
896  *
897  *      check_dependencies(result, line, do_get,
898  *                         target, true_target, doing_subtree, out_of_date_tail,
899  *                         old_locals, implicit, command, less, rechecking_target)
900  *
901  *      Return value:
902  *                              True returned if some dependencies left running
903  *
904  *      Parameters:
905  *              result          Pointer to cell we update if build failed
906  *              line            We get the dependencies from here
907  *              do_get          Allow use of sccs get in recursive doname()
908  *              target          The target to chase dependencies for
909  *              true_target     The real one for :: and lib(member)
910  *              doing_subtree   True if building a conditional macro subtree
911  *              out_of_date_tail Used to set the $? list
```



```
912  *              old_locals      Used for resetting the local macros
913  *              implicit        Called when scanning for implicit rules?
914  *              command         Place to stuff command
915  *              less            Set to $< value
916  *
917  *      Global variables used:
918  *              command_changed Set if we suspect .make.state needs rewrite
919  *              debug_level     Should we trace actions?
920  *              force           The Name " FORCE", compared against
921  *              recursion_level Used for tracing
922  *              rewrite_statefile Set if .make.state needs rewriting
923  *              wait_name       The Name ".WAIT", compared against
924  */
925 static Boolean
926 check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
927 {
928         Boolean                 dependencies_running;
929         register Dependency     dependency;
930         Doname                  dep_result;
931         Boolean                 dependency_changed = false;

933         line->body.line.dependency_time = file_doesnt_exist;
934         if (line->body.line.query != NULL) {
935                 delete_query_chain(line->body.line.query);
936         }
937         line->body.line.query = NULL;
938         line->body.line.is_out_of_date = false;
939         dependencies_running = false;
940         /*
941          * Run thru all the dependencies and call doname() recursively
942          * on each of them.
943          */
944         for (dependency = line->body.line.dependencies;
945              dependency != NULL;
946              dependency = dependency->next) {
947                 Boolean this_dependency_changed = false;

949                 if (!dependency->automatic &&
950                     (rechecking_target || target->rechecking_target)) {
951                         /*
952                          * We only bother with the autos when rechecking
953                          */
954                         continue;
955                 }

957                 if (dependency->name == wait_name) {
958                         /*
959                          * The special target .WAIT means finish all of
960                          * the prior dependencies before continuing.
961                          */
962                         if (dependencies_running) {
963                                 break;
964                         }
965                 } else if ((!parallel_ok(dependency->name, false)) &&
966                            (dependencies_running)) {
967                         /*
968                          * If we can't execute the current dependency in
969                          * parallel, hold off the dependency processing
970                          * to preserve the order of the dependencies.
971                          */
972                         break;
973                 } else {
974                         timestruc_t     depe_time = file_doesnt_exist;


977                         if (true_target->is_member) {
```

```
 978                             depe_time = exists(dependency->name);
 979                     }
 980                     if (dependency->built ||
 981                         (dependency->name->state == build_failed)) {
 982                             dep_result = (Doname) dependency->name->state;
 983                     } else {
 984                             dep_result = doname_check(dependency->name,
 985                                                       do_get,
 986                                                       false,
 987                                                       (Boolean) dependency->
 988                     }
 989                     if (true_target->is_member || dependency->name->is_membe
 990                             /* should compare only secs, cause lib members d
 991                             if (depe_time.tv_sec != dependency->name->stat.t
 992                                     this_dependency_changed =
 993                                       dependency_changed =
 994                                         true;
 995                             }
 996                     } else {
 997                             if (depe_time != dependency->name->stat.time) {
 998                                     this_dependency_changed =
 999                                       dependency_changed =
1000                                         true;
1001                             }
1002                     }
1003                     dependency->built = true;
1004                     switch (dep_result) {
1005                     case build_running:
1006                             dependencies_running = true;
1007                             continue;
1008                     case build_failed:
1009                             *result = build_failed;
1010                             break;
1011                     case build_dont_know:
1012 /*
1013  * If make can't figure out how to make a dependency, maybe the dependency
1014  * is out of date. In this case, we just declare the target out of date
1015  * and go on. If we really need the dependency, the make'ing of the target
1016  * will fail. This will only happen for automatic (hidden) dependencies.
1017  */
1018                             if(!recheck_conditionals) {
1019                                     line->body.line.is_out_of_date = true;
1020                             }
1021                             /*
1022                              * Make sure the dependency is not saved
1023                              * in the state file.
1024                              */
1025                             dependency->stale = true;
1026                             rewrite_statefile =
1027                               command_changed =
1028                                 true;
1029                             if (debug_level > 0) {
1030                                     (void) printf(gettext("Target %s rebuilt
1035                                     (void) printf(catgets(catd, 1, 19, "Targ
1031                                                   true_target->string_mb,
1032                                                   dependency->name->string_mb
1033                             }
1034                             break;
1035                     }
1036                     if (dependency->name->depends_on_conditional) {
1037                             target->depends_on_conditional = true;
1038                     }
1039                     if (dependency->name == force) {
1040                             target->stat.time =
1041                               dependency->name->stat.time;
1042                     }
```



```
1043                     /*
1044                      * Propagate new timestamp from "member" to
1045                      * "lib.a(member)".
1046                      */
1047                     (void) exists(dependency->name);

1049                     /* Collect the timestamp of the youngest dependency */
1050                     line->body.line.dependency_time =
1051                       MAX(dependency->name->stat.time,
1052                           line->body.line.dependency_time);

1054                     /* Correction: do not consider nanosecs for members */
1055                     if(true_target->is_member || dependency->name->is_member
1056                             line->body.line.dependency_time.tv_nsec = 0;
1057                     }

1059                     if (debug_level > 1) {
1060                             (void) printf(gettext("%*sDate(%s)=%s \n"),
1065                             (void) printf(catgets(catd, 1, 20, "%*sDate(%s)=
1061                                           recursion_level,
1062                                           "",
1063                                           dependency->name->string_mb,
1064                                           time_to_string(dependency->name->
1065                                                          stat.time));
1066                             if (dependency->name->stat.time > line->body.lin
1067                                     (void) printf(gettext("%*sDate-dependenc
1072                                     (void) printf(catgets(catd, 1, 21, "%*sD
1068                                                   recursion_level,
1069                                                   "",
1070                                                   true_target->string_mb,
1071                                                   time_to_string(line->body.
1072                                                                  dependency_
1073                             }
1074                     }

1076                     /* Build the $? list */
1077                     if (true_target->is_member) {
1078                             if (this_dependency_changed == true) {
1079                                     true_target->stat.time = dependency->nam
1080                                     true_target->stat.time.tv_sec--;
1081                             } else {
1082                                     /* Dina:
1083                                      * The next statement is commented
1084                                      * out as a fix for bug #1051032.
1085                                      * if dependency hasn't changed
1086                                      * then there's no need to invalidate
1087                                      * true_target. This statemnt causes
1088                                      * make to take much longer to process
1089                                      * an already-built archive. Soren
1090                                      * said it was a quick fix for some
1091                                      * problem he doesn't remember.
1092                                     true_target->stat.time = file_no_time;
1093                                      */
1094                                     (void) exists(true_target);
1095                             }
1096                     } else {
1097                             (void) exists(true_target);
1098                     }
1099                     Boolean out_of_date;
1100                     if (true_target->is_member || dependency->name->is_membe
1101                             out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
1102                                                                     dependen
1103                     } else {
1104                             out_of_date = (Boolean) OUT_OF_DATE(true_target-
1105                                                                 dependency->
1106                     }
```

```
1107                        if ((build_unconditional || out_of_date) &&
1108                            (dependency->name != force) &&
1109                            (dependency->stale == false)) {
1110                                *out_of_date_tail = ALLOC(Chain);
1111                                if (dependency->name->is_member &&
1112                                    (get_prop(dependency->name->prop,
1113                                              member_prop) != NULL)) {
1114                                        (*out_of_date_tail)->name =
1115                                          get_prop(dependency->name->prop,
1116                                                   member_prop)->
1117                                                     body.member.member;
1118                                } else {
1119                                        (*out_of_date_tail)->name =
1120                                          dependency->name;
1121                                }
1122                                (*out_of_date_tail)->next = NULL;
1123                                out_of_date_tail = &(*out_of_date_tail)->next;
1124                                if (debug_level > 0) {
1125                                        if (dependency->name->stat.time == file_
1126                                                (void) printf(gettext("%*sBuildi
1131                                                (void) printf(catgets(catd, 1, 2
1127                                                        recursion_level,
1128                                                        "",
1129                                                        true_target->strin
1130                                                        dependency->name->
1131                                        } else {
1132                                                (void) printf(gettext("%*sBuildi
1137                                                (void) printf(catgets(catd, 1, 2
1133                                                        recursion_level,
1134                                                        "",
1135                                                        true_target->strin
1136                                                        dependency->name->
1137                                        }
1138                                }
1139                        }
1140                        if (dependency->name == force) {
1141                                force->stat.time =
1142                                  file_max_time;
1143                                force->state = build_dont_know;
1144                        }
1145                }
1146        }
1147        if (dependencies_running) {
1148                if (doing_subtree) {
1149                        if (target->conditional_cnt > 0) {
1150                                reset_locals(target,
1151                                        old_locals,
1152                                        get_prop(target->prop,
1153                                                 conditional_prop),
1154                                        0);
1155                        }
1156                        return true;
1157                } else {
1158                        target->state = build_running;
1159                        add_pending(target,
1160                                --recursion_level,
1161                            do_get,
1162                            implicit,
1163                            false);
1164                        if (target->conditional_cnt > 0) {
1165                                reset_locals(target,
1166                                        old_locals,
1167                                        get_prop(target->prop,
1168                                                 conditional_prop),
1169                                        0);
1170                        }
```



```
1171                            return true;
1172                    }
1173            }
1174            /*
1175             * Collect the timestamp of the youngest double colon target
1176             * dependency.
1177             */
1178            if (target->is_double_colon_parent) {
1179                    for (dependency = line->body.line.dependencies;
1180                         dependency != NULL;
1181                         dependency = dependency->next) {
1182                            Property        tmp_line;

1184                            if ((tmp_line = get_prop(dependency->name->prop, line_pr
1185                                    if(tmp_line->body.line.dependency_time != file_m
1186                                            target->stat.time =
1187                                              MAX(tmp_line->body.line.dependency_tim
1188                                                  target->stat.time);
1189                                    }
1190                            }
1191                    }
1192            }
1193            if ((true_target->is_member) && (dependency_changed == true)) {
1194                    true_target->stat.time = file_no_time;
1195            }
1196            /*
1197             * After scanning all the dependencies, we check the rule
1198             * if we found one.
1199             */
1200            if (line->body.line.command_template != NULL) {
1201                    if (line->body.line.command_template_redefined) {
1202                            warning(gettext("Too many rules defined for target %s"),
1207                            warning(catgets(catd, 1, 24, "Too many rules defined for
1203                                    target->string_mb);
1204                    }
1205                    *command = line;
1206                    /* Check if the target is out of date */
1207                    Boolean out_of_date;
1208                    if (true_target->is_member) {
1209                            out_of_date = (Boolean) OUT_OF_DATE_SEC(true_target->sta
1210                                                                    line->body.line.
1211                    } else {
1212                            out_of_date = (Boolean) OUT_OF_DATE(true_target->stat.ti
1213                                                                line->body.line.depe
1214                    }
1215                    if (build_unconditional || out_of_date){
1216                            if(!recheck_conditionals) {
1217                                    line->body.line.is_out_of_date = true;
1218                            }
1219                    }
1220                    line->body.line.sccs_command = false;
1221                    line->body.line.target = true_target;
1222                    if(gnu_style) {

1224                            // set $< for explicit rule
1225                            if(line->body.line.dependencies != NULL) {
1226                                    less = line->body.line.dependencies->name;
1227                            }

1229                            // set $* for explicit rule
1230                            Name                    target_body;
1231                            Name                    tt = true_target;
1232                            Property                member;
1233                            register wchar_t        *target_end;
1234                            register Dependency     suffix;
1235                            register int            suffix_length;
```

```
1236                        Wstring                 targ_string;
1237                        Wstring                 suf_string;

1239                        if (true_target->is_member &&
1240                            ((member = get_prop(target->prop, member_prop)) !=
1241                             NULL)) {
1242                               tt = member->body.member.member;
1243                        }
1244                        targ_string.init(tt);
1245                        target_end = targ_string.get_string() + tt->hash.length;
1246                        for (suffix = suffixes; suffix != NULL; suffix = suffix-
1247                                suffix_length = suffix->name->hash.length;
1248                                suf_string.init(suffix->name);
1249                                if (tt->hash.length < suffix_length) {
1250                                        continue;
1251                                } else if (!IS_WEQUALN(suf_string.get_string(),
1252                                                (target_end - suffix_length),
1253                                                suffix_length)) {
1254                                        continue;
1255                                }
1256                                target_body = GETNAME(
1257                                        targ_string.get_string(),
1258                                        (int)(tt->hash.length - suffix_length)
1259                                );
1260                                line->body.line.star = target_body;
1261                        }

1263                        // set result = build_ok so that implicit rules are not
1264                        if(*result == build_dont_know) {
1265                                *result = build_ok;
1266                        }
1267                }
1268                if (less != NULL) {
1269                        line->body.line.less = less;
1270                }
1271        }

1273        return false;
1274 }

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, "$(@F)" translates to the same thing without a preceeding
1289  *      directory path (if one exists).
1290  *      Note, to enter "$@" 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
1427                                        (void) printf(catgets(catd, 1, 25, "%*sD
1423                                                      recursion_level,
1424                                                      "",
1425                                                      dependency->name->string_m
1426                                                      true_target->string_mb);
1427                                }
1428                                for (start = p; iswspace(*start); start++);
1429                                p = start;
1430                        }
1431                } while ((*p != (int) nul_char) &&
1432                         (*p != (int) parenright_char));
```



```
1433                /* If the expansion was of lib.a(members) format we now */
1434                /* enter the proper member cells */
1435                if (first_member != NULL) {
1436                        /* Scan the new dependencies and transform them from */
1437                        /* "foo" to "lib.a(foo)" */
1438                        for (; 1; first_member = first_member->next) {
1439                                /* Build "lib.a(foo)" name */
1440                                INIT_STRING_FROM_STACK(string, buffer);
1441                                APPEND_NAME(lib,
1442                                            &string,
1443                                            (int) lib->hash.length);
1444                                append_char((int) parenleft_char, &string);
1445                                APPEND_NAME(first_member->name,
1446                                            &string,
1447                                            FIND_LENGTH);
1448                                append_char((int) parenright_char, &string);
1449                                member = first_member->name;
1450                                /* Replace "foo" with "lib.a(foo)" */
1451                                first_member->name =
1452                                  GETNAME(string.buffer.start, FIND_LENGTH);
1453                                if (string.free_after_use) {
1454                                        retmem(string.buffer.start);
1455                                }
1456                                if (debug_level > 0) {
1457                                        (void) printf(gettext("%*sDynamic depend
1462                                        (void) printf(catgets(catd, 1, 26, "%*sD
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 *) wschr(wcb.get_string(),
1492                                        (int) parenleft_char)) != NULL) {
1493                                /* lib(*) type pattern */
1494                                library = buffer;
1495                                (void) wsncpy(buffer,
1496                                              wcb.get_string(),
1497                                              start - wcb.get_string());
```

```
1498                                buffer[start-wcb.get_string()] =
1499                                  (int) nul_char;
1500                                (void) wsncpy(pattern,
1501                                              start + 1,
1502 (int) (dependency->name->hash.length-(start-wcb.get_string())-2));
1503                                pattern[dependency->name->hash.length -
1504                                        (start-wcb.get_string()) - 2] =
1505                                          (int) nul_char;
1506                        } else {
1507                                library = NULL;
1508                                (void) wsncpy(pattern,
1509                                              wcb.get_string(),
1510                                              (int) dependency->name->hash.lengt
1511                                pattern[dependency->name->hash.length] =
1512                                  (int) nul_char;
1513                        }
1514                        start = (wchar_t *) wsrchr(pattern, (int) slash_char);
1515                        if (start == NULL) {
1516                                directory = dot;
1517                                p = pattern;
1518                        } else {
1519                                directory = GETNAME(pattern, start-pattern);
1520                                p = start+1;
1521                        }
1522                        /* The expansion is handled by the read_dir() routine*/
1523                        if (read_dir(directory, p, line, library)) {
1524                                *remove = (*remove)->next;
1525                        } else {
1526                                remove = &dependency->next;
1527                        }
1528                } else {
1529                        remove = &dependency->next;
1530                }
1531        }

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

1537 /*
1538  * DONE.
1539  *
1540  *      run_command(line)
1541  *
1542  *      Takes one Cmd_line and runs the commands from it.
1543  *
1544  *      Return value:
1545  *                              Indicates if the command failed or not
1546  *
1547  *      Parameters:
1548  *              line            The command line to run
1549  *
1550  *      Global variables used:
1551  *              commands_done   Set if we do run command
1552  *              current_line    Set to the line we run a command from
1553  *              current_target  Set to the target we run a command for
1554  *              file_number     Used to form temp file name
1555  *              keep_state      Indicates that .KEEP_STATE is on
1556  *              make_state      The Name ".make.state", used to check timestamp
1557  *              parallel        True if currently building in parallel
1558  *              parallel_process_cnt Count of parallel processes running
1559  *              quest           Indicates that make -q is on
1560  *              rewrite_statefile Set if we do run a command
1561  *              sunpro_dependencies The Name "SUNPRO_DEPENDENCIES", set value
1562  *              temp_file_directory Used to form temp fie name
1563  *              temp_file_name  Set to the name of the temp file
```



```
1564  *              touch           Indicates that make -t is on
1565  */
1566 static Doname
1567 run_command(register Property line, Boolean)
1568 {
1569         register Doname         result = build_ok;
1570         register Boolean        remember_only = false;
1571         register Name           target = line->body.line.target;
1572         wchar_t                 *string;
1573         char                    tmp_file_path[MAXPATHLEN];

1575         if (!line->body.line.is_out_of_date && target->rechecking_target) {
1576                 target->rechecking_target = false;
1577                 return build_ok;
1578         }

1580         /*
1581          * Build the command if we know the target is out of date,
1582          * or if we want to check cmd consistency.
1583          */
1584         if (line->body.line.is_out_of_date || keep_state) {
1585                 /* Hack for handling conditional macros in DMake. */
1586                 if (!line->body.line.dont_rebuild_command_used) {
1587                         build_command_strings(target, line);
1588                 }
1589         }
1590         /* Never mind */
1591         if (!line->body.line.is_out_of_date) {
1592                 return build_ok;
1593         }
1594         /* If quest, then exit(1) because the target is out of date */
1595         if (quest) {
1596                 if (posix) {
1597                         result = execute_parallel(line, true);
1598                 }
1599                 exit_status = 1;
1600                 exit(1);
1601         }
1602         /* We actually had to do something this time */
1603         rewrite_statefile = commands_done = true;
1604         /*
1605          * If this is an sccs command, we have to do some extra checking
1606          * and possibly complain. If the file can't be gotten because it's
1607          * checked out, we complain and behave as if the command was
1608          * executed eventhough we ignored the command.
1609          */
1610         if (!touch &&
1611             line->body.line.sccs_command &&
1612             (target->stat.time != file_doesnt_exist) &&
1613             ((target->stat.mode & 0222) != 0)) {
1614                 fatal(gettext("%s is writable so it cannot be sccs gotten"),
1619                 fatal(catgets(catd, 1, 27, "%s is writable so it cannot be sccs
1615                       target->string_mb);
1616                 target->has_complained = remember_only = true;
1617         }
1618         /*
1619          * If KEEP_STATE is on, we make sure we have the timestamp for
1620          * .make.state. If .make.state changes during the command run,
1621          * we reread .make.state after the command. We also setup the
1622          * environment variable that asks utilities to report dependencies.
1623          */
1624         if (!touch &&
1625             keep_state &&
1626             !remember_only) {
1627                 (void) exists(make_state);
1628                 if((strlen(temp_file_directory) == 1) &&
```

```
1629                        (temp_file_directory[0] == '/')) {
1630                    tmp_file_path[0] = '\0';
1631                } else {
1632                    strcpy(tmp_file_path, temp_file_directory);
1633                }
1634                sprintf(mbs_buffer,
1635                                "%s/.make.dependency.%08x.%d.%d",
1640                                NOCATGETS("%s/.make.dependency.%08x.%d.%d"),
1636                                tmp_file_path,
1637                                hostid,
1638                                getpid(),
1639                                file_number++);
1640                MBSTOWCS(wcs_buffer, mbs_buffer);
1641                Boolean fnd;
1642                temp_file_name = getname_fn(wcs_buffer, FIND_LENGTH, false, &fnd
1643                temp_file_name->stat.is_file = true;
1644                int len = 2*MAXPATHLEN + strlen(target->string_mb) + 2;
1645                wchar_t *to = string = ALLOC_WC(len);
1646                for (wchar_t *from = wcs_buffer; *from != (int) nul_char; ) {
1647                        if (*from == (int) space_char) {
1648                                *to++ = (int) backslash_char;
1649                        }
1650                        *to++ = *from++;
1651                }
1652                *to++ = (int) space_char;
1653                MBSTOWCS(to, target->string_mb);
1654                Name sprodep_name = getname_fn(string, FIND_LENGTH, false, &fnd)
1655                (void) SETVAR(sunpro_dependencies,
1656                              sprodep_name,
1657                              false);
1658                retmem(string);
1659        } else {
1660                temp_file_name = NULL;
1661        }

1663        /*
1664         * In case we are interrupted, we need to know what was going on.
1665         */
1666        current_target = target;
1667        /*
1668         * We also need to be able to save an empty command instead of the
1669         * interrupted one in .make.state.
1670         */
1671        current_line = line;
1672        if (remember_only) {
1673                /* Empty block!!! */
1674        } else if (touch) {
1675                result = touch_command(line, target, result);
1676                if (posix) {
1677                        result = execute_parallel(line, true);
1678                }
1679        } else {
1680                /*
1681                 * If this is not a touch run, we need to execute the
1682                 * proper command(s) for the target.
1683                 */
1684                if (parallel) {
1685                        if (!parallel_ok(target, true)) {
1686                                /*
1687                                 * We are building in parallel, but
1688                                 * this target must be built in serial.
1689                                 */
1690                                /*
1691                                 * If nothing else is building,
1692                                 * do this one, else wait.
1693                                 */
```



```
1694                                if (parallel_process_cnt == 0) {
1695                                        result = execute_parallel(line, true, ta
1696                                } else {
1697                                        current_target = NULL;
1698                                        current_line = NULL;
1699 /*
1700                                        line->body.line.command_used = NULL;
1701  */
1702                                        line->body.line.dont_rebuild_command_use
1703                                        return build_serial;
1704                                }
1705                        } else {
1706                                result = execute_parallel(line, false);
1707                                switch (result) {
1708                                case build_running:
1709                                        return build_running;
1710                                case build_serial:
1711                                        if (parallel_process_cnt == 0) {
1712                                                result = execute_parallel(line,
1713                                        } else {
1714                                                current_target = NULL;
1715                                                current_line = NULL;
1716                                                target->parallel = false;
1717                                                line->body.line.command_used =
1718                                                                        NULL;
1719                                                return build_serial;
1720                                        }
1721                                }
1722                        }
1723                } else {
1724                        result = execute_parallel(line, true, target->localhost)
1725                }
1726        }
1727        temp_file_name = NULL;
1728        if (report_dependencies_level == 0){
1729                update_target(line, result);
1730        }
1731        current_target = NULL;
1732        current_line = NULL;
1733        return result;
1734 }

1736 /*
1737  *      execute_serial(line)
1738  *
1739  *      Runs thru the command line for the target and
1740  *      executes the rules one by one.
1741  *
1742  *      Return value:
1743  *                              The result of the command build
1744  *
1745  *      Parameters:
1746  *              line            The command to execute
1747  *
1748  *      Static variables used:
1749  *
1750  *      Global variables used:
1751  *              continue_after_error -k flag
1752  *              do_not_exec_rule -n flag
1753  *              report_dependencies -P flag
1754  *              silent          Don't echo commands before executing
1755  *              temp_file_name  Temp file for auto dependencies
1756  *              vpath_defined   If true, translate path for command
1757  */
1758 Doname
1759 execute_serial(Property line)
```

```
1760 {
1761         int                     child_pid = 0;
1762         Boolean                 printed_serial;
1763         Doname                  result = build_ok;
1764         Cmd_line                rule, cmd_tail, command = NULL;
1765         char                    mbstring[MAXPATHLEN];
1766         int                     filed;
1767         Name                    target = line->body.line.target;

1769         target->has_recursive_dependency = false;
1770         // We have to create a copy of the rules chain for processing because
1771         // the original one can be destroyed during .make.state file rereading.
1772         for (rule = line->body.line.command_used;
1773              rule != NULL;
1774              rule = rule->next) {
1775                 if (command == NULL) {
1776                         command = cmd_tail = ALLOC(Cmd_line);
1777                 } else {
1778                         cmd_tail->next = ALLOC(Cmd_line);
1779                         cmd_tail = cmd_tail->next;
1780                 }
1781                 *cmd_tail = *rule;
1782         }
1783         if (command) {
1784                 cmd_tail->next = NULL;
1785         }
1786         for (rule = command; rule != NULL; rule = rule->next) {
1787                 if (posix && (touch || quest) && !rule->always_exec) {
1788                         continue;
1789                 }
1790                 if (vpath_defined) {
1791                         rule->command_line =
1792                           vpath_translation(rule->command_line);
1793                 }
1794                 /* Echo command line, maybe. */
1795                 if ((rule->command_line->hash.length > 0) &&
1796                     !silent &&
1797                     (!rule->silent || do_not_exec_rule) &&
1798                     (report_dependencies_level == 0)) {
1799                         (void) printf("%s\n", rule->command_line->string_mb);
1800                 }
1801                 if (rule->command_line->hash.length > 0) {
1802                         /* Do assignment if command line prefixed with "=" */
1803                         if (rule->assign) {
1804                                 result = build_ok;
1805                                 do_assign(rule->command_line, target);
1806                         } else if (report_dependencies_level == 0) {
1807                                 /* Execute command line. */
1808                                 setvar_envvar();
1809                                 result = dosys(rule->command_line,
1810                                                (Boolean) rule->ignore_error,
1811                                                (Boolean) rule->make_refd,
1812                                                /* ds 98.04.23 bug #4085164. make
1813                                                false,
1814                                                /* BOOLEAN(rule->silent &&
1815                                                        rule->ignore_error), */
1816                                                (Boolean) rule->always_exec,
1817                                                target);
1818                                 check_state(temp_file_name);
1819                         }
1820                 } else {
1821                         result = build_ok;
1822                 }
1823                 if (result == build_failed) {
1824                         if (silent || rule->silent) {
1825                                 (void) printf(gettext("The following command cau
```



```
1830                                     (void) printf(catgets(catd, 1, 242, "The followi
1826                                             rule->command_line->string_mb);
1827                         }
1828                         if (!rule->ignore_error && !ignore_errors) {
1829                                 if (!continue_after_error) {
1830                                         fatal(gettext("Command failed for target
1835                                         fatal(catgets(catd, 1, 244, "Command fai
1831                                             target->string_mb);
1832                                 }
1833                                 /*
1834                                  * Make sure a failing command is not
1835                                  * saved in .make.state.
1836                                  */
1837                                 line->body.line.command_used = NULL;
1838                                 break;
1839                         } else {
1840                                 result = build_ok;
1841                         }
1842                 }
1843         }
1844         for (rule = command; rule != NULL; rule = cmd_tail) {
1845                 cmd_tail = rule->next;
1846                 free(rule);
1847         }
1848         command = NULL;
1849         if (temp_file_name != NULL) {
1850                 free_name(temp_file_name);
1851         }
1852         temp_file_name = NULL;

1854         Property spro = get_prop(sunpro_dependencies->prop, macro_prop);
1855         if(spro != NULL) {
1856                 Name val = spro->body.macro.value;
1857                 if(val != NULL) {
1858                         free_name(val);
1859                         spro->body.macro.value = NULL;
1860                 }
1861         }
1862         spro = get_prop(sunpro_dependencies->prop, env_mem_prop);
1863         if(spro) {
1864                 char *val = spro->body.env_mem.value;
1865                 if(val != NULL) {
1866                         /*
1867                          * Do not return memory allocated for SUNPRO_DEPENDENCIE
1868                          * It will be returned in setvar_daemon() in macro.cc
1869                          */
1870                         //      retmem_mb(val);
1871                         spro->body.env_mem.value = NULL;
1872                 }
1873         }
1874
1875         return result;
1876 }
_____unchanged_portion_omitted_

2060 /*
2061  *      do_assign(line, target)
2062  *
2063  *      Handles runtime assignments for command lines prefixed with "=".
2064  *
2065  *      Parameters:
2066  *              line            The command that contains an assignment
2067  *              target          The Name of the target, used for error reports
2068  *
2069  *      Global variables used:
2070  *              assign_done     Set to indicate doname needs to reprocess
```

```
 */
static void
do_assign(register Name line, register Name target)
{
        Wstring wcb(line);
        register wchar_t        *string = wcb.get_string();
        register wchar_t        *equal;
        register Name           name;
        register Boolean        append = false;

        /*
         * If any runtime assignments are done, doname() must reprocess all
         * targets in the future since the macro values used to build the
         * command lines for the targets might have changed.
         */
        assign_done = true;
        /* Skip white space. */
        while (iswspace(*string)) {
                string++;
        }
        equal = string;
        /* Find "+=" or "=". */
        while (!iswspace(*equal) &&
               (*equal != (int) plus_char) &&
               (*equal != (int) equal_char)) {
                equal++;
        }
        /* Internalize macro name. */
        name = GETNAME(string, equal - string);
        /* Skip over "+=" "=". */
        while (!((*equal == (int) nul_char) ||
                 (*equal == (int) equal_char) ||
                 (*equal == (int) plus_char))) {
                equal++;
        }
        switch (*equal) {
        case nul_char:
                fatal(gettext("= expected in rule `%s' for target `%s'"),
                fatal(catgets(catd, 1, 31, "= expected in rule `%s' for target `
                      line->string_mb,
                      target->string_mb);
        case plus_char:
                append = true;
                equal++;
                break;
        }
        equal++;
        /* Skip over whitespace in front of value. */
        while (iswspace(*equal)) {
                equal++;
        }
        /* Enter new macro value. */
        enter_equal(name,
                    GETNAME(equal, wcb.get_string() + line->hash.length - equal)
                    append);
}

/*
 *      build_command_strings(target, line)
 *
 *      Builds the command string to used when
 *      building a target. If the string is different from the previous one
 *      is_out_of_date is set.
 *
 *      Parameters:
 *              target          Target to build commands for
```



```
 *              line            Where to stuff result
 *
 *      Global variables used:
 *              c_at            The Name "@", used to set macro value
 *              command_changed Set if command is different from old
 *              debug_level     Should we trace activities?
 *              do_not_exec_rule Always echo when running -n
 *              empty_name      The Name "", used for empty rule
 *              funny           Semantics of characters
 *              ignore_errors   Used to init field for line
 *              is_conditional  Set to false befor evaling macro, checked
 *                              after expanding macros
 *              keep_state      Indicates that .KEEP_STATE is on
 *              make_word_mentioned Set by macro eval, inits field for cmd
 *              query           The Name "?", used to set macro value
 *              query_mentioned Set by macro eval, inits field for cmd
 *              recursion_level Used for tracing
 *              silent          Used to init field for line
 */
static void
build_command_strings(Name target, register Property line)
{
        String_rec              command_line;
        register Cmd_line       command_template = line->body.line.command_templ
        register Cmd_line       *insert = &line->body.line.command_used;
        register Cmd_line       used = *insert;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        wchar_t                 *start;
        Name                    new_command_line;
        register Boolean        new_command_longer = false;
        register Boolean        ignore_all_command_dependency = true;
        Property                member;
        static Name             less_name;
        static Name             percent_name;
        static Name             star;
        Name                    tmp_name;

        if (less_name == NULL) {
                MBSTOWCS(wcs_buffer, "<");
                less_name = GETNAME(wcs_buffer, FIND_LENGTH);
                MBSTOWCS(wcs_buffer, "%");
                percent_name = GETNAME(wcs_buffer, FIND_LENGTH);
                MBSTOWCS(wcs_buffer, "*");
                star = GETNAME(wcs_buffer, FIND_LENGTH);
        }

        /* We have to check if a target depends on conditional macros */
        /* Targets that do must be reprocessed by doname() each time around */
        /* since the macro values used when building the target might have */
        /* changed */
        conditional_macro_used = false;
        /* If we are building a lib.a(member) target $@ should be bound */
        /* to lib.a */
        if (target->is_member &&
            ((member = get_prop(target->prop, member_prop)) != NULL)) {
                target = member->body.member.library;
        }
        /* If we are building a "::" help target $@ should be bound to */
        /* the real target name */
        /* A lib.a(member) target is never :: */
        if (target->has_target_prop) {
                target = get_prop(target->prop, target_prop)->
                  body.target.target;
        }
        /* Bind the magic macros that make supplies */
        tmp_name = target;
```

```
2202         if(tmp_name != NULL) {
2203                 if (tmp_name->has_vpath_alias_prop) {
2204                         tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
2205                                         body.vpath_alias.alias;
2206                 }
2207         }
2208         (void) SETVAR(c_at, tmp_name, false);

2210         tmp_name = line->body.line.star;
2211         if(tmp_name != NULL) {
2212                 if (tmp_name->has_vpath_alias_prop) {
2213                         tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
2214                                         body.vpath_alias.alias;
2215                 }
2216         }
2217         (void) SETVAR(star, tmp_name, false);

2219         tmp_name = line->body.line.less;
2220         if(tmp_name != NULL) {
2221                 if (tmp_name->has_vpath_alias_prop) {
2222                         tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
2223                                         body.vpath_alias.alias;
2224                 }
2225         }
2226         (void) SETVAR(less_name, tmp_name, false);

2228         tmp_name = line->body.line.percent;
2229         if(tmp_name != NULL) {
2230                 if (tmp_name->has_vpath_alias_prop) {
2231                         tmp_name = get_prop(tmp_name->prop, vpath_alias_prop)->
2232                                         body.vpath_alias.alias;
2233                 }
2234         }
2235         (void) SETVAR(percent_name, tmp_name, false);

2237         /* $? is seldom used and it is expensive to build */
2238         /* so we store the list form and build the string on demand */
2239         Chain query_list = NULL;
2240         Chain *query_list_tail = &query_list;

2242         for (Chain ch = line->body.line.query; ch != NULL; ch = ch->next) {
2243                 *query_list_tail = ALLOC(Chain);
2244                 (*query_list_tail)->name = ch->name;
2245                 if ((*query_list_tail)->name->has_vpath_alias_prop) {
2246                         (*query_list_tail)->name =
2247                                 get_prop((*query_list_tail)->name->prop,
2248                                         vpath_alias_prop)->body.vpath_alias.alia
2249                 }
2250                 (*query_list_tail)->next = NULL;
2251                 query_list_tail = &(*query_list_tail)->next;
2252         }
2253         (void) setvar_daemon(query,
2254                              (Name) query_list,
2255                              false,
2256                              chain_daemon,
2257                              false,
2258                              debug_level);

2260         /* build $^ */
2261         Chain hat_list = NULL;
2262         Chain *hat_list_tail = &hat_list;

2264         for (Dependency dependency = line->body.line.dependencies;
2265                 dependency != NULL;
2266                 dependency = dependency->next) {
2267                 /* skip automatic dependencies */
```



```
2268                 if (!dependency->automatic) {
2269                         if ((dependency->name != force) &&
2270                                 (dependency->stale == false)) {
2271                                 *hat_list_tail = ALLOC(Chain);
2272
2273                                 if (dependency->name->is_member &&
2274                                         (get_prop(dependency->name->prop, member
2275                                         (*hat_list_tail)->name =
2276                                                 get_prop(dependency->nam
2277                                                         member_prop)->bo
2278                                 } else {
2279                                         (*hat_list_tail)->name = dependency->nam
2280                                 }

2282                                 if((*hat_list_tail)->name != NULL) {
2283                                         if ((*hat_list_tail)->name->has_vpath_al
2284                                                 (*hat_list_tail)->name =
2285                                                         get_prop((*hat_list_tail
2286                                                                 vpath_alias_prop
2287                                         }
2288                                 }

2290                                 (*hat_list_tail)->next = NULL;
2291                                 hat_list_tail = &(*hat_list_tail)->next;
2292                         }
2293                 }
2294         }
2295         (void) setvar_daemon(hat,
2296                              (Name) hat_list,
2297                              false,
2298                              chain_daemon,
2299                              false,
2300                              debug_level);

2302 /* We have two command sequences we need to handle */
2303 /* The old one that we probably read from .make.state */
2304 /* and the new one we are building that will replace the old one */
2305 /* Even when KEEP_STATE is not on we build a new command sequence and store */
2306 /* it in the line prop. This command sequence is then executed by */
2307 /* run_command(). If KEEP_STATE is on it is also later written to */
2308 /* .make.state. The routine replaces the old command line by line with the */
2309 /* new one trying to reuse Cmd_lines */

2311         /* If there is no old command_used we have to start creating */
2312         /* Cmd_lines to keep the new cmd in */
2313         if (used == NULL) {
2314                 new_command_longer = true;
2315                 *insert = used = ALLOC(Cmd_line);
2316                 used->next = NULL;
2317                 used->command_line = NULL;
2318                 insert = &used->next;
2319         }
2320         /* Run thru the template for the new command and build the expanded */
2321         /* new command lines */
2322         for (;
2323              command_template != NULL;
2324              command_template = command_template->next, insert = &used->next, us
2325                 /* If there is no old command_used Cmd_line we need to */
2326                 /* create one and say that cmd consistency failed */
2327                 if (used == NULL) {
2328                         new_command_longer = true;
2329                         *insert = used = ALLOC(Cmd_line);
2330                         used->next = NULL;
2331                         used->command_line = empty_name;
2332                 }
2333                 /* Prepare the Cmd_line for the processing */
```

```
2334                /* The command line prefixes "@-=?" are stripped and that */
2335                /* information is saved in the Cmd_line */
2336                used->assign = false;
2337                used->ignore_error = ignore_errors;
2338                used->silent = silent;
2339                used->always_exec = false;
2340                /* Expand the macros in the command line */
2341                INIT_STRING_FROM_STACK(command_line, buffer);
2342                make_word_mentioned =
2343                  query_mentioned =
2344                    false;
2345                expand_value(command_template->command_line, &command_line, true
2346                /* If the macro $(MAKE) is mentioned in the command */
2347                /* "make -n" runs actually execute the command */
2348                used->make_refd = make_word_mentioned;
2349                used->ignore_command_dependency = query_mentioned;
2350                /* Strip the prefixes */
2351                start = command_line.buffer.start;
2352                for (;
2353                     iswspace(*start) ||
2354                     (get_char_semantics_value(*start) & (int) command_prefix_se
2355                     start++) {
2356                        switch (*start) {
2357                        case question_char:
2358                                used->ignore_command_dependency = true;
2359                                break;
2360                        case exclam_char:
2361                                used->ignore_command_dependency = false;
2362                                break;
2363                        case equal_char:
2364                                used->assign = true;
2365                                break;
2366                        case hyphen_char:
2367                                used->ignore_error = true;
2368                                break;
2369                        case at_char:
2370                                if (!do_not_exec_rule) {
2371                                        used->silent = true;
2372                                }
2373                                break;
2374                        case plus_char:
2375                                if(posix) {
2376                                  used->always_exec  = true;
2377                                }
2378                                break;
2379                        }
2380                }
2381                /* If all command lines of the template are prefixed with "?"*/
2382                /* the VIRTUAL_ROOT is not used for cmd consistency checks */
2383                if (!used->ignore_command_dependency) {
2384                        ignore_all_command_dependency = false;
2385                }
2386                /* Internalize the expanded and stripped command line */
2387                new_command_line = GETNAME(start, FIND_LENGTH);
2388                if ((used->command_line == NULL) &&
2389                    (line->body.line.sccs_command)) {
2390                        used->command_line = new_command_line;
2391                        new_command_longer = false;
2392                }
2393                /* Compare it with the old one for command consistency */
2394                if (used->command_line != new_command_line) {
2395                        Name vpath_translated = vpath_translation(new_command_li
2396                        if (keep_state &&
2397                            !used->ignore_command_dependency && (vpath_translate
2398                                if (debug_level > 0) {
2399                                        if (used->command_line != NULL
```



```
2400                                            && *used->command_line->string_mb !=
2401                                            '\0') {
2402                                                (void) printf(gettext("%*sBuildi
2407                                                (void) printf(catgets(catd, 1, 3
2403                                                                      recursion_level,
2404                                                                      "",
2405                                                                      target->string_mb,
2406                                                                      vpath_translated->
2407                                                                      recursion_level,
2408                                                                      "",
2409                                                                      used->
2410                                                                      command_line->
2411                                                                      string_mb);
2412                                        } else {
2413                                                (void) printf(gettext("%*sBuildi
2418                                                (void) printf(catgets(catd, 1, 3
2414                                                                      recursion_level,
2415                                                                      "",
2416                                                                      target->string_mb,
2417                                                                      vpath_translated->
2418                                                                      recursion_level,
2419                                                                      "");
2420                                        }
2421                                }
2422                                command_changed = true;
2423                                line->body.line.is_out_of_date = true;
2424                        }
2425                        used->command_line = new_command_line;
2426                }
2427                if (command_line.free_after_use) {
2428                        retmem(command_line.buffer.start);
2429                }
2430        }
2431        /* Check if the old command is longer than the new for */
2432        /* command consistency */
2433        if (used != NULL) {
2434                *insert = NULL;
2435                if (keep_state &&
2436                    !ignore_all_command_dependency) {
2437                        if (debug_level > 0) {
2438                                (void) printf(gettext("%*sBuilding %s because ne
2443                                (void) printf(catgets(catd, 1, 34, "%*sBuilding
2439                                              recursion_level,
2440                                              "",
2441                                              target->string_mb);
2442                        }
2443                        command_changed = true;
2444                        line->body.line.is_out_of_date = true;
2445                }
2446        }
2447        /* Check if the new command is longer than the old command for */
2448        /* command consistency */
2449        if (new_command_longer &&
2450            !ignore_all_command_dependency &&
2451            keep_state) {
2452                if (debug_level > 0) {
2453                        (void) printf(gettext("%*sBuilding %s because new comman
2458                        (void) printf(catgets(catd, 1, 35, "%*sBuilding %s becau
2454                                      recursion_level,
2455                                      "",
2456                                      target->string_mb);
2457                }
2458                command_changed = true;
2459                line->body.line.is_out_of_date = true;
2460        }
2461        /* Unbind the magic macros */
```

```
2462         (void) SETVAR(c_at, (Name) NULL, false);
2463         (void) SETVAR(star, (Name) NULL, false);
2464         (void) SETVAR(less_name, (Name) NULL, false);
2465         (void) SETVAR(percent_name, (Name) NULL, false);
2466         (void) SETVAR(query, (Name) NULL, false);
2467         if (query_list != NULL) {
2468                 delete_query_chain(query_list);
2469         }
2470         (void) SETVAR(hat, (Name) NULL, false);
2471         if (hat_list != NULL) {
2472                 delete_query_chain(hat_list);
2473         }

2475         if (conditional_macro_used) {
2476                 target->conditional_macro_list = cond_macro_list;
2477                 cond_macro_list = NULL;
2478                 target->depends_on_conditional = true;
2479         }
2480 }

2482 /*
2483  *      touch_command(line, target, result)
2484  *
2485  *      If this is an "make -t" run we do this.
2486  *      We touch all targets in the target group ("foo + fie:") if any.
2487  *
2488  *      Return value:
2489  *                              Indicates if the command failed or not
2490  *
2491  *      Parameters:
2492  *              line            The command line to update
2493  *              target          The target we are touching
2494  *              result          Initial value for the result we return
2495  *
2496  *      Global variables used:
2497  *              do_not_exec_rule Indicates that -n is on
2498  *              silent          Do not echo commands
2499  */
2500 static Doname
2501 touch_command(register Property line, register Name target, Doname result)
2502 {
2503         Name                    name;
2504         register Chain          target_group;
2505         String_rec              touch_string;
2506         wchar_t                 buffer[MAXPATHLEN];
2507         Name                    touch_cmd;
2508         Cmd_line                rule;

2510         for (name = target, target_group = NULL; name != NULL;) {
2511                 if (!name->is_member) {
2512                         /*
2513                          * Build a touch command that can be passed
2514                          * to dosys(). If KEEP_STATE is on, "make -t"
2515                          * will save the proper command, not the
2516                          * "touch" in .make.state.
2517                          */
2518                         INIT_STRING_FROM_STACK(touch_string, buffer);
2519                         MBSTOWCS(wcs_buffer, "touch ");
2524                         MBSTOWCS(wcs_buffer, NOCATGETS("touch "));
2520                         append_string(wcs_buffer, &touch_string, FIND_LENGTH);
2521                         touch_cmd = name;
2522                         if (name->has_vpath_alias_prop) {
2523                                 touch_cmd = get_prop(name->prop,
2524                                                      vpath_alias_prop)->
2525                                                        body.vpath_alias.alias;
2526                         }
```



```
2527                         APPEND_NAME(touch_cmd,
2528                                     &touch_string,
2529                                     FIND_LENGTH);
2530                         touch_cmd = GETNAME(touch_string.buffer.start,
2531                                             FIND_LENGTH);
2532                         if (touch_string.free_after_use) {
2533                                 retmem(touch_string.buffer.start);
2534                         }
2535                         if (!silent ||
2536                             do_not_exec_rule &&
2537                             (target_group == NULL)) {
2538                                 (void) printf("%s\n", touch_cmd->string_mb);
2539                         }
2540                         /* Run the touch command, or simulate it */
2541                         if (!do_not_exec_rule) {
2542                                 result = dosys(touch_cmd,
2543                                                false,
2544                                                false,
2545                                                false,
2546                                                false,
2547                                                name);
2548                         } else {
2549                                 result = build_ok;
2550                         }
2551                 } else {
2552                         result = build_ok;
2553                 }
2554                 if (target_group == NULL) {
2555                         target_group = line->body.line.target_group;
2556                 } else {
2557                         target_group = target_group->next;
2558                 }
2559                 if (target_group != NULL) {
2560                         name = target_group->name;
2561                 } else {
2562                         name = NULL;
2563                 }
2564         }
2565         return result;
2566 }
```

*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
2689                        fatal(catgets(catd, 1, 95, "Can't read symbolic link '%s
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"),
2699                        fatal(catgets(catd, 1, 36, "Can't read symbolic link '%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 *) wsrchr(wcb.get_string(), slash_char)) != N
2704                        append_string(wcb.get_string(), &string, p - wcb.get_str
2705                }
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 becaus
2771                                (void) printf(catgets(catd, 1, 37, "%*sSccs gett
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.
```

```
 *
 *      Parameters:
 *              file            The file we need to read dir for
 *
 *      Global variables used:
 *              dot             The Name ".", used as the default dir
 */
void
read_directory_of_file(register Name file)
{

        Wstring file_string(file);
        wchar_t * wcb = file_string.get_string();
        wchar_t usr_include_buf[MAXPATHLEN];
        wchar_t usr_include_sys_buf[MAXPATHLEN];

        register Name           directory = dot;
        register wchar_t        *p = (wchar_t *) wsrchr(wcb,
                                                        (int) slash_char);
        register int            length = p - wcb;
        static Name             usr_include;
        static Name             usr_include_sys;

        if (usr_include == NULL) {
                MBSTOWCS(usr_include_buf, "/usr/include");
                MBSTOWCS(usr_include_buf, NOCATGETS("/usr/include"));
                usr_include = GETNAME(usr_include_buf, FIND_LENGTH);
                MBSTOWCS(usr_include_sys_buf, "/usr/include/sys");
                MBSTOWCS(usr_include_sys_buf, NOCATGETS("/usr/include/sys"));
                usr_include_sys = GETNAME(usr_include_sys_buf, FIND_LENGTH);
        }

        /*
         * If the filename contains a "/" we have to extract the path
         * Else the path defaults to ".".
         */
        if (p != NULL) {
                /*
                 * Check some popular directories first to possibly
                 * save time. Compare string length first to gain speed.
                 */
                if ((usr_include->hash.length == length) &&
                    IS_WEQUALN(usr_include_buf,
                               wcb,
                               length)) {
                        directory = usr_include;
                } else if ((usr_include_sys->hash.length == length) &&
                           IS_WEQUALN(usr_include_sys_buf,
                                      wcb,
                                      length)) {
                        directory = usr_include_sys;
                } else {
                        directory = GETNAME(wcb, length);
                }
        }
        (void) read_dir(directory,
                        (wchar_t *) NULL,
                        (Property) NULL,
                        (wchar_t *) NULL);
}
```

*_____unchanged_portion_omitted_*

```
/*
 *      set_locals(target, old_locals)
 *
 *      Sets any conditional macros for the target.
```



```
 *      Each target carries a possibly empty set of conditional properties.
 *
 *      Parameters:
 *              target          The target to set conditional macros for
 *              old_locals      Space to store old values in
 *
 *      Global variables used:
 *              debug_level     Should we trace activity?
 *              is_conditional  We need to preserve this value
 *              recursion_level Used for tracing
 */
void
set_locals(register Name target, register Property old_locals)
{
        register Property       conditional;
        register int            i;
        register Boolean        saved_conditional_macro_used;
        Chain                   cond_name;
        Chain                   cond_chain;

        if (target->dont_activate_cond_values) {
                return;
        }

        saved_conditional_macro_used = conditional_macro_used;

        /* Scan the list of conditional properties and apply each one */
        for (conditional = get_prop(target->prop, conditional_prop), i = 0;
             conditional != NULL;
             conditional = get_prop(conditional->next, conditional_prop),
             i++) {
                /* Save the old value */
                old_locals[i].body.macro =
                  maybe_append_prop(conditional->body.conditional.name,
                                    macro_prop)->body.macro;
                if (debug_level > 1) {
                        (void) printf(gettext("%*sActivating conditional value:
                        (void) printf(catgets(catd, 1, 38, "%*sActivating condit
                                      recursion_level,
                                      "");
                }
                /* Set the conditional value. Macros are expanded when the */
                /* macro is refd as usual */
                if ((conditional->body.conditional.name != virtual_root) ||
                    (conditional->body.conditional.value != virtual_root)) {
                        (void) SETVAR(conditional->body.conditional.name,
                                      conditional->body.conditional.value,
                                      (Boolean) conditional->body.conditional.ap
                }
                cond_name = ALLOC(Chain);
                cond_name->name = conditional->body.conditional.name;
        }
        /* Put this target on the front of the chain of conditional targets */
        cond_chain = ALLOC(Chain);
        cond_chain->name = target;
        cond_chain->next = conditional_targets;
        conditional_targets = cond_chain;
        conditional_macro_used = saved_conditional_macro_used;
}

/*
 *      reset_locals(target, old_locals, conditional, index)
 *
 *      Removes any conditional macros for the target.
 *
 *      Parameters:
```

```
2992  *              target          The target we are retoring values for
2993  *              old_locals      The values to restore
2994  *              conditional     The first conditional block for the target
2995  *              index           into the old_locals vector
2996  *      Global variables used:
2997  *              debug_level     Should we trace activities?
2998  *              recursion_level Used for tracing
2999  */
3000 void
3001 reset_locals(register Name target, register Property old_locals, register Proper
3002 {
3003         register Property       this_conditional;
3004         Chain                   cond_chain;

3006         if (target->dont_activate_cond_values) {
3007                 return;
3008         }

3010         /* Scan the list of conditional properties and restore the old value */
3011         /* to each one Reverse the order relative to when we assigned macros */
3012         this_conditional = get_prop(conditional->next, conditional_prop);
3013         if (this_conditional != NULL) {
3014                 reset_locals(target, old_locals, this_conditional, index+1);
3015         } else {
3016                 /* Remove conditional target from chain */
3017                 if (conditional_targets == NULL ||
3018                     conditional_targets->name != target) {
3019                         warning(gettext("Internal error: reset target not at hea
3024                         warning(catgets(catd, 1, 39, "Internal error: reset targ
3020                 } else {
3021                         cond_chain = conditional_targets->next;
3022                         retmem_mb((caddr_t) conditional_targets);
3023                         conditional_targets = cond_chain;
3024                 }
3025         }
3026         get_prop(conditional->body.conditional.name->prop,
3027                  macro_prop)->body.macro = old_locals[index].body.macro;
3028         if (conditional->body.conditional.name == virtual_root) {
3029                 (void) SETVAR(virtual_root, getvar(virtual_root), false);
3030         }
3031         if (debug_level > 1) {
3032                 if (old_locals[index].body.macro.value != NULL) {
3033                         (void) printf(gettext("%*sdeactivating conditional value
3038                         (void) printf(catgets(catd, 1, 40, "%*sdeactivating cond
3034                                       recursion_level,
3035                                       "",
3036                                       conditional->body.conditional.name->
3037                                       string_mb,
3038                                       old_locals[index].body.macro.value->
3039                                       string_mb);
3040                 } else {
3041                         (void) printf(gettext("%*sdeactivating conditional value
3046                         (void) printf(catgets(catd, 1, 41, "%*sdeactivating cond
3042                                       recursion_level,
3043                                       "",
3044                                       conditional->body.conditional.name->
3045                                       string_mb);
3046                 }
3047         }
3048 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   18355 Wed May 20 12:19:48 2015
new/usr/src/cmd/make/bin/files.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2003 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      files.c
 *
 *      Various file related routines:
 *              Figure out if file exists
 *              Wildcard resolution for directory reader
 *              Directory reader
 */



/*
 * Included files
 */
#include <dirent.h>             /* opendir() */
#include <errno.h>              /* errno */
#include <mk/defs.h>
#include <mksh/macro.h>         /* getvar() */
#include <mksh/misc.h>          /* get_prop(), append_prop() */
#include <sys/stat.h>           /* lstat() */
#include <libintl.h>
#endif /* ! codereview */

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */

/*
 * File table of contents
```



```
 */
extern  timestruc_t&    exists(register Name target);
extern  void            set_target_stat(register Name target, struct stat buf);
static  timestruc_t&    vpath_exists(register Name target);
static  Name            enter_file_name(wchar_t *name_string, wchar_t *library);
static  Boolean         star_match(register char *string, register char *pattern
static  Boolean         amatch(register wchar_t *string, register wchar_t *patte

/*
 *      exists(target)
 *
 *      Figure out the timestamp for one target.
 *
 *      Return value:
 *                              The time the target was created
 *
 *      Parameters:
 *              target          The target to check
 *
 *      Global variables used:
 *              debug_level     Should we trace the stat call?
 *              recursion_level Used for tracing
 *              vpath_defined   Was the variable VPATH defined in environment?
 */
timestruc_t&
exists(register Name target)
{
        struct stat             buf;
        register int            result;

        /* We cache stat information. */
        if (target->stat.time != file_no_time) {
                return target->stat.time;
        }

        /*
         * If the target is a member, we have to extract the time
         * from the archive.
         */
        if (target->is_member &&
            (get_prop(target->prop, member_prop) != NULL)) {
                return read_archive(target);
        }

        if (debug_level > 1) {
                (void) printf("%*sstat(%s)\n",
                (void) printf(NOCATGETS("%*sstat(%s)\n"),
                                recursion_level,
                                "",
                                target->string_mb);
        }

        result = lstat_vroot(target->string_mb, &buf, NULL, VROOT_DEFAULT);
        if ((result != -1) && ((buf.st_mode & S_IFMT) == S_IFLNK)) {
                /*
                 * If the file is a symbolic link, we remember that
                 * and then we get the status for the refd file.
                 */
                target->stat.is_sym_link = true;
                result = stat_vroot(target->string_mb, &buf, NULL, VROOT_DEFAULT
        } else {
                target->stat.is_sym_link = false;
        }

        if (result < 0) {
                target->stat.time = file_doesnt_exist;
```

```
127                 target->stat.stat_errno = errno;
128                 if ((errno == ENOENT) &&
129                     vpath_defined &&
130 /* azv, fixing bug 1262942, VPATH works with a leaf name
131  * but not a directory name.
132  */
133                         (target->string_mb[0] != (int) slash_char) ) {
134 /* BID_1214655 */
135 /* azv */
136                         vpath_exists(target);
137                         // return vpath_exists(target);
138                 }
139         } else {
140                 /* Save all the information we need about the file */
141                 target->stat.stat_errno = 0;
142                 target->stat.is_file = true;
143                 target->stat.mode = buf.st_mode & 0777;
144                 target->stat.size = buf.st_size;
145                 target->stat.is_dir =
146                   BOOLEAN((buf.st_mode & S_IFMT) == S_IFDIR);
147                 if (target->stat.is_dir) {
148                         target->stat.time = file_is_dir;
149                 } else {
150                         /* target->stat.time = buf.st_mtime; */
151 /* BID_1129806 */
152 /* vis@nbsp.nsk.su */
153                         target->stat.time = MAX(buf.st_mtim, file_min_time);
154                 }
155         }
156         if ((target->colon_splits > 0) &&
157             (get_prop(target->prop, time_prop) == NULL)) {
158                 append_prop(target, time_prop)->body.time.time =
159                   target->stat.time;
160         }
161         return target->stat.time;
162 }
```

*unchanged_portion_omitted_*

```
261 /*
262  *      read_dir(dir, pattern, line, library)
263  *
264  *      Used to enter the contents of directories into makes namespace.
265  *      Presence of a file is important when scanning for implicit rules.
266  *      read_dir() is also used to expand wildcards in dependency lists.
267  *
268  *      Return value:
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  *                              we find as dependencies for this line
276  *              library         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;

303         if(dir->hash.length >= MAXPATHLEN) {
304                 return 0;
305         }

307         Wstring wcb(dir);
308         Wstring vps;

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. */
318         if (vpath_defined && (dir == dot)) {
319                 vps.init(getvar(vpath_name));
320                 vpath = vps.get_string();
321         }

323         /*
324          * Prepare the string where we build the full name of the
325          * files in the directory.
326          */
327         if ((dir->hash.length > 1) || (wcb.get_string()[0] != (int) period_char)
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 + wslen(file_name);
332         }

334         /* Open the directory. */
335 vpath_loop:
336         dir_fd = opendir(dir->string_mb);
337         if (dir_fd == NULL) {
338                 return 0;
339         }

341         /* Read all the directory entries. */
342         while ((dp = readdir(dir_fd)) != NULL) {
343                 /* We ignore "." and ".." */
344                 if ((dp->d_fileno == 0) ||
345                     ((dp->d_name[0] == (int) period_char) &&
346                      ((dp->d_name[1] == 0) ||
347                       ((dp->d_name[1] == (int) period_char) &&
348                        (dp->d_name[2] == 0))))) {
349                         continue;
350                 }
351                 /*
352                  * Build the full name of the file using whatever
353                  * path supplied to the function.
354                  */
```

```
355                 MBSTOWCS(tmp_wcs_buffer, dp->d_name);
356                 (void) wscpy(file_name_p, tmp_wcs_buffer);
357                 file = enter_file_name(file_name, library);
358                 if ((pattern != NULL) && amatch(tmp_wcs_buffer, pattern)) {
359                         /*
360                          * If we are expanding a wildcard pattern, we
361                          * enter the file as a dependency for the target.
362                          */
363                         if (debug_level > 0){
364                                 WCSTOMBS(mbs_buffer, pattern);
365                                 (void) printf(gettext("'%s: %s' due to %s expans
303                                 (void) printf(catgets(catd, 1, 231, "'%s: %s' du
366                                               line->body.line.target->string_mb,
367                                               file->string_mb,
368                                               mbs_buffer);
369                         }
370                         enter_dependency(line, file, false);
371                         result++;
372                 } else {
373                         /*
374                          * If the file has an SCCS/s. file,
375                          * we will detect that later on.
376                          */
377                         file->stat.has_sccs = NO_SCCS;
378                 /*
379                  * If this is an s. file, we also enter it as if it
380                  * existed in the plain directory.
381                  */
382                 if ((dp->d_name[0] == 's') &&
383                     (dp->d_name[1] == (int) period_char)) {
384
385                         MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
386                         plain_file_name_p = plain_file_name;
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 %
340                                         (void) printf(catgets(catd, 1, 232, "'%s
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  */

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) wsncpy(plain_file_name,
441                       file_name,
442                       file_name_p - file_name);
443         plain_file_name[file_name_p - file_name] = 0;
444         plain_file_name_p = plain_file_name + wslen(plain_file_name);

446         if(!svr4) {

448           if (sccs_dir_path != NULL) {
449                 wchar_t         tmp_wchar;
450                 wchar_t         path[MAXPATHLEN];
451                 char            mb_path[MAXPATHLEN];

453                 if (file_name_p - file_name > 0) {
454                         tmp_wchar = *file_name_p;
455                         *file_name_p = 0;
456                         WCSTOMBS(mbs_buffer, file_name);
457                         (void) sprintf(mb_path, "%s/%s/SCCS",
395                         (void) sprintf(mb_path, NOCATGETS("%s/%s/SCCS"),
458                                                 sccs_dir_path,
459                                                 mbs_buffer);
460                         *file_name_p = tmp_wchar;
461                 } else {
462                         (void) sprintf(mb_path, "%s/SCCS", sccs_dir_path);
400                         (void) sprintf(mb_path, NOCATGETS("%s/SCCS"), sccs_dir_p
463                 }
464                 MBSTOWCS(path, mb_path);
465                 (void) wscpy(file_name, path);
466           } else {
467                 MBSTOWCS(wcs_buffer, "SCCS");
405                 MBSTOWCS(wcs_buffer, NOCATGETS("SCCS"));
468                 (void) wscpy(file_name_p, wcs_buffer);
469           }
470         } else {
471                 MBSTOWCS(wcs_buffer, ".");
409                 MBSTOWCS(wcs_buffer, NOCATGETS("."));
472                 (void) wscpy(file_name_p, wcs_buffer);
473         }
474         /* Internalize the constructed SCCS dir name. */
475         (void) exists(dir = GETNAME(file_name, FIND_LENGTH));
476         /* Just give up if the directory file doesnt exist. */
477         if (!dir->stat.is_file) {
478                 return result;
479         }
480         /* Open the directory. */
```

```
481         dir_fd = opendir(dir->string_mb);
482         if (dir_fd == NULL) {
483                 return result;
484         }
485         MBSTOWCS(wcs_buffer, "/");
486         (void) wscat(file_name, wcs_buffer);
487         file_name_p = file_name + wslen(file_name);

489         while ((dp = readdir(dir_fd)) != NULL) {
490                 if ((dp->d_fileno == 0) ||
491                     ((dp->d_name[0] == (int) period_char) &&
492                      ((dp->d_name[1] == 0) ||
493                       ((dp->d_name[1] == (int) period_char) &&
494                        (dp->d_name[2] == 0))))) {
495                         continue;
496                 }
497                 /* Construct and internalize the true name of the SCCS file. */
498                 MBSTOWCS(wcs_buffer, dp->d_name);
499                 (void) wscpy(file_name_p, wcs_buffer);
500                 file = GETNAME(file_name, FIND_LENGTH);
501                 file->stat.is_file = true;
502                 file->stat.has_sccs = NO_SCCS;
503                 /*
504                  * If this is an s. file, we also enter it as if it
505                  * existed in the plain directory.
506                  */
507                 if ((dp->d_name[0] == 's') &&
508                     (dp->d_name[1] == (int) period_char)) {
509
510                         MBSTOWCS(wcs_buffer, dp->d_name + 2);
511                         (void) wscpy(plain_file_name_p, wcs_buffer);
512                         plain_file = GETNAME(plain_file_name, FIND_LENGTH);
513                         plain_file->stat.is_file = true;
514                         plain_file->stat.has_sccs = HAS_SCCS;
515                                 /* if sccs dependency is already set,skip */
516                         if(plain_file->prop) {
517                                 Property sprop = get_prop(plain_file->prop,sccs_
518                                 if(sprop != NULL) {
519                                         if (sprop->body.sccs.file) {
520                                                 goto try_pattern;
521                                         }
522                                 }
523                         }

525                         /*
526                          * Enter the s. file as a dependency for the
527                          * plain file.
528                          */
529                         maybe_append_prop(plain_file, sccs_prop)->
530                           body.sccs.file = file;
531 try_pattern:
532                         MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
533                         if ((pattern != NULL) &&
534                             amatch(tmp_wcs_buffer, pattern)) {
535                                 if (debug_level > 0) {
536                                         WCSTOMBS(mbs_buffer, pattern);
537                                         (void) printf(gettext("'%s: %s' due to %
475                                         (void) printf(catgets(catd, 1, 233, "'%s
538                                                       line->body.line.target->
539                                                       string_mb,
540                                                       plain_file->string_mb,
541                                                       mbs_buffer);
542                                 }
543                                 enter_dependency(line, plain_file, false);
544                                 result++;
545                         }
```



```
546                 }
547         }
548         (void) closedir(dir_fd);

550         return result;
551 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    4478 Wed May 20 12:19:48 2015
new/usr/src/cmd/make/bin/globals.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

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

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

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

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

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



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

        Boolean         no_parallel = false;
        Name            no_parallel_name;
        Name            not_auto;
        Boolean         only_parallel;
        Boolean         parallel;
        Name            parallel_name;
        Name            localhost_name;
        int             parallel_process_cnt;
        Percent         percent_list;
        Dyntarget       dyntarget_list;
        Name            plus;
```

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

178 /*
179  * File table of contents
180  */
```

```
**********************************************************
   43021 Wed May 20 12:19:49 2015
new/usr/src/cmd/make/bin/implicit.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

/*
 *      implicit.c
 *
 *      Handle suffix and percent rules
 */

/*
 * Included files
 */
#include <mk/defs.h>
#include <mksh/macro.h>         /* expand_value() */
#include <mksh/misc.h>          /* retmem() */
#include <libintl.h>
#endif /* ! codereview */

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static  wchar_t         WIDE_NULL[1] = {(wchar_t) nul_char};

/*
 * File table of contents
 */
extern  Doname          find_suffix_rule(Name target, Name target_body, Name tar
extern  Doname          find_ar_suffix_rule(register Name target, Name true_targ
extern  Doname          find_double_suffix_rule(register Name target, Property *
extern  void            build_suffix_list(register Name target_suffix);
extern  Doname          find_percent_rule(register Name target, Property *comman
```



```
static  void            create_target_group_and_dependencies_list(Name target, P
static  Boolean         match_found_with_pattern(Name target, Percent pat_rule,
static  void            construct_string_from_pattern(Percent pat_rule, String p
static  Boolean         dependency_exists(Name target, Property line);
extern  Property        maybe_append_prop(Name, Property_id);
extern  void            add_target_to_chain(Name target, Chain * query);

/*
 *      find_suffix_rule(target, target_body, target_suffix, command, rechecking
 *
 *      Does the lookup for single and double suffix rules.
 *      It calls build_suffix_list() to build the list of possible suffixes
 *      for the given target.
 *      It then scans the list to find the first possible source file that
 *      exists. This is done by concatenating the body of the target name
 *      (target name less target suffix) and the source suffix and checking
 *      if the resulting file exists.
 *
 *      Return value:
 *                              Indicates if search failed or not
 *
 *      Parameters:
 *              target          The target we need a rule for
 *              target_body     The target name without the suffix
 *              target_suffix   The suffix of the target
 *              command         Pointer to slot to deposit cmd in if found
 *              rechecking      true if we are rechecking target which depends
 *                              on conditional macro and keep_state is set
 *
 *      Global variables used:
 *              debug_level     Indicates how much tracing to do
 *              recursion_level Used for tracing
 */

extern int printf (const char *, ...);

static Boolean actual_doname = false;

/* /tolik/
 * fix bug 1247448: Suffix Rules failed when combine with Pattern Matching Rules
 * When make attemps to apply % rule it didn't look for a single suffix rule bec
 * if "doname" is called from "find_percent_rule" argument "implicit" is set to
 * and find_suffix_rule was not called. I've commented the checking of "implicit
 * in "doname" and make got infinite recursion for SVR4 tilde rules.
 * Usage of "we_are_in_tilde" is intended to avoid this recursion.
 */

static Boolean we_are_in_tilde = false;

Doname
find_suffix_rule(Name target, Name target_body, Name target_suffix, Property *co
{
        static wchar_t          static_string_buf_3M [ 3 * MAXPATHLEN ];
        Name                    true_target = target;
        wchar_t                 *sourcename = (wchar_t*)static_string_buf_3M;
        register wchar_t        *put_suffix;
        register Property       source_suffix;
        register Name           source;
        Doname                  result;
        register Property       line;
        extern Boolean          tilde_rule;
        Boolean                 name_found = true;
        Boolean                 posix_tilde_attempt = true;
        int                     src_len = MAXPATHLEN + strlen(target_body->strin

        /*
```

```
128          * To avoid infinite recursion
129          */
130         if(we_are_in_tilde) {
131                 we_are_in_tilde = false;
132                 return(build_dont_know);
133         }

135         /*
136          * If the target is a constructed one for a "::" target,
137          * we need to consider that.
138          */
139         if (target->has_target_prop) {
140                 true_target = get_prop(target->prop,
141                                        target_prop)->body.target.target;
142         }
143         if (debug_level > 1) {
144                 (void) printf("%*sfind_suffix_rule(%s,%s,%s)\n",
 38                 (void) printf(NOCATGETS("%*sfind_suffix_rule(%s,%s,%s)\n"),
145                                 recursion_level,
146                                 "",
147                                 true_target->string_mb,
148                                 target_body->string_mb,
149                                 target_suffix->string_mb);
150         }
151         if (command != NULL) {
152                 if ((true_target->suffix_scan_done == true) && (*command == NULL
153                         return build_ok;
154                 }
155         }
156         true_target->suffix_scan_done = true;
157         /*
158          * Enter all names from the directory where the target lives as
159          * files that makes sense.
160          * This will make finding the synthesized source possible.
161          */
162         read_directory_of_file(target_body);
163         /* Cache the suffixes for this target suffix if not done. */
164         if (!target_suffix->has_read_suffixes) {
165                 build_suffix_list(target_suffix);
166         }
167         /* Preload the sourcename vector with the head of the target name. */
168         if (src_len >= sizeof(static_string_buf_3M)) {
169                 sourcename = ALLOC_WC(src_len);
170         }
171         (void) mbstowcs(sourcename,
172                       target_body->string_mb,
173                       (int) target_body->hash.length);
174         put_suffix = sourcename + target_body->hash.length;
175         /* Scan the suffix list for the target if one exists. */
176         if (target_suffix->has_suffixes) {
177 posix_attempts:
178                 for (source_suffix = get_prop(target_suffix->prop,
179                                               suffix_prop);
180                      source_suffix != NULL;
181                      source_suffix = get_prop(source_suffix->next,
182                                               suffix_prop)) {
183                         /* Build the synthesized source name. */
184                         (void) mbstowcs(put_suffix,
185                                         source_suffix->body.
186                                         suffix.suffix->string_mb,
187                                         (int) source_suffix->body.
188                                         suffix.suffix->hash.length);
189                         put_suffix[source_suffix->body.
190                                    suffix.suffix->hash.length] =
191                           (int) nul_char;
192                         if (debug_level > 1) {
```



```
193                                 WCSTOMBS(mbs_buffer, sourcename);
194                                 (void) printf(gettext("%*sTrying %s\n"),
 88                                 (void) printf(catgets(catd, 1, 218, "%*sTrying %
195                                         recursion_level,
196                                         "",
197                                         mbs_buffer);
198                         }
199                         source = getname_fn(sourcename, FIND_LENGTH, false, &nam
200                         /*
201                          * If the source file is not registered as
202                          * a file, this source suffix did not match.
203                          */
204                         if(vpath_defined && !posix && !svr4) {
205                                 (void) exists(source);
206                         }
207                         if (!source->stat.is_file) {
208                            if(!(posix|svr4))
209                            {
210                                 if(!name_found) {
211                                         free_name(source);
212                                 }
213                                 continue;
214                            }

216                            /* following code will ensure that the corresponding
217                            ** tilde rules are executed when corresponding s. fil
218                            ** exists in the current directory. Though the curren
219                            ** target ends with a ~ character, there wont be any
220                            ** any file in the current directory with that suffix
221                            ** as it's fictitious. Even if it exists, it'll
222                            ** execute all the rules for the ~ target.
223                            */

225                            if(source->string_mb[source->hash.length - 1] == '~'
226                               ( svr4 || posix_tilde_attempt ) )
227                            {
228                                 char *p, *np;
229                                 char *tmpbuf;

231                                 tmpbuf = getmem(source->hash.length + 8);
232                                 /* + 8 to add "s." or "SCCS/s." */
233                                 memset(tmpbuf,0,source->hash.length + 8);
234                                 source->string_mb[source->hash.length - 1] = '\0
235                                 if(p = (char *) memchr((char *)source->string_mb
236                                 {
237                                   while(1) {
238                                     if(np = (char *) memchr((char *)p+1,'/',sour
239                                       p = np;
240                                     } else {break;}
241                                   }
242                                   /* copy everything including '/' */
243                                   strncpy(tmpbuf, source->string_mb, p - source-
244                                   strcat(tmpbuf, "s.");
138                                   strcat(tmpbuf, NOCATGETS("s."));
245                                   strcat(tmpbuf, p+1);
246                                   retmem((wchar_t *) source->string_mb);
247                                   source->string_mb = tmpbuf;
248
249                                 } else {
250                                   strcpy(tmpbuf, "s.");
144                                   strcpy(tmpbuf, NOCATGETS("s."));
251                                   strcat(tmpbuf, source->string_mb);
252                                   retmem((wchar_t *) source->string_mb);
253                                   source->string_mb = tmpbuf;
254
255                                 }
```

```
256                                 source->hash.length = strlen(source->string_mb);
257                                 if(exists(source) == file_doesnt_exist)
258                                   continue;
259                                 tilde_rule = true;
260                                 we_are_in_tilde = true;
261                             } else {
262                                 if(!name_found) {
263                                         free_name(source);
264                                 }
265                                 continue;
266                             }
267                         } else {
268                             if(posix && posix_tilde_attempt) {
269                                 if(exists(source) == file_doesnt_exist) {
270                                         if(!name_found) {
271                                                 free_name(source);
272                                         }
273                                         continue;
274                                 }
275                             }
276                         }
277
278                         if (command != NULL) {
279                                 if(!name_found) {
280                                         store_name(source);
281                                 }
282                                 /*
283                                  * The source file is a file.
284                                  * Make sure it is up to date.
285                                  */
286                                 if (dependency_exists(source,
287                                                       get_prop(target->prop,
288                                                                line_prop))) {
289                                         result = (Doname) source->state;
290                                 } else {
291 #if 0  /* with_squiggle sends false, which is buggy. : djay */
292                                         result = doname(source,
293                                                         (Boolean) source_suffix-
294                                                         suffix.suffix->with_squi
295                                                         true);
296 #else
297                                         result = doname(source,
298                                                         true,
299                                                         true);
300 #endif
301                                 }
302                         } else {
303                                 result = target_can_be_built(source);
304
305                                 if (result == build_ok) {
306                                         return result;
307                                 } else {
308                                         if(!name_found) {
309                                                 free_name(source);
310                                         }
311                                         continue;
312                                 }
313                         }
314
315                         switch (result) {
316                         case build_dont_know:
317                                 /*
318                                  * If we still can't build the source,
319                                  * this rule is not a match,
320                                  * try the next one.
321                                  */
```



```
322                                 if (source->stat.time == file_doesnt_exist) {
323                                         if(!name_found) {
324                                                 free_name(source);
325                                         }
326                                         continue;
327                                 }
328                         case build_running:
329                                 if(!name_found) {
330                                         store_name(source);
331                                 }
332                                 true_target->suffix_scan_done = false;
333                                 line = maybe_append_prop(target, line_prop);
334                                 enter_dependency(line, source, false);
335                                 line->body.line.target = true_target;
336                                 return build_running;
337                         case build_ok:
338                                 if(!name_found) {
339                                         store_name(source);
340                                 }
341                                 break;
342                         case build_failed:
343                                 if(!name_found) {
344                                         store_name(source);
345                                 }
346                                 if (sourcename != static_string_buf_3M) {
347                                         retmem(sourcename);
348                                 }
349                                 return build_failed;
350                         }
351
352                         if (debug_level > 1) {
353                                 WCSTOMBS(mbs_buffer, sourcename);
354                                 (void) printf(gettext("%*sFound %s\n"),
248                                 (void) printf(catgets(catd, 1, 219, "%*sFound %s
355                                               recursion_level,
356                                               "",
357                                               mbs_buffer);
358                         }
359
360                         if (source->depends_on_conditional) {
361                                 target->depends_on_conditional = true;
362                         }
363 /*
364  * Since it is possible that the same target is built several times during
365  * the make run, we have to patch the target with all information we found
366  * here. Thus, the target will have an explicit rule the next time around.
367  */
368                         line = maybe_append_prop(target, line_prop);
369                         if (*command == NULL) {
370                                 *command = line;
371                         }
372                         if ((source->stat.time > (*command)->body.line.dependenc
373                             (debug_level > 1)) {
374                                 (void) printf(gettext("%*sDate(%s)=%s Date-depen
268                                 (void) printf(catgets(catd, 1, 220, "%*sDate(%s)
375                                               recursion_level,
376                                               "",
377                                               source->string_mb,
378                                               time_to_string(source->
379                                                              stat.time),
380                                               true_target->string_mb,
381                                               time_to_string((*command)->
382                                                              body.line.
383                                                              dependency_time));
384                         }
385                         /*
```

```
386                          * Determine if this new dependency made the
387                          * target out of date.
388                          */
389                         (*command)->body.line.dependency_time =
390                           MAX((*command)->body.line.dependency_time,
391                               source->stat.time);
392                         Boolean out_of_date;
393                         if (target->is_member) {
394                                out_of_date = (Boolean) OUT_OF_DATE_SEC(target->
395                                                                        (*comman
396                         } else {
397                                out_of_date = (Boolean) OUT_OF_DATE(target->stat
398                                                                    (*command)->
399                         }
400                         if (build_unconditional || out_of_date) {
401                                if(!rechecking) {
402                                        line->body.line.is_out_of_date = true;
403                                }
404                                if (debug_level > 0) {
405                                        (void) printf(gettext("%*sBuilding %s us
299                                        (void) printf(catgets(catd, 1, 221, "%*s
406                                                      recursion_level,
407                                                      "",
408                                                      true_target->string_mb,
409                                                      source_suffix->body.suffix
410                                                      target_suffix->string_mb,
411                                                      source->string_mb);
412                                }
413                         }
414                         /*
415                          * Add the implicit rule as the target's explicit
416                          * rule if none actually given, and register
417                          * dependency.
418                          * The time checking above really should be
419                          * conditional on actual use of implicit rule
420                          * as well.
421                          */
422                         line->body.line.sccs_command = false;
423                         if (line->body.line.command_template == NULL) {
424                                line->body.line.command_template =
425                                  source_suffix->body.suffix.command_template;
426                         }
427                         enter_dependency(line, source, false);
428                         line->body.line.target = true_target;
429                         /*
430                          * Also make sure the rule is built with
431                          * $* and $< bound properly.
432                          */
433                         line->body.line.star = target_body;
434                         if(svr4|posix) {
435                           char * p;
436                           char tstr[256];
437                           extern Boolean dollarless_flag;
438                           extern Name dollarless_value;

440                           if(tilde_rule) {
441                               MBSTOWCS(wcs_buffer, source->string_mb);
335                               MBSTOWCS(wcs_buffer, NOCATGETS(source->string_mb))
442                               dollarless_value = GETNAME(wcs_buffer,FIND_LENGTH)
443                           }
444                           else {
445                                 dollarless_flag = false;
446                           }
447                         }
448                         line->body.line.less = source;
449                         line->body.line.percent = NULL;
```



```
450                                 add_target_to_chain(source, &(line->body.line.query));
451                                 if (sourcename != static_string_buf_3M) {
452                                         retmem(sourcename);
453                                 }
454                                 return build_ok;
455                         }
456                         if(posix && posix_tilde_attempt) {
457                                 posix_tilde_attempt = false;
458                                 goto posix_attempts;
459                         }
460                         if ((command != NULL) &&
461                             ((*command) != NULL) &&
462                             ((*command)->body.line.star == NULL)) {
463                                 (*command)->body.line.star = target_body;
464                         }
465                 }
466                 if (sourcename != static_string_buf_3M) {
467                         retmem(sourcename);
468                 }
469                 /* Return here in case no rule matched the target */
470                 return build_dont_know;
471 }

473 /*
474  *      find_ar_suffix_rule(target, true_target, command, rechecking)
475  *
476  *      Scans the .SUFFIXES list and tries
477  *      to find a suffix on it that matches the tail of the target member name.
478  *      If it finds a matching suffix it calls find_suffix_rule() to find
479  *      a rule for the target using the suffix ".a".
480  *
481  *      Return value:
482  *                              Indicates if search failed or not
483  *
484  *      Parameters:
485  *              target          The target we need a rule for
486  *              true_target     The proper name
487  *              command         Pointer to slot where we stuff cmd, if found
488  *              rechecking      true if we are rechecking target which depends
489  *                              on conditional macro and keep_state is set
490  *
491  *      Global variables used:
492  *              debug_level     Indicates how much tracing to do
493  *              dot_a           The Name ".a", compared against
494  *              recursion_level Used for tracing
495  *              suffixes        List of suffixes used for scan (from .SUFFIXES)
496  */
497 Doname
498 find_ar_suffix_rule(register Name target, Name true_target, Property *command, B
499 {
500         wchar_t                 *target_end;
501         register Dependency     suffix;
502         register int            suffix_length;
503         Property                line;
504         Name                    body;
505         static Name             dot_a;

507         Wstring                 targ_string(true_target);
508         Wstring                 suf_string;

510         if (dot_a == NULL) {
511                 MBSTOWCS(wcs_buffer, ".a");
405                 MBSTOWCS(wcs_buffer, NOCATGETS(".a"));
512                 dot_a = GETNAME(wcs_buffer, FIND_LENGTH);
513         }
514         target_end = targ_string.get_string() + true_target->hash.length;
```

```
516         /*
517          * We compare the tail of the target name with the suffixes
518          * from .SUFFIXES.
519          */
520         if (debug_level > 1) {
521                 (void) printf("%*sfind_ar_suffix_rule(%s)\n",
415                 (void) printf(NOCATGETS("%*sfind_ar_suffix_rule(%s)\n"),
522                               recursion_level,
523                               "",
524                               true_target->string_mb);
525         }
526         /*
527          * Scan the .SUFFIXES list to see if the target matches any of
528          * those suffixes.
529          */
530         for (suffix = suffixes; suffix != NULL; suffix = suffix->next) {
531                 /* Compare one suffix. */
532                 suffix_length = suffix->name->hash.length;
533                 suf_string.init(suffix->name);
534                 if (!IS_WEQUALN(suf_string.get_string(),
535                                 target_end - suffix_length,
536                                 suffix_length)) {
537                         goto not_this_one;
538                 }
539                 /*
540                  * The target tail matched a suffix from the .SUFFIXES list.
541                  * Now check for a rule to match.
542                  */
543                 target->suffix_scan_done = false;
544                 body = GETNAME(targ_string.get_string(),
545                                (int)(true_target->hash.length -
546                                      suffix_length));
547                 we_are_in_tilde = false;
548                 switch (find_suffix_rule(target,
549                                          body,
550                                          dot_a,
551                                          command,
552                                          rechecking)) {
553                 case build_ok:
554                         line = get_prop(target->prop, line_prop);
555                         line->body.line.star = body;
556                         return build_ok;
557                 case build_running:
558                         return build_running;
559                 }
560                 /*
561                  * If no rule was found, we try the next suffix to see
562                  * if it matches the target tail, and so on.
563                  * Go here if the suffix did not match the target tail.
564                  */
565         not_this_one:;
566         }
567         return build_dont_know;
568 }

570 /*
571  *      find_double_suffix_rule(target, command, rechecking)
572  *
573  *      Scans the .SUFFIXES list and tries
574  *      to find a suffix on it that matches the tail of the target name.
575  *      If it finds a matching suffix it calls find_suffix_rule() to find
576  *      a rule for the target.
577  *
578  *      Return value:
579  *                              Indicates if scan failed or not
```



```
580  *
581  *      Parameters:
582  *              target          Target we need a rule for
583  *              command         Pointer to slot where we stuff cmd, if found
584  *              rechecking      true if we are rechecking target which depends
585  *                              on conditional macro and keep_state is set
586  *
587  *      Global variables used:
588  *              debug_level     Indicates how much tracing to do
589  *              recursion_level Used for tracing
590  *              suffixes        List of suffixes used for scan (from .SUFFIXES)
591  */
592 Doname
593 find_double_suffix_rule(register Name target, Property *command, Boolean recheck
594 {
595         Name                    true_target = target;
596         Name                    target_body;
597         register wchar_t        *target_end;
598         register Dependency     suffix;
599         register int            suffix_length;
600         Boolean                 scanned_once = false;
601         Boolean                 name_found = true;

603         Wstring                 targ_string;
604         Wstring                 suf_string;

606         /*
607          * If the target is a constructed one for a "::" target,
608          * we need to consider that.
609          */
610         if (target->has_target_prop) {
611                 true_target = get_prop(target->prop,
612                                        target_prop)->body.target.target;
613         }
614         targ_string.init(true_target);

616         /*
617          * We compare the tail of the target name with the
618          * suffixes from .SUFFIXES.
619          */
620         target_end = targ_string.get_string() + true_target->hash.length;
621         if (debug_level > 1) {
622                 (void) printf("%*sfind_double_suffix_rule(%s)\n",
516                 (void) printf(NOCATGETS("%*sfind_double_suffix_rule(%s)\n"),
623                               recursion_level,
624                               "",
625                               true_target->string_mb);
626         }
627         /*
628          * Scan the .SUFFIXES list to see if the target matches
629          * any of those suffixes.
630          */
631         for (suffix = suffixes; suffix != NULL; suffix = suffix->next) {
632                 target->suffix_scan_done = false;
633                 true_target->suffix_scan_done = false;
634                 /* Compare one suffix. */
635                 suffix_length = suffix->name->hash.length;
636                 suf_string.init(suffix->name);
637                 /* Check the lengths, or else RTC will report rua. */
638                 if (true_target->hash.length < suffix_length) {
639                         goto not_this_one;
640                 } else if (!IS_WEQUALN(suf_string.get_string(),
641                                        (target_end - suffix_length),
642                                        suffix_length)) {
643                         goto not_this_one;
644                 }
```

```
645                 /*
646                  * The target tail matched a suffix from the .SUFFIXES list.
647                  * Now check for a rule to match.
648                  */
649                 we_are_in_tilde = false;
650                 target_body = GETNAME(
651                         targ_string.get_string(),
652                         (int)(true_target->hash.length - suffix_length)
653                 );
654                 switch (find_suffix_rule(target,
655                                          target_body,
656                                          suffix->name,
657                                          command,
658                                          rechecking)) {
659                 case build_ok:
660                         return build_ok;
661                 case build_running:
662                         return build_running;
663                 }
664                 if (true_target->suffix_scan_done == true) {
665                         scanned_once = true;
666                 }
667                 /*
668                  * If no rule was found, we try the next suffix to see
669                  * if it matches the target tail. And so on.
670                  * Go here if the suffix did not match the target tail.
671                  */
672         not_this_one:;
673         }
674         if (scanned_once)
675                 true_target->suffix_scan_done = true;
676         return build_dont_know;
677 }

679 /*
680  *      build_suffix_list(target_suffix)
681  *
682  *      Scans the .SUFFIXES list and figures out
683  *      which suffixes this target can be derived from.
684  *      The target itself is not know here, we just know the suffix of the
685  *      target. For each suffix on the list the target can be derived iff
686  *      a rule exists for the name "<suffix-on-list><target-suffix>".
687  *      A list of all possible building suffixes is built, with the rule for
688  *      each, and tacked to the target suffix nameblock.
689  *
690  *      Parameters:
691  *              target_suffix   The suffix we build a match list for
692  *
693  *      Global variables used:
694  *              debug_level     Indicates how much tracing to do
695  *              recursion_level Used for tracing
696  *              suffixes        List of suffixes used for scan (from .SUFFIXES)
697  *              working_on_targets Indicates that this is a real target
698  */
699 void
700 build_suffix_list(register Name target_suffix)
701 {
702         register Dependency     source_suffix;
703         wchar_t                 rule_name[MAXPATHLEN];
704         register Property       line;
705         register Property       suffix;
706         Name                    rule;

708         /* If this is before default.mk has been read we just return to try */
709         /* again later */
710         if ((suffixes == NULL) || !working_on_targets) {
```



```
711                 return;
712         }
713         if (debug_level > 1) {
714                 (void) printf("%*sbuild_suffix_list(%s) ",
608                 (void) printf(NOCATGETS("%*sbuild_suffix_list(%s) "),
715                                 recursion_level,
716                                 "",
717                                 target_suffix->string_mb);
718         }
719         /* Mark the target suffix saying we cashed its list */
720         target_suffix->has_read_suffixes = true;
721         /* Scan the .SUFFIXES list */
722         for (source_suffix = suffixes;
723              source_suffix != NULL;
724              source_suffix = source_suffix->next) {
725                 /*
726                  * Build the name "<suffix-on-list><target-suffix>".
727                  * (a popular one would be ".c.o").
728                  */
729                 (void) mbstowcs(rule_name,
730                                 source_suffix->name->string_mb,
731                                 (int) source_suffix->name->hash.length);
732                 (void) mbstowcs(rule_name + source_suffix->name->hash.length,
733                                 target_suffix->string_mb,
734                                 (int) target_suffix->hash.length);
735                 /*
736                  * Check if that name has a rule. If not, it cannot match
737                  * any implicit rule scan and is ignored.
738                  * The GETNAME() call only checks for presence, it will not
739                  * enter the name if it is not defined.
740                  */
741                 if (((rule = getname_fn(rule_name,
742                                         (int) (source_suffix->name->
743                                                hash.length +
744                                                target_suffix->hash.length),
745                                         true)) != NULL) &&
746                     ((line = get_prop(rule->prop, line_prop)) != NULL)) {
747                         if (debug_level > 1) {
748                                 (void) printf("%s ", rule->string_mb);
749                         }
750                         /*
751                          * This makes it possible to quickly determine if
752                          * it will pay to look for a suffix property.
753                          */
754                         target_suffix->has_suffixes = true;
755                         /*
756                          * Add the suffix property to the target suffix
757                          * and save the rule with it.
758                          * All information the implicit rule scanner need
759                          * is saved in the suffix property.
760                          */
761                         suffix = append_prop(target_suffix, suffix_prop);
762                         suffix->body.suffix.suffix = source_suffix->name;
763                         suffix->body.suffix.command_template =
764                           line->body.line.command_template;
765                 }
766         }
767         if (debug_level > 1) {
768                 (void) printf("\n");
769         }
770 }

772 /*
773  *      find_percent_rule(target, command, rechecking)
774  *
775  *      Tries to find a rule from the list of wildcard matched rules.
```

```
 776  *      It scans the list attempting to match the target.
 777  *      For each target match it checks if the corresponding source exists.
 778  *      If it does the match is returned.
 779  *      The percent_list is built at makefile read time.
 780  *      Each percent rule get one entry on the list.
 781  *
 782  *      Return value:
 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  */
 797 Doname
 798 find_percent_rule(register Name target, Property *command, Boolean rechecking)
 799 {
 800         register Percent        pat_rule, pat_depe;
 801         register Name           depe_to_check;
 802         register Dependency     depe;
 803         register Property       line;
 804         String_rec              string;
 805         wchar_t                 string_buf[STRING_BUFFER_LENGTH];
 806         String_rec              percent;
 807         wchar_t                 percent_buf[STRING_BUFFER_LENGTH];
 808         Name                    true_target = target;
 809         Name                    less;
 810         Boolean                 nonpattern_less;
 811         Boolean                 dep_name_found = false;
 812         Doname                  result = build_dont_know;
 813         Percent                 rule_candidate = NULL;
 814         Boolean                 rule_maybe_ok;
 815         Boolean                 is_pattern;

 817         /* If the target is constructed for a "::" target we consider that */
 818         if (target->has_target_prop) {
 819                 true_target = get_prop(target->prop,
 820                                        target_prop)->body.target.target;
 821         }
 822         if (target->has_long_member_name) {
 823                 true_target = get_prop(target->prop,
 824                                        long_member_name_prop)->body.long_member_
 825         }
 826         if (debug_level > 1) {
 827                 (void) printf(gettext("%*sLooking for %% rule for %s\n"),
 721                 (void) printf(catgets(catd, 1, 222, "%*sLooking for %% rule for
 828                               recursion_level,
 829                               "",
 830                               true_target->string_mb);
 831         }
 832         for (pat_rule = percent_list;
 833              pat_rule != NULL;
 834              pat_rule = pat_rule->next) {
 835                 /* Avoid infinite recursion when expanding patterns */
 836                 if (pat_rule->being_expanded == true) {
 837                         continue;
 838                 }

 840                 /* Mark this pat_rule as "maybe ok". If no % rule is found
```



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

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

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

 870                          dep_name_found = true;
 871                          if (pat_depe->name->percent) {
 872                                  is_pattern = true;
 873                                  /* build dependency name */
 874                                  INIT_STRING_FROM_STACK(string, string_bu
 875                                  construct_string_from_pattern(pat_depe,
 876                                  depe_to_check = getname_fn(string.buffer
 877                                          FIND_LENGTH,
 878                                          false,
 879                                          &dep_name_found
 880                                  );

 882                                  if ((less == NULL) || nonpattern_less) {
 883                                          less = depe_to_check;
 884                                          nonpattern_less = false;
 885                                  }
 886                          } else {
 887                                  /* nonpattern dependency */
 888                                  is_pattern = false;
 889                                  depe_to_check = pat_depe->name;
 890                                  if(depe_to_check->dollar) {
 891                                          INIT_STRING_FROM_STACK(string, s
 892                                          expand_value(depe_to_check, &str
 893                                          depe_to_check = getname_fn(strin
 894                                                  FIND_LENGTH,
 895                                                  false,
 896                                                  &dep_name_found
 897                                          );
 898                                  }
 899                                  if (less == NULL) {
 900                                          less = depe_to_check;
 901                                  }
 902                          }

 904                          if (depe_to_check == empty_name) {
 905                                          result = build_ok;
 906                          } else {
```

```
907                                     if (debug_level > 1) {
908                                             (void) printf(gettext("%*sTrying
802                                             (void) printf(catgets(catd, 1, 2
909                                                     recursion_level,
910                                                     "",
911                                                     depe_to_check->str
912                                     }

914                                     pat_rule->being_expanded = true;

916                                     /* suppress message output */
917                                     int save_debug_level = debug_level;
918                                     debug_level = 0;

920                                     /* check whether dependency can be built
921                                     if (dependency_exists(depe_to_check,
922                                         get_prop(target->prop,
923                                                 line_prop)))
924                                     {
925                                             result = (Doname) depe_to_check-
926                                     } else {
927                                             if(actual_doname) {
928                                                     result = doname(depe_to_
929                                             } else {
930                                                     result = target_can_be_b
931                                             }
932                                             if(!dep_name_found) {
933                                                     if(result != build_ok &&
934                                                             free_name(depe_t
935                                                     } else {
936                                                             store_name(depe_
937                                                     }
938                                             }
939                                     }
940                                     if(result != build_ok && is_pattern) {
941                                             rule_maybe_ok = false;
942                                     }

944                                     /* restore debug_level */
945                                     debug_level = save_debug_level;
946                             }

948                             if (pat_depe->name->percent) {
949                                     if (string.free_after_use) {
950                                             retmem(string.buffer.start);
951                                     }
952                             }
953                             /* make can't figure out how to make this depend
954                             if (result != build_ok && result != build_runnin
955                                     pat_rule->being_expanded = false;
956                                     break;
957                             }
958                     }
959             } else {
960                     result = build_ok;
961             }

963             /* this pattern rule is the needed one since all dependencies co
964             if (result == build_ok || result == build_running) {
965                     break;
966             }

968             /* Make does not know how to build some of dependencies from thi
969                But if all "pattern" dependencies can be built, we remember t
970                as a candidate for the case if no other pattern rule found.
971             */
```



```
972             if(rule_maybe_ok && rule_candidate == NULL) {
973                     rule_candidate = pat_rule;
974             }
975     }

977     /* if no pattern matching rule was found, use the remembered candidate
978        or return build_dont_know if there is no candidate.
979     */
980     if (result != build_ok && result != build_running) {
981             if(rule_candidate) {
982                     pat_rule = rule_candidate;
983             } else {
984                     return build_dont_know;
985             }
986     }

988     /* if we are performing only check whether dependency could be built wit
989        return success */
990     if (command == NULL) {
991             if(pat_rule != NULL) {
992                     pat_rule->being_expanded = false;
993             }
994             return result;
995     }

997     if (debug_level > 1) {
998             (void) printf(gettext("%*sMatched %s:"),
892             (void) printf(catgets(catd, 1, 224, "%*sMatched %s:"),
999                                     recursion_level,
1000                                    "",
1001                                    target->string_mb);

1003            for (pat_depe = pat_rule->dependencies;
1004                 pat_depe != NULL;
1005                 pat_depe = pat_depe->next) {
1006                    if (pat_depe->name->percent) {
1007                            INIT_STRING_FROM_STACK(string, string_buf);
1008                            construct_string_from_pattern(pat_depe, &percent
1009                            depe_to_check = GETNAME(string.buffer.start, FIN
1010                    } else {
1011                            depe_to_check = pat_depe->name;
1012                            if(depe_to_check->dollar) {
1013                                    INIT_STRING_FROM_STACK(string, string_bu
1014                                    expand_value(depe_to_check, &string, fal
1015                                    depe_to_check = GETNAME(string.buffer.st
1016                            }
1017                    }

1019                    if (depe_to_check != empty_name) {
1020                            (void) printf(" %s", depe_to_check->string_mb);
1021                    }
1022            }

1024            (void) printf(gettext(" from: %s:"),
918             (void) printf(catgets(catd, 1, 225, " from: %s:"),
1025                                    pat_rule->name->string_mb);

1027            for (pat_depe = pat_rule->dependencies;
1028                 pat_depe != NULL;
1029                 pat_depe = pat_depe->next) {
1030                    (void) printf(" %s", pat_depe->name->string_mb);
1031            }

1033            (void) printf("\n");
1034    }
```

```
1036        if (true_target->colons == no_colon) {
1037                true_target->colons = one_colon;
1038        }

1040        /* create deppendency list and target group from matched pattern rule */
1041        create_target_group_and_dependencies_list(target, pat_rule, &percent);

1043        /* save command */
1044        line = get_prop(target->prop, line_prop);
1045        *command = line;

1047        /* free query chain if one exist */
1048        while(line->body.line.query != NULL) {
1049                Chain to_free = line->body.line.query;
1050                line->body.line.query = line->body.line.query->next;
1051                retmem_mb((char *) to_free);
1052        }

1054        if (line->body.line.dependencies != NULL) {
1055                /* build all collected dependencies */
1056                for (depe = line->body.line.dependencies;
1057                     depe != NULL;
1058                     depe = depe->next) {
1059                   actual_doname = true;
1060                   result = doname_check(depe->name, true, true, depe->auto

1062                   actual_doname = false;
1063                   if (result == build_failed) {
1064                           pat_rule->being_expanded = false;
1065                           return build_failed;
1066                   }
1067                   if (result == build_running) {
1068                           pat_rule->being_expanded = false;
1069                           return build_running;
1070                   }

1072                   if ((depe->name->stat.time > line->body.line.dependency_
1073                       (debug_level > 1)) {
1074                           (void) printf(gettext("%*sDate(%s)=%s Date-depen
 968                           (void) printf(catgets(catd, 1, 226, "%*sDate(%s)
1075                                         recursion_level,
1076                                         "",
1077                                         depe->name->string_mb,
1078                                         time_to_string(depe->name->stat.ti
1079                                         true_target->string_mb,
1080                                         time_to_string(line->body.line.dep
1081                   }

1083                   line->body.line.dependency_time =
1084                     MAX(line->body.line.dependency_time, depe->name->stat.

1086                   /* determine whether this dependency made target out of
1087                   Boolean out_of_date;
1088                   if (target->is_member || depe->name->is_member) {
1089                           out_of_date = (Boolean) OUT_OF_DATE_SEC(target->
1090                   } else {
1091                           out_of_date = (Boolean) OUT_OF_DATE(target->stat
1092                   }
1093                   if (build_unconditional || out_of_date) {
1094                           if(!rechecking) {
1095                                   line->body.line.is_out_of_date = true;
1096                           }
1097                           add_target_to_chain(depe->name, &(line->body.lin

1099                           if (debug_level > 0) {
1100                                   (void) printf(gettext("%*sBuilding %s us
```



```
 994                                   (void) printf(catgets(catd, 1, 227, "%*s
1101                                                 recursion_level,
1102                                                 "",
1103                                                 true_target->string_mb,
1104                                                 pat_rule->name->string_mb)

1106                                   for (pat_depe = pat_rule->dependencies;
1107                                        pat_depe != NULL;
1108                                        pat_depe = pat_depe->next) {
1109                                           (void) printf(" %s", pat_depe->n
1110                                   }

1112                                   (void) printf(gettext(" because it is ou
1006                                   (void) printf(catgets(catd, 1, 228, " be
1113                                                 depe->name->string_mb);
1114                           }
1115                   }
1116           }
1117   } else {
1118           if ((true_target->stat.time <= file_doesnt_exist) ||
1119               (true_target->stat.time < line->body.line.dependency_time))
1120                   if(!rechecking) {
1121                           line->body.line.is_out_of_date = true;
1122                   }
1123                   if (debug_level > 0) {
1124                           (void) printf(gettext("%*sBuilding %s using patt
1018                           (void) printf(catgets(catd, 1, 229, "%*sBuilding
1125                                         recursion_level,
1126                                         "",
1127                                         true_target->string_mb,
1128                                         pat_rule->name->string_mb,
1129                                         (target->stat.time > file_doesnt_e
1130                                         gettext("because it is out of date
1131                                         gettext("because it does not exist
1024                                         catgets(catd, 1, 230, "because it
1025                                         catgets(catd, 1, 236, "because it
1132                   }
1133           }
1134   }

1136        /* enter explicit rule from percent rule */
1137        Name lmn_target = true_target;
1138        if (true_target->has_long_member_name) {
1139                lmn_target = get_prop(true_target->prop, long_member_name_prop)-
1140        }
1141        line->body.line.sccs_command = false;
1142        line->body.line.target = true_target;
1143        line->body.line.command_template = pat_rule->command_template;
1144        line->body.line.star = GETNAME(percent.buffer.start, FIND_LENGTH);
1145        line->body.line.less = less;

1147        if (lmn_target->parenleft) {
1148                Wstring lmn_string(lmn_target);

1150                wchar_t *left = (wchar_t *) wschr(lmn_string.get_string(), (int)
1151                wchar_t *right = (wchar_t *) wschr(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                }
1158        } else {
1159                line->body.line.percent = NULL;
1160        }
1161        pat_rule->being_expanded = false;
```

```
1163         return result;
1164 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   86805 Wed May 20 12:19:50 2015
new/usr/src/cmd/make/bin/main.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

/*
 *      main.cc
 *
 *      make program main routine plus some helper routines
 */

/*
 * Included files
 */
#if defined(TEAMWARE_MAKE_CMN)
#       include <avo/intl.h>
#endif

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


#include <locale.h>             /* setlocale() */
#include <libgen.h>
#include <mk/defs.h>
#include <mksdmsi18n/mksdmsi18n.h>      /* libmksdmsi18n_init() */
#include <mksh/macro.h>         /* getvar() */
#include <mksh/misc.h>          /* getmem(), setup_char_semantics() */

#if defined(TEAMWARE_MAKE_CMN)
#endif

#include <pwd.h>                /* getpwnam() */
#include <setjmp.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/errno.h>          /* ENOENT */
#include <sys/stat.h>           /* fstat() */
#include <fcntl.h>              /* open() */

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



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

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

// From parallel.cc
#define MAXJOBS_ADJUST_RFE4694000

#ifdef MAXJOBS_ADJUST_RFE4694000
extern void job_adjust_fini();
#endif /* MAXJOBS_ADJUST_RFE4694000 */


/*
 * Defined macros
 */
#define LD_SUPPORT_ENV_VAR      "SGS_SUPPORT_32"
#define LD_SUPPORT_ENV_VAR_32   "SGS_SUPPORT_32"
#define LD_SUPPORT_ENV_VAR_64   "SGS_SUPPORT_64"
#define LD_SUPPORT_MAKE_LIB     "libmakestate.so.1"
#define MAKE_PREFIX             NOCATGETS("/usr")
#define LD_SUPPORT_ENV_VAR      NOCATGETS("SGS_SUPPORT_32")
#define LD_SUPPORT_ENV_VAR_32   NOCATGETS("SGS_SUPPORT_32")
#define LD_SUPPORT_ENV_VAR_64   NOCATGETS("SGS_SUPPORT_64")
#define LD_SUPPORT_MAKE_LIB     NOCATGETS("libmakestate.so.1")
#ifdef __i386
#define LD_SUPPORT_MAKE_ARCH    "i386"
#define LD_SUPPORT_MAKE_ARCH    NOCATGETS("i386")
#elif __sparc
#define LD_SUPPORT_MAKE_ARCH    "sparc"
#define LD_SUPPORT_MAKE_ARCH    NOCATGETS("sparc")
#else
#error "Unsupported architecture"
#endif

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static  char            *argv_zero_string;
static  Boolean         build_failed_ever_seen;
static  Boolean         continue_after_error_ever_seen; /* '-k' */
static  Boolean         dmake_group_specified;          /* '-g' */
static  Boolean         dmake_max_jobs_specified;       /* '-j' */
static  Boolean         dmake_mode_specified;           /* '-m' */
static  Boolean         dmake_add_mode_specified;       /* '-x' */
static  Boolean         dmake_output_mode_specified;    /* '-x DMAKE_OUTPUT_MODE
static  Boolean         dmake_compat_mode_specified;    /* '-x SUN_MAKE_COMPAT_M
static  Boolean         dmake_odir_specified;           /* '-o' */
static  Boolean         dmake_rcfile_specified;         /* '-c' */
static  Boolean         env_wins;                       /* '-e' */
static  Boolean         ignore_default_mk;              /* '-r' */
static  Boolean         list_all_targets;               /* '-T' */
static  int             mf_argc;
static  char            **mf_argv;
static  Dependency_rec  not_auto_depen_struct;
static  Dependency      not_auto_depen = &not_auto_depen_struct;
static  Boolean         pmake_cap_r_specified;          /* '-R' */
static  Boolean         pmake_machinesfile_specified;   /* '-M' */
static  Boolean         stop_after_error_ever_seen;     /* '-S' */
```

```
113 static  Boolean         trace_status;                   /* '-p' */

115 #ifdef DMAKE_STATISTICS
116 static  Boolean         getname_stat = false;
117 #endif

119         static  time_t          start_time;
120         static  int             g_argc;
121         static  char            **g_argv;

123 /*
124  * File table of contents
125  */
126         extern "C" void         cleanup_after_exit(void);

128 extern "C" {
129         extern  void            dmake_exit_callback(void);
130         extern  void            dmake_message_callback(char *);
131 }

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

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

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

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

154 static const char       verstring[] = "illumos make";

156 jmp_buf jmpbuffer;
166 extern nl_catd catd;

158 /*
159  *      main(argc, argv)
160  *
161  *      Parameters:
162  *              argc                    You know what this is
163  *              argv                    You know what this is
164  *
165  *      Static variables used:
166  *              list_all_targets        make -T seen
167  *              trace_status            make -p seen
168  *
169  *      Global variables used:
170  *              debug_level             Should we trace make actions?
171  *              keep_state              Set if .KEEP_STATE seen
172  *              makeflags               The Name "MAKEFLAGS", used to get macro
173  *              remote_command_name     Name of remote invocation cmd ("on")
174  *              running_list            List of parallel running processes
175  *              stdout_stderr_same      true if stdout and stderr are the same
176  *              auto_dependencies       The Name "SUNPRO_DEPENDENCIES"
177  *              temp_file_directory     Set to the dir where we create tmp file
```



```
178  *              trace_reader            Set to reflect tracing status
179  *              working_on_targets      Set when building user targets
180  */
181 int
182 main(int argc, char *argv[])
183 {
184         /*
185          * cp is a -> to the value of the MAKEFLAGS env var,
186          * which has to be regular chars.
187          */
188         register char           *cp;
189         char                    make_state_dir[MAXPATHLEN];
190         Boolean                 parallel_flag = false;
191         char                    *prognameptr;
192         char                    *slash_ptr;
193         mode_t                  um;
194         int                     i;
195         struct itimerval        value;
196         char                    def_dmakerc_path[MAXPATHLEN];
197         Name                    dmake_name, dmake_name2;
198         Name                    dmake_value, dmake_value2;
199         Property                prop, prop2;
200         struct stat             statbuf;
201         int                     statval;

203         struct stat             out_stat, err_stat;
204         hostid = gethostid();
205         bsd_signals();

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


210 #ifdef DMAKE_STATISTICS
211         if (getenv("DMAKE_STATISTICS")) {
221         if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
212                 getname_stat = true;
213         }
214 #endif

216 #ifndef TEXT_DOMAIN
217 #define TEXT_DOMAIN     "SYS_TEST"
218 #endif
219         textdomain(TEXT_DOMAIN);
226         catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);

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


231 /*
232  * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
233  * from avo_util library.
234  */
235         libmksdmsi18n_init();


238         textdomain(NOCATGETS("SUNW_SPRO_MAKE"));

221         g_argc = argc;
222         g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
223         for (i = 0; i < argc; i++) {
224                 g_argv[i] = argv[i];
225         }
226         g_argv[i] = NULL;

228         /*
229          * Set argv_zero_string to some form of argv[0] for
```

```
230          * recursive MAKE builds.
231          */

233         if (*argv[0] == (int) slash_char) {
234                 /* argv[0] starts with a slash */
235                 argv_zero_string = strdup(argv[0]);
236         } else if (strchr(argv[0], (int) slash_char) == NULL) {
237                 /* argv[0] contains no slashes */
238                 argv_zero_string = strdup(argv[0]);
239         } else {
240                 /*
241                  * argv[0] contains at least one slash,
242                  * but doesn't start with a slash
243                  */
244                 char    *tmp_current_path;
245                 char    *tmp_string;

247                 tmp_current_path = get_current_path();
248                 tmp_string = getmem(strlen(tmp_current_path) + 1 +
249                                 strlen(argv[0]) + 1);
250                 (void) sprintf(tmp_string,
251                                 "%s/%s",
252                                 tmp_current_path,
253                                 argv[0]);
254                 argv_zero_string = strdup(tmp_string);
255                 retmem_mb(tmp_string);
256         }

258         /*
259          * The following flags are reset if we don't have the
260          * (.nse_depinfo or .make.state) files locked and only set
261          * AFTER the file has been locked. This ensures that if the user
262          * interrupts the program while file_lock() is waiting to lock
263          * the file, the interrupt handler doesn't remove a lock
264          * that doesn't belong to us.
265          */
266         make_state_lockfile = NULL;
267         make_state_locked = false;


270         /*
271          * look for last slash char in the path to look at the binary
272          * name. This is to resolve the hard link and invoke make
273          * in svr4 mode.
274          */

276         /* Sun OS make standart */
277         svr4 = false;
278         posix = false;
279         if(!strcmp(argv_zero_string, "/usr/xpg4/bin/make")) {
298         if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
280                 svr4 = false;
281                 posix = true;
282         } else {
283                 prognameptr = strrchr(argv[0], '/');
284                 if(prognameptr) {
285                         prognameptr++;
286                 } else {
287                         prognameptr = argv[0];
288                 }
289                 if(!strcmp(prognameptr, "svr4.make")) {
308                 if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
290                         svr4 = true;
291                         posix = false;
292                 }
293         }
```



```
294         if (getenv(USE_SVR4_MAKE) || getenv("USE_SVID")){
313         if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
295                 svr4 = true;
296                 posix = false;
297         }

299         /*
300          * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
301          */
302         char * dmake_compat_mode_var = getenv("SUN_MAKE_COMPAT_MODE");
321         char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
303         if (dmake_compat_mode_var != NULL) {
304                 if (0 == strcasecmp(dmake_compat_mode_var, "GNU")) {
323                 if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
305                         gnu_style = true;
306                 }
307                 //svr4 = false;
308                 //posix = false;
309         }

311         /*
312          * Temporary directory set up.
313          */
314         char * tmpdir_var = getenv("TMPDIR");
333         char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
315         if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
316                 strcpy(mbs_buffer, tmpdir_var);
317                 for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
318                         *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
319                         *tmpdir_var = '\0');
320                 if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
321                         sprintf(mbs_buffer2, "%s/dmake.tst.%d.XXXXXX",
340                         sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
322                                 mbs_buffer, getpid());
323                         int fd = mkstemp(mbs_buffer2);
324                         if (fd >= 0) {
325                                 close(fd);
326                                 unlink(mbs_buffer2);
327                                 tmpdir = strdup(mbs_buffer);
328                         }
329                 }
330         }

332         /* find out if stdout and stderr point to the same place */
333         if (fstat(1, &out_stat) < 0) {
334                 fatal(gettext("fstat of standard out failed: %s"), errmsg(errno)
353                 fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
335         }
336         if (fstat(2, &err_stat) < 0) {
337                 fatal(gettext("fstat of standard error failed: %s"), errmsg(errn
356                 fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
338         }
339         if ((out_stat.st_dev == err_stat.st_dev) &&
340             (out_stat.st_ino == err_stat.st_ino)) {
341                 stdout_stderr_same = true;
342         } else {
343                 stdout_stderr_same = false;
344         }
345         /* Make the vroot package scan the path using shell semantics */
346         set_path_style(0);

348         setup_char_semantics();

350         setup_for_projectdir();

352         /*
```

```
353          * If running with .KEEP_STATE, curdir will be set with
354          * the connected directory.
355          */
356         (void) atexit(cleanup_after_exit);

358         load_cached_names();

360 /*
361  *      Set command line flags
362  */
363         setup_makeflags_argv();
364         read_command_options(mf_argc, mf_argv);
365         read_command_options(argc, argv);
366         if (debug_level > 0) {
367                 cp = getenv(makeflags->string_mb);
368                 (void) printf(gettext("MAKEFLAGS value: %s\n"), cp == NULL ? ""
387                 (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
369         }

371         setup_interrupt(handle_interrupt);

373         read_files_and_state(argc, argv);

375         /*
376          * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
377          */
378         MBSTOWCS(wcs_buffer, "DMAKE_OUTPUT_MODE");
397         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
379         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
380         prop2 = get_prop(dmake_name2->prop, macro_prop);
381         if (prop2 == NULL) {
382                 /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
383                 output_mode = txt1_mode;
384         } else {
385                 dmake_value2 = prop2->body.macro.value;
386                 if ((dmake_value2 == NULL) ||
387                     (IS_EQUAL(dmake_value2->string_mb, "TXT1"))) {
406                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
388                         output_mode = txt1_mode;
389                 } else if (IS_EQUAL(dmake_value2->string_mb, "TXT2")) {
408                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
390                         output_mode = txt2_mode;
391                 } else if (IS_EQUAL(dmake_value2->string_mb, "HTML1")) {
410                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))
392                         output_mode = html1_mode;
393                 } else {
394                         warning(gettext("Unsupported value '%s' for DMAKE_OUTPUT
413                         warning(catgets(catd, 1, 352, "Unsupported value '%s' fo
395                                 dmake_value2->string_mb);
396                 }
397         }
398         /*
399          * Find the dmake_mode: parallel, or serial.
400          */
401     if ((!pmake_cap_r_specified) &&
402         (!pmake_machinesfile_specified)) {
403         char *s = strdup(argv[0]);
404
405         MBSTOWCS(wcs_buffer, "DMAKE_MODE");
424         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
406         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
407         prop2 = get_prop(dmake_name2->prop, macro_prop);
408         // If we're invoked as 'make' run serially, regardless of DMAKE_MODE
409         // If we're invoked as 'make' but passed -j, run parallel
410         // If we're invoked as 'dmake', without DMAKE_MODE, default parallel
411         // If we're invoked as 'dmake' and DMAKE_MODE is set, honour it.
```



```
412         if ((strcmp(basename(s), "make") == 0) &&
431         if ((strcmp(basename(s), NOCATGETS("make")) == 0) &&
413             !dmake_max_jobs_specified) {
414                 dmake_mode_type = serial_mode;
415                 no_parallel = true;
416         } else if (prop2 == NULL) {
417                 /* DMAKE_MODE not defined, default based on our name */
418                 char *s = strdup(argv[0]);

420                 if (strcmp(basename(s), "dmake") == 0) {
439                 if (strcmp(basename(s), NOCATGETS("dmake")) == 0) {
421                         dmake_mode_type = parallel_mode;
422                         no_parallel = false;
423                 }
424         } else {
425                 dmake_value2 = prop2->body.macro.value;
426                 if (IS_EQUAL(dmake_value2->string_mb, "parallel")) {
445                 if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel"))) {
427                         dmake_mode_type = parallel_mode;
428                         no_parallel = false;
429                 } else if (IS_EQUAL(dmake_value2->string_mb, "serial")) {
448                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
430                         dmake_mode_type = serial_mode;
431                         no_parallel = true;
432                 } else {
433                         fatal(gettext("Unknown dmake mode argument '%s' after -m
452                         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
434                 }
435         }
436         free(s);
437     }

439         parallel_flag = true;
440         putenv(strdup("DMAKE_CHILD=TRUE"));
459         putenv(strdup(NOCATGETS("DMAKE_CHILD=TRUE")));

442 //
443 // If dmake is running with -t option, set dmake_mode_type to serial.
444 // This is done because doname() calls touch_command() that runs serially.
445 // If we do not do that, maketool will have problems.
446 //
447         if(touch) {
448                 dmake_mode_type = serial_mode;
449                 no_parallel = true;
450         }

452         /*
453          * Check whether stdout and stderr are physically same.
454          * This is in order to decide whether we need to redirect
455          * stderr separately from stdout.
456          * This check is performed only if __DMAKE_SEPARATE_STDERR
457          * is not set. This variable may be used in order to preserve
458          * the 'old' behaviour.
459          */
460         out_err_same = true;
461         char * dmake_sep_var = getenv("__DMAKE_SEPARATE_STDERR");
462         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, "NO"))) {
480         char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
481         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
463                 struct stat stdout_stat;
464                 struct stat stderr_stat;
465                 if( (fstat(1, &stdout_stat) == 0)
466                  && (fstat(2, &stderr_stat) == 0) )
467                 {
468                         if( (stdout_stat.st_dev != stderr_stat.st_dev)
469                          || (stdout_stat.st_ino != stderr_stat.st_ino) )
```

```
470                         {
471                                 out_err_same = false;
472                         }
473                 }
474         }

476
477 /*
478  *      Enable interrupt handler for alarms
479  */
480         (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

482 /*
483  *      Check if make should report
484  */
485         if (getenv(sunpro_dependencies->string_mb) != NULL) {
486                 FILE    *report_file;

488                 report_dependency("");
489                 report_file = get_report_file();
490                 if ((report_file != NULL) && (report_file != (FILE*)-1)) {
491                         (void) fprintf(report_file, "\n");
492                 }
493         }

495 /*
496  *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
497  */
498         if (keep_state) {
499                 maybe_append_prop(sunpro_dependencies, macro_prop)->
500                   body.macro.exported = true;
501         } else {
502                 maybe_append_prop(sunpro_dependencies, macro_prop)->
503                   body.macro.exported = false;
504         }

506         working_on_targets = true;
507         if (trace_status) {
508                 dump_make_state();
509                 fclose(stdout);
510                 fclose(stderr);
511                 exit_status = 0;
512                 exit(0);
513         }
514         if (list_all_targets) {
515                 dump_target_list();
516                 fclose(stdout);
517                 fclose(stderr);
518                 exit_status = 0;
519                 exit(0);
520         }
521         trace_reader = false;

523         /*
524          * Set temp_file_directory to the directory the .make.state
525          * file is written to.
526          */
527         if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
528                 temp_file_directory = strdup(get_current_path());
529         } else {
530                 *slash_ptr = (int) nul_char;
531                 (void) strcpy(make_state_dir, make_state->string_mb);
532                 *slash_ptr = (int) slash_char;
533                    /* when there is only one slash and it's the first
534                    ** character, make_state_dir should point to '/'.
535                    */
```



```
536                 if(make_state_dir[0] == '\0') {
537                    make_state_dir[0] = '/';
538                    make_state_dir[1] = '\0';
539                 }
540                 if (make_state_dir[0] == (int) slash_char) {
541                         temp_file_directory = strdup(make_state_dir);
542                 } else {
543                         char    tmp_current_path2[MAXPATHLEN];
544
545                         (void) sprintf(tmp_current_path2,
546                                         "%s/%s",
547                                         get_current_path(),
548                                         make_state_dir);
549                         temp_file_directory = strdup(tmp_current_path2);
550                 }
551         }


554         report_dir_enter_leave(true);

556         make_targets(argc, argv, parallel_flag);

558         report_dir_enter_leave(false);

560         if (build_failed_ever_seen) {
561                 if (posix) {
562                         exit_status = 1;
563                 }
564                 exit(1);
565         }
566         exit_status = 0;
567         exit(0);
568         /* NOTREACHED */
569 }

571 /*
572  *      cleanup_after_exit()
573  *
574  *      Called from exit(), performs cleanup actions.
575  *
576  *      Parameters:
577  *              status          The argument exit() was called with
578  *              arg             Address of an argument vector to
579  *                              cleanup_after_exit()
580  *
581  *      Global variables used:
582  *              command_changed Set if we think .make.state should be rewritten
583  *              current_line    Is set we set commands_changed
584  *              do_not_exec_rule
585  *                              True if -n flag on
586  *              done            The Name ".DONE", rule we run
587  *              keep_state      Set if .KEEP_STATE seen
588  *              parallel        True if building in parallel
589  *              quest           If -q is on we do not run .DONE
590  *              report_dependencies
591  *                              True if -P flag on
592  *              running_list    List of parallel running processes
593  *              temp_file_name  The temp file is removed, if any
613  *              catd    the message catalog file
594  */
595 extern "C" void
596 cleanup_after_exit(void)
597 {
598         Running         rp;

600 extern long     getname_bytes_count;
```

```
extern long     getname_names_count;
extern long     getname_struct_count;
extern long     freename_bytes_count;
extern long     freename_names_count;
extern long     freename_struct_count;
extern long     other_alloc;

extern long     env_alloc_num;
extern long     env_alloc_bytes;


#ifdef DMAKE_STATISTICS
if(getname_stat) {
        printf(">>> Getname statistics:\n");
        printf("  Allocated:\n");
        printf("        Names: %ld\n", getname_names_count);
        printf("      Strings: %ld Kb (%ld bytes)\n", getname_bytes_count/1000,
        printf("      Structs: %ld Kb (%ld bytes)\n", getname_struct_count/1000,
        printf("  Total bytes: %ld Kb (%ld bytes)\n", getname_struct_count/1000

        printf("\n  Unallocated: %ld\n", freename_names_count);
        printf("        Names: %ld\n", freename_names_count);
        printf("      Strings: %ld Kb (%ld bytes)\n", freename_bytes_count/1000,
        printf("      Structs: %ld Kb (%ld bytes)\n", freename_struct_count/1000
        printf("  Total bytes: %ld Kb (%ld bytes)\n", freename_struct_count/1000
        printf(NOCATGETS(">>> Getname statistics:\n"));
        printf(NOCATGETS("  Allocated:\n"));
        printf(NOCATGETS("        Names: %ld\n"), getname_names_count);
        printf(NOCATGETS("      Strings: %ld Kb (%ld bytes)\n"), getname_bytes_c
        printf(NOCATGETS("      Structs: %ld Kb (%ld bytes)\n"), getname_struct_
        printf(NOCATGETS("  Total bytes: %ld Kb (%ld bytes)\n"), getname_struct_

        printf(NOCATGETS("\n  Unallocated: %ld\n"), freename_names_count);
        printf(NOCATGETS("        Names: %ld\n"), freename_names_count);
        printf(NOCATGETS("      Strings: %ld Kb (%ld bytes)\n"), freename_bytes_
        printf(NOCATGETS("      Structs: %ld Kb (%ld bytes)\n"), freename_struct
        printf(NOCATGETS("  Total bytes: %ld Kb (%ld bytes)\n"), freename_struct

        printf("\n  Total used: %ld Kb (%ld bytes)\n", (getname_struct_count/100
        printf(NOCATGETS("\n  Total used: %ld Kb (%ld bytes)\n"), (getname_struc

        printf("\n>>> Other:\n");
        printf(NOCATGETS("\n>>> Other:\n"));
        printf(
                "       Env (%ld): %ld Kb (%ld bytes)\n",
                NOCATGETS("       Env (%ld): %ld Kb (%ld bytes)\n"),
                env_alloc_num,
                env_alloc_bytes/1000,
                env_alloc_bytes
        );

}
#endif

        parallel = false;
        /* If we used the SVR4_MAKE, don't build .DONE or .FAILED */
        if (!getenv(USE_SVR4_MAKE)){
            /* Build the target .DONE or .FAILED if we caught an error */
            if (!quest && !list_all_targets) {
                Name            failed_name;

                MBSTOWCS(wcs_buffer, ".FAILED");
                MBSTOWCS(wcs_buffer, NOCATGETS(".FAILED"));
                failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
                if ((exit_status != 0) && (failed_name->prop != NULL)) {
                        /*
```



```
                         * [tolik] switch DMake to serial mode
                         */
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
                        (void) doname(failed_name, false, true);
                } else {
                    if (!trace_status) {
                        /*
                         * Switch DMake to serial mode
                         */
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
                        (void) doname(done, false, true);
                    }
                }
            }
        }
        /*
         * Remove the temp file utilities report dependencies thru if it
         * is still around
         */
        if (temp_file_name != NULL) {
                (void) unlink(temp_file_name->string_mb);
        }
        /*
         * Do not save the current command in .make.state if make
         * was interrupted.
         */
        if (current_line != NULL) {
                command_changed = true;
                current_line->body.line.command_used = NULL;
        }
        /*
         * For each parallel build process running, remove the temp files
         * and zap the command line so it won't be put in .make.state
         */
        for (rp = running_list; rp != NULL; rp = rp->next) {
                if (rp->temp_file != NULL) {
                        (void) unlink(rp->temp_file->string_mb);
                }
                if (rp->stdout_file != NULL) {
                        (void) unlink(rp->stdout_file);
                        retmem_mb(rp->stdout_file);
                        rp->stdout_file = NULL;
                }
                if (rp->stderr_file != NULL) {
                        (void) unlink(rp->stderr_file);
                        retmem_mb(rp->stderr_file);
                        rp->stderr_file = NULL;
                }
                command_changed = true;
/*
                line = get_prop(rp->target->prop, line_prop);
                if (line != NULL) {
                        line->body.line.command_used = NULL;
                }
 */
        }
        /* Remove the statefile lock file if the file has been locked */
        if ((make_state_lockfile != NULL) && (make_state_locked)) {
                (void) unlink(make_state_lockfile);
                make_state_lockfile = NULL;
                make_state_locked = false;
        }
        /* Write .make.state */
        write_state_file(1, (Boolean) 1);
```

```
#if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
        job_adjust_fini();
#endif

#ifdef TEAMWARE_MAKE_CMN
        catclose(catd);
#endif
}

/*
 *      handle_interrupt()
 *
 *      This is where C-C traps are caught.
 *
 *      Parameters:
 *
 *      Global variables used (except DMake 1.0):
 *              current_target          Sometimes the current target is removed
 *              do_not_exec_rule        But not if -n is on
 *              quest                   or -q
 *              running_list            List of parallel running processes
 *              touch                   Current target is not removed if -t on
 */
void
handle_interrupt(int)
{
        Property                member;
        Running                 rp;

        (void) fflush(stdout);
        if (childPid > 0) {
                kill(childPid, SIGTERM);
                childPid = -1;
        }
        for (rp = running_list; rp != NULL; rp = rp->next) {
                if (rp->state != build_running) {
                        continue;
                }
                if (rp->pid > 0) {
                        kill(rp->pid, SIGTERM);
                        rp->pid = -1;
                }
        }
        if (getpid() == getpgrp()) {
                bsd_signal(SIGTERM, SIG_IGN);
                kill (-getpid(), SIGTERM);
        }
        /* Clean up all parallel children already finished */
        finish_children(false);

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

        while (wait((int *) NULL) != -1);
        /* Delete the current targets unless they are precious */
        if ((current_target != NULL) &&
            current_target->is_member &&
            ((member = get_prop(current_target->prop, member_prop)) != NULL)) {
                current_target = member->body.member.library;
        }
        if (!do_not_exec_rule &&
            !touch &&
            !quest &&
            (current_target != NULL) &&
            !(current_target->stat.is_precious || all_precious)) {
```



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

                if (exists(current_target) != file_doesnt_exist) {
                        (void) fprintf(stderr,
                                       "\n*** %s ",
                                       current_target->string_mb);
                        if (current_target->stat.is_dir) {
                                (void) fprintf(stderr,
                                               gettext("not removed.\n"),
                                               catgets(catd, 1, 168, "not remove
                                               current_target->string_mb);
                        } else if (unlink(current_target->string_mb) == 0) {
                                (void) fprintf(stderr,
                                               gettext("removed.\n"),
                                               catgets(catd, 1, 169, "removed.\n
                                               current_target->string_mb);
                        } else {
                                (void) fprintf(stderr,
                                               gettext("could not be removed: %s
                                               catgets(catd, 1, 170, "could not
                                               current_target->string_mb,
                                               errmsg(errno));
                        }
                }
        }
        for (rp = running_list; rp != NULL; rp = rp->next) {
                if (rp->state != build_running) {
                        continue;
                }
                if (rp->target->is_member &&
                    ((member = get_prop(rp->target->prop, member_prop)) !=
                     NULL)) {
                        rp->target = member->body.member.library;
                }
                if (!do_not_exec_rule &&
                    !touch &&
                    !quest &&
                    !(rp->target->stat.is_precious || all_precious)) {

                        rp->target->stat.time = file_no_time;
                        if (exists(rp->target) != file_doesnt_exist) {
                                (void) fprintf(stderr,
                                               "\n*** %s ",
                                               rp->target->string_mb);
                                if (rp->target->stat.is_dir) {
                                        (void) fprintf(stderr,
                                                       gettext("not removed.\n")
                                                       catgets(catd, 1, 171, "no
                                                       rp->target->string_mb);
                                } else if (unlink(rp->target->string_mb) == 0) {
                                        (void) fprintf(stderr,
                                                       gettext("removed.\n"),
                                                       catgets(catd, 1, 172, "re
                                                       rp->target->string_mb);
                                } else {
                                        (void) fprintf(stderr,
                                                       gettext("could not be rem
                                                       catgets(catd, 1, 173, "co
                                                       rp->target->string_mb,
                                                       errmsg(errno));
                                }
                        }
                }
        }
```

```
841         /* Have we locked .make.state or .nse_depinfo? */
842         if ((make_state_lockfile != NULL) && (make_state_locked)) {
843                 unlink(make_state_lockfile);
844                 make_state_lockfile = NULL;
845                 make_state_locked = false;
846         }
847         /*
848          * Re-read .make.state file (it might be changed by recursive make)
849          */
850         check_state(NULL);

852         report_dir_enter_leave(false);

854         exit_status = 2;
855         exit(2);
856 }
_____unchanged_portion_omitted_


877 /*
878  *      read_command_options(argc, argv)
879  *
880  *      Scan the cmd line options and process the ones that start with "-"
881  *
882  *      Return value:
883  *                              -M argument, if any
884  *
885  *      Parameters:
886  *              argc            You know what this is
887  *              argv            You know what this is
888  *
889  *      Global variables used:
890  */
891 static void
892 read_command_options(register int argc, register char **argv)
893 {
894         register int            ch;
895         int                     current_optind = 1;
896         int                     last_optind_with_double_hyphen = 0;
897         int                     last_optind;
898         int                     last_current_optind;
899         register int            i;
900         register int            j;
901         register int            k;
902         register int            makefile_next = 0; /*
903                                                     * flag to note options:
904                                                     * -c, f, g, j, m, o
905                                                     */
906         const char              *tptr;
907         const char              *CMD_OPTS;

909         extern char             *optarg;
910         extern int              optind, opterr, optopt;

912 #define SUNPRO_CMD_OPTS "-~Bbc:Ddef:g:ij:K:kM:m:NnO:o:PpqRrSsTtuVvwx:"

914 #       define SVR4_CMD_OPTS   "-c:ef:g:ij:km:nO:o:pqrsTtVv"

916         /*
917          * Added V in SVR4_CMD_OPTS also, which is going to be a hidden
918          * option, just to make sure that the getopt doesn't fail when some
919          * users leave their USE_SVR4_MAKE set and try to use the makefiles
920          * that are designed to issue commands like $(MAKE) -V. Anyway it
921          * sets the same flag but ensures that getopt doesn't fail.
```



```
922          */

924         opterr = 0;
925         optind = 1;
926         while (1) {
927                 last_optind=optind;                     /* Save optind and curre
928                 last_current_optind=current_optind;     /* in case we have to re
929                 if (svr4) {
930                         CMD_OPTS=SVR4_CMD_OPTS;
931                         ch = getopt(argc, argv, SVR4_CMD_OPTS);
932                 } else {
933                         CMD_OPTS=SUNPRO_CMD_OPTS;
934                         ch = getopt(argc, argv, SUNPRO_CMD_OPTS);
935                 }
936                 if (ch == EOF) {
937                         if(optind < argc) {
938                                 /*
939                                  * Fixing bug 4102537:
940                                  *    Strange behaviour of command make using --
941                                  * Not all argv have been processed
942                                  * Skip non-flag argv and continue processing.
943                                  */
944                                 optind++;
945                                 current_optind++;
946                                 continue;
947                         } else {
948                                 break;
949                         }

951                 }
952                 if (ch == '?') {
953                         if (optopt == '-') {
954                                 /* Bug 5060758: getopt() changed behavior (s10_6
955                                  * and now we have to deal with cases when optio
956                                  * with double hyphen appear here, from -$(MAKEF
957                                  */
958                                 i = current_optind;
959                                 if (argv[i][0] == '-') {
960                                   if (argv[i][1] == '-') {
961                                     if (argv[i][2] != '\0') {
962                                       /* Check if this option is allowed */
963                                       tptr = strchr(CMD_OPTS, argv[i][2]);
964                                       if (tptr) {
965                                         if (last_optind_with_double_hyphen != cu
966                                           /* This is first time we are trying to
967                                            * problem with this option. If we com
968                                            * time, we will go to fatal error.
969                                            */
970                                           last_optind_with_double_hyphen = curre
971
972                                           /* Eliminate first hyphen character */
973                                           for (j=0; argv[i][j] != '\0'; j++) {
974                                             argv[i][j] = argv[i][j+1];
975                                           }
976
977                                           /* Repeat the processing of this argum
978                                           optind=last_optind;
979                                           current_optind=last_current_optind;
980                                           continue;
981                                         }
982                                       }
983                                     }
984                                   }
985                                 }
986                         }
987                 }
```

```
 989                 if (ch == '?') {
 990                         if (svr4) {
 991                                 fprintf(stderr,
 992                                         gettext("Usage : dmake [ -f makefile ][
1016                                         catgets(catd, 1, 267, "Usage : dmake [ -
 993                                 fprintf(stderr,
 994                                         gettext("             [ -j dmake_max_jo
1018                                         catgets(catd, 1, 268, "               [ -
 995                                 fprintf(stderr,
 996                                         gettext("             [ -e ][ -i ][ -k
1020                                         catgets(catd, 1, 269, "               [ -
 997                                 tptr = strchr(SVR4_CMD_OPTS, optopt);
 998                         } else {
 999                                 fprintf(stderr,
1000                                         gettext("Usage : dmake [ -f makefile ][
1024                                         catgets(catd, 1, 272, "Usage : dmake [ -
1001                                 fprintf(stderr,
1002                                         gettext("             [ -j dmake_max_jo
1026                                         catgets(catd, 1, 273, "               [ -
1003                                 fprintf(stderr,
1004                                         gettext("             [ -d ][ -dd ][ -D
1028                                         catgets(catd, 1, 274, "               [ -
1005                                 fprintf(stderr,
1006                                         gettext("             [ -q ][ -r ][ -s
1030                                         catgets(catd, 1, 275, "               [ -
1007                                 tptr = strchr(SUNPRO_CMD_OPTS, optopt);
1008                         }
1009                         if (!tptr) {
1010                                 fatal(gettext("Unknown option '-%c'"), optopt);
1034                                 fatal(catgets(catd, 1, 279, "Unknown option '-%c
1011                         } else {
1012                                 fatal(gettext("Missing argument after '-%c'"), o
1036                                 fatal(catgets(catd, 1, 280, "Missing argument af
1013                         }
1014                 }



1018                 makefile_next |= parse_command_option(ch);
1019                 /*
1020                  * If we're done processing all of the options of
1021                  * ONE argument string...
1022                  */
1023                 if (current_optind < optind) {
1024                         i = current_optind;
1025                         k = 0;
1026                         /* If there's an argument for an option... */
1027                         if ((optind - current_optind) > 1) {
1028                                 k = i + 1;
1029                         }
1030                         switch (makefile_next) {
1031                         case 0:
1032                                 argv[i] = NULL;
1033                                 /* This shouldn't happen */
1034                                 if (k) {
1035                                         argv[k] = NULL;
1036                                 }
1037                                 break;
1038                         case 1: /* -f seen */
1039                                 argv[i] = (char *)"-f";
1063                                 argv[i] = (char *)NOCATGETS("-f");
1040                                 break;
1041                         case 2: /* -c seen */
1042                                 argv[i] = (char *)"-c";
1066                                 argv[i] = (char *)NOCATGETS("-c");
```



```
1043                                 break;
1044                         case 4: /* -g seen */
1045                                 argv[i] = (char *)"-g";
1069                                 argv[i] = (char *)NOCATGETS("-g");
1046                                 break;
1047                         case 8: /* -j seen */
1048                                 argv[i] = (char *)"-j";
1072                                 argv[i] = (char *)NOCATGETS("-j");
1049                                 break;
1050                         case 16: /* -M seen */
1051                                 argv[i] = (char *)"-M";
1075                                 argv[i] = (char *)NOCATGETS("-M");
1052                                 break;
1053                         case 32: /* -m seen */
1054                                 argv[i] = (char *)"-m";
1078                                 argv[i] = (char *)NOCATGETS("-m");
1055                                 break;
1056                         case 128: /* -O seen */
1057                                 argv[i] = (char *)"-O";
1081                                 argv[i] = (char *)NOCATGETS("-O");
1058                                 break;
1059                         case 256: /* -K seen */
1060                                 argv[i] = (char *)"-K";
1084                                 argv[i] = (char *)NOCATGETS("-K");
1061                                 break;
1062                         case 512:       /* -o seen */
1063                                 argv[i] = (char *)"-o";
1087                                 argv[i] = (char *)NOCATGETS("-o");
1064                                 break;
1065                         case 1024: /* -x seen */
1066                                 argv[i] = (char *)"-x";
1090                                 argv[i] = (char *)NOCATGETS("-x");
1067                                 break;
1068                         default: /* > 1 of -c, f, g, j, K, M, m, O, o, x seen */
1069                                 fatal(gettext("Illegal command line. More than o
1093                                 fatal(catgets(catd, 1, 286, "Illegal command lin
1070                         }

1072                         makefile_next = 0;
1073                         current_optind = optind;
1074                 }
1075         }
1076 }
_____unchanged_portion_omitted_

1135 /*
1136  * Convert the MAKEFLAGS string value into a vector of char *, similar
1137  * to argv.
1138  */
1139 static void
1140 setup_makeflags_argv()
1141 {
1142         char            *cp;
1143         char            *cp1;
1144         char            *cp2;
1145         char            *cp3;
1146         char            *cp_orig;
1147         Boolean         add_hyphen;
1148         int             i;
1149         char            tmp_char;

1151         mf_argc = 1;
1152         cp = getenv(makeflags->string_mb);
1153         cp_orig = cp;

1155         if (cp) {
```

```
1156                 /*
1157                  * If new MAKEFLAGS format, no need to add hyphen.
1158                  * If old MAKEFLAGS format, add hyphen before flags.
1159                  */

1161                 if ((strchr(cp, (int) hyphen_char) != NULL) ||
1162                     (strchr(cp, (int) equal_char) != NULL)) {

1164                         /* New MAKEFLAGS format */

1166                         add_hyphen = false;
1167 #ifdef ADDFIX5060758
1168                         /* Check if MAKEFLAGS value begins with multiple
1169                          * hyphen characters, and remove all duplicates.
1170                          * Usually it happens when the next command is
1171                          * used: $(MAKE) -$(MAKEFLAGS)
1172                          * This is a workaround for BugID 5060758.
1173                          */
1174                         while (*cp) {
1175                                 if (*cp != (int) hyphen_char) {
1176                                         break;
1177                                 }
1178                                 cp++;
1179                                 if (*cp == (int) hyphen_char) {
1180                                         /* There are two hyphens. Skip one */
1181                                         cp_orig = cp;
1182                                         cp++;
1183                                 }
1184                                 if (!(*cp)) {
1185                                         /* There are hyphens only. Skip all */
1186                                         cp_orig = cp;
1187                                         break;
1188                                 }
1189                         }
1190 #endif
1191                 } else {

1193                         /* Old MAKEFLAGS format */

1195                         add_hyphen = true;
1196                 }
1197         }

1199         /* Find the number of arguments in MAKEFLAGS */
1200         while (cp && *cp) {
1201                 /* Skip white spaces */
1202                 while (cp && *cp && isspace(*cp)) {
1203                         cp++;
1204                 }
1205                 if (cp && *cp) {
1206                         /* Increment arg count */
1207                         mf_argc++;
1208                         /* Go to next white space */
1209                         while (cp && *cp && !isspace(*cp)) {
1210                                 if(*cp == (int) backslash_char) {
1211                                         cp++;
1212                                 }
1213                                 cp++;
1214                         }
1215                 }
1216         }
1217         /* Allocate memory for the new MAKEFLAGS argv */
1218         mf_argv = (char **) malloc((mf_argc + 1) * sizeof(char *));
1219         mf_argv[0] = (char *)"MAKEFLAGS";
1243         mf_argv[0] = (char *)NOCATGETS("MAKEFLAGS");
1220         /*
```



```
1221          * Convert the MAKEFLAGS string value into a vector of char *,
1222          * similar to argv.
1223          */
1224         cp = cp_orig;
1225         for (i = 1; i < mf_argc; i++) {
1226                 /* Skip white spaces */
1227                 while (cp && *cp && isspace(*cp)) {
1228                         cp++;
1229                 }
1230                 if (cp && *cp) {
1231                         cp_orig = cp;
1232                         /* Go to next white space */
1233                         while (cp && *cp && !isspace(*cp)) {
1234                                 if(*cp == (int) backslash_char) {
1235                                         cp++;
1236                                 }
1237                                 cp++;
1238                         }
1239                         tmp_char = *cp;
1240                         *cp = (int) nul_char;
1241                         if (add_hyphen) {
1242                                 mf_argv[i] = getmem(2 + strlen(cp_orig));
1243                                 mf_argv[i][0] = '\0';
1244                                 (void) strcat(mf_argv[i], "-");
1245                                 // (void) strcat(mf_argv[i], cp_orig);
1246                                 unquote_str(cp_orig, mf_argv[i]+1);
1247                         } else {
1248                                 mf_argv[i] = getmem(2 + strlen(cp_orig));
1249                                 //mf_argv[i] = strdup(cp_orig);
1250                                 unquote_str(cp_orig, mf_argv[i]);
1251                         }
1252                         *cp = tmp_char;
1253                 }
1254         }
1255         mf_argv[i] = NULL;
1256 }

1258 /*
1259  *      parse_command_option(ch)
1260  *
1261  *      Parse make command line options.
1262  *
1263  *      Return value:
1264  *                              Indicates if any -f -c or -M were seen
1265  *
1266  *      Parameters:
1267  *              ch              The character to parse
1268  *
1269  *      Static variables used:
1270  *              dmake_group_specified   Set for make -g
1271  *              dmake_max_jobs_specified        Set for make -j
1272  *              dmake_mode_specified    Set for make -m
1273  *              dmake_add_mode_specified        Set for make -x
1274  *              dmake_compat_mode_specified     Set for make -x SUN_MAKE_COMPAT_
1275  *              dmake_output_mode_specified     Set for make -x DMAKE_OUTPUT_MOD
1276  *              dmake_odir_specified    Set for make -o
1277  *              dmake_rcfile_specified  Set for make -c
1278  *              env_wins                Set for make -e
1279  *              ignore_default_mk       Set for make -r
1280  *              trace_status            Set for make -p
1281  *
1282  *      Global variables used:
1283  *              .make.state path & name set for make -K
1284  *              continue_after_error    Set for make -k
1285  *              debug_level             Set for make -d
1286  *              do_not_exec_rule        Set for make -n
```

```
 *              filter_stderr           Set for make -X
 *              ignore_errors_all       Set for make -i
 *              no_parallel             Set for make -R
 *              quest                   Set for make -q
 *              read_trace_level        Set for make -D
 *              report_dependencies     Set for make -P
 *              silent_all              Set for make -s
 *              touch                   Set for make -t
 */
static int
parse_command_option(register char ch)
{
        static int              invert_next = 0;
        int                     invert_this = invert_next;

        invert_next = 0;
        switch (ch) {
        case '-':                       /* Ignore "--" */
                return 0;
        case '~':                       /* Invert next option */
                invert_next = 1;
                return 0;
        case 'B':                       /* Obsolete */
                return 0;
        case 'b':                       /* Obsolete */
                return 0;
        case 'c':                       /* Read alternative dmakerc file */
                if (invert_this) {
                        dmake_rcfile_specified = false;
                } else {
                        dmake_rcfile_specified = true;
                }
                return 2;
        case 'D':                       /* Show lines read */
                if (invert_this) {
                        read_trace_level--;
                } else {
                        read_trace_level++;
                }
                return 0;
        case 'd':                       /* Debug flag */
                if (invert_this) {
                        debug_level--;
                } else {
                        debug_level++;
                }
                return 0;
        case 'e':                       /* Environment override flag */
                if (invert_this) {
                        env_wins = false;
                } else {
                        env_wins = true;
                }
                return 0;
        case 'f':                       /* Read alternative makefile(s) */
                return 1;
        case 'g':                       /* Use alternative DMake group */
                if (invert_this) {
                        dmake_group_specified = false;
                } else {
                        dmake_group_specified = true;
                }
                return 4;
        case 'i':                       /* Ignore errors */
                if (invert_this) {
                        ignore_errors_all = false;
```



```
                } else {
                        ignore_errors_all = true;
                }
                return 0;
        case 'j':                       /* Use alternative DMake max jobs */
                if (invert_this) {
                        dmake_max_jobs_specified = false;
                } else {
                        dmake_mode_type = parallel_mode;
                        no_parallel = false;
                        dmake_max_jobs_specified = true;
                }
                return 8;
        case 'K':                       /* Read alternative .make.state */
                return 256;
        case 'k':                       /* Keep making even after errors */
                if (invert_this) {
                        continue_after_error = false;
                } else {
                        continue_after_error = true;
                        continue_after_error_ever_seen = true;
                }
                return 0;
        case 'M':                       /* Read alternative make.machines file
                if (invert_this) {
                        pmake_machinesfile_specified = false;
                } else {
                        pmake_machinesfile_specified = true;
                        dmake_mode_type = parallel_mode;
                        no_parallel = false;
                }
                return 16;
        case 'm':                       /* Use alternative DMake build mode */
                if (invert_this) {
                        dmake_mode_specified = false;
                } else {
                        dmake_mode_specified = true;
                }
                return 32;
        case 'x':                       /* Use alternative DMake mode */
                if (invert_this) {
                        dmake_add_mode_specified = false;
                } else {
                        dmake_add_mode_specified = true;
                }
                return 1024;
        case 'N':                       /* Reverse -n */
                if (invert_this) {
                        do_not_exec_rule = true;
                } else {
                        do_not_exec_rule = false;
                }
                return 0;
        case 'n':                       /* Print, not exec commands */
                if (invert_this) {
                        do_not_exec_rule = false;
                } else {
                        do_not_exec_rule = true;
                }
                return 0;
        case 'O':                       /* Integrate with maketool, obsolete */
                return 0;
        case 'o':                       /* Use alternative dmake output dir */
                if (invert_this) {
                        dmake_odir_specified = false;
                } else {
```

```
1419                    dmake_odir_specified = true;
1420            }
1421            return 512;
1422    case 'P':                       /* Print for selected targets */
1423            if (invert_this) {
1424                    report_dependencies_level--;
1425            } else {
1426                    report_dependencies_level++;
1427            }
1428            return 0;
1429    case 'p':                       /* Print description */
1430            if (invert_this) {
1431                    trace_status = false;
1432                    do_not_exec_rule = false;
1433            } else {
1434                    trace_status = true;
1435                    do_not_exec_rule = true;
1436            }
1437            return 0;
1438    case 'q':                       /* Question flag */
1439            if (invert_this) {
1440                    quest = false;
1441            } else {
1442                    quest = true;
1443            }
1444            return 0;
1445    case 'R':                       /* Don't run in parallel */
1446            if (invert_this) {
1447                    pmake_cap_r_specified = false;
1448                    no_parallel = false;
1449            } else {
1450                    pmake_cap_r_specified = true;
1451                    dmake_mode_type = serial_mode;
1452                    no_parallel = true;
1453            }
1454            return 0;
1455    case 'r':                       /* Turn off internal rules */
1456            if (invert_this) {
1457                    ignore_default_mk = false;
1458            } else {
1459                    ignore_default_mk = true;
1460            }
1461            return 0;
1462    case 'S':                       /* Reverse -k */
1463            if (invert_this) {
1464                    continue_after_error = true;
1465            } else {
1466                    continue_after_error = false;
1467                    stop_after_error_ever_seen = true;
1468            }
1469            return 0;
1470    case 's':                       /* Silent flag */
1471            if (invert_this) {
1472                    silent_all = false;
1473            } else {
1474                    silent_all = true;
1475            }
1476            return 0;
1477    case 'T':                       /* Print target list */
1478            if (invert_this) {
1479                    list_all_targets = false;
1480                    do_not_exec_rule = false;
1481            } else {
1482                    list_all_targets = true;
1483                    do_not_exec_rule = true;
1484            }
```



```
1485            return 0;
1486    case 't':                       /* Touch flag */
1487            if (invert_this) {
1488                    touch = false;
1489            } else {
1490                    touch = true;
1491            }
1492            return 0;
1493    case 'u':                       /* Unconditional flag */
1494            if (invert_this) {
1495                    build_unconditional = false;
1496            } else {
1497                    build_unconditional = true;
1498            }
1499            return 0;
1500    case 'V':                       /* SVR4 mode */
1501            svr4 = true;
1502            return 0;
1503    case 'v':                       /* Version flag */
1504            if (invert_this) {
1505            } else {
1506                    fprintf(stdout, "dmake: %s\n", verstring);
1530                    fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
1507                    exit_status = 0;
1508                    exit(0);
1509            }
1510            return 0;
1511    case 'w':                       /* Unconditional flag */
1512            if (invert_this) {
1513                    report_cwd = false;
1514            } else {
1515                    report_cwd = true;
1516            }
1517            return 0;
1518 #if 0
1519    case 'X':                       /* Filter stdout */
1520            if (invert_this) {
1521                    filter_stderr = false;
1522            } else {
1523                    filter_stderr = true;
1524            }
1525            return 0;
1526 #endif
1527    default:
1528            break;
1529    }
1530    return 0;
1531 }

1533 /*
1534  *      setup_for_projectdir()
1535  *
1536  *      Read the PROJECTDIR variable, if defined, and set the sccs path
1537  *
1538  *      Parameters:
1539  *
1540  *      Global variables used:
1541  *              sccs_dir_path   Set to point to SCCS dir to use
1542  */
1543 static void
1544 setup_for_projectdir(void)
1545 {
1546 static char     path[MAXPATHLEN];
1547 char            cwdpath[MAXPATHLEN];
1548 uid_t uid;
1549 int   done=0;
```

```
1551         /* Check if we should use PROJECTDIR when reading the SCCS dir. */
1552         sccs_dir_path = getenv("PROJECTDIR");
1576         sccs_dir_path = getenv(NOCATGETS("PROJECTDIR"));
1553         if ((sccs_dir_path != NULL) &&
1554             (sccs_dir_path[0] != (int) slash_char)) {
1555                 struct passwd *pwent;

1557                 {
1558                    uid = getuid();
1559                    pwent = getpwuid(uid);
1560                    if (pwent == NULL) {
1561                       fatal(gettext("Bogus USERID "));
1585                       fatal(catgets(catd, 1, 188, "Bogus USERID "));
1562                    }
1563                    if ((pwent = getpwnam(sccs_dir_path)) == NULL) {
1564                           /*empty block : it'll go & check cwd  */
1565                    }
1566                    else {
1567                      (void) sprintf(path, "%s/src", pwent->pw_dir);
1591                      (void) sprintf(path, NOCATGETS("%s/src"), pwent->pw_dir);
1568                      if (access(path, F_OK) == 0) {
1569                           sccs_dir_path = path;
1570                           done = 1;
1571                      } else {
1572                           (void) sprintf(path, "%s/source", pwent->pw_dir);
1596                           (void) sprintf(path, NOCATGETS("%s/source"), pwent->pw_d
1573                           if (access(path, F_OK) == 0) {
1574                                  sccs_dir_path = path;
1575                                  done = 1;
1576                           }
1577                       }
1578                    }
1579                    if (!done) {
1580                        if (getcwd(cwdpath, MAXPATHLEN - 1 )) {

1582                           (void) sprintf(path, "%s/%s", cwdpath,sccs_dir_path);
1606                           (void) sprintf(path, NOCATGETS("%s/%s"), cwdpath,sccs_dir
1583                           if (access(path, F_OK) == 0) {
1584                                  sccs_dir_path = path;
1585                                  done = 1;
1586                           } else {
1587                                  fatal(gettext("Bogus PROJECTDIR '%s'"), sccs_dir
1611                                  fatal(catgets(catd, 1, 189, "Bogus PROJECTDIR '%
1588                           }
1589                       }
1590                    }
1591                }
1592         }
1593 }
```

_____***unchanged_portion_omitted_***

```
1712 /*
1713  *      read_files_and_state(argc, argv)
1714  *
1715  *      Read the makefiles we care about and the environment
1716  *      Also read the = style command line options
1717  *
1718  *      Parameters:
1719  *              argc            You know what this is
1720  *              argv            You know what this is
1721  *
1722  *      Static variables used:
1723  *              env_wins        make -e, determines if env vars are RO
1724  *              ignore_default_mk make -r, determines if make.rules is read
1725  *              not_auto_depen  dwight
```



```
1726  *
1727  *      Global variables used:
1728  *              default_target_to_build Set to first proper target from file
1729  *              do_not_exec_rule Set to false when makfile is made
1730  *              dot             The Name ".", used to read current dir
1731  *              empty_name      The Name "", use as macro value
1732  *              keep_state      Set if KEEP_STATE is in environment
1733  *              make_state      The Name ".make.state", used to read file
1734  *              makefile_type   Set to type of file being read
1735  *              makeflags       The Name "MAKEFLAGS", used to set macro value
1736  *              not_auto        dwight
1737  *              read_trace_level Checked to se if the reader should trace
1738  *              report_dependencies If -P is on we do not read .make.state
1739  *              trace_reader    Set if reader should trace
1740  *              virtual_root    The Name "VIRTUAL_ROOT", used to check value
1741  */
1742 static void
1743 read_files_and_state(int argc, char **argv)
1744 {
1745         wchar_t                 buffer[1000];
1746         wchar_t                 buffer_posix[1000];
1747         register char           ch;
1748         register char           *cp;
1749         Property                def_make_macro = NULL;
1750         Name                    def_make_name;
1751         Name                    default_makefile;
1752         String_rec              dest;
1753         wchar_t                 destbuffer[STRING_BUFFER_LENGTH];
1754         register int            i;
1755         register int            j;
1756         Name                    keep_state_name;
1757         int                     length;
1758         Name                    Makefile;
1759         register Property       macro;
1760         struct stat             make_state_stat;
1761         Name                    makefile_name;
1762         register int            makefile_next = 0;
1763         register Boolean        makefile_read = false;
1764         String_rec              makeflags_string;
1765         String_rec              makeflags_string_posix;
1766         String_rec *            makeflags_string_current;
1767         Name                    makeflags_value_saved;
1768         register Name           name;
1769         Name                    new_make_value;
1770         Boolean                 save_do_not_exec_rule;
1771         Name                    sdotMakefile;
1772         Name                    sdotmakefile_name;
1773         static wchar_t          state_file_str;
1774         static char             state_file_str_mb[MAXPATHLEN];
1775         static struct _Name     state_filename;
1776         Boolean                 temp;
1777         char                    tmp_char;
1778         wchar_t                 *tmp_wcs_buffer;
1779         register Name           value;
1780         ASCII_Dyn_Array         makeflags_and_macro;
1781         Boolean                 is_xpg4;

1783 /*
1784  *      Remember current mode. It may be changed after reading makefile
1785  *      and we will have to correct MAKEFLAGS variable.
1786  */
1787         is_xpg4 = posix;

1789         MBSTOWCS(wcs_buffer, "KEEP_STATE");
1813         MBSTOWCS(wcs_buffer, NOCATGETS("KEEP_STATE"));
1790         keep_state_name = GETNAME(wcs_buffer, FIND_LENGTH);
```

```
1791          MBSTOWCS(wcs_buffer, "Makefile");
1815          MBSTOWCS(wcs_buffer, NOCATGETS("Makefile"));
1792          Makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1793          MBSTOWCS(wcs_buffer, "makefile");
1817          MBSTOWCS(wcs_buffer, NOCATGETS("makefile"));
1794          makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
1795          MBSTOWCS(wcs_buffer, "s.makefile");
1819          MBSTOWCS(wcs_buffer, NOCATGETS("s.makefile"));
1796          sdotmakefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
1797          MBSTOWCS(wcs_buffer, "s.Makefile");
1821          MBSTOWCS(wcs_buffer, NOCATGETS("s.Makefile"));
1798          sdotMakefile = GETNAME(wcs_buffer, FIND_LENGTH);

1800  /*
1801   *      initialize global dependency entry for .NOT_AUTO
1802   */
1803          not_auto_depen->next = NULL;
1804          not_auto_depen->name = not_auto;
1805          not_auto_depen->automatic = not_auto_depen->stale = false;

1807  /*
1808   *      Read internal definitions and rules.
1809   */
1810          if (read_trace_level > 1) {
1811                  trace_reader = true;
1812          }
1813          if (!ignore_default_mk) {
1814                  if (svr4) {
1815                          MBSTOWCS(wcs_buffer, "svr4.make.rules");
1839                          MBSTOWCS(wcs_buffer, NOCATGETS("svr4.make.rules"));
1816                          default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1817                  } else {
1818                          MBSTOWCS(wcs_buffer, "make.rules");
1842                          MBSTOWCS(wcs_buffer, NOCATGETS("make.rules"));
1819                          default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
1820                  }
1821                  default_makefile->stat.is_file = true;

1823                  (void) read_makefile(default_makefile,
1824                                       true,
1825                                       false,
1826                                       true);
1827          }

1829          /*
1830           * If the user did not redefine the MAKE macro in the
1831           * default makefile (make.rules), then we'd like to
1832           * change the macro value of MAKE to be some form
1833           * of argv[0] for recursive MAKE builds.
1834           */
1835          MBSTOWCS(wcs_buffer, "MAKE");
1859          MBSTOWCS(wcs_buffer, NOCATGETS("MAKE"));
1836          def_make_name = GETNAME(wcs_buffer, wslen(wcs_buffer));
1837          def_make_macro = get_prop(def_make_name->prop, macro_prop);
1838          if ((def_make_macro != NULL) &&
1839              (IS_EQUAL(def_make_macro->body.macro.value->string_mb,
1840                        "make"))) {
1864                        NOCATGETS("make")))) {
1841                  MBSTOWCS(wcs_buffer, argv_zero_string);
1842                  new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
1843                  (void) SETVAR(def_make_name,
1844                                new_make_value,
1845                                false);
1846          }

1848          default_target_to_build = NULL;
```



```
1849          trace_reader = false;

1851  /*
1852   *      Read environment args. Let file args which follow override unless
1853   *      -e option seen. If -e option is not mentioned.
1854   */
1855          read_environment(env_wins);
1856          if (getvar(virtual_root)->hash.length == 0) {
1857                  maybe_append_prop(virtual_root, macro_prop)
1858                    ->body.macro.exported = true;
1859                  MBSTOWCS(wcs_buffer, "/");
1860                  (void) SETVAR(virtual_root,
1861                                GETNAME(wcs_buffer, FIND_LENGTH),
1862                                false);
1863          }

1865  /*
1866   * We now scan mf_argv and argv to see if we need to set
1867   * any of the DMake-added options/variables in MAKEFLAGS.
1868   */

1870          makeflags_and_macro.start = 0;
1871          makeflags_and_macro.size = 0;
1872          enter_argv_values(mf_argc, mf_argv, &makeflags_and_macro);
1873          enter_argv_values(argc, argv, &makeflags_and_macro);

1875  /*
1876   *      Set MFLAGS and MAKEFLAGS
1877   *
1878   *      Before reading makefile we do not know exactly which mode
1879   *      (posix or not) is used. So prepare two MAKEFLAGS strings
1880   *      for both posix and solaris modes because they are different.
1881   */
1882          INIT_STRING_FROM_STACK(makeflags_string, buffer);
1883          INIT_STRING_FROM_STACK(makeflags_string_posix, buffer_posix);
1884          append_char((int) hyphen_char, &makeflags_string);
1885          append_char((int) hyphen_char, &makeflags_string_posix);

1887          switch (read_trace_level) {
1888          case 2:
1889                  append_char('D', &makeflags_string);
1890                  append_char('D', &makeflags_string_posix);
1891          case 1:
1892                  append_char('D', &makeflags_string);
1893                  append_char('D', &makeflags_string_posix);
1894          }
1895          switch (debug_level) {
1896          case 2:
1897                  append_char('d', &makeflags_string);
1898                  append_char('d', &makeflags_string_posix);
1899          case 1:
1900                  append_char('d', &makeflags_string);
1901                  append_char('d', &makeflags_string_posix);
1902          }
1903          if (env_wins) {
1904                  append_char('e', &makeflags_string);
1905                  append_char('e', &makeflags_string_posix);
1906          }
1907          if (ignore_errors_all) {
1908                  append_char('i', &makeflags_string);
1909                  append_char('i', &makeflags_string_posix);
1910          }
1911          if (continue_after_error) {
1912                  if (stop_after_error_ever_seen) {
1913                          append_char('S', &makeflags_string_posix);
1914                          append_char((int) space_char, &makeflags_string_posix);
```

```
1915                            append_char((int) hyphen_char, &makeflags_string_posix);
1916                    }
1917                    append_char('k', &makeflags_string);
1918                    append_char('k', &makeflags_string_posix);
1919            } else {
1920                    if (stop_after_error_ever_seen
1921                        && continue_after_error_ever_seen) {
1922                            append_char('k', &makeflags_string_posix);
1923                            append_char((int) space_char, &makeflags_string_posix);
1924                            append_char((int) hyphen_char, &makeflags_string_posix);
1925                            append_char('S', &makeflags_string_posix);
1926                    }
1927            }
1928            if (do_not_exec_rule) {
1929                    append_char('n', &makeflags_string);
1930                    append_char('n', &makeflags_string_posix);
1931            }
1932            switch (report_dependencies_level) {
1933            case 4:
1934                    append_char('P', &makeflags_string);
1935                    append_char('P', &makeflags_string_posix);
1936            case 3:
1937                    append_char('P', &makeflags_string);
1938                    append_char('P', &makeflags_string_posix);
1939            case 2:
1940                    append_char('P', &makeflags_string);
1941                    append_char('P', &makeflags_string_posix);
1942            case 1:
1943                    append_char('P', &makeflags_string);
1944                    append_char('P', &makeflags_string_posix);
1945            }
1946            if (trace_status) {
1947                    append_char('p', &makeflags_string);
1948                    append_char('p', &makeflags_string_posix);
1949            }
1950            if (quest) {
1951                    append_char('q', &makeflags_string);
1952                    append_char('q', &makeflags_string_posix);
1953            }
1954            if (silent_all) {
1955                    append_char('s', &makeflags_string);
1956                    append_char('s', &makeflags_string_posix);
1957            }
1958            if (touch) {
1959                    append_char('t', &makeflags_string);
1960                    append_char('t', &makeflags_string_posix);
1961            }
1962            if (build_unconditional) {
1963                    append_char('u', &makeflags_string);
1964                    append_char('u', &makeflags_string_posix);
1965            }
1966            if (report_cwd) {
1967                    append_char('w', &makeflags_string);
1968                    append_char('w', &makeflags_string_posix);
1969            }
1970            /* -c dmake_rcfile */
1971            if (dmake_rcfile_specified) {
1972                    MBSTOWCS(wcs_buffer, "DMAKE_RCFILE");
1996                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
1973                    dmake_rcfile = GETNAME(wcs_buffer, FIND_LENGTH);
1974                    append_makeflags_string(dmake_rcfile, &makeflags_string);
1975                    append_makeflags_string(dmake_rcfile, &makeflags_string_posix);
1976            }
1977            /* -g dmake_group */
1978            if (dmake_group_specified) {
1979                    MBSTOWCS(wcs_buffer, "DMAKE_GROUP");
```



```
2003                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
1980                    dmake_group = GETNAME(wcs_buffer, FIND_LENGTH);
1981                    append_makeflags_string(dmake_group, &makeflags_string);
1982                    append_makeflags_string(dmake_group, &makeflags_string_posix);
1983            }
1984            /* -j dmake_max_jobs */
1985            if (dmake_max_jobs_specified) {
1986                    MBSTOWCS(wcs_buffer, "DMAKE_MAX_JOBS");
2010                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
1987                    dmake_max_jobs = GETNAME(wcs_buffer, FIND_LENGTH);
1988                    append_makeflags_string(dmake_max_jobs, &makeflags_string);
1989                    append_makeflags_string(dmake_max_jobs, &makeflags_string_posix)
1990            }
1991            /* -m dmake_mode */
1992            if (dmake_mode_specified) {
1993                    MBSTOWCS(wcs_buffer, "DMAKE_MODE");
2017                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
1994                    dmake_mode = GETNAME(wcs_buffer, FIND_LENGTH);
1995                    append_makeflags_string(dmake_mode, &makeflags_string);
1996                    append_makeflags_string(dmake_mode, &makeflags_string_posix);
1997            }
1998            /* -x dmake_compat_mode */
1999 //         if (dmake_compat_mode_specified) {
2000 //                 MBSTOWCS(wcs_buffer, "SUN_MAKE_COMPAT_MODE");
2024 //                 MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE_COMPAT_MODE"));
2001 //                 dmake_compat_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2002 //                 append_makeflags_string(dmake_compat_mode, &makeflags_string);
2003 //                 append_makeflags_string(dmake_compat_mode, &makeflags_string_pos
2004 //         }
2005            /* -x dmake_output_mode */
2006            if (dmake_output_mode_specified) {
2007                    MBSTOWCS(wcs_buffer, "DMAKE_OUTPUT_MODE");
2031                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
2008                    dmake_output_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2009                    append_makeflags_string(dmake_output_mode, &makeflags_string);
2010                    append_makeflags_string(dmake_output_mode, &makeflags_string_pos
2011            }
2012            /* -o dmake_odir */
2013            if (dmake_odir_specified) {
2014                    MBSTOWCS(wcs_buffer, "DMAKE_ODIR");
2038                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2015                    dmake_odir = GETNAME(wcs_buffer, FIND_LENGTH);
2016                    append_makeflags_string(dmake_odir, &makeflags_string);
2017                    append_makeflags_string(dmake_odir, &makeflags_string_posix);
2018            }
2019            /* -M pmake_machinesfile */
2020            if (pmake_machinesfile_specified) {
2021                    MBSTOWCS(wcs_buffer, "PMAKE_MACHINESFILE");
2045                    MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
2022                    pmake_machinesfile = GETNAME(wcs_buffer, FIND_LENGTH);
2023                    append_makeflags_string(pmake_machinesfile, &makeflags_string);
2024                    append_makeflags_string(pmake_machinesfile, &makeflags_string_po
2025            }
2026            /* -R */
2027            if (pmake_cap_r_specified) {
2028                    append_char((int) space_char, &makeflags_string);
2029                    append_char((int) hyphen_char, &makeflags_string);
2030                    append_char('R', &makeflags_string);
2031                    append_char((int) space_char, &makeflags_string_posix);
2032                    append_char((int) hyphen_char, &makeflags_string_posix);
2033                    append_char('R', &makeflags_string_posix);
2034            }

2036 /*
2037  *      Make sure MAKEFLAGS is exported
2038  */
```

```
2039         maybe_append_prop(makeflags, macro_prop)->
2040           body.macro.exported = true;

2042         if (makeflags_string.buffer.start[1] != (int) nul_char) {
2043                 if (makeflags_string.buffer.start[1] != (int) space_char) {
2044                         MBSTOWCS(wcs_buffer, "MFLAGS");
2068                         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2045                         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2046                                       GETNAME(makeflags_string.buffer.start,
2047                                               FIND_LENGTH),
2048                                       false);
2049                 } else {
2050                         MBSTOWCS(wcs_buffer, "MFLAGS");
2074                         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2051                         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2052                                       GETNAME(makeflags_string.buffer.start + 2,
2053                                               FIND_LENGTH),
2054                                       false);
2055                 }
2056         }

2058 /*
2059  *      Add command line macro to POSIX makeflags_string
2060  */
2061         if (makeflags_and_macro.start) {
2062                 tmp_char = (char) space_char;
2063                 cp = makeflags_and_macro.start;
2064                 do {
2065                         append_char(tmp_char, &makeflags_string_posix);
2066                 } while ( tmp_char = *cp++ );
2067                 retmem_mb(makeflags_and_macro.start);
2068         }

2070 /*
2071  *      Now set the value of MAKEFLAGS macro in accordance
2072  *      with current mode.
2073  */
2074         macro = maybe_append_prop(makeflags, macro_prop);
2075         temp = (Boolean) macro->body.macro.read_only;
2076         macro->body.macro.read_only = false;
2077         if(posix || gnu_style) {
2078                 makeflags_string_current = &makeflags_string_posix;
2079         } else {
2080                 makeflags_string_current = &makeflags_string;
2081         }
2082         if (makeflags_string_current->buffer.start[1] == (int) nul_char) {
2083                 makeflags_value_saved =
2084                         GETNAME( makeflags_string_current->buffer.start + 1
2085                                , FIND_LENGTH
2086                                );
2087         } else {
2088                 if (makeflags_string_current->buffer.start[1] != (int) space_cha
2089                         makeflags_value_saved =
2090                                 GETNAME( makeflags_string_current->buffer.start
2091                                        , FIND_LENGTH
2092                                        );
2093                 } else {
2094                         makeflags_value_saved =
2095                                 GETNAME( makeflags_string_current->buffer.start
2096                                        , FIND_LENGTH
2097                                        );
2098                 }
2099         }
2100         (void) SETVAR( makeflags
2101                      , makeflags_value_saved
2102                      , false
```



```
2103                         );
2104         macro->body.macro.read_only = temp;

2106 /*
2107  *      Read command line "-f" arguments and ignore -c, g, j, K, M, m, O and o a
2108  */
2109         save_do_not_exec_rule = do_not_exec_rule;
2110         do_not_exec_rule = false;
2111         if (read_trace_level > 0) {
2112                 trace_reader = true;
2113         }

2115         for (i = 1; i < argc; i++) {
2116                 if (argv[i] &&
2117                     (argv[i][0] == (int) hyphen_char) &&
2118                     (argv[i][1] == 'f') &&
2119                     (argv[i][2] == (int) nul_char)) {
2120                         argv[i] = NULL;         /* Remove -f */
2121                         if (i >= argc - 1) {
2122                                 fatal(gettext("No filename argument after -f fla
2146                                 fatal(catgets(catd, 1, 190, "No filename argumen
2123                         }
2124                         MBSTOWCS(wcs_buffer, argv[++i]);
2125                         primary_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2126                         (void) read_makefile(primary_makefile, true, true, true)
2127                         argv[i] = NULL;         /* Remove filename */
2128                         makefile_read = true;
2129                 } else if (argv[i] &&
2130                            (argv[i][0] == (int) hyphen_char) &&
2131                            (argv[i][1] == 'c' ||
2132                             argv[i][1] == 'g' ||
2133                             argv[i][1] == 'j' ||
2134                             argv[i][1] == 'K' ||
2135                             argv[i][1] == 'M' ||
2136                             argv[i][1] == 'm' ||
2137                             argv[i][1] == 'O' ||
2138                             argv[i][1] == 'o') &&
2139                            (argv[i][2] == (int) nul_char)) {
2140                         argv[i] = NULL;
2141                         argv[++i] = NULL;
2142                 }
2143         }

2145 /*
2146  *      If no command line "-f" args then look for "makefile", and then for
2147  *      "Makefile" if "makefile" isn't found.
2148  */
2149         if (!makefile_read) {
2150                 (void) read_dir(dot,
2151                                 (wchar_t *) NULL,
2152                                 (Property) NULL,
2153                                 (wchar_t *) NULL);
2154             if (!posix) {
2155                 if (makefile_name->stat.is_file) {
2156                         if (Makefile->stat.is_file) {
2157                                 warning(gettext("Both 'makefile' and 'Makefile'
2181                                 warning(catgets(catd, 1, 310, "Both 'makefile' a
2158                         }
2159                         primary_makefile = makefile_name;
2160                         makefile_read = read_makefile(makefile_name,
2161                                                       false,
2162                                                       false,
2163                                                       true);
2164                 }
2165                 if (!makefile_read &&
2166                     Makefile->stat.is_file) {
```

```
2167                        primary_makefile = Makefile;
2168                        makefile_read = read_makefile(Makefile,
2169                                                      false,
2170                                                      false,
2171                                                      true);
2172                }
2173            } else {

2175                enum sccs_stat save_m_has_sccs = NO_SCCS;
2176                enum sccs_stat save_M_has_sccs = NO_SCCS;

2178                if (makefile_name->stat.is_file) {
2179                        if (Makefile->stat.is_file) {
2180                                warning(gettext("Both ‘makefile’ and ‘Makefile’
2204                                warning(catgets(catd, 1, 191, "Both ‘makefile’ a
2181                        }
2182                }
2183                if (makefile_name->stat.is_file) {
2184                        if (makefile_name->stat.has_sccs == NO_SCCS) {
2185                           primary_makefile = makefile_name;
2186                           makefile_read = read_makefile(makefile_name,
2187                                                         false,
2188                                                         false,
2189                                                         true);
2190                        } else {
2191                          save_m_has_sccs = makefile_name->stat.has_sccs;
2192                          makefile_name->stat.has_sccs = NO_SCCS;
2193                          primary_makefile = makefile_name;
2194                          makefile_read = read_makefile(makefile_name,
2195                                                        false,
2196                                                        false,
2197                                                        true);
2198                        }
2199                }
2200                if (!makefile_read &&
2201                    Makefile->stat.is_file) {
2202                        if (Makefile->stat.has_sccs == NO_SCCS) {
2203                           primary_makefile = Makefile;
2204                           makefile_read = read_makefile(Makefile,
2205                                                         false,
2206                                                         false,
2207                                                         true);
2208                        } else {
2209                          save_M_has_sccs = Makefile->stat.has_sccs;
2210                          Makefile->stat.has_sccs = NO_SCCS;
2211                          primary_makefile = Makefile;
2212                          makefile_read = read_makefile(Makefile,
2213                                                        false,
2214                                                        false,
2215                                                        true);
2216                        }
2217                }
2218                if (!makefile_read &&
2219                        makefile_name->stat.is_file) {
2220                           makefile_name->stat.has_sccs = save_m_has_sccs;
2221                           primary_makefile = makefile_name;
2222                           makefile_read = read_makefile(makefile_name,
2223                                                         false,
2224                                                         false,
2225                                                         true);
2226                }
2227                if (!makefile_read &&
2228                    Makefile->stat.is_file) {
2229                           Makefile->stat.has_sccs = save_M_has_sccs;
2230                           primary_makefile = Makefile;
2231                           makefile_read = read_makefile(Makefile,
```



```
2232                                                         false,
2233                                                         false,
2234                                                         true);
2235                }
2236            }
2237        }
2238        do_not_exec_rule = save_do_not_exec_rule;
2239        allrules_read = makefile_read;
2240        trace_reader = false;

2242 /*
2243  *      Now get current value of MAKEFLAGS and compare it with
2244  *      the saved value we set before reading makefile.
2245  *      If they are different then MAKEFLAGS is subsequently set by
2246  *      makefile, just leave it there. Otherwise, if make mode
2247  *      is changed by using .POSIX target in makefile we need
2248  *      to correct MAKEFLAGS value.
2249  */
2250        Name mf_val = getvar(makeflags);
2251        if( (posix != is_xpg4)
2252         && (!strcmp(mf_val->string_mb, makeflags_value_saved->string_mb)))
2253        {
2254                if (makeflags_string_posix.buffer.start[1] == (int) nul_char) {
2255                        (void) SETVAR(makeflags,
2256                                      GETNAME(makeflags_string_posix.buffer.star
2257                                              FIND_LENGTH),
2258                                      false);
2259                } else {
2260                        if (makeflags_string_posix.buffer.start[1] != (int) spac
2261                                (void) SETVAR(makeflags,
2262                                              GETNAME(makeflags_string_posix.buf
2263                                                      FIND_LENGTH),
2264                                              false);
2265                        } else {
2266                                (void) SETVAR(makeflags,
2267                                              GETNAME(makeflags_string_posix.buf
2268                                                      FIND_LENGTH),
2269                                              false);
2270                        }
2271                }
2272        }

2274        if (makeflags_string.free_after_use) {
2275                retmem(makeflags_string.buffer.start);
2276        }
2277        if (makeflags_string_posix.free_after_use) {
2278                retmem(makeflags_string_posix.buffer.start);
2279        }
2280        makeflags_string.buffer.start = NULL;
2281        makeflags_string_posix.buffer.start = NULL;

2283        if (posix) {
2284                /*
2285                 * If the user did not redefine the ARFLAGS macro in the
2286                 * default makefile (make.rules), then we’d like to
2287                 * change the macro value of ARFLAGS to be in accordance
2288                 * with "POSIX" requirements.
2289                 */
2290                MBSTOWCS(wcs_buffer, "ARFLAGS");
2314                MBSTOWCS(wcs_buffer, NOCATGETS("ARFLAGS"));
2291                name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2292                macro = get_prop(name->prop, macro_prop);
2293                if ((macro != NULL) && /* Maybe (macro == NULL) || ? */
2294                    (IS_EQUAL(macro->body.macro.value->string_mb,
2295                              "rv"))) {
2296                        MBSTOWCS(wcs_buffer, "-rv");
```

```
                            NOCATGETS("rv")))) {
                        MBSTOWCS(wcs_buffer, NOCATGETS("-rv"));
                        value = GETNAME(wcs_buffer, wslen(wcs_buffer));
                        (void) SETVAR(name,
                                      value,
                                      false);
                }
        }

        if (!posix && !svr4) {
                set_sgs_support();
        }


/*
 *      Make sure KEEP_STATE is in the environment if KEEP_STATE is on.
 */
        macro = get_prop(keep_state_name->prop, macro_prop);
        if ((macro != NULL) &&
            macro->body.macro.exported) {
                keep_state = true;
        }
        if (keep_state) {
                if (macro == NULL) {
                        macro = maybe_append_prop(keep_state_name,
                                                  macro_prop);
                }
                macro->body.macro.exported = true;
                (void) SETVAR(keep_state_name,
                              empty_name,
                              false);

                /*
                 *      Read state file
                 */

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

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

                if(!stat(make_state->string_mb, &make_state_stat)) {
                   if(!(make_state_stat.st_mode & S_IFREG) ) {
                        /* copy the make_state structure to the other
                        ** and then let make_state point to the new
                        ** one.
                        */
                      memcpy(&state_filename, make_state,sizeof(state_filename))
                      state_filename.string_mb = state_file_str_mb;
                /* Just a kludge to avoid two slashes back to back */
                      if((make_state->hash.length == 1)&&
                                (make_state->string_mb[0] == '/')) {
                         make_state->hash.length = 0;
                         make_state->string_mb[0] = '\0';
                      }
                      sprintf(state_file_str_mb,"%s%s",
                       make_state->string_mb,"/.make.state");
                      sprintf(state_file_str_mb,NOCATGETS("%s%s"),
                       make_state->string_mb,NOCATGETS("/.make.state"));
                      make_state = &state_filename;
```



```
                        /* adjust the length to reflect the appended string */
                      make_state->hash.length += 12;
                   }
                } else { /* the file doesn't exist or no permission */
                   char tmp_path[MAXPATHLEN];
                   char *slashp;

                   if (slashp = strrchr(make_state->string_mb, '/')) {
                      strncpy(tmp_path, make_state->string_mb,
                                (slashp - make_state->string_mb));
                        tmp_path[slashp - make_state->string_mb]=0;
                      if(strlen(tmp_path)) {
                        if(stat(tmp_path, &make_state_stat)) {
                          warning(gettext("directory %s for .KEEP_STATE_FILE doe
                          warning(catgets(catd, 1, 192, "directory %s for .KEEP_
                        }
                        if (access(tmp_path, F_OK) != 0) {
                          warning(gettext("can't access dir %s"),tmp_path);
                          warning(catgets(catd, 1, 193, "can't access dir %s"),t
                        }
                      }
                   }
                }
                if (report_dependencies_level != 1) {
                        Makefile_type   makefile_type_temp = makefile_type;
                        makefile_type = reading_statefile;
                        if (read_trace_level > 1) {
                                trace_reader = true;
                        }
                        (void) read_simple_file(make_state,
                                                false,
                                                false,
                                                false,
                                                false,
                                                false,
                                                true);
                        trace_reader = false;
                        makefile_type = makefile_type_temp;
                }
        }
}

/*
 * Scan the argv for options and "=" type args and make them readonly.
 */
static void
enter_argv_values(int argc, char *argv[], ASCII_Dyn_Array *makeflags_and_macro)
{
        register char           *cp;
        register int            i;
        int                     length;
        register Name           name;
        int                     opt_separator = argc;
        char                    tmp_char;
        wchar_t                 *tmp_wcs_buffer;
        register Name           value;
        Boolean                 append = false;
        Property                macro;
        struct stat             statbuf;


        /* Read argv options and "=" type args and make them readonly. */
        makefile_type = reading_nothing;
        for (i = 1; i < argc; ++i) {
                append = false;
                if (argv[i] == NULL) {
```

```
2423                    continue;
2424            } else if (((argv[i][0] == '-') && (argv[i][1] == '-')) ||
2425                       ((argv[i][0] == (int) ' ') &&
2426                        (argv[i][1] == (int) '-') &&
2427                        (argv[i][2] == (int) ' ') &&
2428                        (argv[i][3] == (int) '-'))) {
2429                    argv[i] = NULL;
2430                    opt_separator = i;
2431                    continue;
2432            } else if ((i < opt_separator) && (argv[i][0] == (int) hyphen_ch
2433                    switch (parse_command_option(argv[i][1])) {
2434                    case 1: /* -f seen */
2435                            ++i;
2436                            continue;
2437                    case 2: /* -c seen */
2438                            if (argv[i+1] == NULL) {
2439                                    fatal(gettext("No dmake rcfile argument
2463                                    fatal(catgets(catd, 1, 194, "No dmake rc
2440                            }
2441                            MBSTOWCS(wcs_buffer, "DMAKE_RCFILE");
2465                            MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2442                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2443                            break;
2444                    case 4: /* -g seen */
2445                            if (argv[i+1] == NULL) {
2446                                    fatal(gettext("No dmake group argument a
2470                                    fatal(catgets(catd, 1, 195, "No dmake gr
2447                            }
2448                            MBSTOWCS(wcs_buffer, "DMAKE_GROUP");
2472                            MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
2449                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2450                            break;
2451                    case 8: /* -j seen */
2452                            if (argv[i+1] == NULL) {
2453                                    fatal(gettext("No dmake max jobs argumen
2477                                    fatal(catgets(catd, 1, 196, "No dmake ma
2454                            }
2455                            MBSTOWCS(wcs_buffer, "DMAKE_MAX_JOBS");
2479                            MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS")
2456                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2457                            break;
2458                    case 16: /* -M seen */
2459                            if (argv[i+1] == NULL) {
2460                                    fatal(gettext("No pmake machinesfile arg
2484                                    fatal(catgets(catd, 1, 323, "No pmake ma
2461                            }
2462                            MBSTOWCS(wcs_buffer, "PMAKE_MACHINESFILE");
2486                            MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFI
2463                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2464                            break;
2465                    case 32: /* -m seen */
2466                            if (argv[i+1] == NULL) {
2467                                    fatal(gettext("No dmake mode argument af
2491                                    fatal(catgets(catd, 1, 197, "No dmake mo
2468                            }
2469                            MBSTOWCS(wcs_buffer, "DMAKE_MODE");
2493                            MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2470                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2471                            break;
2472                    case 256: /* -K seen */
2473                            if (argv[i+1] == NULL) {
2474                                    fatal(gettext("No makestate filename arg
2498                                    fatal(catgets(catd, 1, 288, "No makestat
2475                            }
2476                            MBSTOWCS(wcs_buffer, argv[i+1]);
2477                            make_state = GETNAME(wcs_buffer, FIND_LENGTH);
```



```
2478                            keep_state = true;
2479                            argv[i] = NULL;
2480                            argv[i+1] = NULL;
2481                            continue;
2482                    case 512:       /* -o seen */
2483                            if (argv[i+1] == NULL) {
2484                                    fatal(gettext("No dmake output dir argum
2508                                    fatal(catgets(catd, 1, 312, "No dmake ou
2485                            }
2486                            MBSTOWCS(wcs_buffer, "DMAKE_ODIR");
2510                            MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2487                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2488                            break;
2489                    case 1024: /* -x seen */
2490                            if (argv[i+1] == NULL) {
2491                                    fatal(gettext("No argument after -x flag
2515                                    fatal(catgets(catd, 1, 351, "No argument
2492                            }
2493                            length = strlen( "SUN_MAKE_COMPAT_MODE=");
2494                            if (strncmp(argv[i+1], "SUN_MAKE_COMPAT_MODE=",
2517                            length = strlen( NOCATGETS("SUN_MAKE_COMPAT_MODE
2518                            if (strncmp(argv[i+1], NOCATGETS("SUN_MAKE_COMPA
2495                                    argv[i+1] = &argv[i+1][length];
2496                                    MBSTOWCS(wcs_buffer, "SUN_MAKE_COMPAT_MO
2520                                    MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE
2497                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2498                                    dmake_compat_mode_specified = dmake_add_
2499                                    break;
2500                            }
2501                            length = strlen( "DMAKE_OUTPUT_MODE=");
2502                            if (strncmp(argv[i+1], "DMAKE_OUTPUT_MODE=", len
2525                            length = strlen( NOCATGETS("DMAKE_OUTPUT_MODE=")
2526                            if (strncmp(argv[i+1], NOCATGETS("DMAKE_OUTPUT_M
2503                                    argv[i+1] = &argv[i+1][length];
2504                                    MBSTOWCS(wcs_buffer, "DMAKE_OUTPUT_MODE"
2528                                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OU
2505                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2506                                    dmake_output_mode_specified = dmake_add_
2507                            } else {
2508                                    warning(gettext("Unknown argument '%s' a
2532                                    warning(catgets(catd, 1, 354, "Unknown a
2509                                            argv[i+1]);
2510                                    argv[i] = argv[i + 1] = NULL;
2511                                    continue;
2512                            }
2513                            break;
2514                    default: /* Shouldn't reach here */
2515                            argv[i] = NULL;
2516                            continue;
2517                    }
2518                    argv[i] = NULL;
2519                    if (i == (argc - 1)) {
2520                            break;
2521                    }
2522                    if ((length = strlen(argv[i+1])) >= MAXPATHLEN) {
2523                            tmp_wcs_buffer = ALLOC_WC(length + 1);
2524                            (void) mbstowcs(tmp_wcs_buffer, argv[i+1], lengt
2525                            value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2526                            retmem(tmp_wcs_buffer);
2527                    } else {
2528                            MBSTOWCS(wcs_buffer, argv[i+1]);
2529                            value = GETNAME(wcs_buffer, FIND_LENGTH);
2530                    }
2531                    argv[i+1] = NULL;
2532            } else if ((cp = strchr(argv[i], (int) equal_char)) != NULL) {
2533 /*
```

```
2534  * Combine all macro in dynamic array
2535  */
2536                         if(*(cp-1) == (int) plus_char)
2537                         {
2538                                 if(isspace(*(cp-2))) {
2539                                         append = true;
2540                                         cp--;
2541                                 }
2542                         }
2543                         if(!append)
2544                                 append_or_replace_macro_in_dyn_array(makeflags_a

2546                         while (isspace(*(cp-1))) {
2547                                 cp--;
2548                         }
2549                         tmp_char = *cp;
2550                         *cp = (int) nul_char;
2551                         MBSTOWCS(wcs_buffer, argv[i]);
2552                         *cp = tmp_char;
2553                         name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2554                         while (*cp != (int) equal_char) {
2555                                 cp++;
2556                         }
2557                         cp++;
2558                         while (isspace(*cp) && (*cp != (int) nul_char)) {
2559                                 cp++;
2560                         }
2561                         if ((length = strlen(cp)) >= MAXPATHLEN) {
2562                                 tmp_wcs_buffer = ALLOC_WC(length + 1);
2563                                 (void) mbstowcs(tmp_wcs_buffer, cp, length + 1);
2564                                 value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2565                                 retmem(tmp_wcs_buffer);
2566                         } else {
2567                                 MBSTOWCS(wcs_buffer, cp);
2568                                 value = GETNAME(wcs_buffer, FIND_LENGTH);
2569                         }
2570                         argv[i] = NULL;
2571                 } else {
2572                         /* Illegal MAKEFLAGS argument */
2573                         continue;
2574                 }
2575                 if(append) {
2576                         setvar_append(name, value);
2577                         append = false;
2578                 } else {
2579                         macro = maybe_append_prop(name, macro_prop);
2580                         macro->body.macro.exported = true;
2581                         SETVAR(name, value, false)->body.macro.read_only = true;
2582                 }
2583         }
2584 }

2586 /*
2587  * Append the DMake option and value to the MAKEFLAGS string.
2588  */
2589 static void
2590 append_makeflags_string(Name name, register String makeflags_string)
2591 {
2592         const char      *option;

2594         if (strcmp(name->string_mb, "DMAKE_GROUP") == 0) {
2595                 option = " -g ";
2596         } else if (strcmp(name->string_mb, "DMAKE_MAX_JOBS") == 0) {
2597                 option = " -j ";
2598         } else if (strcmp(name->string_mb, "DMAKE_MODE") == 0) {
2599                 option = " -m ";
```



```
2600         } else if (strcmp(name->string_mb, "DMAKE_ODIR") == 0) {
2601                 option = " -o ";
2602         } else if (strcmp(name->string_mb, "DMAKE_RCFILE") == 0) {
2603                 option = " -c ";
2604         } else if (strcmp(name->string_mb, "PMAKE_MACHINESFILE") == 0) {
2605                 option = " -M ";
2606         } else if (strcmp(name->string_mb, "DMAKE_OUTPUT_MODE") == 0) {
2607                 option = " -x DMAKE_OUTPUT_MODE=";
2608         } else if (strcmp(name->string_mb, "SUN_MAKE_COMPAT_MODE") == 0) {
2609                 option = " -x SUN_MAKE_COMPAT_MODE=";
2618         if (strcmp(name->string_mb, NOCATGETS("DMAKE_GROUP")) == 0) {
2619                 option = NOCATGETS(" -g ");
2620         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_MAX_JOBS")) == 0) {
2621                 option = NOCATGETS(" -j ");
2622         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_MODE")) == 0) {
2623                 option = NOCATGETS(" -m ");
2624         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_ODIR")) == 0) {
2625                 option = NOCATGETS(" -o ");
2626         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_RCFILE")) == 0) {
2627                 option = NOCATGETS(" -c ");
2628         } else if (strcmp(name->string_mb, NOCATGETS("PMAKE_MACHINESFILE")) == 0
2629                 option = NOCATGETS(" -M ");
2630         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_OUTPUT_MODE")) == 0)
2631                 option = NOCATGETS(" -x DMAKE_OUTPUT_MODE=");
2632         } else if (strcmp(name->string_mb, NOCATGETS("SUN_MAKE_COMPAT_MODE")) ==
2633                 option = NOCATGETS(" -x SUN_MAKE_COMPAT_MODE=");
2610         } else {
2611                 fatal(gettext("Internal error: name not recognized in append_mak
2635                 fatal(catgets(catd, 1, 289, "Internal error: name not recognized
2612         }
2613         Property prop = maybe_append_prop(name, macro_prop);
2614         if( prop == 0 || prop->body.macro.value == 0 ||
2615             prop->body.macro.value->string_mb == 0 ) {
2616                 return;
2617         }
2618         char mbs_value[MAXPATHLEN + 100];
2619         strcpy(mbs_value, option);
2620         strcat(mbs_value, prop->body.macro.value->string_mb);
2621         MBSTOWCS(wcs_buffer, mbs_value);
2622         append_string(wcs_buffer, makeflags_string, FIND_LENGTH);
2623 }

2625 /*
2626  *      read_environment(read_only)
2627  *
2628  *      This routine reads the process environment when make starts and enters
2629  *      it as make macros. The environment variable SHELL is ignored.
2630  *
2631  *      Parameters:
2632  *              read_only       Should we make env vars read only?
2633  *
2634  *      Global variables used:
2635  *              report_pwd      Set if this make was started by other make
2636  */
2637 static void
2638 read_environment(Boolean read_only)
2639 {
2640         register char           **environment;
2641         int                     length;
2642         wchar_t                 *tmp_wcs_buffer;
2643         Boolean                 alloced_tmp_wcs_buffer = false;
2644         register wchar_t        *name;
2645         register wchar_t        *value;
2646         register Name           macro;
2647         Property                val;
2648         Boolean                 read_only_saved;
```

```
2650         reading_environment = true;
2651         environment = environ;
2652         for (; *environment; environment++) {
2653                 read_only_saved = read_only;
2654                 if ((length = strlen(*environment)) >= MAXPATHLEN) {
2655                         tmp_wcs_buffer = ALLOC_WC(length + 1);
2656                         alloced_tmp_wcs_buffer = true;
2657                         (void) mbstowcs(tmp_wcs_buffer, *environment, length + 1
2658                         name = tmp_wcs_buffer;
2659                 } else {
2660                         MBSTOWCS(wcs_buffer, *environment);
2661                         name = wcs_buffer;
2662                 }
2663                 value = (wchar_t *) wschr(name, (int) equal_char);

2665                 /*
2666                  * Looks like there's a bug in the system, but sometimes
2667                  * you can get blank lines in *environment.
2668                  */
2669                 if (!value) {
2670                         continue;
2671                 }
2672                 MBSTOWCS(wcs_buffer2, "SHELL=");
2696                 MBSTOWCS(wcs_buffer2, NOCATGETS("SHELL="));
2673                 if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2674                         continue;
2675                 }
2676                 MBSTOWCS(wcs_buffer2, "MAKEFLAGS=");
2700                 MBSTOWCS(wcs_buffer2, NOCATGETS("MAKEFLAGS="));
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                 }

2691                 /*
2692                  * We ignore SUNPRO_DEPENDENCIES. This environment variable is
2693                  * set by make and read by cpp which then writes info to
2694                  * .make.dependency.xxx.  When make is invoked by another make
2695                  * (recursive make), we don't want to read this because then
2696                  * the child make will end up writing to the parent
2697                  * directory's .make.state and clobbering them.
2698                  */
2699                 MBSTOWCS(wcs_buffer2, "SUNPRO_DEPENDENCIES");
2723                 MBSTOWCS(wcs_buffer2, NOCATGETS("SUNPRO_DEPENDENCIES"));
2700                 if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2701                         continue;
2702                 }

2704                 macro = GETNAME(name, value - name);
2705                 maybe_append_prop(macro, macro_prop)->body.macro.exported =
2706                   true;
2707                 if ((value == NULL) || ((value + 1)[0] == (int) nul_char)) {
2708                         val = setvar_daemon(macro,
2709                                             (Name) NULL,
2710                                             false, no_daemon, false, debug_level
2711                 } else {
```



```
2712                         val = setvar_daemon(macro,
2713                                             GETNAME(value + 1, FIND_LENGTH),
2714                                             false, no_daemon, false, debug_level
2715                 }
2716                 val->body.macro.read_only = read_only_saved;
2717                 if (alloced_tmp_wcs_buffer) {
2718                         retmem(tmp_wcs_buffer);
2719                         alloced_tmp_wcs_buffer = false;
2720                 }
2721         }
2722         reading_environment = false;
2723 }
```

*_____unchanged_portion_omitted_*

```
2757 /*
2758  *      make_targets(argc, argv, parallel_flag)
2759  *
2760  *      Call doname on the specified targets
2761  *
2762  *      Parameters:
2763  *              argc            You know what this is
2764  *              argv            You know what this is
2765  *              parallel_flag   True if building in parallel
2766  *
2767  *      Global variables used:
2768  *              build_failed_seen Used to generated message after failed -k
2769  *              commands_done   Used to generate message "Up to date"
2770  *              default_target_to_build First proper target in makefile
2771  *              init            The Name ".INIT", use to run command
2772  *              parallel        Global parallel building flag
2773  *              quest           make -q, suppresses messages
2774  *              recursion_level Initialized, used for tracing
2775  *              report_dependencies make -P, regroves whole process
2776  */
2777 static void
2778 make_targets(int argc, char **argv, Boolean parallel_flag)
2779 {
2780         int                     i;
2781         char                    *cp;
2782         Doname                  result;
2783         register Boolean        target_to_make_found = false;

2785         (void) doname(init, true, true);
2786         recursion_level = 1;
2787         parallel = parallel_flag;
2788 /*
2789  *      make remaining args
2790  */
2791 /*
2792         if ((report_dependencies_level == 0) && parallel) {
2793  */
2794         if (parallel) {
2795                 /*
2796                  * If building targets in parallel, start all of the
2797                  * remaining args to build in parallel.
2798                  */
2799                 for (i = 1; i < argc; i++) {
2800                         if ((cp = argv[i]) != NULL) {
2801                                 commands_done = false;
2802                                 if ((cp[0] == (int) period_char) &&
2803                                     (cp[1] == (int) slash_char)) {
2804                                         cp += 2;
2805                                 }
2806                                  if((cp[0] == (int) ' ') &&
2807                                     (cp[1] == (int) '-') &&
2808                                     (cp[2] == (int) ' ') &&
```

```
2809                                    (cp[3] == (int) '-')) {
2810                                    argv[i] = NULL;
2811                                        continue;
2812                                }
2813                                MBSTOWCS(wcs_buffer, cp);
2814                                //default_target_to_build = GETNAME(wcs_buffer,
2815                                //                                  FIND_LENGTH);
2816                                default_target_to_build = normalize_name(wcs_buf
2817                                                                         wslen(wcs_buff
2818                                if (default_target_to_build == wait_name) {
2819                                        if (parallel_process_cnt > 0) {
2820                                                finish_running();
2821                                        }
2822                                        continue;
2823                                }
2824                                top_level_target = get_wstring(default_target_to
2825                                /*
2826                                 * If we can't execute the current target in
2827                                 * parallel, hold off the target processing
2828                                 * to preserve the order of the targets as they
2829                                 * in command line.
2830                                 */
2831                                if (!parallel_ok(default_target_to_build, false)
2832                                                && parallel_process_cnt > 0) {
2833                                        finish_running();
2834                                }
2835                                result = doname_check(default_target_to_build,
2836                                                      true,
2837                                                      false,
2838                                                      false);
2839                                gather_recursive_deps();
2840                                if (/* !commands_done && */
2841                                    (result == build_ok) &&
2842                                    !quest &&
2843                                    (report_dependencies_level == 0) /*  &&
2844                                    (exists(default_target_to_build) > file_does
2845                                        if (posix) {
2846                                                if (!commands_done) {
2847                                                        (void) printf(gettext("'
2871                                                        (void) printf(catgets(ca
2848                                                                              default_ta
2849                                                } else {
2850                                                        if (no_action_was_taken)
2851                                                                (void) printf(ge
2875                                                                (void) printf(ca
2852                                                                              de
2853                                                        }
2854                                                }
2855                                        } else {
2856                                                default_target_to_build->stat.ti
2857                                                if (!commands_done &&
2858                                                    (exists(default_target_to_bu
2859                                                        (void) printf(gettext("'
2883                                                        (void) printf(catgets(ca
2860                                                                              default_ta
2861                                                }
2862                                        }
2863                                }
2864                        }
2865                }
2866                /* Now wait for all of the targets to finish running */
2867                finish_running();
2868                //              setjmp(jmpbuffer);
2869
2870        }
2871        for (i = 1; i < argc; i++) {
```



```
2872                if ((cp = argv[i]) != NULL) {
2873                        target_to_make_found = true;
2874                        if ((cp[0] == (int) period_char) &&
2875                            (cp[1] == (int) slash_char)) {
2876                                cp += 2;
2877                        }
2878                                if((cp[0] == (int) ' ') &&
2879                                   (cp[1] == (int) '-') &&
2880                                   (cp[2] == (int) ' ') &&
2881                                   (cp[3] == (int) '-')) {
2882                                   argv[i] = NULL;
2883                                      continue;
2884                                }
2885                        MBSTOWCS(wcs_buffer, cp);
2886                        default_target_to_build = normalize_name(wcs_buffer, wsl
2887                        top_level_target = get_wstring(default_target_to_build->
2888                        report_recursion(default_target_to_build);
2889                        commands_done = false;
2890                        if (parallel) {
2891                                result = (Doname) default_target_to_build->state
2892                        } else {
2893                                result = doname_check(default_target_to_build,
2894                                                      true,
2895                                                      false,
2896                                                      false);
2897                        }
2898                        gather_recursive_deps();
2899                        if (build_failed_seen) {
2900                                build_failed_ever_seen = true;
2901                                warning(gettext("Target '%s' not remade because
2925                                warning(catgets(catd, 1, 200, "Target '%s' not r
2902                                        default_target_to_build->string_mb);
2903                        }
2904                        build_failed_seen = false;
2905                        if (report_dependencies_level > 0) {
2906                                print_dependencies(default_target_to_build,
2907                                                   get_prop(default_target_to_bu
2908                                                            line_prop));
2909                        }
2910                        default_target_to_build->stat.time =
2911                          file_no_time;
2912                        if (default_target_to_build->colon_splits > 0) {
2913                                default_target_to_build->state =
2914                                  build_dont_know;
2915                        }
2916                        if (!parallel &&
2917                            /* !commands_done && */
2918                            (result == build_ok) &&
2919                            !quest &&
2920                            (report_dependencies_level == 0) /*  &&
2921                            (exists(default_target_to_build) > file_doesnt_exist
2922                                if (posix) {
2923                                        if (!commands_done) {
2924                                                (void) printf(gettext("'%s' is u
2948                                                (void) printf(catgets(catd, 1, 2
2925                                                              default_target_to_
2926                                        } else {
2927                                                if (no_action_was_taken) {
2928                                                        (void) printf(gettext("'
2952                                                        (void) printf(catgets(ca
2929                                                                              default_ta
2930                                                }
2931                                        }
2932                                } else {
2933                                        if (!commands_done &&
2934                                            (exists(default_target_to_build) > f
```

```
2935                                            (void) printf(gettext("‘%s’ is u
2959                                            (void) printf(catgets(catd, 1, 2
2936                                                                  default_target_to_
2937                                    }
2938                            }
2939                    }
2940            }
2941    }

2943 /*
2944  *      If no file arguments have been encountered,
2945  *      make the first name encountered that doesnt start with a dot
2946  */
2947    if (!target_to_make_found) {
2948            if (default_target_to_build == NULL) {
2949                    fatal(gettext("No arguments to build"));
2973                    fatal(catgets(catd, 1, 202, "No arguments to build"));
2950            }
2951            commands_done = false;
2952            top_level_target = get_wstring(default_target_to_build->string_m
2953            report_recursion(default_target_to_build);


2956            if (getenv("SPRO_EXPAND_ERRORS")){
2957                    (void) printf("::(%s)\n",
2980            if (getenv(NOCATGETS("SPRO_EXPAND_ERRORS"))){
2981                    (void) printf(NOCATGETS("::(%s)\n"),
2958                                    default_target_to_build->string_mb);
2959            }


2962            result = doname_parallel(default_target_to_build, true, false);
2963            gather_recursive_deps();
2964            if (build_failed_seen) {
2965                    build_failed_ever_seen = true;
2966                    warning(gettext("Target ‘%s’ not remade because of error
2990                    warning(catgets(catd, 1, 203, "Target ‘%s’ not remade be
2967                                    default_target_to_build->string_mb);
2968            }
2969            build_failed_seen = false;
2970            if (report_dependencies_level > 0) {
2971                    print_dependencies(default_target_to_build,
2972                                    get_prop(default_target_to_build->
2973                                                    prop,
2974                                                    line_prop));
2975            }
2976            default_target_to_build->stat.time = file_no_time;
2977            if (default_target_to_build->colon_splits > 0) {
2978                    default_target_to_build->state = build_dont_know;
2979            }
2980            if (/* !commands_done && */
2981                (result == build_ok) &&
2982                !quest &&
2983                (report_dependencies_level == 0) /*  &&
2984                (exists(default_target_to_build) > file_doesnt_exist)  */) {
2985                    if (posix) {
2986                            if (!commands_done) {
2987                                    (void) printf(gettext("‘%s’ is updated.\
3011                                    (void) printf(catgets(catd, 1, 299, "‘%s
2988                                                    default_target_to_build->s
2989                            } else {
2990                                    if (no_action_was_taken) {
2991                                            (void) printf(gettext("‘%s’: no
3015                                            (void) printf(catgets(catd, 1, 3
2992                                                                  default_target_to_
2993                                    }
```



```
2994                            }
2995                    } else {
2996                            if (!commands_done &&
2997                                (exists(default_target_to_build) > file_does
2998                                    (void) printf(gettext("‘%s’ is up to dat
3022                                    (void) printf(catgets(catd, 1, 301, "‘%s
2999                                                          default_target_to_build->s
3000                            }
3001                    }
3002            }
3003    }
3004 }
______unchanged_portion_omitted_

3167 static void
3168 report_dir_enter_leave(Boolean entering)
3169 {
3170            char    rcwd[MAXPATHLEN];
3171 static     char *  mlev = NULL;
3172            char *  make_level_str = NULL;
3173            int     make_level_val = 0;

3175            make_level_str = getenv("MAKELEVEL");
3199            make_level_str = getenv(NOCATGETS("MAKELEVEL"));
3176            if(make_level_str) {
3177                    make_level_val = atoi(make_level_str);
3178            }
3179            if(mlev == NULL) {
3180                    mlev = (char*) malloc(MAXPATHLEN);
3181            }
3182            if(entering) {
3183                    sprintf(mlev, "MAKELEVEL=%d", make_level_val + 1);
3207                    sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val + 1);
3184            } else {
3185                    make_level_val--;
3186                    sprintf(mlev, "MAKELEVEL=%d", make_level_val);
3210                    sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val);
3187            }
3188            putenv(mlev);

3190            if(report_cwd) {
3191                    if(make_level_val <= 0) {
3192                            if(entering) {
3193                                    sprintf( rcwd
3194                                            , gettext("dmake: Entering directory ‘%s’
3218                                            , catgets(catd, 1, 329, "dmake: Entering
3195                                            , get_current_path());
3196                            } else {
3197                                    sprintf( rcwd
3198                                            , gettext("dmake: Leaving directory ‘%s’\
3222                                            , catgets(catd, 1, 331, "dmake: Leaving d
3199                                            , get_current_path());
3200                            }
3201                    } else {
3202                            if(entering) {
3203                                    sprintf( rcwd
3204                                            , gettext("dmake[%d]: Entering directory
3228                                            , catgets(catd, 1, 333, "dmake[%d]: Enter
3205                                            , make_level_val, get_current_path());
3206                            } else {
3207                                    sprintf( rcwd
3208                                            , gettext("dmake[%d]: Leaving directory ‘
3232                                            , catgets(catd, 1, 335, "dmake[%d]: Leavi
3209                                            , make_level_val, get_current_path());
3210                            }
3211                    }
```

```
3212                    printf("%s", rcwd);
3236                    printf(NOCATGETS("%s"), rcwd);
3213            }
3214 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   19494 Wed May 20 12:19:51 2015
new/usr/src/cmd/make/bin/misc.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

/*
 *      misc.cc
 *
 *      This file contains various unclassified routines. Some main groups:
 *              getname
 *              Memory allocation
 *              String handling
 *              Property handling
 *              Error message handling
 *              Make internal state dumping
 *              main routine support
 */

/*
 * Included files
 */
#include <errno.h>
#include <mk/defs.h>
#include <mksh/macro.h>         /* SETVAR() */
#include <mksh/misc.h>          /* enable_interrupt() */
#include <stdarg.h>             /* va_list, va_start(), va_end() */
#include <vroot/report.h>       /* SUNPRO_DEPENDENCIES */
#include <libintl.h>
#endif /* ! codereview */


#define MAXJOBS_ADJUST_RFE4694000

#ifdef MAXJOBS_ADJUST_RFE4694000
extern void job_adjust_fini();
#endif /* MAXJOBS_ADJUST_RFE4694000 */


/*
 * Defined macros
 */
```



```
/*
 * typedefs & structs
 */

/*
 * Static variables
 */

/*
 * File table of contents
 */
static  void            print_rule(register Name target);
static  void            print_target_n_deps(register Name target);

/*************************************
 *
 *      getname
 */

/*************************************
 *
 *      Memory allocation
 */

/*
 *      free_chain()
 *
 *      frees a chain of Name_vector's
 *
 *      Parameters:
 *              ptr             Pointer to the first element in the chain
 *                              to be freed.
 *
 *      Global variables used:
 */
void
free_chain(Name_vector ptr)
{
        if (ptr != NULL) {
                if (ptr->next != NULL) {
                        free_chain(ptr->next);
                }
                free((char *) ptr);
        }
}

/*************************************
 *
 *      String manipulation
 */

/*************************************
 *
 *      Nameblock property handling
 */

/*************************************
 *
 *      Error message handling
 */

/*
 *      fatal(format, args...)
 *
 *      Print a message and die
```

```
 *
 *      Parameters:
 *              format          printf type format string
 *              args            Arguments to match the format
 *
 *      Global variables used:
 *              fatal_in_progress Indicates if this is a recursive call
 *              parallel_process_cnt Do we need to wait for anything?
 *              report_pwd      Should we report the current path?
 */
/*VARARGS*/
void
fatal(const char *message, ...)
{
        va_list args;

        va_start(args, message);
        (void) fflush(stdout);
        (void) fprintf(stderr, gettext("make: Fatal error: "));
        (void) fprintf(stderr, catgets(catd, 1, 263, "make: Fatal error: "));
        (void) vfprintf(stderr, message, args);
        (void) fprintf(stderr, "\n");
        va_end(args);
        if (report_pwd) {
                (void) fprintf(stderr,
                                gettext("Current working directory %s\n"),
                                catgets(catd, 1, 156, "Current working directory
                                get_current_path());
        }
        (void) fflush(stderr);
        if (fatal_in_progress) {
                exit_status = 1;
                exit(1);
        }
        fatal_in_progress = true;
        /* Let all parallel children finish */
        if ((dmake_mode_type == parallel_mode) &&
            (parallel_process_cnt > 0)) {
                (void) fprintf(stderr,
                                gettext("Waiting for %d %s to finish\n"),
                                catgets(catd, 1, 157, "Waiting for %d %s to finis
                                parallel_process_cnt,
                                parallel_process_cnt == 1 ?
                                gettext("job") : gettext("jobs"));
                                catgets(catd, 1, 158, "job") : catgets(catd, 1, 1
                (void) fflush(stderr);
        }

        while (parallel_process_cnt > 0) {
                await_parallel(true);
                finish_children(false);
        }

#if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
        job_adjust_fini();
#endif

        exit_status = 1;
        exit(1);
}

/*
 *      warning(format, args...)
 *
 *      Print a message and continue.
 *
```



```
 *      Parameters:
 *              format          printf type format string
 *              args            Arguments to match the format
 *
 *      Global variables used:
 *              report_pwd      Should we report the current path?
 */
/*VARARGS*/
void
warning(char * message, ...)
{
        va_list args;

        va_start(args, message);
        (void) fflush(stdout);
        (void) fprintf(stderr, gettext("make: Warning: "));
        (void) fprintf(stderr, catgets(catd, 1, 265, "make: Warning: "));
        (void) vfprintf(stderr, message, args);
        (void) fprintf(stderr, "\n");
        va_end(args);
        if (report_pwd) {
                (void) fprintf(stderr,
                                gettext("Current working directory %s\n"),
                                catgets(catd, 1, 161, "Current working directory
                                get_current_path());
        }
        (void) fflush(stderr);
}

/*
 *      time_to_string(time)
 *
 *      Take a numeric time value and produce
 *      a proper string representation.
 *
 *      Return value:
 *                              The string representation of the time
 *
 *      Parameters:
 *              time            The time we need to translate
 *
 *      Global variables used:
 */
char *
time_to_string(const timestruc_t &time)
{
        struct tm               *tm;
        char                    buf[128];

        if (time == file_doesnt_exist) {
                return gettext("File does not exist");
                return catgets(catd, 1, 163, "File does not exist");
        }
        if (time == file_max_time) {
                return gettext("Younger than any file");
                return catgets(catd, 1, 164, "Younger than any file");
        }
        tm = localtime(&time.tv_sec);
        strftime(buf, sizeof (buf), "%c %Z", tm);
        strftime(buf, sizeof (buf), NOCATGETS("%c %Z"), tm);
        buf[127] = (int) nul_char;
        return strdup(buf);
}
```

*unchanged_portion_omitted_*

```
/****************************************
```

```
 542  *
 543  *      main() support
 544  */

 546 /*
 547  *      load_cached_names()
 548  *
 549  *      Load the vector of cached names
 550  *
 551  *      Parameters:
 552  *
 553  *      Global variables used:
 554  *              Many many pointers to Name blocks.
 555  */
 556 void
 557 load_cached_names(void)
 558 {
 559         char            *cp;
 560         Name            dollar;

 562         /* Load the cached_names struct */
 563         MBSTOWCS(wcs_buffer, ".BUILT_LAST_MAKE_RUN");
 465         MBSTOWCS(wcs_buffer, NOCATGETS(".BUILT_LAST_MAKE_RUN"));
 564         built_last_make_run = GETNAME(wcs_buffer, FIND_LENGTH);
 565         MBSTOWCS(wcs_buffer, "@");
 467         MBSTOWCS(wcs_buffer, NOCATGETS("@"));
 566         c_at = GETNAME(wcs_buffer, FIND_LENGTH);
 567         MBSTOWCS(wcs_buffer, " *conditionals* ");
 469         MBSTOWCS(wcs_buffer, NOCATGETS(" *conditionals* "));
 568         conditionals = GETNAME(wcs_buffer, FIND_LENGTH);
 569         /*
 570          * A version of make was released with NSE 1.0 that used
 571          * VERSION-1.1 but this version is identical to VERSION-1.0.
 572          * The version mismatch code makes a special case for this
 573          * situation.  If the version number is changed from 1.0
 574          * it should go to 1.2.
 575          */
 576         MBSTOWCS(wcs_buffer, "VERSION-1.0");
 478         MBSTOWCS(wcs_buffer, NOCATGETS("VERSION-1.0"));
 577         current_make_version = GETNAME(wcs_buffer, FIND_LENGTH);
 578         MBSTOWCS(wcs_buffer, ".SVR4");
 480         MBSTOWCS(wcs_buffer, NOCATGETS(".SVR4"));
 579         svr4_name = GETNAME(wcs_buffer, FIND_LENGTH);
 580         MBSTOWCS(wcs_buffer, ".POSIX");
 482         MBSTOWCS(wcs_buffer, NOCATGETS(".POSIX"));
 581         posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
 582         MBSTOWCS(wcs_buffer, ".DEFAULT");
 484         MBSTOWCS(wcs_buffer, NOCATGETS(".DEFAULT"));
 583         default_rule_name = GETNAME(wcs_buffer, FIND_LENGTH);
 584         MBSTOWCS(wcs_buffer, "$");
 486         MBSTOWCS(wcs_buffer, NOCATGETS("$"));
 585         dollar = GETNAME(wcs_buffer, FIND_LENGTH);
 586         MBSTOWCS(wcs_buffer, ".DONE");
 488         MBSTOWCS(wcs_buffer, NOCATGETS(".DONE"));
 587         done = GETNAME(wcs_buffer, FIND_LENGTH);
 588         MBSTOWCS(wcs_buffer, ".");
 490         MBSTOWCS(wcs_buffer, NOCATGETS("."));
 589         dot = GETNAME(wcs_buffer, FIND_LENGTH);
 590         MBSTOWCS(wcs_buffer, ".KEEP_STATE");
 492         MBSTOWCS(wcs_buffer, NOCATGETS(".KEEP_STATE"));
 591         dot_keep_state = GETNAME(wcs_buffer, FIND_LENGTH);
 592         MBSTOWCS(wcs_buffer, ".KEEP_STATE_FILE");
 494         MBSTOWCS(wcs_buffer, NOCATGETS(".KEEP_STATE_FILE"));
 593         dot_keep_state_file = GETNAME(wcs_buffer, FIND_LENGTH);
 594         MBSTOWCS(wcs_buffer, "");
 496         MBSTOWCS(wcs_buffer, NOCATGETS(""));
```



```
 595         empty_name = GETNAME(wcs_buffer, FIND_LENGTH);
 596         MBSTOWCS(wcs_buffer, " FORCE");
 498         MBSTOWCS(wcs_buffer, NOCATGETS(" FORCE"));
 597         force = GETNAME(wcs_buffer, FIND_LENGTH);
 598         MBSTOWCS(wcs_buffer, "HOST_ARCH");
 500         MBSTOWCS(wcs_buffer, NOCATGETS("HOST_ARCH"));
 599         host_arch = GETNAME(wcs_buffer, FIND_LENGTH);
 600         MBSTOWCS(wcs_buffer, "HOST_MACH");
 502         MBSTOWCS(wcs_buffer, NOCATGETS("HOST_MACH"));
 601         host_mach = GETNAME(wcs_buffer, FIND_LENGTH);
 602         MBSTOWCS(wcs_buffer, ".IGNORE");
 504         MBSTOWCS(wcs_buffer, NOCATGETS(".IGNORE"));
 603         ignore_name = GETNAME(wcs_buffer, FIND_LENGTH);
 604         MBSTOWCS(wcs_buffer, ".INIT");
 506         MBSTOWCS(wcs_buffer, NOCATGETS(".INIT"));
 605         init = GETNAME(wcs_buffer, FIND_LENGTH);
 606         MBSTOWCS(wcs_buffer, ".LOCAL");
 508         MBSTOWCS(wcs_buffer, NOCATGETS(".LOCAL"));
 607         localhost_name = GETNAME(wcs_buffer, FIND_LENGTH);
 608         MBSTOWCS(wcs_buffer, ".make.state");
 510         MBSTOWCS(wcs_buffer, NOCATGETS(".make.state"));
 609         make_state = GETNAME(wcs_buffer, FIND_LENGTH);
 610         MBSTOWCS(wcs_buffer, "MAKEFLAGS");
 512         MBSTOWCS(wcs_buffer, NOCATGETS("MAKEFLAGS"));
 611         makeflags = GETNAME(wcs_buffer, FIND_LENGTH);
 612         MBSTOWCS(wcs_buffer, ".MAKE_VERSION");
 514         MBSTOWCS(wcs_buffer, NOCATGETS(".MAKE_VERSION"));
 613         make_version = GETNAME(wcs_buffer, FIND_LENGTH);
 614         MBSTOWCS(wcs_buffer, ".NO_PARALLEL");
 516         MBSTOWCS(wcs_buffer, NOCATGETS(".NO_PARALLEL"));
 615         no_parallel_name = GETNAME(wcs_buffer, FIND_LENGTH);
 616         MBSTOWCS(wcs_buffer, ".NOT_AUTO");
 518         MBSTOWCS(wcs_buffer, NOCATGETS(".NOT_AUTO"));
 617         not_auto = GETNAME(wcs_buffer, FIND_LENGTH);
 618         MBSTOWCS(wcs_buffer, ".PARALLEL");
 520         MBSTOWCS(wcs_buffer, NOCATGETS(".PARALLEL"));
 619         parallel_name = GETNAME(wcs_buffer, FIND_LENGTH);
 620         MBSTOWCS(wcs_buffer, "PATH");
 522         MBSTOWCS(wcs_buffer, NOCATGETS("PATH"));
 621         path_name = GETNAME(wcs_buffer, FIND_LENGTH);
 622         MBSTOWCS(wcs_buffer, "+");
 524         MBSTOWCS(wcs_buffer, NOCATGETS("+"));
 623         plus = GETNAME(wcs_buffer, FIND_LENGTH);
 624         MBSTOWCS(wcs_buffer, ".PRECIOUS");
 526         MBSTOWCS(wcs_buffer, NOCATGETS(".PRECIOUS"));
 625         precious = GETNAME(wcs_buffer, FIND_LENGTH);
 626         MBSTOWCS(wcs_buffer, "?");
 528         MBSTOWCS(wcs_buffer, NOCATGETS("?"));
 627         query = GETNAME(wcs_buffer, FIND_LENGTH);
 628         MBSTOWCS(wcs_buffer, "^");
 530         MBSTOWCS(wcs_buffer, NOCATGETS("^"));
 629         hat = GETNAME(wcs_buffer, FIND_LENGTH);
 630         MBSTOWCS(wcs_buffer, ".RECURSIVE");
 532         MBSTOWCS(wcs_buffer, NOCATGETS(".RECURSIVE"));
 631         recursive_name = GETNAME(wcs_buffer, FIND_LENGTH);
 632         MBSTOWCS(wcs_buffer, ".SCCS_GET");
 534         MBSTOWCS(wcs_buffer, NOCATGETS(".SCCS_GET"));
 633         sccs_get_name = GETNAME(wcs_buffer, FIND_LENGTH);
 634         MBSTOWCS(wcs_buffer, ".SCCS_GET_POSIX");
 536         MBSTOWCS(wcs_buffer, NOCATGETS(".SCCS_GET_POSIX"));
 635         sccs_get_posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
 636         MBSTOWCS(wcs_buffer, ".GET");
 538         MBSTOWCS(wcs_buffer, NOCATGETS(".GET"));
 637         get_name = GETNAME(wcs_buffer, FIND_LENGTH);
 638         MBSTOWCS(wcs_buffer, ".GET_POSIX");
 540         MBSTOWCS(wcs_buffer, NOCATGETS(".GET_POSIX"));
```

```
639         get_posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
640         MBSTOWCS(wcs_buffer, "SHELL");
542         MBSTOWCS(wcs_buffer, NOCATGETS("SHELL"));
641         shell_name = GETNAME(wcs_buffer, FIND_LENGTH);
642         MBSTOWCS(wcs_buffer, ".SILENT");
544         MBSTOWCS(wcs_buffer, NOCATGETS(".SILENT"));
643         silent_name = GETNAME(wcs_buffer, FIND_LENGTH);
644         MBSTOWCS(wcs_buffer, ".SUFFIXES");
546         MBSTOWCS(wcs_buffer, NOCATGETS(".SUFFIXES"));
645         suffixes_name = GETNAME(wcs_buffer, FIND_LENGTH);
646         MBSTOWCS(wcs_buffer, SUNPRO_DEPENDENCIES);
647         sunpro_dependencies = GETNAME(wcs_buffer, FIND_LENGTH);
648         MBSTOWCS(wcs_buffer, "TARGET_ARCH");
550         MBSTOWCS(wcs_buffer, NOCATGETS("TARGET_ARCH"));
649         target_arch = GETNAME(wcs_buffer, FIND_LENGTH);
650         MBSTOWCS(wcs_buffer, "TARGET_MACH");
552         MBSTOWCS(wcs_buffer, NOCATGETS("TARGET_MACH"));
651         target_mach = GETNAME(wcs_buffer, FIND_LENGTH);
652         MBSTOWCS(wcs_buffer, "VIRTUAL_ROOT");
554         MBSTOWCS(wcs_buffer, NOCATGETS("VIRTUAL_ROOT"));
653         virtual_root = GETNAME(wcs_buffer, FIND_LENGTH);
654         MBSTOWCS(wcs_buffer, "VPATH");
556         MBSTOWCS(wcs_buffer, NOCATGETS("VPATH"));
655         vpath_name = GETNAME(wcs_buffer, FIND_LENGTH);
656         MBSTOWCS(wcs_buffer, ".WAIT");
558         MBSTOWCS(wcs_buffer, NOCATGETS(".WAIT"));
657         wait_name = GETNAME(wcs_buffer, FIND_LENGTH);

659         wait_name->state = build_ok;

661         /* Mark special targets so that the reader treats them properly */
662         svr4_name->special_reader = svr4_special;
663         posix_name->special_reader = posix_special;
664         built_last_make_run->special_reader = built_last_make_run_special;
665         default_rule_name->special_reader = default_special;
666         dot_keep_state->special_reader = keep_state_special;
667         dot_keep_state_file->special_reader = keep_state_file_special;
668         ignore_name->special_reader = ignore_special;
669         make_version->special_reader = make_version_special;
670         no_parallel_name->special_reader = no_parallel_special;
671         parallel_name->special_reader = parallel_special;
672         localhost_name->special_reader = localhost_special;
673         precious->special_reader = precious_special;
674         sccs_get_name->special_reader = sccs_get_special;
675         sccs_get_posix_name->special_reader = sccs_get_posix_special;
676         get_name->special_reader = get_special;
677         get_posix_name->special_reader = get_posix_special;
678         silent_name->special_reader = silent_special;
679         suffixes_name->special_reader = suffixes_special;

681         /* The value of $$ is $ */
682         (void) SETVAR(dollar, dollar, false);
683         dollar->dollar = false;

685         /* Set the value of $(SHELL) */
686         if (posix) {
687           MBSTOWCS(wcs_buffer, "/usr/xpg4/bin/sh");
589           MBSTOWCS(wcs_buffer, NOCATGETS("/usr/xpg4/bin/sh"));
688         } else {
689           MBSTOWCS(wcs_buffer, "/bin/sh");
591           MBSTOWCS(wcs_buffer, NOCATGETS("/bin/sh"));
690         }
691         (void) SETVAR(shell_name, GETNAME(wcs_buffer, FIND_LENGTH), false);

693         /*
694          * Use " FORCE" to simulate a FRC dependency for :: type
```



```
695          * targets with no dependencies.
696          */
697         (void) append_prop(force, line_prop);
698         force->stat.time = file_max_time;

700         /* Make sure VPATH is defined before current dir is read */
701         if ((cp = getenv(vpath_name->string_mb)) != NULL) {
702                 MBSTOWCS(wcs_buffer, cp);
703                 (void) SETVAR(vpath_name,
704                               GETNAME(wcs_buffer, FIND_LENGTH),
705                               false);
706         }

708         /* Check if there is NO PATH variable. If not we construct one. */
709         if (getenv(path_name->string_mb) == NULL) {
710                 vroot_path = NULL;
711                 add_dir_to_path(".", &vroot_path, -1);
712                 add_dir_to_path("/bin", &vroot_path, -1);
713                 add_dir_to_path("/usr/bin", &vroot_path, -1);
613                 add_dir_to_path(NOCATGETS("."), &vroot_path, -1);
614                 add_dir_to_path(NOCATGETS("/bin"), &vroot_path, -1);
615                 add_dir_to_path(NOCATGETS("/usr/bin"), &vroot_path, -1);
714         }
715 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   45629 Wed May 20 12:19:51 2015
new/usr/src/cmd/make/bin/parallel.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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


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

/*
 * Included files
 */
#include <errno.h>              /* errno */
#include <fcntl.h>
#include <mk/defs.h>
#include <mksh/dosys.h>         /* redirect_io() */
#include <mksh/macro.h>         /* expand_value() */
#include <mksh/misc.h>          /* getmem() */
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netdb.h>
#include <libintl.h>
#endif /* ! codereview */



/*
 * Defined macros
 */
#define MAXRULES                100

/*
 * This const should be in avo_dms/include/AvoDmakeCommand.h
 */
```



```
const int local_host_mask = 0x20;


/*
 * typedefs & structs
 */


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


/*
 * File table of contents
 */
static  void            delete_running_struct(Running rp);
static  Boolean         dependency_conflict(Name target);
static  Doname          distribute_process(char **commands, Property line);
static  void            doname_subtree(Name target, Boolean do_get, Boolean impl
static  void            dump_out_file(char *filename, Boolean err);
static  void            finish_doname(Running rp);
static  void            maybe_reread_make_state(void);
static  void            process_next(void);
static  void            reset_conditionals(int cnt, Name *targets, Property *loc
static  pid_t           run_rule_commands(char *host, char **commands);
static  Property        *set_conditionals(int cnt, Name *targets);
static  void            store_conditionals(Running rp);


/*
 *      execute_parallel(line, waitflg)
 *
 *      DMake 2.x:
 *      parallel mode: spawns a parallel process to execute the command group.
 *
 *      Return value:
 *                              The result of the execution
 *
 *      Parameters:
 *              line            The command group to execute
 */
Doname
execute_parallel(Property line, Boolean waitflg, Boolean local)
{
        int                     argcnt;
        int                     cmd_options = 0;
        char                    *commands[MAXRULES + 5];
        char                    *cp;
        Name                    dmake_name;
        Name                    dmake_value;
        int                     ignore;
        Name                    make_machines_name;
        char                    **p;
        Property                prop;
        Doname                  result = build_ok;
        Cmd_line                rule;
        Boolean                 silent_flag;
```

```
128         Name                    target = line->body.line.target;
129         Boolean                 wrote_state_file = false;

131         if ((pmake_max_jobs == 0) &&
132             (dmake_mode_type == parallel_mode)) {
133                 if (local_host[0] == '\0') {
134                         (void) gethostname(local_host, MAXNAMELEN);
135                 }
136                 MBSTOWCS(wcs_buffer, "DMAKE_MAX_JOBS");
 49                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
137                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
138                 if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
139                     ((dmake_value = prop->body.macro.value) != NULL)) {
140                         pmake_max_jobs = atoi(dmake_value->string_mb);
141                         if (pmake_max_jobs <= 0) {
142                                 warning(gettext("DMAKE_MAX_JOBS cannot be less t
143                                 warning(gettext("setting DMAKE_MAX_JOBS to %d.")
 55                                 warning(catgets(catd, 1, 308, "DMAKE_MAX_JOBS ca
 56                                 warning(catgets(catd, 1, 309, "setting DMAKE_MAX
144                                 pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
145                         }
146                 } else {
147                         /*
148                          * For backwards compatibility w/ PMake 1.x, when
149                          * DMake 2.x is being run in parallel mode, DMake
150                          * should parse the PMake startup file
151                          * $(HOME)/.make.machines to get the pmake_max_jobs.
152                          */
153                         MBSTOWCS(wcs_buffer, "PMAKE_MACHINESFILE");
 66                         MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
154                         dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
155                         if (((prop = get_prop(dmake_name->prop, macro_prop)) !=
156                             ((dmake_value = prop->body.macro.value) != NULL)) {
157                                 make_machines_name = dmake_value;
158                         } else {
159                                 make_machines_name = NULL;
160                         }
161                         if ((pmake_max_jobs = read_make_machines(make_machines_n
162                                 pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
163                         }
164                 }
165         }

167         if ((dmake_mode_type == serial_mode) ||
168             ((dmake_mode_type == parallel_mode) && (waitflg))) {
169                 return (execute_serial(line));
170         }

172         {
173                 p = commands;
174         }

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



```
191                 if (rule->command_line->hash.length > 0) {
192                         if (++argcnt == MAXRULES) {
193                                 return build_serial;
194                         }
195                         {
196                                 if (rule->silent && !silent) {
197                                         silent_flag = true;
198                                 }
199                                 if (rule->ignore_error) {
200                                         ignore++;
201                                 }
202                                 /* XXX - need to add support for + prefix */
203                                 if (silent_flag || ignore) {
204                                         *p = getmem((silent_flag ? 1 : 0) +
205                                                     ignore +
206                                                     (strlen(rule->
207                                                             command_line->
208                                                             string_mb)) +
209                                                     1);
210                                         cp = *p++;
211                                         if (silent_flag) {
212                                                 *cp++ = (int) at_char;
213                                         }
214                                         if (ignore) {
215                                                 *cp++ = (int) hyphen_char;
216                                         }
217                                         (void) strcpy(cp, rule->command_line->st
218                                 } else {
219                                         *p++ = rule->command_line->string_mb;
220                                 }
221                         }
222                 }
223         }
224         if ((argcnt == 0) ||
225             (report_dependencies_level > 0)) {
226                 return build_ok;
227         }
228         {
229                 *p = NULL;

231                 Doname res = distribute_process(commands, line);
232                 if (res == build_running) {
233                         parallel_process_cnt++;
234                 }

236                 /*
237                  * Return only those memory that were specially allocated
238                  * for part of commands.
239                  */
240                 for (int i = 0; commands[i] != NULL; i++) {
241                         if ((commands[i][0] == (int) at_char) ||
242                             (commands[i][0] == (int) hyphen_char)) {
243                                 retmem_mb(commands[i]);
244                         }
245                 }
246                 return res;
247         }
248 }
______unchanged_portion_omitted_

295 /*
296  *  M2 adjust mode data and functions
297  *
298  *  m2_init()           - initializes M2 shared semaphore
299  *  m2_acquire_job()    - decrements M2 semaphore counter
```

```
 300  *  m2_release_job()    - increments M2 semaphore counter
 301  *  m2_fini()           - destroys M2 semaphore and shared memory*
 302  *
 303  *  Environment variables:
 304  *      __DMAKE_M2_FILE__
 305  *
 306  *  External functions:
 307  *      ftok(), shmget(), shmat(), shmdt(), shmctl()
 308  *      sem_init(), sem_trywait(), sem_post(), sem_destroy()
 309  *      creat(), close(), unlink()
 310  *      getenv(), putenv()
 311  *
 312  *  Static variables:
 313  *      m2_file         - tmp file name to create ipc key for shared memory
 314  *      m2_shm_id       - shared memory id
 315  *      m2_shm_sem      - shared memory semaphore
 316  */

 318 static char     m2_file[MAXPATHLEN];
 319 static int      m2_shm_id = -1;
 320 static sem_t*   m2_shm_sem = 0;

 322 static int
 323 m2_init() {
 324         char    *var;
 325         key_t   key;

 327         if ((var = getenv("__DMAKE_M2_FILE__")) == 0) {
 240         if ((var = getenv(NOCATGETS("__DMAKE_M2_FILE__"))) == 0) {
 328                 /* compose tmp file name */
 329                 sprintf(m2_file, "%s/dmake.m2.%d.XXXXXX", tmpdir, getpid());
 242                 sprintf(m2_file, NOCATGETS("%s/dmake.m2.%d.XXXXXX"), tmpdir, get

 331                 /* create tmp file */
 332                 int fd = mkstemp(m2_file);
 333                 if (fd < 0) {
 334                         return -1;
 335                 } else {
 336                         close(fd);
 337                 }
 338         } else {
 339                 /* using existing semaphore */
 340                 strcpy(m2_file, var);
 341         }

 343         /* combine IPC key */
 344         if ((key = ftok(m2_file, 38)) == (key_t) -1) {
 345                 return -1;
 346         }

 348         /* create shared memory */
 349         if ((m2_shm_id = shmget(key, sizeof(*m2_shm_sem), 0666 | (var ? 0 : IPC_
 350                 return -1;
 351         }

 353         /* attach shared memory */
 354         if ((m2_shm_sem = (sem_t*) shmat(m2_shm_id, 0, 0666)) == (sem_t*)-1) {
 355                 return -1;
 356         }

 358         /* root process */
 359         if (var == 0) {
 360                 /* initialize semaphore */
 361                 if (sem_init(m2_shm_sem, 1, pmake_max_jobs)) {
 362                         return -1;
 363                 }
```



```
 365                 /* alloc memory for env variable */
 366                 if ((var = (char*) malloc(MAXPATHLEN)) == 0) {
 367                         return -1;
 368                 }

 370                 /* put key to env */
 371                 sprintf(var, "__DMAKE_M2_FILE__=%s", m2_file);
 284                 sprintf(var, NOCATGETS("__DMAKE_M2_FILE__=%s"), m2_file);
 372                 if (putenv(var)) {
 373                         return -1;
 374                 }
 375         }
 376         return 0;
 377 }
______unchanged_portion_omitted_

 464 /*
 465  *  void job_adjust_error()
 466  *
 467  *  Description:
 468  *      Prints warning message, cleans up job adjust data, and disables job adju
 469  *
 470  *  Environment:
 471  *      DMAKE_ADJUST_MAX_JOBS
 472  *
 473  *  External functions:
 474  *      putenv()
 475  *
 476  *  Static variables:
 477  *      job_adjust_mode Current job adjust mode
 478  */
 479 static void
 480 job_adjust_error() {
 481         if (job_adjust_mode != ADJUST_NONE) {
 482                 /* cleanup internals */
 483                 job_adjust_fini();

 485                 /* warning message for the user */
 486                 warning(gettext("Encountered max jobs auto adjustment error - di
 399                 warning(catgets(catd, 1, 339, "Encountered max jobs auto adjustm

 488                 /* switch off job adjustment for the children */
 489                 putenv(strdup("DMAKE_ADJUST_MAX_JOBS=NO"));
 402                 putenv(strdup(NOCATGETS("DMAKE_ADJUST_MAX_JOBS=NO")));

 491                 /* and for this dmake */
 492                 job_adjust_mode = ADJUST_NONE;
 493         }
 494 }

 496 /*
 497  *  void job_adjust_init()
 498  *
 499  *  Description:
 500  *      Parses DMAKE_ADJUST_MAX_JOBS env variable
 501  *      and performs appropriate initializations.
 502  *
 503  *  Environment:
 504  *      DMAKE_ADJUST_MAX_JOBS
 505  *        DMAKE_ADJUST_MAX_JOBS == "NO" - no adjustment
 506  *        DMAKE_ADJUST_MAX_JOBS == "M2" - M2 adjust mode
 507  *        other                         - M1 adjust mode
 508  *
 509  *  External functions:
 510  *      getenv()
```

```
 511  *
 512  *  Static variables:
 513  *      job_adjust_mode Current job adjust mode
 514  */
 515 static void
 516 job_adjust_init() {
 517         if (job_adjust_mode == ADJUST_UNKNOWN) {
 518                 /* default mode */
 519                 job_adjust_mode = ADJUST_M1;

 521                 /* determine adjust mode */
 522                 if (char *var = getenv("DMAKE_ADJUST_MAX_JOBS")) {
 523                         if (strcasecmp(var, "NO") == 0) {
 435                 if (char *var = getenv(NOCATGETS("DMAKE_ADJUST_MAX_JOBS"))) {
 436                         if (strcasecmp(var, NOCATGETS("NO")) == 0) {
 524                                 job_adjust_mode = ADJUST_NONE;
 525                         } else if (strcasecmp(var, "M2") == 0) {
 438                         } else if (strcasecmp(var, NOCATGETS("M2")) == 0) {
 526                                 job_adjust_mode = ADJUST_M2;
 527                         }
 528                 }

 530                 /* M2 specific initialization */
 531                 if (job_adjust_mode == ADJUST_M2) {
 532                         if (m2_init()) {
 533                                 job_adjust_error();
 534                         }
 535                 }
 536         }
 537 }

 539 #endif /* MAXJOBS_ADJUST_RFE4694000 */

 541 /*
 542  *      distribute_process(char **commands, Property line)
 543  *
 544  *      Parameters:
 545  *              commands        argv vector of commands to execute
 546  *
 547  *      Return value:
 548  *                              The result of the execution
 549  *
 550  *      Static variables used:
 551  *              process_running Set to the pid of the process set running
 552  * #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
 553  *              job_adjust_mode Current job adjust mode
 554  * #endif
 555  */
 556 static Doname
 557 distribute_process(char **commands, Property line)
 558 {
 559         static unsigned file_number = 0;
 560         wchar_t         string[MAXPATHLEN];
 561         char            mbstring[MAXPATHLEN];
 562         int             filed;
 563         int             res;
 564         int             tmp_index;
 565         char            *tmp_index_str_ptr;

 567 #if !defined (TEAMWARE_MAKE_CMN) || !defined (MAXJOBS_ADJUST_RFE4694000)
 568         while (parallel_process_cnt >= pmake_max_jobs) {
 569                 await_parallel(false);
 570                 finish_children(true);
 571         }
 572 #else /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
 573         /* initialize adjust mode, if not initialized */
```



```
 574         if (job_adjust_mode == ADJUST_UNKNOWN) {
 575                 job_adjust_init();
 576         }

 578         /* actions depend on adjust mode */
 579         switch (job_adjust_mode) {
 580         case ADJUST_M1:
 581                 while (parallel_process_cnt >= adjust_pmake_max_jobs (pmake_max_
 582                         await_parallel(false);
 583                         finish_children(true);
 584                 }
 585                 break;
 586         case ADJUST_M2:
 587                 if ((res = m2_acquire_job()) == 0) {
 588                         if (parallel_process_cnt > 0) {
 589                                 await_parallel(false);
 590                                 finish_children(true);

 592                                 if ((res = m2_acquire_job()) == 0) {
 593                                         return build_serial;
 594                                 }
 595                         } else {
 596                                 return build_serial;
 597                         }
 598                 }
 599                 if (res < 0) {
 600                         /* job adjustment error */
 601                         job_adjust_error();

 603                         /* no adjustment */
 604                         while (parallel_process_cnt >= pmake_max_jobs) {
 605                                 await_parallel(false);
 606                                 finish_children(true);
 607                         }
 608                 }
 609                 break;
 610         default:
 611                 while (parallel_process_cnt >= pmake_max_jobs) {
 612                         await_parallel(false);
 613                         finish_children(true);
 614                 }
 615         }
 616 #endif /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
 617         setvar_envvar();
 618         /*
 619          * Tell the user what DMake is doing.
 620          */
 621         if (!silent && output_mode != txt2_mode) {
 622                 /*
 623                  * Print local_host --> x job(s).
 624                  */
 625                 (void) fprintf(stdout,
 626                                 gettext("%s --> %d %s\n"),
 539                                 catgets(catd, 1, 325, "%s --> %d %s\n"),
 627                                 local_host,
 628                                 parallel_process_cnt + 1,
 629                                 (parallel_process_cnt == 0) ? gettext("job") : ge
 542                                 (parallel_process_cnt == 0) ? catgets(catd, 1, 12

 631                 /* Print command line(s). */
 632                 tmp_index = 0;
 633                 while (commands[tmp_index] != NULL) {
 634                     /* No @ char. */
 635                     /* XXX - need to add [2] when + prefix is added */
 636                     if ((commands[tmp_index][0] != (int) at_char) &&
 637                         (commands[tmp_index][1] != (int) at_char)) {
```

```
638                         tmp_index_str_ptr = commands[tmp_index];
639                         if (*tmp_index_str_ptr == (int) hyphen_char) {
640                                 tmp_index_str_ptr++;
641                         }
642                         (void) fprintf(stdout, "%s\n", tmp_index_str_ptr);
643                     }
644                     tmp_index++;
645                 }
646                 (void) fflush(stdout);
647         }

649         (void) sprintf(mbstring,
650                         "%s/dmake.stdout.%d.%d.XXXXXX",
563                         NOCATGETS("%s/dmake.stdout.%d.%d.XXXXXX"),
651                         tmpdir,
652                         getpid(),
653                         file_number++);

655         mktemp(mbstring);

657         stdout_file = strdup(mbstring);
658         stderr_file = NULL;

660         if (!out_err_same) {
661                 (void) sprintf(mbstring,
662                                 "%s/dmake.stderr.%d.%d.XXXXXX",
575                                 NOCATGETS("%s/dmake.stderr.%d.%d.XXXXXX"),
663                                 tmpdir,
664                                 getpid(),
665                                 file_number++);

667                 mktemp(mbstring);

669                 stderr_file = strdup(mbstring);
670         }

672         process_running = run_rule_commands(local_host, commands);

674         return build_running;
675 }
```

unchanged_portion_omitted_

```
767 /*
768  *      process_next()
769  *
770  *      Searches the running list for any targets which can start processing.
771  *      This can be a pending target, a serial target, or a subtree target.
772  *
773  *      Parameters:
774  *
775  *      Static variables used:
776  *              running_tail            The end of the list of running procs
777  *              subtree_conflict        A target which conflicts with a subtree
778  *              subtree_conflict2       The other target which conflicts
779  *
780  *      Global variables used:
781  *              commands_done           True if commands executed
782  *              debug_level             Controls debug output
783  *              parallel_process_cnt    Number of parallel process running
784  *              recursion_level         Indentation for debug output
785  *              running_list            List of running processes
786  */
787 static void
788 process_next(void)
789 {
790         Running         rp;
```



```
791         Running         *rp_prev;
792         Property        line;
793         Chain           target_group;
794         Dependency      dep;
795         Boolean         quiescent = true;
796         Running         *subtree_target;
797         Boolean         saved_commands_done;
798         Property        *conditionals;

800         subtree_target = NULL;
801         subtree_conflict = NULL;
802         subtree_conflict2 = NULL;
803         /*
804          * If nothing currently running, build a serial target, if any.
805          */
806 start_loop_1:
807         for (rp_prev = &running_list, rp = running_list;
808              rp != NULL && parallel_process_cnt == 0;
809              rp = rp->next) {
810                 if (rp->state == build_serial) {
811                         *rp_prev = rp->next;
812                         if (rp->next == NULL) {
813                                 running_tail = rp_prev;
814                         }
815                         recursion_level = rp->recursion_level;
816                         rp->target->state = build_pending;
817                         (void) doname_check(rp->target,
818                                             rp->do_get,
819                                             rp->implicit,
820                                             false);
821                         quiescent = false;
822                         delete_running_struct(rp);
823                         goto start_loop_1;
824                 } else {
825                         rp_prev = &rp->next;
826                 }
827         }
828         /*
829          * Find a target to build.  The target must be pending, have all
830          * its dependencies built, and not be in a target group with a target
831          * currently building.
832          */
833 start_loop_2:
834         for (rp_prev = &running_list, rp = running_list;
835              rp != NULL;
836              rp = rp->next) {
837                 if (!(rp->state == build_pending ||
838                       rp->state == build_subtree)) {
839                         quiescent = false;
840                         rp_prev = &rp->next;
841                 } else if (rp->state == build_pending) {
842                         line = get_prop(rp->target->prop, line_prop);
843                         for (dep = line->body.line.dependencies;
844                              dep != NULL;
845                              dep = dep->next) {
846                                 if (dep->name->state == build_running ||
847                                     dep->name->state == build_pending ||
848                                     dep->name->state == build_serial) {
849                                         break;
850                                 }
851                         }
852                         if (dep == NULL) {
853                                 for (target_group = line->body.line.target_group
854                                      target_group != NULL;
855                                      target_group = target_group->next) {
856                                         if (is_running(target_group->name)) {
```

```
857                                             break;
858                                     }
859                             }
860                             if (target_group == NULL) {
861                                     *rp_prev = rp->next;
862                                     if (rp->next == NULL) {
863                                             running_tail = rp_prev;
864                                     }
865                                     recursion_level = rp->recursion_level;
866                                     rp->target->state = rp->redo ?
867                                       build_dont_know : build_pending;
868                                     saved_commands_done = commands_done;
869                                     conditionals =
870                                             set_conditionals
871                                                 (rp->conditional_cnt,
872                                                  rp->conditional_targets);
873                                     rp->target->dont_activate_cond_values =
874                                     if ((doname_check(rp->target,
875                                                       rp->do_get,
876                                                       rp->implicit,
877                                                       rp->target->has_target
878                                         build_running) &&
879                                         !commands_done) {
880                                             commands_done =
881                                               saved_commands_done;
882                                     }
883                                     rp->target->dont_activate_cond_values =
884                                     reset_conditionals
885                                             (rp->conditional_cnt,
886                                              rp->conditional_targets,
887                                              conditionals);
888                                     quiescent = false;
889                                     delete_running_struct(rp);
890                                     goto start_loop_2;
891                             } else {
892                                     rp_prev = &rp->next;
893                             }
894                     } else {
895                             rp_prev = &rp->next;
896                     }
897             } else {
898                     rp_prev = &rp->next;
899             }
900         }
901         /*
902          * If nothing has been found to build and there exists a subtree
903          * target with no dependency conflicts, build it.
904          */
905         if (quiescent) {
906 start_loop_3:
907                 for (rp_prev = &running_list, rp = running_list;
908                      rp != NULL;
909                      rp = rp->next) {
910                         if (rp->state == build_subtree) {
911                                 if (!dependency_conflict(rp->target)) {
912                                         *rp_prev = rp->next;
913                                         if (rp->next == NULL) {
914                                                 running_tail = rp_prev;
915                                         }
916                                         recursion_level = rp->recursion_level;
917                                         doname_subtree(rp->target,
918                                                        rp->do_get,
919                                                        rp->implicit);
920                                         quiescent = false;
921                                         delete_running_struct(rp);
922                                         goto start_loop_3;
```



```
923                                 } else {
924                                         subtree_target = rp_prev;
925                                         rp_prev = &rp->next;
926                                 }
927                         } else {
928                                 rp_prev = &rp->next;
929                         }
930                 }
931         }
932         /*
933          * If still nothing found to build, we either have a deadlock
934          * or a subtree with a dependency conflict with something waiting
935          * to build.
936          */
937         if (quiescent) {
938                 if (subtree_target == NULL) {
939                         fatal(gettext("Internal error: deadlock detected in proc
852                         fatal(catgets(catd, 1, 126, "Internal error: deadlock de
940                 } else {
941                         rp = *subtree_target;
942                         if (debug_level > 0) {
943                                 warning(gettext("Conditional macro conflict enco
856                                 warning(catgets(catd, 1, 127, "Conditional macro
944                                         subtree_conflict2->string_mb,
945                                         rp->target->string_mb,
946                                         subtree_conflict->string_mb);
947                         }
948                         *subtree_target = (*subtree_target)->next;
949                         if (rp->next == NULL) {
950                                 running_tail = subtree_target;
951                         }
952                         recursion_level = rp->recursion_level;
953                         doname_subtree(rp->target, rp->do_get, rp->implicit);
954                         delete_running_struct(rp);
955                 }
956         }
957 }
______unchanged_portion_omitted_

1102 /*
1103  *      await_parallel(waitflg)
1104  *
1105  *      Waits for parallel children to exit and finishes their processing.
1106  *      If waitflg is false, the function returns after update_delay.
1107  *
1108  *      Parameters:
1109  *              waitflg         dwight
1110  */
1111 void
1112 await_parallel(Boolean waitflg)
1113 {
1114         Boolean         nohang;
1115         pid_t           pid;
1116         int             status;
1117         Running         rp;
1118         int             waiterr;

1120         nohang = false;
1121         for ( ; ; ) {
1122                 if (!nohang) {
1123                         (void) alarm((int) update_delay);
1124                 }
1125                 pid = waitpid((pid_t)-1,
1126                               &status,
1127                               nohang ? WNOHANG : 0);
1128                 waiterr = errno;
```

```
1129                 if (!nohang) {
1130                         (void) alarm(0);
1131                 }
1132                 if (pid <= 0) {
1133                         if (waiterr == EINTR) {
1134                                 if (waitflg) {
1135                                         continue;
1136                                 } else {
1137                                         return;
1138                                 }
1139                         } else {
1140                                 return;
1141                         }
1142                 }
1143                 for (rp = running_list;
1144                      (rp != NULL) && (rp->pid != pid);
1145                      rp = rp->next) {
1146                         ;
1147                 }
1148                 if (rp == NULL) {
1149                         fatal(gettext("Internal error: returned child pid not in
1062                         fatal(catgets(catd, 1, 128, "Internal error: returned ch
1150                 } else {
1151                         rp->state = (WIFEXITED(status) && WEXITSTATUS(status) ==
1152                 }
1153                 nohang = true;
1154                 parallel_process_cnt--;

1156 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
1157                 if (job_adjust_mode == ADJUST_M2) {
1158                         if (m2_release_job()) {
1159                                 job_adjust_error();
1160                         }
1161                 }
1162 #endif
1163         }
1164 }

1166 /*
1167  *      finish_children(docheck)
1168  *
1169  *      Finishes the processing for all targets which were running
1170  *      and have now completed.
1171  *
1172  *      Parameters:
1173  *              docheck         Completely check the finished target
1174  *
1175  *      Static variables used:
1176  *              running_tail    The tail of the running list
1177  *
1178  *      Global variables used:
1179  *              continue_after_error  -k flag
1180  *              fatal_in_progress  True if we are finishing up after fatal err
1181  *              running_list    List of running processes
1182  */
1183 void
1184 finish_children(Boolean docheck)
1185 {
1186         int             cmds_length;
1187         Property        line;
1188         Property        line2;
1189         struct stat     out_buf;
1190         Running         rp;
1191         Running         *rp_prev;
1192         Cmd_line        rule;
1193         Boolean         silent_flag;
```



```
1195         for (rp_prev = &running_list, rp = running_list;
1196              rp != NULL;
1197              rp = rp->next) {
1198 bypass_for_loop_inc_4:
1199                 /*
1200                  * If the state is ok or failed, then this target has
1201                  * finished building.
1202                  * In parallel_mode, output the accumulated stdout/stderr.
1203                  * Read the auto dependency stuff, handle a failed build,
1204                  * update the target, then finish the doname process for
1205                  * that target.
1206                  */
1207                 if (rp->state == build_ok || rp->state == build_failed) {
1208                         *rp_prev = rp->next;
1209                         if (rp->next == NULL) {
1210                                 running_tail = rp_prev;
1211                         }
1212                         if ((line2 = rp->command) == NULL) {
1213                                 line2 = get_prop(rp->target->prop, line_prop);
1214                         }


1217                         /*
1218                          * Check if there were any job output
1219                          * from the parallel build.
1220                          */
1221                         if (rp->stdout_file != NULL) {
1222                                 if (stat(rp->stdout_file, &out_buf) < 0) {
1223                                         fatal(gettext("stat of %s failed: %s"),
1136                                         fatal(catgets(catd, 1, 130, "stat of %s
1224                                             rp->stdout_file,
1225                                             errmsg(errno));
1226                                 }

1228                                 if ((line2 != NULL) &&
1229                                     (out_buf.st_size > 0)) {
1230                                         cmds_length = 0;
1231                                         for (rule = line2->body.line.command_use
1232                                                 silent_flag = silent;
1233                                              rule != NULL;
1234                                              rule = rule->next) {
1235                                                 cmds_length += rule->command_lin
1236                                                 silent_flag = BOOLEAN(silent_fla
1237                                         }
1238                                         if (out_buf.st_size != cmds_length || si
1239                                             output_mode == txt2_mode) {
1240                                                 dump_out_file(rp->stdout_file, f
1241                                         }
1242                                 }
1243                                 (void) unlink(rp->stdout_file);
1244                                 retmem_mb(rp->stdout_file);
1245                                 rp->stdout_file = NULL;
1246                         }

1248                         if (!out_err_same && (rp->stderr_file != NULL)) {
1249                                 if (stat(rp->stderr_file, &out_buf) < 0) {
1250                                         fatal(gettext("stat of %s failed: %s"),
1163                                         fatal(catgets(catd, 1, 130, "stat of %s
1251                                             rp->stderr_file,
1252                                             errmsg(errno));
1253                                 }
1254                                 if ((line2 != NULL) &&
1255                                     (out_buf.st_size > 0)) {
1256                                         dump_out_file(rp->stderr_file, true);
1257                                 }
```

```
1258                                (void) unlink(rp->stderr_file);
1259                                retmem_mb(rp->stderr_file);
1260                                rp->stderr_file = NULL;
1261                        }
1262
1263                        check_state(rp->temp_file);
1264                        if (rp->temp_file != NULL) {
1265                                free_name(rp->temp_file);
1266                        }
1267                        rp->temp_file = NULL;
1268                        if (rp->state == build_failed) {
1269                                line = get_prop(rp->target->prop, line_prop);
1270                                if (line != NULL) {
1271                                        line->body.line.command_used = NULL;
1272                                }
1273                                if (continue_after_error ||
1274                                    fatal_in_progress ||
1275                                    !docheck) {
1276                                        warning(gettext("Command failed for targ
1189                                        warning(catgets(catd, 1, 256, "Command f
1277                                                rp->command ? line2->body.line.t
1278                                        build_failed_seen = true;
1279                                } else {
1280                                        /*
1281                                         * XXX??? - DMake needs to exit(),
1282                                         * but shouldn't call fatal().
1283                                         */
1284 #ifdef PRINT_EXIT_STATUS
1285                                        warning("I'm in finish_children. rp->sta
1198                                        warning(NOCATGETS("I'm in finish_childre
1286 #endif
1287
1288                                        fatal(gettext("Command failed for target
1201                                        fatal(catgets(catd, 1, 258, "Command fai
1289                                                rp->command ? line2->body.line.t
1290                                }
1291                        }
1292                        if (!docheck) {
1293                                delete_running_struct(rp);
1294                                rp = *rp_prev;
1295                                if (rp == NULL) {
1296                                        break;
1297                                } else {
1298                                        goto bypass_for_loop_inc_4;
1299                                }
1300                        }
1301                        update_target(get_prop(rp->target->prop, line_prop),
1302                                      rp->state);
1303                        finish_doname(rp);
1304                        delete_running_struct(rp);
1305                        rp = *rp_prev;
1306                        if (rp == NULL) {
1307                                break;
1308                        } else {
1309                                goto bypass_for_loop_inc_4;
1310                        }
1311                } else {
1312                        rp_prev = &rp->next;
1313                }
1314        }
1315 }

1317 /*
1318  *      dump_out_file(filename, err)
1319  *
1320  *      Write the contents of the file to stdout, then unlink the file.
```



```
1321  *
1322  *      Parameters:
1323  *              filename        Name of temp file containing output
1324  *
1325  *      Global variables used:
1326  */
1327 static void
1328 dump_out_file(char *filename, Boolean err)
1329 {
1330         int             chars_read;
1331         char            copybuf[BUFSIZ];
1332         int             fd;
1333         int             out_fd = (err ? 2 : 1);

1335         if ((fd = open(filename, O_RDONLY)) < 0) {
1336                 fatal(gettext("open failed for output file %s: %s"),
1249                 fatal(catgets(catd, 1, 141, "open failed for output file %s: %s"
1337                       filename,
1338                       errmsg(errno));
1339         }
1340         if (!silent && output_mode != txt2_mode) {
1341                 (void) fprintf(err ? stderr : stdout,
1342                                err ?
1343                                 gettext("%s --> Job errors\n") :
1344                                 gettext("%s --> Job output\n"),
1256                                 catgets(catd, 1, 338, "%s --> Job errors\n") :
1257                                 catgets(catd, 1, 259, "%s --> Job output\n"),
1345                               local_host);
1346                 (void) fflush(err ? stderr : stdout);
1347         }
1348         for (chars_read = read(fd, copybuf, BUFSIZ);
1349              chars_read > 0;
1350              chars_read = read(fd, copybuf, BUFSIZ)) {
1351                 /*
1352                  * Read buffers from the source file until end or error.
1353                  */
1354                 if (write(out_fd, copybuf, chars_read) < 0) {
1355                         fatal(gettext("write failed for output file %s: %s"),
1268                         fatal(catgets(catd, 1, 260, "write failed for output fil
1356                               filename,
1357                               errmsg(errno));
1358                 }
1359         }
1360         (void) close(fd);
1361         (void) unlink(filename);
1362 }

1364 /*
1365  *      finish_doname(rp)
1366  *
1367  *      Completes the processing for a target which was left running.
1368  *
1369  *      Parameters:
1370  *              rp              Running list entry for target
1371  *
1372  *      Global variables used:
1373  *              debug_level     Debug flag
1374  *              recursion_level Indentation for debug output
1375  */
1376 static void
1377 finish_doname(Running rp)
1378 {
1379         int             auto_count = rp->auto_count;
1380         Name            *automatics = rp->automatics;
1381         Doname          result = rp->state;
1382         Name            target = rp->target;
```

```
        Name            true_target = rp->true_target;
        Property        *conditionals;

        recursion_level = rp->recursion_level;
        if (result == build_ok) {
                if (true_target == NULL) {
                        (void) printf("Target = %s\n", target->string_mb);
                        (void) printf(" State = %d\n", result);
                        fatal("Internal error: NULL true_target in finish_doname
                        (void) printf(NOCATGETS("Target = %s\n"), target->string
                        (void) printf(NOCATGETS(" State = %d\n"), result);
                        fatal(NOCATGETS("Internal error: NULL true_target in fin
                }
                /* If all went OK, set a nice timestamp */
                if (true_target->stat.time == file_doesnt_exist) {
                        true_target->stat.time = file_max_time;
                }
        }
        target->state = result;
        if (target->is_member) {
                Property member;

                /* Propagate the timestamp from the member file to the member */
                if ((target->stat.time != file_max_time) &&
                    ((member = get_prop(target->prop, member_prop)) != NULL) &&
                    (exists(member->body.member.member) > file_doesnt_exist)) {
                        target->stat.time =
/*
                          exists(member->body.member.member);
 */
                          member->body.member.member->stat.time;
                }
        }
        /*
         * Check if we found any new auto dependencies when we
         * built the target.
         */
        if ((result == build_ok) && check_auto_dependencies(target,
                                                            auto_count,
                                                            automatics)) {
                if (debug_level > 0) {
                        (void) printf(gettext("%*sTarget `%s' acquired new depen
                        (void) printf(catgets(catd, 1, 261, "%*sTarget `%s' acqu
                                      recursion_level,
                                      "",
                                      true_target->string_mb);
                }
                target->rechecking_target = true;
                target->state = build_running;

                /* [tolik, Tue Mar 25 1997]
                 * Fix for bug 4038824:
                 *       command line options set by conditional macros get drop
                 * rp->conditional_cnt and rp->conditional_targets must be copie
                 * to new 'rp' during add_pending(). Set_conditionals() stores
                 * rp->conditional_targets to the global variable 'conditional_t
                 * Add_pending() will use this variable to set up 'rp'.
                 */
                conditionals = set_conditionals(rp->conditional_cnt, rp->conditi
                add_pending(target,
                            recursion_level,
                            rp->do_get,
                            rp->implicit,
                            true);
                reset_conditionals(rp->conditional_cnt, rp->conditional_targets,
        }
```



```
}
______unchanged_portion_omitted_

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

static pid_t
run_rule_commands(char *host, char **commands)
{
        Boolean         always_exec;
        Name            command;
        Boolean         ignore;
        int             length;
        Doname          result;
        Boolean         silent_flag;
        wchar_t         *tmp_wcs_buffer;

        childPid = fork();
        switch (childPid) {
        case -1:        /* Error */
                fatal(gettext("Could not fork child process for dmake job: %s"),
                fatal(catgets(catd, 1, 337, "Could not fork child process for dm
                      errmsg(errno));
                break;
        case 0:         /* Child */
                /* To control the processed targets list is not the child's busi
                running_list = NULL;
                if(out_err_same) {
                        redirect_io(stdout_file, (char*)NULL);
                } else {
                        redirect_io(stdout_file, stderr_file);
                }
                for (commands = commands;
                     (*commands != (char *)NULL);
                     commands++) {
                        silent_flag = silent;
                        ignore = false;
                        always_exec = false;
                        while ((**commands == (int) at_char) ||
                               (**commands == (int) hyphen_char) ||
                               (**commands == (int) plus_char)) {
                                if (**commands == (int) at_char) {
                                        silent_flag = true;
                                }
                                if (**commands == (int) hyphen_char) {
                                        ignore = true;
                                }
                                if (**commands == (int) plus_char) {
                                        always_exec = true;
                                }
                                (*commands)++;
                        }
                        if ((length = strlen(*commands)) >= MAXPATHLEN) {
                                tmp_wcs_buffer = ALLOC_WC(length + 1);
                                (void) mbstowcs(tmp_wcs_buffer, *commands, lengt
                                command = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
                                retmem(tmp_wcs_buffer);
                        } else {
                                MBSTOWCS(wcs_buffer, *commands);
                                command = GETNAME(wcs_buffer, FIND_LENGTH);
                        }
                        if ((command->hash.length > 0) &&
                            !silent_flag) {
                                (void) printf("%s\n", command->string_mb);
```

```
1832                        }
1833                        result = dosys(command,
1834                                       ignore,
1835                                       false,
1836                                       false, /* bugs #4085164 & #4990057 */
1837                                       /* BOOLEAN(silent_flag && ignore), */
1838                                       always_exec,
1839                                       (Name) NULL);
1840                        if (result == build_failed) {
1841                                if (silent_flag) {
1842                                        (void) printf(gettext("The following com
1755                                        (void) printf(catgets(catd, 1, 152, "The
1843                                }
1844                                if (!ignore) {
1845                                        _exit(1);
1846                                }
1847                        }
1848                }
1849                _exit(0);
1850                break;
1851        default:
1852                break;
1853        }
1854        return childPid;
1855 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   11050 Wed May 20 12:19:52 2015
new/usr/src/cmd/make/bin/pmake.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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


/*
 * Included files
 */
#include <arpa/inet.h>
#include <mk/defs.h>
#include <mksh/misc.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>
#include <rpc/rpc.h>            /* host2netname(), netname2host() */
#include <libintl.h>
#endif /* ! codereview */

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */

/*
 * File table of contents
 */
static int              get_max(wchar_t **ms_address, wchar_t *hostname);
static Boolean          pskip_comment(wchar_t **cp_address);
static void             pskip_till_next_word(wchar_t **cp);
static Boolean          pskip_white_space(wchar_t **cp_address);
```



```
/*
 *      read_make_machines(Name make_machines_name)
 *
 *      For backwards compatibility w/ PMake 1.x, when DMake 2.x is
 *      being run in parallel mode, DMake should parse the PMake startup
 *      file $(HOME)/.make.machines to get the PMake max jobs.
 *
 *      Return value:
 *              int of PMake max jobs
 *
 *      Parameters:
 *              make_machines_name      Name of .make.machines file
 *
 */
int
read_make_machines(Name make_machines_name)
{
        wchar_t                 c;
        Boolean                 default_make_machines;
        struct hostent          *hp;
        wchar_t                 local_host[MAX_HOSTNAMELEN + 1];
        char                    local_host_mb[MAX_HOSTNAMELEN + 1] = "";
        int                     local_host_wslen;
        wchar_t                 full_host[MAXNETNAMELEN + 1];
        int                     full_host_wslen = 0;
        char                    *homedir;
        Name                    MAKE_MACHINES;
        struct stat             make_machines_buf;
        FILE                    *make_machines_file;
        wchar_t                 *make_machines_list = NULL;
        char                    *make_machines_list_mb = NULL;
        wchar_t                 make_machines_path[MAXPATHLEN];
        char                    mb_make_machines_path[MAXPATHLEN];
        wchar_t                 *mp;
        wchar_t                 *ms;
        int                     pmake_max_jobs = 0;
        struct utsname          uts_info;


        MBSTOWCS(wcs_buffer, "MAKE_MACHINES");
        MBSTOWCS(wcs_buffer, NOCATGETS("MAKE_MACHINES"));
        MAKE_MACHINES = GETNAME(wcs_buffer, FIND_LENGTH);
        /* Did the user specify a .make.machines file on the command line? */
        default_make_machines = false;
        if (make_machines_name == NULL) {
                /* Try reading the default .make.machines file, in $(HOME). */
                homedir = getenv("HOME");
                homedir = getenv(NOCATGETS("HOME"));
                if ((homedir != NULL) && (strlen(homedir) < (sizeof(mb_make_mach
                        sprintf(mb_make_machines_path,
                         "%s/.make.machines", homedir);
                         NOCATGETS("%s/.make.machines"), homedir);
                        MBSTOWCS(make_machines_path, mb_make_machines_path);
                        make_machines_name = GETNAME(make_machines_path, FIND_LE
                        default_make_machines = true;
                }
                if (make_machines_name == NULL) {
                        /*
                         * No $(HOME)/.make.machines file.
                         * Return 0 for PMake max jobs.
                         */
                        return(0);
                }
        }
```

```
/*
        make_machines_list_mb = getenv(MAKE_MACHINES->string_mb);
 */
        /* Open the .make.machines file. */
        if ((make_machines_file = fopen(make_machines_name->string_mb, "r")) ==
                if (!default_make_machines) {
                        /* Error opening .make.machines file. */
                        fatal(gettext("Open of %s failed: %s"),
                        fatal(catgets(catd, 1, 314, "Open of %s failed: %s"),
                              make_machines_name->string_mb,
                              errmsg(errno));
                } else {
                        /*
                         * No $(HOME)/.make.machines file.
                         * Return 0 for PMake max jobs.
                         */
                        return(0);
                }
        /* Stat the .make.machines file to get the size of the file.  */
        } else if (fstat(fileno(make_machines_file), &make_machines_buf) < 0) {
                /* Error stat'ing .make.machines file. */
                fatal(gettext("Stat of %s failed: %s"),
                fatal(catgets(catd, 1, 315, "Stat of %s failed: %s"),
                      make_machines_name->string_mb,
                      errmsg(errno));
        } else {
                /* Allocate memory for "MAKE_MACHINES=<contents of .m.m>" */
                make_machines_list_mb =
                  (char *) getmem((int) (strlen(MAKE_MACHINES->string_mb) +
                                         2 +
                                         make_machines_buf.st_size));
                sprintf(make_machines_list_mb,
                        "%s=",
                        MAKE_MACHINES->string_mb);
                /* Read in the .make.machines file. */
                if (fread(make_machines_list_mb + strlen(MAKE_MACHINES->string_m
                          sizeof(char),
                          (int) make_machines_buf.st_size,
                          make_machines_file) != make_machines_buf.st_size) {
                        /*
                         * Error reading .make.machines file.
                         * Return 0 for PMake max jobs.
                         */
                        warning(gettext("Unable to read %s"),
                        warning(catgets(catd, 1, 316, "Unable to read %s"),
                                make_machines_name->string_mb);
                        (void) fclose(make_machines_file);
                        retmem_mb((caddr_t) make_machines_list_mb);
                        return(0);
                } else {
                        (void) fclose(make_machines_file);
                        /* putenv "MAKE_MACHINES=<contents of .m.m>" */
                        *(make_machines_list_mb +
                          strlen(MAKE_MACHINES->string_mb) +
                          1 +
                          make_machines_buf.st_size) = (int) nul_char;
                        if (putenv(make_machines_list_mb) != 0) {
                                warning(gettext("Couldn't put contents of %s in
                                warning(catgets(catd, 1, 317, "Couldn't put cont
                                        make_machines_name->string_mb);
                        } else {
                                make_machines_list_mb += strlen(MAKE_MACHINES->s
                                make_machines_list = ALLOC_WC(strlen(make_machin
                                (void) mbstowcs(make_machines_list,
                                                make_machines_list_mb,
                                                (strlen(make_machines_list_mb) +
```



```
                        }
                }
        }

        uname(&uts_info);
        strcpy(local_host_mb, &uts_info.nodename[0]);
        MBSTOWCS(local_host, local_host_mb);
        local_host_wslen = wslen(local_host);

        // There is no getdomainname() function on Solaris.
        // And netname2host() function does not work on Linux.
        // So we have to use different APIs.
        if (host2netname(mbs_buffer, NULL, NULL) &&
            netname2host(mbs_buffer, mbs_buffer2, MAXNETNAMELEN+1)) {
                MBSTOWCS(full_host, mbs_buffer2);
                full_host_wslen = wslen(full_host);
        }

        for (ms = make_machines_list;
             (ms) && (*ms );
             ) {
                /*
                 * Skip white space and comments till you reach
                 * a machine name.
                 */
                pskip_till_next_word(&ms);

                /*
                 * If we haven't reached the end of file, process the
                 * machine name.
                 */
                if (*ms) {
                        /*
                         * If invalid machine name decrement counter
                         * and skip line.
                         */
                        mp = ms;
                        SKIPWORD(ms);
                        c = *ms;
                        *ms++ = '\0'; /* Append null to machine name. */
                        /*
                         * If this was the beginning of a comment
                         * (we overwrote a # sign) and it's not
                         * end of line yet, shift the # sign.
                         */
                        if ((c == '#') && (*ms != '\n') && (*ms)) {
                                *ms = '#';
                        }
                        WCSTOMBS(mbs_buffer, mp);
                        /*
                         * Print "Ignoring unknown host" if:
                         * 1) hostname is longer than MAX_HOSTNAMELEN, or
                         * 2) hostname is unknown
                         */
                        if ((wslen(mp) > MAX_HOSTNAMELEN) ||
                            ((hp = gethostbyname(mbs_buffer)) == NULL)) {
                                warning(gettext("Ignoring unknown host %s"),
                                warning(catgets(catd, 1, 318, "Ignoring unknown
                                        mbs_buffer);
                                SKIPTOEND(ms);
                                /* Increment ptr if not end of file. */
                                if (*ms) {
                                        ms++;
                                }
                        } else {
                                /* Compare current hostname with local_host. */
```

```
252                                 if (wslen(mp) == local_host_wslen &&
253                                     IS_WEQUALN(mp, local_host, local_host_wslen)
254                                         /*
255                                          * Bingo, local_host is in .make.machine
256                                          * Continue reading.
257                                          */
258                                         pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
259                                 /* Compare current hostname with full_host. */
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");
216                                         MBSTOWCS(wcs_buffer, NOCATGETS("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
227                                                         warning(catgets(catd, 1,
291                                                         SKIPTOEND(ms);
292                                                         break;
293                                                 }
294                                         }
295                                 }
296                                 break; /* out of outermost for() loop. */
297                         }
298                 }
299         }
300         retmem(make_machines_list);
301         return(pmake_max_jobs);
302 }
```

_____***unchanged_portion_omitted_***

```
387 static int
388 get_max(wchar_t **ms_address, wchar_t *hostname)
389 {
390         wchar_t         *ms = *ms_address;
391         int             limit = PMAKE_DEF_MAX_JOBS; /* Default setting. */

393         WCSTOMBS(mbs_buffer, hostname);
394         /* Look for '='. */
395         SKIPSPACE(ms);
396         if ((!*ms) || (*ms == '\n') || (*ms != '=')) {
397                 SKIPTOEND(ms);
```



```
398                 warning(gettext("expected '=' after max, ignoring rest of line f
335                 warning(catgets(catd, 1, 319, "expected '=' after max, ignoring
399                                 mbs_buffer);
400                 *ms_address = ms;
401                 return((int) limit);
402         } else {
403                 ms++;
404                 SKIPSPACE(ms);
405                 if ((*ms != '\n') && (*ms != '\0')) {
406                         /* We've found, hopefully, a valid "max" value. */
407                         limit = (int) wcstol(ms, &ms, 10);
408                         if (limit < 1) {
409                                 limit = PMAKE_DEF_MAX_JOBS;
410                                 warning(gettext("max value cannot be less than o
347                                 warning(catgets(catd, 1, 320, "max value cannot
411                         }
412                 } else {
413                         /* No "max" value after "max=". */
414                         warning(gettext("no max value specified for host %s"), m
351                         warning(catgets(catd, 1, 321, "no max value specified fo
415                 }
416                 *ms_address = ms;
417                 return(limit);
418         }
419 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   56687 Wed May 20 12:19:52 2015
new/usr/src/cmd/make/bin/read.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

/*
 *      read.c
 *
 *      This file contains the makefile reader.
 */

/*
 * Included files
 */
#include <alloca.h>             /* alloca() */
#include <errno.h>              /* errno */
#include <fcntl.h>              /* fcntl() */
#include <mk/defs.h>
#include <mksh/macro.h>         /* expand_value(), expand_macro() */
#include <mksh/misc.h>          /* getmem() */
#include <mksh/read.h>          /* get_next_block_fn() */
#include <sys/uio.h>            /* read() */
#include <unistd.h>             /* read(), unlink() */
#include <libintl.h>
#endif /* ! codereview */


/*
 * typedefs & structs
 */

/*
 * Static variables
 */

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

/*
 * File table of contents
 */
static  void            parse_makefile(register Name true_makefile_name, registe
```



```
static  Source          push_macro_value(register Source bp, register wchar_t *b
extern  void            enter_target_groups_and_dependencies(Name_vector target,
extern  Name            normalize_name(register wchar_t *name_string, register i

/*
 *      read_simple_file(makefile_name, chase_path, doname_it,
 *              complain, must_exist, report_file, lock_makefile)
 *
 *      Make the makefile and setup to read it. Actually read it if it is stdio
 *
 *      Return value:
 *                              false if the read failed
 *
 *      Parameters:
 *              makefile_name   Name of the file to read
 *              chase_path      Use the makefile path when opening file
 *              doname_it       Call doname() to build the file first
 *              complain        Print message if doname/open fails
 *              must_exist      Generate fatal if file is missing
 *              report_file     Report file when running -P
 *              lock_makefile   Lock the makefile when reading
 *
 *      Static variables used:
 *
 *      Global variables used:
 *              do_not_exec_rule Is -n on?
 *              file_being_read Set to the name of the new file
 *              line_number     The number of the current makefile line
 *              makefiles_used  A list of all makefiles used, appended to
 */


Boolean
read_simple_file(register Name makefile_name, register Boolean chase_path, regis
{
        static short            max_include_depth;
        register Property       makefile = maybe_append_prop(makefile_name,
                                                             makefile_prop);
        Boolean                 forget_after_parse = false;
        static pathpt           makefile_path;
        register int            n;
        char                    *path;
        register Source         source = ALLOC(Source);
        Property                orig_makefile = makefile;
        Dependency              *dpp;
        Dependency              dp;
        register int            length;
        wchar_t                 *previous_file_being_read = file_being_read;
        int                     previous_line_number = line_number;
        wchar_t                 previous_current_makefile[MAXPATHLEN];
        Makefile_type           save_makefile_type;
        Name                    normalized_makefile_name;
        register wchar_t        *string_start;
        register wchar_t        *string_end;



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

        if (max_include_depth++ >= 40) {
                fatal(gettext("Too many nested include statements"));
                fatal(catgets(catd, 1, 66, "Too many nested include statements")
        }
        if (makefile->body.makefile.contents != NULL) {
                retmem(makefile->body.makefile.contents);
        }
```

```
127         source->inp_buf =
128           source->inp_buf_ptr =
129             source->inp_buf_end = NULL;
130         source->error_converting = false;
131         makefile->body.makefile.contents = NULL;
132         makefile->body.makefile.size = 0;
133         if ((makefile_name->hash.length != 1) ||
134             (wcb[0] != (int) hyphen_char)) {
135                 if ((makefile->body.makefile.contents == NULL) &&
136                     (doname_it)) {
137                         if (makefile_path == NULL) {
138                                 char *pfx = make_install_prefix();
139                                 char *path;
141                                 add_dir_to_path(".",
142                                                 &makefile_path,
143                                                 -1);
145                                 // As regularly installed
146                                 asprintf(&path, "%s/../share/lib/make", pfx);
147                                 add_dir_to_path(path, &makefile_path, -1);
148                                 free(path);
150                                 // Tools build
151                                 asprintf(&path, "%s/../../share/", pfx);
152                                 add_dir_to_path(path, &makefile_path, -1);
153                                 free(path);
154
155                                 add_dir_to_path("/usr/share/lib/make",
 77                                 add_dir_to_path(NOCATGETS("/usr/share/lib/make")
156                                                 &makefile_path,
157                                                 -1);
158                                 add_dir_to_path("/etc/default",
 80                                 add_dir_to_path(NOCATGETS("/etc/default"),
159                                                 &makefile_path,
160                                                 -1);
162                                 free(pfx);
163                         }
164                         save_makefile_type = makefile_type;
165                         makefile_type = reading_nothing;
166                         if (doname(makefile_name, true, false) == build_dont_kno
167                                 /* Try normalized filename */
168                                 string_start=get_wstring(makefile_name->string_m
169                                 for (string_end=string_start+1; *string_end != L
170                                 normalized_makefile_name=normalize_name(string_s
171                                 if ((strcmp(makefile_name->string_mb, normalized
172                                         (doname(normalized_makefile_name, true,
173                                         n = access_vroot(makefile_name->string_m
174                                                  4,
175                                                  chase_path ?
176                                                  makefile_path : NULL,
177                                                  VROOT_DEFAULT);
178                                         if (n == 0) {
179                                                 get_vroot_path((char **) NULL,
180                                                        &path,
181                                                        (char **) NULL);
182                                                 if ((path[0] == (int) period_cha
183                                                     (path[1] == (int) slash_char
184                                                          path += 2;
185                                                 }
186                                                 MBSTOWCS(wcs_buffer, path);
187                                                 makefile_name = GETNAME(wcs_buff
188                                                                 FIND_LENGTH);
189                                         }
190                                 }
```



```
191                                 retmem(string_start);
192                                 /*
193                                  * Commented out: retmem_mb(normalized_makefile_
194                                  * We have to return this memory, but it seems t
195                                  * in dmake or in Sun C++ 5.7 compiler (it works
196                                  * is compiled using Sun C++ 5.6).
197                                  */
198                                 // retmem_mb(normalized_makefile_name->string_mb
199                         }
200                         makefile_type = save_makefile_type;
201                 }
202                 source->string.free_after_use = false;
203                 source->previous = NULL;
204                 source->already_expanded = false;
205                 /* Lock the file for read, but not when -n. */
206                 if (lock_makefile &&
207                     !do_not_exec_rule) {
209                         make_state_lockfile = getmem(strlen(make_state->string_
131                         make_state_lockfile = getmem(strlen(make_state->string_
210                         (void) sprintf(make_state_lockfile,
211                                                 "%s.lock",
133                                                 NOCATGETS("%s.lock"),
212                                                 make_state->string_mb);
213                         (void) file_lock(make_state->string_mb,
214                                          make_state_lockfile,
215                                          (int *) &make_state_locked,
216                                          0);
217                         if(!make_state_locked) {
218                                 printf("-- NO LOCKING for read\n");
140                                 printf(NOCATGETS("-- NO LOCKING for read\n"));
219                                 retmem_mb(make_state_lockfile);
220                                 make_state_lockfile = 0;
221                                 return failed;
222                         }
223                 }
224                 if (makefile->body.makefile.contents == NULL) {
225                         save_makefile_type = makefile_type;
226                         makefile_type = reading_nothing;
227                         if ((doname_it) &&
228                             (doname(makefile_name, true, false) == build_failed)
229                                 if (complain) {
230                                         (void) fprintf(stderr,
231                                                 gettext("make: Couldn't m
153                                                 catgets(catd, 1, 237, "ma
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':
170                                                 fatal(catgets(catd, 1, 68, "Can'
249                                                       makefile_name->string_mb,
250                                                       errmsg(makefile_name->
251                                                              stat.stat_errno));
```

```
252                                     } else {
253                                             warning(gettext("Can't find '%s'
175                                             warning(catgets(catd, 1, 69, "Ca
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':
220                                                     fatal(catgets(catd, 1, 70, "Can'
299                                                           makefile_name->string_mb,
300                                                           errmsg(errno));
301                                             } else {
302                                                     warning(gettext("Can't open '%s'
224                                                     warning(catgets(catd, 1, 71, "Ca
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;

344                             MBSTOWCS(wcs_buffer, "Standard in");
266                             MBSTOWCS(wcs_buffer, NOCATGETS("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");
275                             MBSTOWCS(wcs_buffer, NOCATGETS("standard input"));
354                             file_being_read = (wchar_t *) wsdup(wcs_buffer);
355                             line_number = 0;
356                             while ((n = read(fileno(stdin),
357                                              stdin_text_p,
358                                              length)) > 0) {
359                                     length -= n;
360                                     stdin_text_p += n;
361                                     if (length == 0) {
362                                             p_mb = getmem(length = 1024 +
363                                                           (stdin_buffer_end -
364                                                            stdin_buffer_start));
365                                             (void) strncpy(p_mb,
366                                                            stdin_buffer_start,
367                                                            (stdin_buffer_end -
368                                                             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"),
301                     fatal(catgets(catd, 1, 72, "Error reading standard input
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;

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;
391             source->previous = NULL;
392             source->bytes_left_in_file = 0;
393             source->fd = -1;
394             source->already_expanded = false;
395             source->string.buffer.start =
396               source->string.text.p = ALLOC_WC(num_mb_chars + 1);
397             source->string.buffer.end =
398                 source->string.text.p + num_mb_chars;
399             num_wc_chars = mbstowcs(source->string.text.p,
400                                     stdin_text_p,
401                                     num_mb_chars);
402             if ((int) num_wc_chars >= 0) {
403                     source->string.text.end =
404                       source->string.text.p + num_wc_chars;
405             }
406             (void) retmem_mb(stdin_text_p);
407     }
408     line_number = 1;
409     if (trace_reader) {
410             (void) printf(gettext(">>>>>>>>>>>>>>>> Reading makefile %s\n"),
332             (void) printf(catgets(catd, 1, 73, ">>>>>>>>>>>>>>>> Reading mak
411                           makefile_name->string_mb);
412     }
413     parse_makefile(makefile_name, source);
414     if (trace_reader) {
415             (void) printf(gettext(">>>>>>>>>>>>>>>> End of makefile %s\n"),
337             (void) printf(catgets(catd, 1, 74, ">>>>>>>>>>>>>>>> End of make
416                           makefile_name->string_mb);
417     }
418     if(file_being_read) {
419             retmem(file_being_read);
420     }
421     file_being_read = previous_file_being_read;
422     line_number = previous_line_number;
423     makefile_type = reading_nothing;
424     max_include_depth--;
425     if (make_state_locked) {
426             /* Unlock .make.state. */
427             unlink(make_state_lockfile);
428             make_state_locked = false;
429             retmem_mb(make_state_lockfile);
430     }
431     if (forget_after_parse) {
432             retmem(makefile->body.makefile.contents);
433             makefile->body.makefile.contents = NULL;
434     }
435     retmem_mb((char *)source);
436     return succeeded;
437 }

439 /*
440  *      parse_makefile(true_makefile_name, source)
441  *
```



```
442  *      Strings are read from Sources.
443  *      When macros are found, their values are represented by a
444  *      Source that is pushed on a stack. At end of string
445  *      (that is returned from GET_CHAR() as 0), the block is popped.
446  *
447  *      Parameters:
448  *              true_makefile_name      The name of makefile we are parsing
449  *              source                  The source block to read from
450  *
451  *      Global variables used:
452  *              do_not_exec_rule Is -n on?
453  *              line_number     The number of the current makefile line
454  *              makefile_type   What kind of makefile are we reading?
455  *              empty_name      The Name ""
456  */
457 static void
458 parse_makefile(register Name true_makefile_name, register Source source)
459 {
460 /*
461         char                    mb_buffer[MB_LEN_MAX];
462  */
463         register wchar_t        *source_p;
464         register wchar_t        *source_end;
465         register wchar_t        *string_start;
466         wchar_t                 *string_end;
467         register Boolean        macro_seen_in_string;
468         Boolean                 append;
469         String_rec              name_string;
470         wchar_t                 name_buffer[STRING_BUFFER_LENGTH];
471         register int            distance;
472         register int            paren_count;
473         int                     brace_count;
474         int                     char_number;
475         Cmd_line                command;
476         Cmd_line                command_tail;
477         Name                    macro_value;

479         Name_vector_rec         target;
480         Name_vector_rec         depes;
481         Name_vector_rec         extra_name_vector;
482         Name_vector             current_names;
483         Name_vector             extra_names = &extra_name_vector;
484         Name_vector             nvp;
485         Boolean                 target_group_seen;

487         register Reader_state   state;
488         register Reader_state   on_eoln_state;
489         register Separator      separator;

491         wchar_t                 buffer[4 * STRING_BUFFER_LENGTH];
492         Source                  extrap;

494         Boolean                 save_do_not_exec_rule = do_not_exec_rule;
495         Name                    makefile_name;

497         static Name             sh_name;
498         static Name             shell_name;
499         int                     i;

501         static wchar_t          include_space[10];
502         static wchar_t          include_tab[10];
503         int                     tmp_bytes_left_in_string;
504         Boolean                 tmp_maybe_include = false;
505         int                     emptycount = 0;
506         Boolean                 first_target;
```

```
508         String_rec              include_name;
509         wchar_t                 include_buffer[STRING_BUFFER_LENGTH];

511         target.next = depes.next = NULL;
512         /* Move some values from their struct to register declared locals */
513         CACHE_SOURCE(0);

515  start_new_line:
516         /*
517          * Read whitespace on old line. Leave pointer on first char on
518          * next line.
519          */
520         first_target = true;
521         on_eoln_state = exit_state;
522 /*
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 firs
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         }

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 (; 1; source_p++) {
666                         switch (GET_CHAR()) {
667                         case newline_char:
668                                 goto eoln_1;
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_1:
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         }

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");
656                 MBSTOWCS(include_space, NOCATGETS("include "));
657                 MBSTOWCS(include_tab, NOCATGETS("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;

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;

759                                 default:
760                                         source_p++;
761                                         break;
762                                 }
763                         }

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                                         append_string(string_start,
777                                                       &name_string,
778                                                       source_p - string_start);
779                                         macro_seen_in_string = true;
780                                         GET_NEXT_BLOCK(source);
781                                         string_start = source_p;
782                                         if (source == NULL) {
783                                                 GOTO_STATE(on_eoln_state);
784                                         }
785                                         break;
786
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 = wsrchr(include_name.buf
831                                         if (slash != NULL) {
832                                                 include_name.text.p = slash + 1;
833                                                 append_string(name_start,
834                                                               &include_name,
835                                                               name_length);
```



```
837                                                 name_start = include_name.buffer
838                                                 name_length = include_name.text.
839                                         }
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;
851
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                                 fatal_reader(catgets(catd, 1, 75, "Read of inclu
880                                              makefile_name->string_mb);
881                         }
882                         makefile_type = save_makefile_type;
883                         do_not_exec_rule = save_do_not_exec_rule;
884                         CACHE_SOURCE(0);
885                         goto start_new_line;
886                     } else {
887                         source_p -= 7;
888                     }
889                 } else {
890                     /* Check if the word include was split across 8K boundary. */
891
892                     tmp_bytes_left_in_string = source->string.text.end - source_p;
893                     if (tmp_bytes_left_in_string < 8) {
894                             tmp_maybe_include = false;
895                             if (IS_WEQUALN(source_p,
896                                            include_space,
897                                            tmp_bytes_left_in_string)) {
898                                 tmp_maybe_include = true;
899                             }
900                             if (tmp_maybe_include) {
                                    GET_NEXT_BLOCK(source);
```

```
901                                 tmp_maybe_include = false;
902                                 goto line_evald;
903                         }
904                 }
905             }
906         }

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 firs
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"));
 986                                 fatal_reader(catgets(catd, 1, 76, "Unmatched `('
1065                         }
1066                         if (brace_count > 0) {
1067                                 fatal_reader(gettext("Unmatched `{' on line"));
 989                                 fatal_reader(catgets(catd, 1, 77, "Unmatched `{'
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"));
1042                         fatal_reader(catgets(catd, 1, 78, "Unexpected comment se
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"));
1077                                 fatal_reader(catgets(catd, 1, 79, "Unmatched `)'
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"));
1085                                fatal_reader(catgets(catd, 1, 80, "Unmatched `}'
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
1130                                fatal_reader(catgets(catd, 1, 81, "Extra `:', `:
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"));
1162                                  fatal_reader(catgets(catd, 1, 82, "syntax erro
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"))
1185                                fatal_reader(catgets(catd, 1, 83, "Unexpected co
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                                first_target = false;
1277                                if (extra_names == NULL) {
1278                                        extra_names = (Name_vector)
1279                                          alloca((int)
1280                                                 sizeof (Name_vector_rec));
1281                                }
1282                        }
1283                        source_p++;
1284                        /* Make sure to enter a rule even if the is */
1285                        /* no text here */
1286                        command = command_tail = ALLOC(Cmd_line);
1287                        command->next = NULL;
1288                        command->command_line = empty_name;
```

```
1289                        command->make_refd = false;
1290                        command->ignore_command_dependency = false;
1291                        command->assign = false;
1292                        command->ignore_error = false;
1293                        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 (; 1; 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                                                append_string(string_start,
1312                                                              &name_string,
1313                                                              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_strin
1328                                                                INIT_STRING_FROM
1329                                                                               n
1330                                                        }
1331                                                        append_string(string_sta
1332                                                                      &name_stri
1333                                                                      source_p -
1334                                                        macro_seen_in_string = t
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
1312                                          fatal_reader(catgets(catd, 1, 84, "Must be a s
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");
1339                                                MBSTOWCS(wcs_buffer, NOCATGETS("sh"));
1418                                                sh_name = GETNAME(wcs_buffer, FIND_LENGT
```

```
1419                                    MBSTOWCS(wcs_buffer, "shell");
1341                                    MBSTOWCS(wcs_buffer, NOCATGETS("shell"))
1420                                    shell_name = GETNAME(wcs_buffer, FIND_LE
1421                            }

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                            );

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_LENG
1439                                  (!svr4)) {
1440                                    String_rec      macro_name;
1441                                    wchar_t         buffer[100];

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"));
1410                              fatal_reader(catgets(catd, 1, 85, "syntax erro
1489                            }
1490                            else {
1491                              fatal_reader(gettext("Macro assignment on depe
1413                              fatal_reader(catgets(catd, 1, 86, "Macro assig
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                                            append_char((int)
1547                                                        backslash_char,
```

```
1548                                                    &name_string);
1549                                            append_char(*source_p,
1550                                                    &name_string);
1551                                            string_start = source_p+1;
1552                                            goto macro_value_start;
1553                                    } else {
1554                                        if (source->fd >= 0) {
1555                                            line_number++;
1556                                        }
1557                                    }
1558                                    break;
1559                            case newline_char:
1560                            case numbersign_char:
1561                                    string_start = source_p;
1562                                    goto macro_value_end;
1563                            case tab_char:
1564                            case space_char:
1565                                    break;
1566                            default:
1567                                    string_start = source_p;
1568                                    goto macro_value_start;
1569                            }
1570                    }
1571            macro_value_start:
1572                    /* Find the end of the value */
1573                    for (; 1; source_p++) {
1574                            if (distance != 0) {
1575                                    *source_p = *(source_p + distance);
1576                            }
1577                            switch (GET_CHAR()) {
1578                            case nul_char:
1579                                    if (!macro_seen_in_string) {
1580                                            macro_seen_in_string = true;
1581                                            INIT_STRING_FROM_STACK(name_stri
1582                                                                   name_buff
1583                                    }
1584                                    append_string(string_start,
1585                                                  &name_string,
1586                                                  source_p - string_start);
1587                                    GET_NEXT_BLOCK(source);
1588                                    string_start = source_p;
1589                                    source_p--;
1590                                    if (source == NULL) {
1591                                            GOTO_STATE(on_eoln_state);
1592                                    }
1593                                    break;
1594                            case backslash_char:
1595                                    source_p++;
1596                                    if (distance != 0) {
1597                                            *source_p =
1598                                              *(source_p + distance);
1599                                    }
1600                                    if (*source_p == (int) nul_char) {
1601                                            if (!macro_seen_in_string) {
1602                                                    macro_seen_in_string =
1603                                                      true;
1604                                                    INIT_STRING_FROM_STACK(n
1605                                                                           n
1606                                            }

1608 /*  BID_1225561 */
1609                                            *(source_p - 1) = (int) space_ch
1610                                            append_string(string_start,
1611                                                          &name_string,
1612                                                          source_p -
1613                                                          string_start - 1);
```



```
1614                                            GET_NEXT_BLOCK(source);
1615                                            string_start = source_p;
1616                                            if (source == NULL) {
1617                                                    GOTO_STATE(on_eoln_state
1618                                            }
1619                                            if (distance != 0) {
1620                                                    *source_p =
1621                                                      *(source_p +
1622                                                        distance);
1623                                            }
1624                                            if (*source_p == (int) newline_c
1625                                                    append_char((int) space_
1626                                            } else {
1627                                                    append_char((int) backsl
1628                                            }
1629 /****************/
1630                                    }
1631                                    if (*source_p == (int) newline_char) {
1632                                            source_p--;
1633                                            line_number++;
1634                                            distance++;
1635                                            *source_p = (int) space_char;
1636                                            while ((*(source_p +
1637                                                      distance + 1) ==
1638                                                    (int) tab_char) ||
1639                                                   (*(source_p +
1640                                                      distance + 1) ==
1641                                                    (int) space_char)) {
1642                                                    distance++;
1643                                            }
1644                                    }
1645                                    break;
1646                            case newline_char:
1647                            case numbersign_char:
1648                                    goto macro_value_end;
1649                            }
1650                    }
1651            macro_value_end:
1652                    /* Complete the value in the string */
1653                    if (!macro_seen_in_string) {
1654                            macro_seen_in_string = true;
1655                            INIT_STRING_FROM_STACK(name_string,
1656                                                   name_buffer);
1657                    }
1658                    append_string(string_start,
1659                                  &name_string,
1660                                  source_p - string_start);
1661                    if (name_string.buffer.start != name_string.text.p) {
1662                            macro_value =
1663                              GETNAME(name_string.buffer.start,
1664                                      FIND_LENGTH);
1665                    }
1666                    if (name_string.free_after_use) {
1667                            retmem(name_string.buffer.start);
1668                    }
1669                    for (; distance > 0; distance--) {
1670                            *source_p++ = (int) space_char;
1671                    }
1672                    GOTO_STATE(on_eoln_state);
1673            }
1674    }

1676 /*************************************************************
1677  *      enter dependencies state
1678  */
1679  case enter_dependencies_state:
```

```
1680  enter_dependencies_label:
1681 /* Expects pointer on first non whitespace char after last dependency. (On */
1682 /* next line.) We end up here after having read a "targets : dependencies" */
1683 /* line. The state checks if there is a rule to read and if so dispatches */
1684 /* to scan_command_state scan_command_state reads one rule line and the */
1685 /* returns here */

1687         /* First check if the first char on the next line is special */
1688         switch (GET_CHAR()) {
1689         case nul_char:
1690                 GET_NEXT_BLOCK(source);
1691                 if (source == NULL) {
1692                         break;
1693                 }
1694                 goto enter_dependencies_label;
1695         case exclam_char:
1696                 /* The line should be evaluate before it is read */
1697                 macro_seen_in_string = false;
1698                 string_start = source_p + 1;
1699                 for (; 1; source_p++) {
1700                         switch (GET_CHAR()) {
1701                         case newline_char:
1702                                 goto eoln_2;
1703                         case nul_char:
1704                                 if (source->fd > 0) {
1705                                         if (!macro_seen_in_string) {
1706                                                 macro_seen_in_string = true;
1707                                                 INIT_STRING_FROM_STACK(name_stri
1708                                                                        name_buff
1709                                         }
1710                                         append_string(string_start,
1711                                                       &name_string,
1712                                                       source_p - string_start);
1713                                         GET_NEXT_BLOCK(source);
1714                                         string_start = source_p;
1715                                         source_p--;
1716                                         break;
1717                                 }
1718                         eoln_2:
1719                                 if (!macro_seen_in_string) {
1720                                         INIT_STRING_FROM_STACK(name_string,
1721                                                                name_buffer);
1722                                 }
1723                                 append_string(string_start,
1724                                               &name_string,
1725                                               source_p - string_start);
1726                                 extrap = (Source)
1727                                   alloca((int) sizeof (Source_rec));
1728                                 extrap->string.buffer.start = NULL;
1729                                 extrap->inp_buf =
1730                                   extrap->inp_buf_ptr =
1731                                     extrap->inp_buf_end = NULL;
1732                                 extrap->error_converting = false;
1733                                 expand_value(GETNAME(name_string.buffer.start,
1734                                                      FIND_LENGTH),
1735                                              &extrap->string,
1736                                              false);
1737                                 if (name_string.free_after_use) {
1738                                         retmem(name_string.buffer.start);
1739                                 }
1740                                 UNCACHE_SOURCE();
1741                                 extrap->string.text.p =
1742                                   extrap->string.buffer.start;
1743                                 extrap->fd = -1;
1744                                 extrap->previous = source;
1745                                 source = extrap;
```



```
1746                                 CACHE_SOURCE(0);
1747                                 goto enter_dependencies_label;
1748                         }
1749                 }
1750         case dollar_char:
1751                 if (source->already_expanded) {
1752                         break;
1753                 }
1754                 source_p++;
1755                 UNCACHE_SOURCE();
1756                 {
1757                         Source t = (Source) alloca((int) sizeof (Source_rec));
1758                         source = push_macro_value(t,
1759                                                   buffer,
1760                                                   sizeof buffer,
1761                                                   source);
1762                 }
1763                 CACHE_SOURCE(0);
1764                 goto enter_dependencies_label;
1765         case numbersign_char:
1766                 if (makefile_type != reading_makefile) {
1767                         source_p++;
1768                         GOTO_STATE(scan_command_state);
1769                 }
1770                 for (; 1; source_p++) {
1771                         switch (GET_CHAR()) {
1772                         case nul_char:
1773                                 GET_NEXT_BLOCK_NOCHK(source);
1774                                 if (source == NULL) {
1775                                         GOTO_STATE(on_eoln_state);
1776                                 }
1777                                 if (source->error_converting) {
1778                                 // Illegal byte sequence - skip its first byte
1779                                         source->inp_buf_ptr++;
1780                                 }
1781                                 source_p--;
1782                                 break;
1783                         case backslash_char:
1784                                 if (*++source_p == (int) nul_char) {
1785                                         GET_NEXT_BLOCK_NOCHK(source);
1786                                         if (source == NULL) {
1787                                                 GOTO_STATE(on_eoln_state);
1788                                         }
1789                                         if (source->error_converting) {
1790                                         // Illegal byte sequence - skip its firs
1791                                                 source->inp_buf_ptr++;
1792                                                 source_p--;
1793                                                 break;
1794                                         }
1795                                 }
1796                                 if(*source_p == (int) newline_char) {
1797                                         if (source->fd >= 0) {
1798                                                 line_number++;
1799                                         }
1800                                 }
1801                                 break;
1802                         case newline_char:
1803                                 source_p++;
1804                                 if (source->fd >= 0) {
1805                                         line_number++;
1806                                 }
1807                                 goto enter_dependencies_label;
1808                         }
1809                 }

1811         case tab_char:
```

```
1812                 GOTO_STATE(scan_command_state);
1813         }

1815         /* We read all the command lines for the target/dependency line. */
1816         /* Enter the stuff */
1817         enter_target_groups_and_dependencies( &target, &depes, command,
1818                                               separator, target_group_seen);

1820         goto start_new_line;

1822 /*************************************************************
1823  *      scan command state
1824  */
1825 case scan_command_state:
1826         /* We need to read one rule line. Do that and return to */
1827         /* the enter dependencies state */
1828         string_start = source_p;
1829         macro_seen_in_string = false;
1830         for (; 1; source_p++) {
1831                 switch (GET_CHAR()) {
1832                 case backslash_char:
1833                         if (!macro_seen_in_string) {
1834                                 INIT_STRING_FROM_STACK(name_string,
1835                                                        name_buffer);
1836                         }
1837                         append_string(string_start,
1838                                       &name_string,
1839                                       source_p - string_start);
1840                         macro_seen_in_string = true;
1841                         if (*++source_p == (int) nul_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);

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        }

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;

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"));
1912        fatal_reader(catgets(catd, 1, 340, "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***
1917                        fatal_reader(catgets(catd, 1, 90, "Unexpected end of lin
1996                }
1997        }
1998        fatal_reader(gettext("Unexpected end of line seen"));
1920        fatal_reader(catgets(catd, 1, 87, "Unexpected end of line seen"));
1999 case poorly_formed_macro_state:
2000        fatal_reader(gettext("Badly formed macro assignment"));
1922        fatal_reader(catgets(catd, 1, 88, "Badly formed macro assignment"));
2001 case exit_state:
2002        return;
2003 default:
2004        fatal_reader(gettext("Internal error. Unknown reader state"));
1926        fatal_reader(catgets(catd, 1, 89, "Internal error. Unknown reader state"
```



```
2005 }
______unchanged_portion_omitted_

2052 /*
2053  *      enter_target_groups_and_dependencies(target, depes, command, separator,
2054  *                                           target_group_seen)
2055  *
2056  *      Parameters:
2057  *              target          Structure that shows the target(s) on the line
2058  *                              we are currently parsing. This can looks like
2059  *                              target1 .. targetN : dependencies
2060  *                                                     commands
2061  *                              or
2062  *                              target1 + .. + targetN : dependencies
2063  *                                                       commands
2064  *              depes           Dependencies
2065  *              command         Points to the command(s) to be executed for
2066  *                              this target.
2067  *              separator       : or :: or :=
2068  *              target_group_seen       Set if we have target1 + .. + targetN
2069  *
2070  *
2071  *      After reading the command lines for a target, this routine
2072  *      is called to setup the dependencies and the commands for it.
2073  *      If the target is a % pattern or part of a target group, then
2074  *      the appropriate routines are called.
2075  */
2076
2077 void
2078 enter_target_groups_and_dependencies(Name_vector target, Name_vector depes, Cmd_
2079 {
2080        int                     i;
2081        Boolean                 reset= true;
2082        Chain                   target_group_member;
2083        Percent                 percent_ptr;

2085        for (; target != NULL; target = target->next) {
2086                for (i = 0; i < target->used; i++) {
2087                        if (target->names[i] != NULL) {
2088                                if (target_group_seen) {
2089                                        target_group_member =
2090                                          find_target_groups(target, i, reset);
2091                                        if(target_group_member == NULL) {
2092                                                fatal_reader(gettext("Unexpected
2014                                                fatal_reader(catgets(catd, 1, 32
2093                                        }
2094                                }
2095                                reset = false;

2097                                /* If we saw it in the makefile it must be
2098                                 * a file */
2099                                target->names[i]->stat.is_file = true;
2100                                /* Make sure that we use dependencies
2101                                 * entered for makefiles */
2102                                target->names[i]->state = build_dont_know;

2104                                /* If the target is special we delegate
2105                                 * the processing */
2106                                if (target->names[i]->special_reader
2107                                    != no_special) {
2108                                        special_reader(target->names[i],
2109                                                       depes,
2110                                                       command);
2111                                }
2112                                /* Check if this is a "a%b : x%y" type rule */
2113                                else if (target->names[i]->percent) {
```

```
2114                                        percent_ptr =
2115                                          enter_percent(target->names[i],
2116                                                        target->target_group[i],
2117                                                        depes, command);
2118                                        if (target_group_seen) {
2119                                                target_group_member->percent_mem
2120                                                  percent_ptr;
2121                                        }
2122                                } else if (target->names[i]->dollar) {
2123                                        enter_dyntarget(target->names[i]);
2124                                        enter_dependencies
2125                                          (target->names[i],
2126                                           target->target_group[i],
2127                                           depes,
2128                                           command,
2129                                           separator);
2130                                } else {
2131                                        if (target_group_seen) {
2132                                                target_group_member->percent_mem
2133                                                  NULL;
2134                                        }
2135
2136                                        enter_dependencies
2137                                          (target->names[i],
2138                                           target->target_group[i],
2139                                           depes,
2140                                           command,
2141                                           separator);
2142                                }
2143                        }
2144                }
2145        }
2146 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   51149 Wed May 20 12:19:53 2015
new/usr/src/cmd/make/bin/read2.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

/*
 *      read.c
 *
 *      This file contains the makefile reader.
 */

/*
 * Included files
 */
#include <mk/defs.h>
#include <mksh/dosys.h>         /* sh_command2string() */
#include <mksh/macro.h>         /* expand_value() */
#include <mksh/misc.h>          /* retmem() */
#include <stdarg.h>             /* va_list, va_start(), va_end() */
#include <libintl.h>
#endif /* ! codereview */

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static  Boolean         built_last_make_run_seen;

/*
 * File table of contents
 */
static  Name_vector     enter_member_name(register wchar_t *lib_start, register
extern  Name            normalize_name(register wchar_t *name_string, register i
static  void            read_suffixes_list(register Name_vector depes);
```



```
static  void            make_relative(wchar_t *to, wchar_t *result);
static  void            print_rule(register Cmd_line command);
static  void            sh_transform(Name *name, Name *value);


/*
 *      enter_name(string, tail_present, string_start, string_end,
 *            current_names, extra_names, target_group_seen)
 *
 *      Take one string and enter it as a name. The string is passed in
 *      two parts. A make string and possibly a C string to append to it.
 *      The result is stuffed in the vector current_names.
 *      extra_names points to a vector that is used if current_names overflows.
 *      This is allocad in the calling routine.
 *      Here we handle the "lib.a[members]" notation.
 *
 *      Return value:
 *                              The name vector that was used
 *
 *      Parameters:
 *              tail_present    Indicates if both C and make string was passed
 *              string_start    C string
 *              string_end      Pointer to char after last in C string
 *              string          make style string with head of name
 *              current_names   Vector to deposit the name in
 *              extra_names     Where to get next name vector if we run out
 *              target_group_seen Pointer to boolean that is set if "+" is seen
 *
 *      Global variables used:
 *              makefile_type   When we read a report file we normalize paths
 *              plus            Points to the Name "+"
 */

Name_vector
enter_name(String string, Boolean tail_present, register wchar_t *string_start,
{
        Name                    name;
        register wchar_t        *cp;
        wchar_t                 ch;

        /* If we were passed a separate tail of the name we append it to the */
        /* make string with the rest of it */
        if (tail_present) {
                append_string(string_start, string, string_end - string_start);
                string_start = string->buffer.start;
                string_end = string->text.p;
        }
        ch = *string_end;
        *string_end = (int) nul_char;
        /*
         * Check if there are any ( or [ that are not prefixed with $.
         * If there are, we have to deal with the lib.a(members) format.
         */
        for (cp = (wchar_t *) wschr(string_start, (int) parenleft_char);
             cp != NULL;
             cp = (wchar_t *) wschr(cp + 1, (int) parenleft_char)) {
                if (*(cp - 1) != (int) dollar_char) {
                        *string_end = ch;
                        return enter_member_name(string_start,
                                                 cp,
                                                 string_end,
                                                 current_names,
                                                 extra_names);
                }
        }
        *string_end = ch;
```

```
129         if (makefile_type == reading_cpp_file) {
130                 /* Remove extra ../ constructs if we are reading from a report f
131                 name = normalize_name(string_start, string_end - string_start);
132         } else {
133                 /*
134                  * /tolik, fix bug 1197477/
135                  * Normalize every target name before entering.
136                  * ..//obj/a.o and ../obj//a.o are not two different targets.
137                  * There is only one target ../obj/a.o
138                  */
139                 /*name = GETNAME(string_start, string_end - string_start);*/
140                 name = normalize_name(string_start, string_end - string_start);
141         }

143         /* Internalize the name. Detect the name "+" (target group here) */
144 if(current_names->used != 0 && current_names->names[current_names->used-1] == pl
145         if(name == plus) {
146                 return current_names;
147         }
148 }
149         /* If the current_names vector is full we patch in the one from */
150         /* extra_names */
151         if (current_names->used == VSIZEOF(current_names->names)) {
152                 if (current_names->next != NULL) {
153                         current_names = current_names->next;
154                 } else {
155                         current_names->next = *extra_names;
156                         *extra_names = NULL;
157                         current_names = current_names->next;
158                         current_names->used = 0;
159                         current_names->next = NULL;
160                 }
161         }
162         current_names->target_group[current_names->used] = NULL;
163         current_names->names[current_names->used++] = name;
164         if (name == plus) {
165                 *target_group_seen = true;
166         }
167         if (tail_present && string->free_after_use) {
168                 retmem(string->buffer.start);
169         }
170         return current_names;
171 }

173 /*
174  *      enter_member_name(lib_start, member_start, string_end,
175  *                current_names, extra_names)
176  *
177  *      A string has been found to contain member names.
178  *      (The "lib.a[members]" and "lib.a(members)" notation)
179  *      Handle it pretty much as enter_name() does for simple names.
180  *
181  *      Return value:
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  */
193 static Name_vector
```



```
194 enter_member_name(register wchar_t *lib_start, register wchar_t *member_start, r
195 {
196         register Boolean        entry = false;
197         wchar_t                 buffer[STRING_BUFFER_LENGTH];
198         Name                    lib;
199         Name                    member;
200         Name                    name;
201         Property                prop;
202         wchar_t                 *memberp;
203         wchar_t                 *q;
204         register int            paren_count;
205         register Boolean        has_dollar;
206         register wchar_t        *cq;
207         Name                    long_member_name = NULL;

209         /* Internalize the name of the library */
210         lib = GETNAME(lib_start, member_start - lib_start);
211         lib->is_member = true;
212         member_start++;
213         if (*member_start == (int) parenleft_char) {
214                 /* This is really the "lib.a((entries))" format */
215                 entry = true;
216                 member_start++;
217         }
218         /* Move the library name to the buffer where we intend to build the */
219         /* "lib.a(member)" for each member */
220         (void) wsncpy(buffer, lib_start, member_start - lib_start);
221         memberp = buffer + (member_start-lib_start);
222         while (1) {
223                 long_member_name = NULL;
224                 /* Skip leading spaces */
225                 for (;
226                      (member_start < string_end) && iswspace(*member_start);
227                      member_start++);
228                 /* Find the end of the member name. Allow nested (). Detect $*/
229                 for (cq = memberp, has_dollar = false, paren_count = 0;
230                      (member_start < string_end) &&
231                      ((*member_start != (int) parenright_char) ||
232                       (paren_count > 0)) &&
233                      !iswspace(*member_start);
234                      *cq++ = *member_start++) {
235                         switch (*member_start) {
236                         case parenleft_char:
237                                 paren_count++;
238                                 break;
239                         case parenright_char:
240                                 paren_count--;
241                                 break;
242                         case dollar_char:
243                                 has_dollar = true;
244                         }
245                 }
246                 /* Internalize the member name */
247                 member = GETNAME(memberp, cq - memberp);
248                 *cq = 0;
249                 if ((q = (wchar_t *) wsrchr(memberp, (int) slash_char)) == NULL)
250                         q = memberp;
251                 }
252                 if ((cq - q > (int) ar_member_name_len) &&
253                     !has_dollar) {
254                         *cq++ = (int) parenright_char;
255                         if (entry) {
256                                 *cq++ = (int) parenright_char;
257                         }
258                         long_member_name = GETNAME(buffer, cq - buffer);
259                         cq = q + (int) ar_member_name_len;
```

```
260                 }
261                 *cq++ = (int) parenright_char;
262                 if (entry) {
263                         *cq++ = (int) parenright_char;
264                 }
265                 /* Internalize the "lib.a(member)" notation for this member */
266                 name = GETNAME(buffer, cq - buffer);
267                 name->is_member = lib->is_member;
268                 if (long_member_name != NULL) {
269                         prop = append_prop(name, long_member_name_prop);
270                         name->has_long_member_name = true;
271                         prop->body.long_member_name.member_name =
272                           long_member_name;
273                 }
274                 /* And add the member prop */
275                 prop = append_prop(name, member_prop);
276                 prop->body.member.library = lib;
277                 if (entry) {
278                         /* "lib.a((entry))" notation */
279                         prop->body.member.entry = member;
280                         prop->body.member.member = NULL;
281                 } else {
282                         /* "lib.a(member)" Notation */
283                         prop->body.member.entry = NULL;
284                         prop->body.member.member = member;
285                 }
286                 /* Handle overflow of current_names */
287                 if (current_names->used == VSIZEOF(current_names->names)) {
288                         if (current_names->next != NULL) {
289                                 current_names = current_names->next;
290                         } else {
291                                 if (*extra_names == NULL) {
292                                         current_names =
293                                           current_names->next =
294                                             ALLOC(Name_vector);
295                                 } else {
296                                         current_names =
297                                           current_names->next =
298                                             *extra_names;
299                                         *extra_names = NULL;
300                                 }
301                                 current_names->used = 0;
302                                 current_names->next = NULL;
303                         }
304                 }
305                 current_names->target_group[current_names->used] = NULL;
306                 current_names->names[current_names->used++] = name;
307                 while (iswspace(*member_start)) {
308                         member_start++;
309                 }
310                 /* Check if there are more members */
311                 if ((*member_start == (int) parenright_char) ||
312                     (member_start >= string_end)) {
313                         return current_names;
314                 }
315         }
316         /* NOTREACHED */
317 }
318
319 /*
320  *      normalize_name(name_string, length)
321  *
322  *      Take a namestring and remove redundant ../, // and ./ constructs
323  *
324  *      Return value:
325  *                              The normalized name
```



```
326  *
327  *      Parameters:
328  *              name_string     Path string to normalize
329  *              length          Length of that string
330  *
331  *      Global variables used:
332  *              dot             The Name ".", compared against
333  *              dotdot          The Name "..", compared against
334  */
335 Name
336 normalize_name(register wchar_t *name_string, register int length)
337 {
338         static Name             dotdot;
339         register wchar_t        *string = ALLOC_WC(length + 1);
340         register wchar_t        *string2;
341         register wchar_t        *cdp;
342         wchar_t                 *current_component;
343         Name                    name;
344         register int            count;
345
346         if (dotdot == NULL) {
347                 MBSTOWCS(wcs_buffer, "..");
348                 dotdot = GETNAME(wcs_buffer, FIND_LENGTH);
349         }
350
351         /*
352          * Copy string removing ./ and //.
353          * First strip leading ./
354          */
355         while ((length > 1) &&
356                (name_string[0] == (int) period_char) &&
357                (name_string[1] == (int) slash_char)) {
358                 name_string += 2;
359                 length -= 2;
360                 while ((length > 0) && (name_string[0] == (int) slash_char)) {
361                         name_string++;
362                         length--;
363                 }
364         }
365         /* Then copy the rest of the string removing /./ & // */
366         cdp = string;
367         while (length > 0) {
368                 if (((length > 2) &&
369                      (name_string[0] == (int) slash_char) &&
370                      (name_string[1] == (int) period_char) &&
371                      (name_string[2] == (int) slash_char)) ||
372                     ((length == 2) &&
373                      (name_string[0] == (int) slash_char) &&
374                      (name_string[1] == (int) period_char))) {
375                         name_string += 2;
376                         length -= 2;
377                         continue;
378                 }
379                 if ((length > 1) &&
380                     (name_string[0] == (int) slash_char) &&
381                     (name_string[1] == (int) slash_char)) {
382                         name_string++;
383                         length--;
384                         continue;
385                 }
386                 *cdp++ = *name_string++;
387                 length--;
388         }
389         *cdp = (int) nul_char;
390         /*
391          * Now scan for <name>/../ and remove such combinations iff <name>
```

```
392          * is not another ..
393          * Each time something is removed, the whole process is restarted.
394          */
395 removed_one:
396         name_string = string;
397         string2 = name_string;          /*save for free*/
398         current_component =
399           cdp =
400             string =
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) &&
410                      (name_string[1] == (int) period_char) &&
411                      (name_string[2] == (int) period_char))) {
412                    /* Positioned on the / that starts a /.. sequence */
413                    if (((count = cdp - current_component) != 0) &&
414                        (exists(name = GETNAME(string, cdp - string)) > file
415                        (!name->stat.is_sym_link)) {
416                            name = GETNAME(current_component, count);
417                            if(name != dotdot) {
418                                    cdp = current_component;
419                                    name_string += 3;
420                                    length -= 3;
421                                    if (length > 0) {
422                                            name_string++;  /* skip slash */
423                                            length--;
424                                            while (length > 0) {
425                                                    *cdp++ = *name_string++;
426                                                    length--;
427                                            }
428                                    }
429                                    *cdp = (int) nul_char;
430                                    retmem(string2);
431                                    goto removed_one;
432                            }
433                    }
434                 }
435                 if ((*cdp++ = *name_string++) == (int) slash_char) {
436                         current_component = cdp;
437                 }
438                 length--;
439         }
440         *cdp = (int) nul_char;
441         if (string[0] == (int) nul_char) {
442                 name = dot;
443         } else {
444                 name = GETNAME(string, FIND_LENGTH);
445         }
446         retmem(string);
447         retmem(string2);
448         return name;
449 }

451 /*
452  *      find_target_groups(target_list)
453  *
454  *      If a "+" was seen when the target list was scanned we need to extract
455  *      the groups. Each target in the name vector that is a member of a
456  *      group gets a pointer to a chain of all the members stuffed in its
457  *      target_group vector slot
```



```
458  *
459  *      Parameters:
460  *              target_list     The list of targets that contains "+"
461  *
462  *      Global variables used:
463  *              plus            The Name "+", compared against
464  */
465 Chain
466 find_target_groups(register Name_vector target_list, register int i, Boolean res
467 {
468         static Chain            target_group = NULL;
469         static Chain            tail_target_group = NULL;
470         static Name             *next;
471         static Boolean  clear_target_group = false;

473         if (reset) {
474                 target_group = NULL;
475                 tail_target_group = NULL;
476                 clear_target_group = false;
477         }

479         /* Scan the list of targets */
480         /* If the previous target terminated a group */
481         /* we flush the pointer to that member chain */
482         if (clear_target_group) {
483                 clear_target_group = false;
484                 target_group = NULL;
485         }
486         /* Pick up a pointer to the cell with */
487         /* the next target */
488         if (i + 1 != target_list->used) {
489                 next = &target_list->names[i + 1];
490         } else {
491                 next = (target_list->next != NULL) ?
492                   &target_list->next->names[0] : NULL;
493         }
494         /* We have four states here :
495          *      0:      No target group started and next element is not "+"
496          *              This is not interesting.
497          *      1:      A target group is being built and the next element
498          *              is not "+". This terminates the group.
499          *      2:      No target group started and the next member is "+"
500          *              This is the first target in a group.
501          *      3:      A target group started and the next member is a "+"
502          *              The group continues.
503          */
504         switch ((target_group ? 1 : 0) +
505                 (next && (*next == plus) ?
506                  2 : 0)) {
507               case 0:   /* Not target_group */
508                 break;
509               case 1:   /* Last group member */
510                 /* We need to keep this pointer so */
511                 /* we can stuff it for last member */
512                 clear_target_group = true;
513                 /* fall into */
514               case 3:   /* Middle group member */
515                 /* Add this target to the */
516                 /* current chain */
517                 tail_target_group->next = ALLOC(Chain);
518                 tail_target_group = tail_target_group->next;
519                 tail_target_group->next = NULL;
520                 tail_target_group->name = target_list->names[i];
521                 break;
522               case 2:   /* First group member */
523                 /* Start a new chain */
```

```
524                 target_group = tail_target_group = ALLOC(Chain);
525                 target_group->next = NULL;
526                 target_group->name = target_list->names[i];
527                 break;
528         }
529         /* Stuff the current chain, if any, in the */
530         /* targets group slot */
531         target_list->target_group[i] = target_group;
532         if ((next != NULL) &&
533             (*next == plus)) {
534                 *next = NULL;
535         }
536         return (tail_target_group);
537 }

539 /*
540  *      enter_dependencies(target, target_group, depes, command, separator)
541  *
542  *      Take one target and a list of dependencies and process the whole thing.
543  *      The target might be special in some sense in which case that is handled
544  *
545  *      Parameters:
546  *              target          The target we want to enter
547  *              target_group    Non-NULL if target is part of a group this time
548  *              depes           A list of dependencies for the target
549  *              command         The command the target should be entered with
550  *              separator       Indicates if this is a ":" or a "::" rule
551  *
552  *      Static variables used:
553  *              built_last_make_run_seen If the previous target was
554  *                                       .BUILT_LAST_MAKE_RUN we say to rewrite
555  *                                       the state file later on
556  *
557  *      Global variables used:
558  *              command_changed Set to indicate if .make.state needs rewriting
559  *              default_target_to_build Set to the target if reading makefile
560  *                                      and this is the first regular target
561  *              force           The Name " FORCE", used with "::" targets
562  *              makefile_type   We do different things for makefile vs. report
563  *              not_auto        The Name ".NOT_AUTO", compared against
564  *              recursive_name  The Name ".RECURSIVE", compared against
565  *              temp_file_number Used to figure out when to clear stale
566  *                                      automatic dependencies
567  *              trace_reader    Indicates that we should echo stuff we read
568  */
569 void
570 enter_dependencies(register Name target, Chain target_group, register Name_vecto
571 {
572         register int            i;
573         register Property       line;
574         Name                    name;
575         Name                    directory;
576         wchar_t                 *namep;
577         char                    *mb_namep;
578         Dependency              dp;
579         Dependency              *dpp;
580         Property                line2;
581         wchar_t                 relative[MAXPATHLEN];
582         register int            recursive_state;
583         Boolean                 register_as_auto;
584         Boolean                 not_auto_found;
585         char                    *slash;
586         Wstring                 depstr;

588         /* Check if this is a .RECURSIVE line */
589         if ((depes->used >= 3) &&
```



```
590             (depes->names[0] == recursive_name)) {
591                 target->has_recursive_dependency = true;
592                 depes->names[0] = NULL;
593                 recursive_state = 0;
594                 dp = NULL;
595                 dpp = &dp;
596                 /* Read the dependencies. They are "<directory> <target-made>*/
597                 /* <makefile>*" */
598                 for (; depes != NULL; depes = depes->next) {
599                         for (i = 0; i < depes->used; i++) {
600                                 if (depes->names[i] != NULL) {
601                                         switch (recursive_state++) {
602                                         case 0: /* Directory */
603                                         {
604                                                 depstr.init(depes->names[i]);
605                                                 make_relative(depstr.get_string(
606                                                                 relative);
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                 /* Check if this recursion already has been reported else */
629                 /* enter the recursive prop for the target */
630                 /* The has_built flag is used to tell if this .RECURSIVE */
631                 /* was discovered from this run (read from a tmp file) */
632                 /* or was from discovered from the original .make.state */
633                 /* file */
634                 for (line = get_prop(target->prop, recursive_prop);
635                      line != NULL;
636                      line = get_prop(line->next, recursive_prop)) {
637                         if ((line->body.recursive.directory == directory) &&
638                             (line->body.recursive.target == name)) {
639                                 line->body.recursive.makefiles = dp;
640                                 line->body.recursive.has_built =
641                                   (Boolean)
642                                     (makefile_type == reading_cpp_file);
643                                 return;
644                         }
645                 }
646                 line2 = append_prop(target, recursive_prop);
647                 line2->body.recursive.directory = directory;
648                 line2->body.recursive.target = name;
649                 line2->body.recursive.makefiles = dp;
650                 line2->body.recursive.has_built =
651                     (Boolean) (makefile_type == reading_cpp_file);
652                 line2->body.recursive.in_depinfo = false;
653                 return;
654         }
655         /* If this is the first target that doesnt start with a "." in the */
```

```
656         /* makefile we remember that */
657         Wstring tstr(target);
658         wchar_t * wcb = tstr.get_string();
659         if ((makefile_type == reading_makefile) &&
660             (default_target_to_build == NULL) &&
661             ((wcb[0] != (int) period_char) ||
662              wschr(wcb, (int) slash_char))) {

664 /* BID 1181577: $(EMPTY_MACRO) + $(EMPTY_MACRO):
665 ** The target with empty name cannot be default_target_to_build
666 */
667                 if (target->hash.length != 0)
668                         default_target_to_build = target;
669         }
670         /* Check if the line is ":" or "::" */
671         if (makefile_type == reading_makefile) {
672                 if (target->colons == no_colon) {
673                         target->colons = separator;
674                 } else {
675                         if (target->colons != separator) {
676                                 fatal_reader(gettext(":/:: conflict for target ‘
 40                              fatal_reader(catgets(catd, 1, 92, ":/:: conflict
677                                             target->string_mb);
678                         }
679                 }
680                 if (target->colons == two_colon) {
681                         if (depes->used == 0) {
682                                 /* If this is a "::" type line with no */
683                                 /* dependencies we add one "FRC" type */
684                                 /* dependency for free */
685                                 depes->used = 1; /* Force :: targets with no
686                                                     * depes to always run */
687                                 depes->names[0] = force;
688                         }
689                         /* Do not delete "::" type targets when interrupted */
690                         target->stat.is_precious = true;
691                         /*
692                          * Build a synthetic target "<number>%target"
693                          * for "target".
694                          */
695                         mb_namep = getmem((int) (strlen(target->string_mb) + 10)
696                         namep = ALLOC_WC((int) (target->hash.length + 10));
697                         slash = strrchr(target->string_mb, (int) slash_char);
698                         if (slash == NULL) {
699                                 (void) sprintf(mb_namep,
700                                                "%d@%s",
701                                                target->colon_splits++,
702                                                target->string_mb);
703                         } else {
704                                 *slash = 0;
705                                 (void) sprintf(mb_namep,
706                                                "%s/%d@%s",
707                                                target->string_mb,
708                                                target->colon_splits++,
709                                                slash + 1);
710                                 *slash = (int) slash_char;
711                         }
712                         MBSTOWCS(namep, mb_namep);
713                         retmem_mb(mb_namep);
714                         name = GETNAME(namep, FIND_LENGTH);
715                         retmem(namep);
716                         if (trace_reader) {
717                                 (void) printf("%s:\t", target->string_mb);
718                         }
719                         /* Make "target" depend on "<number>%target */
720                         line2 = maybe_append_prop(target, line_prop);
```



```
721                         enter_dependency(line2, name, true);
722                         line2->body.line.target = target;
723                         /* Put a prop on "<number>%target that makes */
724                         /* appear as "target" */
725                         /* when it is processed */
726                         maybe_append_prop(name, target_prop)->
727                           body.target.target = target;
728                         target->is_double_colon_parent = true;
729                         name->is_double_colon = true;
730                         name->has_target_prop = true;
731                         if (trace_reader) {
732                                 (void) printf("\n");
733                         }
734                         (target = name)->stat.is_file = true;
735                 }
736         }
737         /* This really is a regular dependency line. Just enter it */
738         line = maybe_append_prop(target, line_prop);
739         line->body.line.target = target;
740         /* Depending on what kind of makefile we are reading we have to */
741         /* treat things differently */
742         switch (makefile_type) {
743         case reading_makefile:
744                 /* Reading regular makefile. Just notice whether this */
745                 /* redefines the rule for the  target */
746                 if (command != NULL) {
747                         if (line->body.line.command_template != NULL) {
748                                 line->body.line.command_template_redefined =
749                                   true;
750                                 if ((wcb[0] == (int) period_char) &&
751                                     !wschr(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 {
759                         if ((wcb[0] == (int) period_char) &&
760                             !wschr(wcb, (int) slash_char)) {
761                                 line->body.line.command_template = command;
762                         }
763                 }
764                 break;
765         case rereading_statefile:
766                 /* Rereading the statefile. We only enter thing that changed */
767                 /* since the previous time we read it */
768                 if (!built_last_make_run_seen) {
769                         for (Cmd_line next, cmd = command; cmd != NULL; cmd = ne
770                                 next = cmd->next;
771                                 free(cmd);
772                         }
773                         return;
774                 }
775                 built_last_make_run_seen = false;
776                 command_changed = true;
777                 target->ran_command = true;
778         case reading_statefile:
779                 /* Reading the statefile for the first time. Enter the rules */
780                 /* as "Commands used" not "templates to use" */
781                 if (command != NULL) {
782                         for (Cmd_line next, cmd = line->body.line.command_used;
783                              cmd != NULL; cmd = next) {
784                                 next = cmd->next;
785                                 free(cmd);
786                         }
```

```
787                         line->body.line.command_used = command;
788                 }
789         case reading_cpp_file:
790                 /* Reading report file from programs that reports */
791                 /* dependencies. If this is the first time the target is */
792                 /* read from this reportfile we clear all old */
793                 /* automatic depes */
794                 if (target->temp_file_number == temp_file_number) {
795                         break;
796                 }
797                 target->temp_file_number = temp_file_number;
798                 command_changed = true;
799                 if (line != NULL) {
800                         for (dp = line->body.line.dependencies;
801                              dp != NULL;
802                              dp = dp->next) {
803                                 if (dp->automatic) {
804                                         dp->stale = true;
805                                 }
806                         }
807                 }
808                 break;
809         default:
810                 fatal_reader(gettext("Internal error. Unknown makefile type %d")
174                 fatal_reader(catgets(catd, 1, 93, "Internal error. Unknown makef
811                              makefile_type);
812         }
813         /* A target may only be involved in one target group */
814         if (line->body.line.target_group != NULL) {
815                 if (target_group != NULL) {
816                         fatal_reader(gettext("Too many target groups for target
180                         fatal_reader(catgets(catd, 1, 94, "Too many target group
817                                      target->string_mb);
818                 }
819         } else {
820                 line->body.line.target_group = target_group;
821         }

823         if (trace_reader) {
824                 (void) printf("%s:\t", target->string_mb);
825         }
826         /* Enter the dependencies */
827         register_as_auto = BOOLEAN(makefile_type != reading_makefile);
828         not_auto_found = false;
829         for (;
830              (depes != NULL) && !not_auto_found;
831              depes = depes->next) {
832                 for (i = 0; i < depes->used; i++) {
833                         /* the dependency .NOT_AUTO signals beginning of
834                          * explicit dependancies which were put at end of
835                          * list in .make.state file - we stop entering
836                          * dependencies at this point
837                          */
838                         if (depes->names[i] == not_auto) {
839                                 not_auto_found = true;
840                                 break;
841                         }
842                         enter_dependency(line,
843                                          depes->names[i],
844                                          register_as_auto);
845                 }
846         }
847         if (trace_reader) {
848                 (void) printf("\n");
849                 print_rule(command);
850         }
```



```
 851 }
______unchanged_portion_omitted_

1052 /*
1053  *      enter_dyntarget(target)
1054  *
1055  *      Enter "$$(MACRO) : b" type lines
1056  *
1057  *      Parameters:
1058  *              target          Left hand side of pattern
1059  *
1060  *      Global variables used:
1061  *              dyntarget_list  The list of all percent rules, added to
1062  *              trace_reader    Indicates that we should echo stuff we read
1063  */
1064 Dyntarget
1065 enter_dyntarget(register Name target)
1066 {
1067         register Dyntarget      result = ALLOC(Dyntarget);
1068         Dyntarget               p;
1069         Dyntarget               *insert;
1070         int                             i;

1072         result->next = NULL;
1073         result->name = target;


1076         /* Find the end of the dyntarget list and append the new pattern */
1077         for (insert = &dyntarget_list, p = *insert;
1078              p != NULL;
1079              insert = &p->next, p = *insert);
1080         *insert = result;

1082         if (trace_reader) {
1083                 (void) printf("Dynamic target %s:\n", result->name->string_mb);
 447                 (void) printf(NOCATGETS("Dynamic target %s:\n"), result->name->s
1084         }
1085         return( result);
1086 }


1089 /*
1090  *      special_reader(target, depes, command)
1091  *
1092  *      Read the pseudo targets make knows about
1093  *      This handles the special targets that should not be entered as regular
1094  *      target/dependency sets.
1095  *
1096  *      Parameters:
1097  *              target          The special target
1098  *              depes           The list of dependencies it was entered with
1099  *              command         The command it was entered with
1100  *
1101  *      Static variables used:
1102  *              built_last_make_run_seen Set to indicate .BUILT_LAST... seen
1103  *
1104  *      Global variables used:
1105  *              all_parallel    Set to indicate that everything runs parallel
1106  *              svr4            Set when ".SVR4" target is read
1107  *              svr4_name       The Name ".SVR4"
1108  *              posix           Set when ".POSIX" target is read
1109  *              posix_name      The Name ".POSIX"
1110  *              current_make_version The Name "<current version number>"
1111  *              default_rule    Set when ".DEFAULT" target is read
1112  *              default_rule_name The Name ".DEFAULT", used for tracing
1113  *              dot_keep_state  The Name ".KEEP_STATE", used for tracing
```

```
 *              ignore_errors   Set if ".IGNORE" target is read
 *              ignore_name     The Name ".IGNORE", used for tracing
 *              keep_state      Set if ".KEEP_STATE" target is read
 *              no_parallel_name The Name ".NO_PARALLEL", used for tracing
 *              only_parallel   Set to indicate only some targets runs parallel
 *              parallel_name   The Name ".PARALLEL", used for tracing
 *              precious        The Name ".PRECIOUS", used for tracing
 *              sccs_get_name   The Name ".SCCS_GET", used for tracing
 *              sccs_get_posix_name The Name ".SCCS_GET_POSIX", used for tracing
 *              get_name        The Name ".GET", used for tracing
 *              sccs_get_rule   Set when ".SCCS_GET" target is read
 *              silent          Set when ".SILENT" target is read
 *              silent_name     The Name ".SILENT", used for tracing
 *              trace_reader    Indicates that we should echo stuff we read
 */
void
special_reader(Name target, register Name_vector depes, Cmd_line command)
{
        register int            n;

        switch (target->special_reader) {

        case svr4_special:
                if (depes->used != 0) {
                        fatal_reader(gettext("Illegal dependencies for target `%
 502                    fatal_reader(catgets(catd, 1, 98, "Illegal dependencies
                                     target->string_mb);
                }
                svr4  = true;
                posix  = false;
                keep_state = false;
                all_parallel = false;
                only_parallel = false;
                if (trace_reader) {
                        (void) printf("%s:\n", svr4_name->string_mb);
                }
                break;

        case posix_special:
                if(svr4)
                  break;
                if (depes->used != 0) {
                        fatal_reader(gettext("Illegal dependencies for target `%
 519                    fatal_reader(catgets(catd, 1, 99, "Illegal dependencies
                                     target->string_mb);
                }
                posix  = true;
                        /* with posix on, use the posix get rule */
                sccs_get_rule = sccs_get_posix_rule;
                        /* turn keep state off being SunPro make specific */
                keep_state = false;
                /* Use /usr/xpg4/bin/sh on Solaris */
                MBSTOWCS(wcs_buffer, "/usr/xpg4/bin/sh");
 528            MBSTOWCS(wcs_buffer, NOCATGETS("/usr/xpg4/bin/sh"));
                (void) SETVAR(shell_name, GETNAME(wcs_buffer, FIND_LENGTH), fals
                if (trace_reader) {
                        (void) printf("%s:\n", posix_name->string_mb);
                }
                break;

        case built_last_make_run_special:
                built_last_make_run_seen = true;
                break;

        case default_special:
                if (depes->used != 0) {
```



```
                        warning(gettext("Illegal dependency list for target `%s'
 541                    warning(catgets(catd, 1, 100, "Illegal dependency list f
                                target->string_mb);
                }
                default_rule = command;
                if (trace_reader) {
                        (void) printf("%s:\n",
                                      default_rule_name->string_mb);
                        print_rule(command);
                }
                break;


        case ignore_special:
                if ((depes->used != 0) &&(!posix)){
                        fatal_reader(gettext("Illegal dependencies for target `%
 555                    fatal_reader(catgets(catd, 1, 101, "Illegal dependencies
                                     target->string_mb);
                }
                if (depes->used == 0)
                {
                   ignore_errors_all = true;
                }
                if(svr4) {
                  ignore_errors_all = true;
                  break;
                }
                for (; depes != NULL; depes = depes->next) {
                        for (n = 0; n < depes->used; n++) {
                                depes->names[n]->ignore_error_mode = true;
                        }
                }
                if (trace_reader) {
                        (void) printf("%s:\n", ignore_name->string_mb);
                }
                break;

        case keep_state_special:
                if(svr4)
                  break;
                        /* ignore keep state, being SunPro make specific */
                if(posix)
                  break;
                if (depes->used != 0) {
                        fatal_reader(gettext("Illegal dependencies for target `%
 583                    fatal_reader(catgets(catd, 1, 102, "Illegal dependencies
                                     target->string_mb);
                }
                keep_state = true;
                if (trace_reader) {
                        (void) printf("%s:\n",
                                      dot_keep_state->string_mb);
                }
                break;

        case keep_state_file_special:
                if(svr4)
                  break;
                if(posix)
                  break;
                        /* it's not necessary to specify KEEP_STATE, if this
                        ** is given, so set the keep_state.
                        */
                keep_state = true;
                if (depes->used != 0) {
                  if((!make_state) ||(!strcmp(make_state->string_mb,".make.stat
```

```
 603                  if((!make_state) ||(!strcmp(make_state->string_mb,NOCATGETS("
1240                    make_state = depes->names[0];
1241                  }
1242                }
1243                break;
1244        case make_version_special:
1245                if(svr4)
1246                  break;
1247                if (depes->used != 1) {
1248                        fatal_reader(gettext("Illegal dependency list for target
 612                        fatal_reader(catgets(catd, 1, 103, "Illegal dependency l
1249                                        target->string_mb);
1250                }
1251                if (depes->names[0] != current_make_version) {
1252                        /*
1253                         * Special case the fact that version 1.0 and 1.1
1254                         * are identical.
1255                         */
1256                        if (!IS_EQUAL(depes->names[0]->string_mb,
1257                                        "VERSION-1.1") ||
 621                                        NOCATGETS("VERSION-1.1")) ||
1258                            !IS_EQUAL(current_make_version->string_mb,
1259                                        "VERSION-1.0")) {
 623                                        NOCATGETS("VERSION-1.0"))) {
1260                                /*
1261                                 * Version mismatches should cause the
1262                                 * .make.state file to be skipped.
1263                                 * This is currently not true - it is read
1264                                 * anyway.
1265                                 */
1266                                warning(gettext("Version mismatch between curren
 630                                warning(catgets(catd, 1, 104, "Version mismatch
1267                                        current_make_version->string_mb,
1268                                        depes->names[0]->string_mb);
1269                        }
1270                }
1271                break;

1273        case no_parallel_special:
1274                if(svr4)
1275                  break;
1276                /* Set the no_parallel bit for all the targets on */
1277                /* the dependency list */
1278                if (depes->used == 0) {
1279                        /* only those explicitly made parallel */
1280                        only_parallel = true;
1281                        all_parallel = false;
1282                }
1283                for (; depes != NULL; depes = depes->next) {
1284                        for (n = 0; n < depes->used; n++) {
1285                                if (trace_reader) {
1286                                        (void) printf("%s:\t%s\n",
1287                                                        no_parallel_name->string_m
1288                                                        depes->names[n]->string_mb
1289                                }
1290                                depes->names[n]->no_parallel = true;
1291                                depes->names[n]->parallel = false;
1292                        }
1293                }
1294                break;

1296        case parallel_special:
1297                if(svr4)
1298                  break;
1299                if (depes->used == 0) {
1300                        /* everything runs in parallel */
```



```
1301                        all_parallel = true;
1302                        only_parallel = false;
1303                }
1304                /* Set the parallel bit for all the targets on */
1305                /* the dependency list */
1306                for (; depes != NULL; depes = depes->next) {
1307                        for (n = 0; n < depes->used; n++) {
1308                                if (trace_reader) {
1309                                        (void) printf("%s:\t%s\n",
1310                                                        parallel_name->string_mb,
1311                                                        depes->names[n]->string_mb
1312                                }
1313                                depes->names[n]->parallel = true;
1314                                depes->names[n]->no_parallel = false;
1315                        }
1316                }
1317                break;

1319        case localhost_special:
1320                if(svr4)
1321                  break;
1322                /* Set the no_parallel bit for all the targets on */
1323                /* the dependency list */
1324                if (depes->used == 0) {
1325                        /* only those explicitly made parallel */
1326                        only_parallel = true;
1327                        all_parallel = false;
1328                }
1329                for (; depes != NULL; depes = depes->next) {
1330                        for (n = 0; n < depes->used; n++) {
1331                                if (trace_reader) {
1332                                        (void) printf("%s:\t%s\n",
1333                                                        localhost_name->string_mb,
1334                                                        depes->names[n]->string_mb
1335                                }
1336                                depes->names[n]->no_parallel = true;
1337                                depes->names[n]->parallel = false;
1338                                depes->names[n]->localhost = true;
1339                        }
1340                }
1341                break;

1343        case precious_special:
1344                if (depes->used == 0) {
1345                        /* everything is precious      */
1346                        all_precious = true;
1347                } else {
1348                        all_precious = false;
1349                }
1350                if(svr4) {
1351                  all_precious = true;
1352                  break;
1353                }
1354                /* Set the precious bit for all the targets on */
1355                /* the dependency list */
1356                for (; depes != NULL; depes = depes->next) {
1357                        for (n = 0; n < depes->used; n++) {
1358                                if (trace_reader) {
1359                                        (void) printf("%s:\t%s\n",
1360                                                        precious->string_mb,
1361                                                        depes->names[n]->string_mb
1362                                }
1363                                depes->names[n]->stat.is_precious = true;
1364                        }
1365                }
1366                break;
```

```
1368         case sccs_get_special:
1369                 if (depes->used != 0) {
1370                         fatal_reader(gettext("Illegal dependencies for target ‘%
 734                         fatal_reader(catgets(catd, 1, 105, "Illegal dependencies
1371                                         target->string_mb);
1372                 }
1373                 sccs_get_rule = command;
1374                 sccs_get_org_rule = command;
1375                 if (trace_reader) {
1376                         (void) printf("%s:\n", sccs_get_name->string_mb);
1377                         print_rule(command);
1378                 }
1379                 break;

1381         case sccs_get_posix_special:
1382                 if (depes->used != 0) {
1383                         fatal_reader(gettext("Illegal dependencies for target ‘%
 747                         fatal_reader(catgets(catd, 1, 106, "Illegal dependencies
1384                                         target->string_mb);
1385                 }
1386                 sccs_get_posix_rule = command;
1387                 if (trace_reader) {
1388                         (void) printf("%s:\n", sccs_get_posix_name->string_mb);
1389                         print_rule(command);
1390                 }
1391                 break;

1393         case get_posix_special:
1394                 if (depes->used != 0) {
1395                         fatal_reader(gettext("Illegal dependencies for target ‘%
 759                         fatal_reader(catgets(catd, 1, 107, "Illegal dependencies
1396                                         target->string_mb);
1397                 }
1398                 get_posix_rule = command;
1399                 if (trace_reader) {
1400                         (void) printf("%s:\n", get_posix_name->string_mb);
1401                         print_rule(command);
1402                 }
1403                 break;

1405         case get_special:
1406                 if(!svr4) {
1407                   break;
1408                 }
1409                 if (depes->used != 0) {
1410                         fatal_reader(gettext("Illegal dependencies for target ‘%
 774                         fatal_reader(catgets(catd, 1, 108, "Illegal dependencies
1411                                         target->string_mb);
1412                 }
1413                 get_rule = command;
1414                 sccs_get_rule = command;
1415                 if (trace_reader) {
1416                         (void) printf("%s:\n", get_name->string_mb);
1417                         print_rule(command);
1418                 }
1419                 break;

1421         case silent_special:
1422                 if ((depes->used != 0) && (!posix)){
1423                         fatal_reader(gettext("Illegal dependencies for target ‘%
 787                         fatal_reader(catgets(catd, 1, 109, "Illegal dependencies
1424                                         target->string_mb);
1425                 }
1426                 if (depes->used == 0)
1427                 {
```



```
1428                    silent_all = true;
1429                 }
1430                 if(svr4) {
1431                   silent_all = true;
1432                   break;
1433                 }
1434                 for (; depes != NULL; depes = depes->next) {
1435                         for (n = 0; n < depes->used; n++) {
1436                                 depes->names[n]->silent_mode = true;
1437                         }
1438                 }
1439                 if (trace_reader) {
1440                         (void) printf("%s:\n", silent_name->string_mb);
1441                 }
1442                 break;

1444         case suffixes_special:
1445                 read_suffixes_list(depes);
1446                 break;

1448         default:

1450                 fatal_reader(gettext("Internal error: Unknown special reader"));
 814                 fatal_reader(catgets(catd, 1, 110, "Internal error: Unknown spec
1451         }
1452 }
______unchanged_portion_omitted_

1727 /*
1728  *      enter_equal(name, value, append)
1729  *
1730  *      Enter "MACRO= value" constructs
1731  *
1732  *      Parameters:
1733  *              name            The name of the macro
1734  *              value           The value for the macro
1735  *              append          Indicates if the assignment is appending or not
1736  *
1737  *      Global variables used:
1738  *              trace_reader    Indicates that we should echo stuff we read
1739  */
1740 void
1741 enter_equal(Name name, Name value, register Boolean append)
1742 {
1743         wchar_t         *string;
1744         Name            temp;

1746         if (name->colon) {
1747                 sh_transform(&name, &value);
1748         }
1749         (void) SETVAR(name, value, append);

1751         /* if we’re setting FC, we want to set F77 to the same value. */
1752         Wstring nms(name);
1753         wchar_t * wcb = nms.get_string();
1754         string = wcb;
1755         if (string[0]==’F’ &&
1756             string[1]==’C’ &&
1757             string[2]==’\0’) {
1758                 MBSTOWCS(wcs_buffer, "F77");
1122                 MBSTOWCS(wcs_buffer, NOCATGETS("F77"));
1759                 temp = GETNAME(wcs_buffer, FIND_LENGTH);
1760                 (void) SETVAR(temp, value, append);
1761 /*
1762                 fprintf(stderr, gettext("warning: FC is obsolete, use F77 instea
1126                 fprintf(stderr, catgets(catd, 1, 111, "warning: FC is obsolete,
```

```
 */
        }

        if (trace_reader) {
                if (value == NULL) {
                        (void) printf("%s %c=\n",
                                      name->string_mb,
                                      append ?
                                      (int) plus_char : (int) space_char);
                } else {
                        (void) printf("%s %c= %s\n",
                                      name->string_mb,
                                      append ?
                                      (int) plus_char : (int) space_char,
                                      value->string_mb);
                }
        }
}

/*
 *      sh_transform(name, value)
 *
 *      Parameters:
 *              name    The name of the macro we might transform
 *              value   The value to transform
 *
 */
static void
sh_transform(Name *name, Name *value)
{
        /* Check if we need :sh transform */
        wchar_t         *colon;
        String_rec      command;
        String_rec      destination;
        wchar_t         buffer[1000];
        wchar_t         buffer1[1000];

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

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

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

                if(*value == NULL) {
                        buffer[0] = 0;
                } else {
                        Wstring wcb1((*value));
                        if (IS_WEQUAL(colon, colon_shell)) {
                                INIT_STRING_FROM_STACK(command, buffer1);
                                expand_value(*value, &command, false);
                        } else {
                                command.text.p = wcb1.get_string() + (*value)->h
                                command.text.end = command.text.p;
                                command.buffer.start = wcb1.get_string();
                                command.buffer.end = command.text.p;
                        }
```



```
                        sh_command2string(&command, &destination);
                }

                (*value) = GETNAME(destination.buffer.start, FIND_LENGTH);
                *colon = (int) nul_char;
                (*name) = GETNAME(wcb, FIND_LENGTH);
                *colon = (int) colon_char;
        }
}

/*
 *      fatal_reader(format, args...)
 *
 *      Parameters:
 *              format          printf style format string
 *              args            arguments to match the format
 *
 *      Global variables used:
 *              file_being_read Name of the makefile being read
 *              line_number     Line that is being read
 *              report_pwd      Indicates whether current path should be shown
 *              temp_file_name  When reading tempfile we report that name
 */
/*VARARGS*/
void
fatal_reader(char * pattern, ...)
{
        va_list args;
        char    message[1000];

        va_start(args, pattern);
        if (file_being_read != NULL) {
                WCSTOMBS(mbs_buffer, file_being_read);
                if (line_number != 0) {
                        (void) sprintf(message,
                                       gettext("%s, line %d: %s"),
                                       catgets(catd, 1, 112, "%s, line %d: %s"),
                                       mbs_buffer,
                                       line_number,
                                       pattern);
                } else {
                        (void) sprintf(message,
                                       "%s: %s",
                                       mbs_buffer,
                                       pattern);
                }
                pattern = message;
        }

        (void) fflush(stdout);
        (void) fprintf(stderr, gettext("make: Fatal error in reader: "));
        (void) fprintf(stderr, catgets(catd, 1, 238, "make: Fatal error in reade
        (void) vfprintf(stderr, pattern, args);
        (void) fprintf(stderr, "\n");
        va_end(args);

        if (temp_file_name != NULL) {
                (void) fprintf(stderr,
                               gettext("make: Temp-file %s not removed\n"),
                               catgets(catd, 1, 239, "make: Temp-file %s not rem
                               temp_file_name->string_mb);
                temp_file_name = NULL;
        }

        if (report_pwd) {
                (void) fprintf(stderr,
```

```
1890                                gettext("Current working directory %s\n"),
1254                                catgets(catd, 1, 115, "Current working directory
1891                                get_current_path());
1892            }
1893            (void) fflush(stderr);
1894            exit_status = 1;
1895            exit(1);
1896    }
```

*unchanged_portion_omitted_*

```
**********************************************************
    9635 Wed May 20 12:19:53 2015
new/usr/src/cmd/make/bin/rep.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2003 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      rep.c
 *
 *      This file handles the .nse_depinfo file
 */

/*
 * Included files
 */
#include <mk/defs.h>
#include <mksh/misc.h>          /* retmem() */
#include <vroot/report.h>       /* NSE_DEPINFO */

/*
 * Static variables
 */
static  Recursive_make  recursive_list;
static  Recursive_make  *bpatch = &recursive_list;
static  Boolean         changed;

/*
 * File table of contents
 */


/*
 *      report_recursive_init()
 *
 *      Read the .nse_depinfo file and make a list of all the
 *      .RECURSIVE entries.
 *
 *      Parameters:
 *
 *      Static variables used:
 *              bpatch          Points to slot where next cell should be added
 *
```



```
 *      Global variables used:
 *              recursive_name  The Name ".RECURSIVE", compared against
 */

void
report_recursive_init(void)
{
        char            *search_dir;
        char            nse_depinfo[MAXPATHLEN];
        FILE            *fp;
        int             line_size, line_index;
        wchar_t         *line;
        wchar_t         *bigger_line;
        wchar_t         *colon;
        wchar_t         *dollar;
        Recursive_make  rp;

        /*
         * This routine can be called more than once,  don't do
         * anything after the first time.
         */
        if (depinfo_already_read) {
                return;
        } else {
                depinfo_already_read = true;
        }

        search_dir = getenv("NSE_DEP");
        search_dir = getenv(NOCATGETS("NSE_DEP"));
        if (search_dir == NULL) {
                return;
        }
        (void) sprintf(nse_depinfo, "%s/%s", search_dir, NSE_DEPINFO);
        fp = fopen(nse_depinfo, "r");
        if (fp == NULL) {
                return;
        }
        line_size = MAXPATHLEN;
        line_index = line_size - 1;
        line = ALLOC_WC(line_size);
        Wstring rns(recursive_name);
        wchar_t * wcb = rns.get_string();
        while (fgetws(line, line_size, fp) != NULL) {
                while (wslen(line) == line_index) {
                        if (line[wslen(line) - 1] == '\n') {
                                continue;
                        }
                        bigger_line = ALLOC_WC(2 * line_size);
                        wscpy(bigger_line, line);
                        retmem(line);
                        line = bigger_line;
                        if (fgetws(&line[line_index], line_size, fp) == NULL)
                                continue;
                        line_index = 2 * line_index;
                        line_size = 2 * line_size;
                }

                colon = (wchar_t *) wschr(line, (int) colon_char);
                if (colon == NULL) {
                        continue;
                }
                dollar = (wchar_t *) wschr(line, (int) dollar_char);
                line[wslen(line) - 1] = (int) nul_char;
                if (IS_WEQUALN(&colon[2], wcb,
                    (int) recursive_name->hash.length)) {
                        /*
```

```
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 *) wsdup(line);
140                         if ( dollar != NULL ){
141                                 rp->cond_macrostring =
142                                     (wchar_t *) wsdup(dollar - VER_LEN + 1);
143                         }
144                         /*
145                          * get target name into recursive struct
146                          */
147                         *colon = (int) nul_char;
148                         rp->target = (wchar_t *) wsdup(line);
149                         *bpatch = rp;
150                         bpatch = &rp->next;
151                 }
152         }
153         (void) fclose(fp);
154 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   12306 Wed May 20 12:19:54 2015
new/usr/src/cmd/make/bin/state.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
______unchanged_portion_omitted_

  96 static  void            print_auto_depes(register Dependency dependency, registe

  98 /*
  99  *      write_state_file(report_recursive, exiting)
 100  *
 101  *      Write a new version of .make.state
 102  *
 103  *      Parameters:
 104  *              report_recursive        Should only be done at end of run
 105  *              exiting                 true if called from the exit handler
 106  *
 107  *      Global variables used:
 108  *              built_last_make_run The Name ".BUILT_LAST_MAKE_RUN", written
 109  *              command_changed If no command changed we do not need to write
 110  *              current_make_version The Name "<current version>", written
 111  *              do_not_exec_rule If -n is on we do not write statefile
 112  *              hashtab         The hashtable that contains all names
 113  *              keep_state      If .KEEP_STATE is no on we do not write file
 114  *              make_state      The Name ".make.state", used for opening file
 115  *              make_version    The Name ".MAKE_VERSION", written
 116  *              recursive_name  The Name ".RECURSIVE", written
 117  *              rewrite_statefile Indicates that something changed
 118  */

 120 void
 121 write_state_file(int, Boolean exiting)
 122 {
 123         register FILE           *fd;
 124         int                     lock_err;
 125         char                    buffer[MAXPATHLEN];
 126         char                    make_state_tempfile[MAXPATHLEN];
 127         jmp_buf                 long_jump;
 128         register int            attempts = 0;
 129         Name_set::iterator      np, e;
 130         register Property       lines;
 131         register int            m;
 132         Dependency              dependency;
 133         register Boolean        name_printed;
 134         Boolean                 built_this_run = false;
 135         char                    *target_name;
 136         int                     line_length;
 137         register Cmd_line       cp;


 140         if (!rewrite_statefile ||
 141             !command_changed ||
 142             !keep_state ||
 143             do_not_exec_rule ||
 144             (report_dependencies_level > 0)) {
 145                 return;
 146         }
 147         /* Lock the file for writing. */
 148         make_state_lockfile = getmem(strlen(make_state->string_mb) + strlen(".lo
 148         make_state_lockfile = getmem(strlen(make_state->string_mb) + strlen(NOCA
 149         (void) sprintf(make_state_lockfile,
 150                         "%s.lock",
 150                         NOCATGETS("%s.lock"),
 151                         make_state->string_mb);
 152         if (lock_err = file_lock(make_state->string_mb,
```



```
 153                                  make_state_lockfile,
 154                                  (int *) &make_state_locked, 0)) {
 155                 retmem_mb(make_state_lockfile);
 156                 make_state_lockfile = NULL;
 157
 158                 /*
 159                  * We need to make sure that we are not being
 160                  * called by the exit handler so we don't call
 161                  * it again.
 162                  */
 163
 164                 if (exiting) {
 165                         (void) sprintf(buffer, "%s/.make.state.%d.XXXXXX", tmpdi
 165                         (void) sprintf(buffer, NOCATGETS("%s/.make.state.%d.XXXX
 166                         report_pwd = true;
 167                         warning(gettext("Writing to %s"), buffer);
 167                         warning(catgets(catd, 1, 60, "Writing to %s"), buffer);
 168                         int fdes = mkstemp(buffer);
 169                         if ((fdes < 0) || (fd = fdopen(fdes, "w")) == NULL) {
 170                                 fprintf(stderr,
 171                                         gettext("Could not open statefile '%s':
 171                                         catgets(catd, 1, 61, "Could not open sta
 172                                         buffer,
 173                                         errmsg(errno));
 174                                 return;
 175                         }
 176                 } else {
 177                         report_pwd = true;
 178                         fatal(gettext("Can't lock .make.state"));
 178                         fatal(catgets(catd, 1, 62, "Can't lock .make.state"));
 179                 }
 180         }

 182         (void) sprintf(make_state_tempfile,
 183                         "%s.tmp",
 183                         NOCATGETS("%s.tmp"),
 184                         make_state->string_mb);
 185         /* Delete old temporary statefile (in case it exists) */
 186         (void) unlink(make_state_tempfile);
 187         if ((fd = fopen(make_state_tempfile, "w")) == NULL) {
 188                 lock_err = errno; /* Save it! unlink() can change errno */
 189                 (void) unlink(make_state_lockfile);
 190                 retmem_mb(make_state_lockfile);
 191                 make_state_lockfile = NULL;
 192                 make_state_locked = false;
 193                 fatal(gettext("Could not open temporary statefile '%s': %s"),
 193                 fatal(catgets(catd, 1, 59, "Could not open temporary statefile '
 194                       make_state_tempfile,
 195                       errmsg(lock_err));
 196         }
 197         /*
 198          * Set a trap for failed writes. If a write fails, the routine
 199          * will try saving the .make.state file under another name in /tmp.
 200          */
 201         if (setjmp(long_jump)) {
 202                 (void) fclose(fd);
 203                 if (attempts++ > 5) {
 204                         if ((make_state_lockfile != NULL) &&
 205                             make_state_locked) {
 206                                 (void) unlink(make_state_lockfile);
 207                                 retmem_mb(make_state_lockfile);
 208                                 make_state_lockfile = NULL;
 209                                 make_state_locked = false;
 210                         }
 211                         fatal(gettext("Giving up on writing statefile"));
 211                         fatal(catgets(catd, 1, 63, "Giving up on writing statefi
```

```
212                 }
213                 sleep(10);
214                 (void) sprintf(buffer, "%s/.make.state.%d.XXXXXX", tmpdir, getpi
214                 (void) sprintf(buffer, NOCATGETS("%s/.make.state.%d.XXXXXX"), tm
215                 int fdes = mkstemp(buffer);
216                 if ((fdes < 0) || (fd = fdopen(fdes, "w")) == NULL) {
217                         fatal(gettext("Could not open statefile `%s': %s"),
217                         fatal(catgets(catd, 1, 64, "Could not open statefile `%s
218                               buffer,
219                               errmsg(errno));
220                 }
221                 warning(gettext("Initial write of statefile failed. Trying again
221                 warning(catgets(catd, 1, 65, "Initial write of statefile failed.
222                         buffer);
223         }

225         /* Write the version stamp. */
226         XFWRITE(make_version->string_mb,
227                 strlen(make_version->string_mb),
228                 fd);
229         XPUTC(colon_char, fd);
230         XPUTC(tab_char, fd);
231         XFWRITE(current_make_version->string_mb,
232                 strlen(current_make_version->string_mb),
233                 fd);
234         XPUTC(newline_char, fd);

236         /*
237          * Go through all the targets, dump their dependencies and
238          * command used.
239          */
240         for (np = hashtab.begin(), e = hashtab.end(); np != e; np++) {
241                 /*
242                  * If the target has no command used nor dependencies,
243                  * we can go to the next one.
244                  */
245                 if ((lines = get_prop(np->prop, line_prop)) == NULL) {
246                         continue;
247                 }
248                 /* If this target is a special target, don't print. */
249                 if (np->special_reader != no_special) {
250                         continue;
251                 }
252                 /*
253                  * Find out if any of the targets dependencies should
254                  * be written to .make.state.
255                  */
256                 for (m = 0, dependency = lines->body.line.dependencies;
257                      dependency != NULL;
258                      dependency = dependency->next) {
259                         if (m = !dependency->stale
260                             && (dependency->name != force)
261 #ifndef PRINT_EXPLICIT_DEPEN
262                             && dependency->automatic
263 #endif
264                             ) {
265                                 break;
266                         }
267                 }
268                 /* Only print if dependencies listed. */
269                 if (m || (lines->body.line.command_used != NULL)) {
270                         name_printed = false;
271                         /*
272                          * If this target was built during this make run,
273                          * we mark it.
274                          */
```



```
275                         built_this_run = false;
276                         if (np->has_built) {
277                                 built_this_run = true;
278                                 XFWRITE(built_last_make_run->string_mb,
279                                         strlen(built_last_make_run->string_mb),
280                                         fd);
281                                 XPUTC(colon_char, fd);
282                                 XPUTC(newline_char, fd);
283                         }
284                         /* If the target has dependencies, we dump them. */
285                         target_name = escape_target_name(np);
286                         if (np->has_long_member_name) {
287                                 target_name =
288                                   get_prop(np->prop, long_member_name_prop)
289                                     ->body.long_member_name.member_name->
290                                       string_mb;
291                         }
292                         if (m) {
293                                 XFPUTS(target_name, fd);
294                                 XPUTC(colon_char, fd);
295                                 XFPUTS("\t", fd);
296                                 name_printed = true;
297                                 line_length = 0;
298                                 for (dependency =
299                                      lines->body.line.dependencies;
300                                      dependency != NULL;
301                                      dependency = dependency->next) {
302                                         print_auto_depes(dependency,
303                                                          fd,
304                                                          built_this_run,
305                                                          &line_length,
306                                                          target_name,
307                                                          long_jump);
308                                 }
309                                 XFPUTS("\n", fd);
310                         }
311                         /* If there is a command used, we dump it. */
312                         if (lines->body.line.command_used != NULL) {
313                                 /*
314                                  * Only write the target name if it
315                                  * wasn't done for the dependencies.
316                                  */
317                                 if (!name_printed) {
318                                         XFPUTS(target_name, fd);
319                                         XPUTC(colon_char, fd);
320                                         XPUTC(newline_char, fd);
321                                 }
322                                 /*
323                                  * Write the command lines.
324                                  * Prefix each textual line with a tab.
325                                  */
326                                 for (cp = lines->body.line.command_used;
327                                      cp != NULL;
328                                      cp = cp->next) {
329                                         char            *csp;
330                                         int             n;

332                                         XPUTC(tab_char, fd);
333                                         if (cp->command_line != NULL) {
334                                                 for (csp = cp->
335                                                           command_line->
336                                                           string_mb,
337                                                      n = strlen(cp->
338                                                                 command_line->
339                                                                 string_mb);
340                                                      n > 0;
```

```
341                                                 n--, csp++) {
342                                                    XPUTC(*csp, fd);
343                                                    if (*csp ==
344                                                        (int) newline_char)
345                                                            XPUTC(tab_char,
346                                                                  fd);
347                                                    }
348                                             }
349                                         }
350                                         XPUTC(newline_char, fd);
351                                 }
352                         }
353                         (void)free(target_name);
354                 }
355         }
356         if (fclose(fd) == EOF) {
357                 longjmp(long_jump, LONGJUMP_VALUE);
358         }
359         if (attempts == 0) {
360                 if (unlink(make_state->string_mb) != 0 && errno != ENOENT) {
361                         lock_err = errno; /* Save it! unlink() can change errno
362                         /* Delete temporary statefile */
363                         (void) unlink(make_state_tempfile);
364                         (void) unlink(make_state_lockfile);
365                         retmem_mb(make_state_lockfile);
366                         make_state_lockfile = NULL;
367                         make_state_locked = false;
368                         fatal(gettext("Could not delete old statefile ‘%s’: %s")
368                         fatal(catgets(catd, 1, 356, "Could not delete old statef
369                               make_state->string_mb,
370                               errmsg(lock_err));
371                 }
372                 if (rename(make_state_tempfile, make_state->string_mb) != 0) {
373                         lock_err = errno; /* Save it! unlink() can change errno
374                         /* Delete temporary statefile */
375                         (void) unlink(make_state_tempfile);
376                         (void) unlink(make_state_lockfile);
377                         retmem_mb(make_state_lockfile);
378                         make_state_lockfile = NULL;
379                         make_state_locked = false;
380                         fatal(gettext("Could not rename ‘%s’ to ‘%s’: %s"),
380                         fatal(catgets(catd, 1, 357, "Could not rename ‘%s’ to ‘%
381                               make_state_tempfile,
382                               make_state->string_mb,
383                               errmsg(lock_err));
384                 }
385         }
386         if ((make_state_lockfile != NULL) && make_state_locked) {
387                 (void) unlink(make_state_lockfile);
388                 retmem_mb(make_state_lockfile);
389                 make_state_lockfile = NULL;
390                 make_state_locked = false;
391         }
392 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   13975 Wed May 20 12:19:55 2015
new/usr/src/cmd/make/include/mk/defs.h
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
______unchanged_portion_omitted_


/*
 * Typedefs for all structs
 */
typedef struct _Cmd_line        *Cmd_line, Cmd_line_rec;
typedef struct _Dependency      *Dependency, Dependency_rec;
typedef struct _Macro           *Macro, Macro_rec;
typedef struct _Name_vector     *Name_vector, Name_vector_rec;
typedef struct _Percent         *Percent, Percent_rec;
typedef struct _Dyntarget       *Dyntarget;
typedef struct _Recursive_make  *Recursive_make, Recursive_make_rec;
typedef struct _Running         *Running, Running_rec;


/*
 *      extern declarations for all global variables.
 *      The actual declarations are in globals.cc
 */
extern  Boolean         allrules_read;
extern  Name            posix_name;
extern  Name            svr4_name;
extern  Boolean         sdot_target;
extern  Boolean         all_parallel;
extern  Boolean         assign_done;
extern  Boolean         build_failed_seen;
extern  Name            built_last_make_run;
extern  Name            c_at;
extern  Boolean         command_changed;
extern  Boolean         commands_done;
extern  Chain           conditional_targets;
extern  Name            conditionals;
extern  Boolean         continue_after_error;
extern  Property        current_line;
extern  Name            current_make_version;
extern  Name            current_target;
extern  short           debug_level;
extern  Cmd_line        default_rule;
extern  Name            default_rule_name;
extern  Name            default_target_to_build;
extern  Boolean         depinfo_already_read;
extern  Name            dmake_group;
extern  Name            dmake_max_jobs;
extern  Name            dmake_mode;
extern  DMake_mode      dmake_mode_type;
extern  Name            dmake_output_mode;
extern  DMake_output_mode       output_mode;
extern  Name            dmake_odir;
extern  Name            dmake_rcfile;
extern  Name            done;
extern  Name            dot;
extern  Name            dot_keep_state;
extern  Name            dot_keep_state_file;
extern  Name            empty_name;
extern  Boolean         fatal_in_progress;
extern  int             file_number;
extern  Name            force;
extern  Name            ignore_name;
extern  Boolean         ignore_errors;
extern  Boolean         ignore_errors_all;
```



```
extern  Name            init;
extern  int             job_msg_id;
extern  Boolean         keep_state;
extern  Name            make_state;
extern  timestruc_t     make_state_before;
extern  Boolean         make_state_locked;
extern  Dependency      makefiles_used;
extern  Name            makeflags;
extern  Name            make_version;
extern  char            mbs_buffer2[];
extern  char            *mbs_ptr;
extern  char            *mbs_ptr2;
extern  Boolean         no_action_was_taken;
extern  Boolean         no_parallel;
extern  Name            no_parallel_name;
extern  Name            not_auto;
extern  Boolean         only_parallel;
extern  Boolean         parallel;
extern  Name            parallel_name;
extern  Name            localhost_name;
extern  int             parallel_process_cnt;
extern  Percent         percent_list;
extern  Dyntarget       dyntarget_list;
extern  Name            plus;
extern  Name            pmake_machinesfile;
extern  Name            precious;
extern  Name            primary_makefile;
extern  Boolean         quest;
extern  short           read_trace_level;
extern  Boolean         reading_dependencies;
extern  int             recursion_level;
extern  Name            recursive_name;
extern  short           report_dependencies_level;
extern  Boolean         report_pwd;
extern  Boolean         rewrite_statefile;
extern  Running         running_list;
extern  char            *sccs_dir_path;
extern  Name            sccs_get_name;
extern  Name            sccs_get_posix_name;
extern  Cmd_line        sccs_get_rule;
extern  Cmd_line        sccs_get_org_rule;
extern  Cmd_line        sccs_get_posix_rule;
extern  Name            get_name;
extern  Name            get_posix_name;
extern  Cmd_line        get_rule;
extern  Cmd_line        get_posix_rule;
extern  Boolean         all_precious;
extern  Boolean         report_cwd;
extern  Boolean         silent_all;
extern  Boolean         silent;
extern  Name            silent_name;
extern  char            *stderr_file;
extern  char            *stdout_file;
extern  Boolean         stdout_stderr_same;
extern  Dependency      suffixes;
extern  Name            suffixes_name;
extern  Name            sunpro_dependencies;
extern  Boolean         target_variants;
extern  const char      *tmpdir;
extern  const char      *temp_file_directory;
extern  Name            temp_file_name;
extern  short           temp_file_number;
extern  wchar_t         *top_level_target;
extern  Boolean         touch;
extern  Boolean         trace_reader;
extern  Boolean         build_unconditional;
```

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

293 /*
294  * Declarations of system defined variables
295  */
296 /* On linux this variable is defined in 'signal.h' */
297 extern  char            *sys_siglist[];

299 /*
300  * Declarations of system supplied functions
301  */
302 extern  int             file_lock(char *, char *, int *, int);

304 /*
305  * Declarations of functions declared and used by make
306  */
307 extern  void            add_pending(Name target, int recursion_level, Boolean do
308 extern  void            add_running(Name target, Name true_target, Property comm
309 extern  void            add_serial(Name target, int recursion_level, Boolean do_
310 extern  void            add_subtree(Name target, int recursion_level, Boolean do
311 extern  void            append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar
312 extern  void            await_parallel(Boolean waitflg);
313 extern  void            build_suffix_list(Name target_suffix);
314 extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
315 extern  void            check_state(Name temp_file_name);
316 extern  void            cond_macros_into_string(Name np, String_rec *buffer);
317 extern  void            construct_target_string();
318 extern  void            create_xdrs_ptr(void);
319 extern  void            depvar_add_to_list (Name name, Boolean cmdline);
320 extern  Doname          doname(register Name target, register Boolean do_get, re
321 extern  Doname          doname_check(register Name target, register Boolean do_g
322 extern  Doname          doname_parallel(Name target, Boolean do_get, Boolean imp
323 extern  Doname          dosys(register Name command, register Boolean ignore_err
324 extern  void            dump_make_state(void);
325 extern  void            dump_target_list(void);
326 extern  void            enter_conditional(register Name target, Name name, Name
327 extern  void            enter_dependencies(register Name target, Chain target_gr
328 extern  void            enter_dependency(Property line, register Name depe, Bool
329 extern  void            enter_equal(Name name, Name value, register Boolean appe
330 extern  Percent         enter_percent(register Name target, Chain target_group,
331 extern  Dyntarget       enter_dyntarget(register Name target);
332 extern  Name_vector     enter_name(String string, Boolean tail_present, register
333 extern  Boolean         exec_vp(register char *name, register char **argv, char
334 extern  Doname          execute_parallel(Property line, Boolean waitflg, Boolean
335 extern  Doname          execute_serial(Property line);
336 extern  timestruc_t&    exists(register Name target);
337 extern  void            fatal(const char *, ...);
338 extern  void            fatal_reader(char *, ...);
339 extern  Doname          find_ar_suffix_rule(register Name target, Name true_targ
340 extern  Doname          find_double_suffix_rule(register Name target, Property *
341 extern  Doname          find_percent_rule(register Name target, Property *comman
342 extern  int             find_run_directory (char *cmd, char *cwd, char *dir, cha
343 extern  Doname          find_suffix_rule(Name target, Name target_body, Name tar
344 extern  Chain           find_target_groups(register Name_vector target_list, reg
345 extern  void            finish_children(Boolean docheck);
346 extern  void            finish_running(void);
347 extern  void            free_chain(Name_vector ptr);
348 extern  void            gather_recursive_deps(void);
349 extern  char            *get_current_path(void);
350 extern  int             get_job_msg_id(void);
```



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

unchanged_portion_omitted_

```
**********************************************************
   22507 Wed May 20 12:19:56 2015
new/usr/src/cmd/make/include/mksh/defs.h
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

```
   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 <avo/intl.h>
  28 #include <limits.h>             /* MB_LEN_MAX */
  29 #include <stdio.h>
  30 #include <stdlib.h>             /* wchar_t */
  31 #include <string.h>             /* strcmp() */
  33 #include <nl_types.h>           /* catgets() */
  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>


  42 /*
  43  * A type and some utilities for boolean values
  44  */

  46 #define false   BOOLEAN_false
  47 #define true    BOOLEAN_true

  49 typedef enum {
  50         false =         0,
  51         true =          1,
  52         failed =        0,
  53         succeeded =     1
  54 } Boolean;
______unchanged_portion_omitted_

 854 /*
 855  *      extern declarations for all global variables.
 856  *      The actual declarations are in globals.cc
```



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

 900 /*
 901  * RFE 1257407: make does not use fine granularity time info available from stat
 902  * High resolution time comparison.
 903  */

 905 inline int
 906 operator==(const timestruc_t &t1, const timestruc_t &t2) {
 907         return ((t1.tv_sec == t2.tv_sec) && (t1.tv_nsec == t2.tv_nsec));
 908 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2301 Wed May 20 12:19:56 2015
new/usr/src/cmd/make/include/vroot/vroot.h
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
______unchanged_portion_omitted_
  39 typedef patht           *pathpt;

  41 extern  void            add_dir_to_path(const char *path, register pathpt *point
  42 extern  void            flush_path_cache(void);
  43 extern  void            flush_vroot_cache(void);
  44 extern  const char      *get_path_name(void);
  45 extern  char            *get_vroot_path(register char **vroot, register char **p
  46 extern  const char      *get_vroot_name(void);
  47 extern  int             open_vroot(char *path, int flags, int mode, pathpt vroot
  48 extern  pathpt          parse_path_string(register char *string, register int re
  49 extern  void            scan_path_first(void);
  50 extern  void            scan_vroot_first(void);
  51 extern  void            set_path_style(int style);

  53 extern  int             access_vroot(char *path, int mode, pathpt vroot_path, pa

  55 extern  int             execve_vroot(char *path, char **argv, char **environ, pa

  57 extern  int             lstat_vroot(char *path, struct stat *buffer, pathpt vroo
  58 extern  int             stat_vroot(char *path, struct stat *buffer, pathpt vroot
  59 extern  int             readlink_vroot(char *path, char *buffer, int buffer_size

  62 extern nl_catd  libvroot_catd;
  61 #endif
```

```
**********************************************************
     741 Wed May 20 12:19:56 2015
new/usr/src/cmd/make/lib/Makefile
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

# Copyright 2015, Richard Lowe.

SUBDIRS=bsd \
        makestate \
        mksdmsi18n \
        mksh \
        vroot

all     :=      TARGET = all
install :=      TARGET = install
clean   :=      TARGET = clean
clobber :=      TARGET = clobber
lint    :=      TARGET = lint
_msg    :=      TARGET = _msg
#endif /* ! codereview */

.KEEP_STATE:

all clean clobber lint install _msg: $(SUBDIRS)
all clean clobber lint install: $(SUBDIRS)

$(SUBDIRS):     FRC
        @cd $@; pwd; $(MAKE) $(TARGET)

FRC:
```

```
**********************************************************
     684 Wed May 20 12:19:57 2015
new/usr/src/cmd/make/lib/bsd/Makefile
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 # Copyright 2015, Richard Lowe.

  14 LIBRARY = libbsd.a
  15 VERS = .1
  16 OBJECTS = bsd.o

  18 include $(SRC)/lib/Makefile.lib
  19 include ../../Makefile.com

  21 LIBS = $(LIBRARY)
  22 SRCDIR = ../
  23 CPPFLAGS += -D_FILE_OFFSET_BITS=64

  25 all: $(LIBS)

  27 install: all

  29 lint:

  31 _msg:

  33 #endif /* ! codereview */
  34 include $(SRC)/lib/Makefile.targ
```

```
**********************************************************
     737 Wed May 20 12:19:57 2015
new/usr/src/cmd/make/lib/makestate/Makefile
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

  12 # Copyright 2015, Richard Lowe.

  14 include $(SRC)/Makefile.master

  16 SUBDIRS=$(MACH) $(MACH64)

  18 all     :=      TARGET = all
  19 install :=      TARGET = install
  20 clean   :=      TARGET = clean
  21 clobber :=      TARGET = clobber
  22 lint    :=      TARGET = lint

  24 .KEEP_STATE:

  26 all clean clobber lint install: $(SUBDIRS)

  28 _msg:

  30 #endif /* ! codereview */
  31 $(SUBDIRS):     FRC
  32         @cd $@; pwd; $(MAKE) $(TARGET)

  34 FRC:
```

```
**********************************************************
     949 Wed May 20 12:19:58 2015
new/usr/src/cmd/make/lib/mksh/Makefile
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

  12 # Copyright 2015, Richard Lowe.

  14 LIBRARY =       libmksh.a
  15 VERS =          .1
  16 OBJECTS =       dosys.o \
  17                 globals.o \
  18                 i18n.o \
  19                 macro.o \
  20                 misc.o \
  21                 mksh.o \
  22                 read.o

  24 include $(SRC)/lib/Makefile.lib
  25 include ../../Makefile.com

  27 POFILE = libmksh.po
  28 POFILES = $(OBJECTS:%.o=%.po)

  30 #endif /* ! codereview */
  31 LIBS = $(LIBRARY)
  32 SRCDIR = ../
  33 MAPFILES=
  34 CPPFLAGS += -D_FILE_OFFSET_BITS=64

  36 all: $(LIBS)

  38 install: all

  40 lint:

  42 $(POFILE): $(POFILES)
  43         $(CAT) $(POFILES) > $@

  45 _msg: $(MSGDOMAIN) $(POFILE)
  46         $(RM) $(MSGDOMAIN)/$(POFILE)
  47         $(CP) $(POFILE) $(MSGDOMAIN)

  49 #endif /* ! codereview */
  50 include $(SRC)/lib/Makefile.targ
```

```
**********************************************************
   15028 Wed May 20 12:19:58 2015
new/usr/src/cmd/make/lib/mksh/dosys.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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


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

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

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

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
```



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

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

/*
 *      void
 *      redirect_io(char *stdout_file, char *stderr_file)
 *
 *      Redirects stdout and stderr for a child mksh process.
 */
void
redirect_io(char *stdout_file, char *stderr_file)
{
        long            descriptor_limit;
        int             i;

        if ((descriptor_limit = ulimit(UL_GDESLIM)) < 0) {
                fatal_mksh(gettext("ulimit() failed: %s"), errmsg(errno));
                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 89, "ulimit() failed:
        }
        for (i = 3; i < descriptor_limit; i++) {
                (void) close(i);
        }
        if ((i = my_open(stdout_file,
                 O_WRONLY | O_CREAT | O_TRUNC | O_DSYNC,
                 S_IREAD | S_IWRITE)) < 0) {
                fatal_mksh(gettext("Couldn't open standard out temp file '%s': %
                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 90, "Couldn't open sta
                      stdout_file,
                      errmsg(errno));
        } else {
                if (dup2(i, 1) == -1) {
                        fatal_mksh("*** Error: dup2(3, 1) failed: %s",
                        fatal_mksh(NOCATGETS("*** Error: dup2(3, 1) failed: %s")
                                errmsg(errno));
                }
                close(i);
        }
        if (stderr_file == NULL) {
                if (dup2(1, 2) == -1) {
                        fatal_mksh("*** Error: dup2(1, 2) failed: %s",
                        fatal_mksh(NOCATGETS("*** Error: dup2(1, 2) failed: %s")
                                errmsg(errno));
                }
        } else if ((i = my_open(stderr_file,
                        O_WRONLY | O_CREAT | O_TRUNC | O_DSYNC,
                        S_IREAD | S_IWRITE)) < 0) {
                fatal_mksh(gettext("Couldn't open standard error temp file '%s':
                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 91, "Couldn't open sta
                      stderr_file,
```

```
122                         errmsg(errno));
123         } else {
124                 if (dup2(i, 2) == -1) {
125                         fatal_mksh("*** Error: dup2(3, 2) failed: %s",
 82                         fatal_mksh(NOCATGETS("*** Error: dup2(3, 2) failed: %s")
126                                 errmsg(errno));
127                 }
128                 close(i);
129         }
130 }

132 /*
133  *      doshell(command, ignore_error)
134  *
135  *      Used to run command lines that include shell meta-characters.
136  *      The make macro SHELL is supposed to contain a path to the shell.
137  *
138  *      Return value:
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 int
150 doshell(wchar_t *command, register Boolean ignore_error, char *stdout_file, char
151 {
152         char                    *argv[6];
153         int                     argv_index = 0;
154         int                     cmd_argv_index;
155         int                     length;
156         char                    nice_prio_buf[MAXPATHLEN];
157         register Name           shell = getvar(shell_name);
158         register char           *shellname;
159         char                    *tmp_mbs_buffer;


162         if (IS_EQUAL(shell->string_mb, "")) {
163                 shell = shell_name;
164         }
165         if ((shellname = strrchr(shell->string_mb, (int) slash_char)) == NULL) {
166                 shellname = shell->string_mb;
167         } else {
168                 shellname++;
169         }

171         /*
172          * Only prepend the /usr/bin/nice command to the original command
173          * if the nice priority, nice_prio, is NOT zero (0).
174          * Nice priorities can be a positive or a negative number.
175          */
176         if (nice_prio != 0) {
177                 argv[argv_index++] = (char *)"nice";
178                 (void) sprintf(nice_prio_buf, "-%d", nice_prio);
134                 argv[argv_index++] = (char *)NOCATGETS("nice");
135                 (void) sprintf(nice_prio_buf, NOCATGETS("-%d"), nice_prio);
179                 argv[argv_index++] = strdup(nice_prio_buf);
180         }
181         argv[argv_index++] = shellname;
182         argv[argv_index++] = (char*)(ignore_error ? "-c" : "-ce");
139         argv[argv_index++] = (char*)(ignore_error ? NOCATGETS("-c") : NOCATGETS(
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 {
190                 WCSTOMBS(mbs_buffer, command);
191                 cmd_argv_index = argv_index;
192                 argv[argv_index++] = strdup(mbs_buffer);
193         }
194         argv[argv_index] = NULL;
195         (void) fflush(stdout);
196         if ((childPid = fork()) == 0) {
197                 enable_interrupt((void (*) (int)) SIG_DFL);
198 #if 0
199                 if (filter_stderr) {
200                         redirect_stderr();
201                 }
202 #endif
203                 if (nice_prio != 0) {
204                         (void) execve("/usr/bin/nice", argv, environ);
205                         fatal_mksh(gettext("Could not load '/usr/bin/nice': %s")
161                         (void) execve(NOCATGETS("/usr/bin/nice"), argv, environ)
162                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 92, "Could not
206                                 errmsg(errno));
207                 } else {
208                         (void) execve(shell->string_mb, argv, environ);
209                         fatal_mksh(gettext("Could not load Shell from '%s': %s")
166                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 93, "Could not
210                                 shell->string_mb,
211                                 errmsg(errno));
212                 }
213         }
214         if (childPid  == -1) {
215                 fatal_mksh(gettext("fork failed: %s"),
172                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 94, "fork failed: %s")
216                       errmsg(errno));
217         }
218         retmem_mb(argv[cmd_argv_index]);
219         return childPid;
220 }

222 /*
223  *      exec_vp(name, argv, envp, ignore_error)
224  *
225  *      Like execve, but does path search.
226  *      This starts command when make invokes it directly (without a shell).
227  *
228  *      Return value:
229  *                              Returns false if the exec failed
230  *
231  *      Parameters:
232  *              name            The name of the command to run
233  *              argv            Arguments for the command
234  *              envp            The environment for it
235  *              ignore_error    Should we abort on error?
236  *
237  *      Global variables used:
238  *              shell_name      The Name "SHELL", used to get the path to shell
239  *              vroot_path      The path used by the vroot package
240  */
241 static Boolean
242 exec_vp(register char *name, register char **argv, char **envp, register Boolean
243 {
244         register Name           shell = getvar(shell_name);
245         register char           *shellname;
```

```
246         char                    *shargv[4];
247         Name                    tmp_shell;
248
249         if (IS_EQUAL(shell->string_mb, "")) {
250                 shell = shell_name;
251         }
252
253         for (int i = 0; i < 5; i++) {
254                 (void) execve_vroot(name,
255                                     argv + 1,
256                                     envp,
257                                     vroot_path,
258                                     VROOT_DEFAULT);
259                 switch (errno) {
260                 case ENOEXEC:
261                 case ENOENT:
262                         /* That failed. Let the shell handle it */
263                         shellname = strrchr(shell->string_mb, (int) slash_char);
264                         if (shellname == NULL) {
265                                 shellname = shell->string_mb;
266                         } else {
267                                 shellname++;
268                         }
269                         shargv[0] = shellname;
270                         shargv[1] = (char*)(ignore_error ? "-c" : "-ce");
227                         shargv[1] = (char*)(ignore_error ? NOCATGETS("-c") : NOC
271                         shargv[2] = argv[0];
272                         shargv[3] = NULL;
273                         tmp_shell = getvar(shell_name);
274                         if (IS_EQUAL(tmp_shell->string_mb, "")) {
275                                 tmp_shell = shell_name;
276                         }
277                         (void) execve_vroot(tmp_shell->string_mb,
278                                             shargv,
279                                             envp,
280                                             vroot_path,
281                                             VROOT_DEFAULT);
282                         return failed;
283                 case ETXTBSY:
284                         /*
285                          * The program is busy (debugged?).
286                          * Wait and then try again.
287                          */
288                         (void) sleep((unsigned) i);
289                 case EAGAIN:
290                         break;
291                 default:
292                         return failed;
293                 }
294         }
295         return failed;
296 }
297
298 /*
299  *      doexec(command, ignore_error)
300  *
301  *      Will scan an argument string and split it into words
302  *      thus building an argument list that can be passed to exec_ve()
303  *
304  *      Return value:
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 int
315 doexec(register wchar_t *command, register Boolean ignore_error, char *stdout_fi
316 {
317         int                     arg_count = 5;
318         char                    **argv;
319         int                     length;
320         char                    nice_prio_buf[MAXPATHLEN];
321         register char           **p;
322         wchar_t                 *q;
323         register wchar_t        *t;
324         char                    *tmp_mbs_buffer;
325
326         /*
327          * Only prepend the /usr/bin/nice command to the original command
328          * if the nice priority, nice_prio, is NOT zero (0).
329          * Nice priorities can be a positive or a negative number.
330          */
331         if (nice_prio != 0) {
332                 arg_count += 2;
333         }
334         for (t = command; *t != (int) nul_char; t++) {
335                 if (iswspace(*t)) {
336                         arg_count++;
337                 }
338         }
339         argv = (char **)alloca(arg_count * (sizeof(char *)));
340         /*
341          * Reserve argv[0] for sh in case of exec_vp failure.
342          * Don't worry about prepending /usr/bin/nice command to argv[0].
343          * In fact, doing it may cause the sh command to fail!
344          */
345         p = &argv[1];
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);
314                 *p++ = strdup(NOCATGETS("/usr/bin/nice"));
315                 (void) sprintf(nice_prio_buf, NOCATGETS("-%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
323                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 95, "Command '
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 = wslen(q)) >= MAXPATHLEN) {
378                         tmp_mbs_buffer = getmem((length * MB_LEN_MAX) + 1);
379                         (void) wcstombs(tmp_mbs_buffer, q, (length * MB_LEN_MAX)
380                         *p++ = strdup(tmp_mbs_buffer);
381                         retmem_mb(tmp_mbs_buffer);
382                 } else {
383                         WCSTOMBS(mbs_buffer, q);
384                         *p++ = strdup(mbs_buffer);
385                 }
386         }
387         *p = NULL;

389         /* Then exec the command with that argument list. */
390         (void) fflush(stdout);
391         if ((childPid = fork()) == 0) {
392                 enable_interrupt((void (*) (int)) SIG_DFL);
393 #if 0
394                 if (filter_stderr) {
395                         redirect_stderr();
396                 }
397 #endif
398                 (void) exec_vp(argv[1], argv, environ, ignore_error, vroot_path)
399                 fatal_mksh(gettext("Cannot load command '%s': %s"), argv[1], err
356                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 96, "Cannot load comma
400         }
401         if (childPid  == -1) {
402                 fatal_mksh(gettext("fork failed: %s"),
359                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 97, "fork failed: %s")
403                       errmsg(errno));
404         }
405         for (int i = 0; argv[i] != NULL; i++) {
406                 retmem_mb(argv[i]);
407         }
408         return childPid;
409 }

411 /*
412  *      await(ignore_error, silent_error, target, command, running_pid)
413  *
414  *      Wait for one child process and analyzes
415  *      the returned status when the child process terminates.
416  *
417  *      Return value:
418  *                              Returns true if commands ran OK
419  *
420  *      Parameters:
421  *              ignore_error    Should we abort on error?
422  *              silent_error    Should error messages be suppressed for dmake?
423  *              target          The target we are building, for error msgs
424  *              command         The command we ran, for error msgs
425  *              running_pid     The pid of the process we are waiting for
426  *
427  *      Static variables used:
428  *              filter_file     The fd for the filter file
429  *              filter_file_name The name of the filter file
430  *
431  *      Global variables used:
432  *              filter_stderr   Set if -X is on
433  */
434 Boolean
435 await(register Boolean ignore_error, register Boolean silent_error, Name target,
436 {
437         int                     status;
```



```
438         char                    *buffer;
439         int                     core_dumped;
440         int                     exit_status;
441         FILE                    *outfp;
442         register pid_t          pid;
443         struct stat             stat_buff;
444         int                     termination_signal;
445         char                    tmp_buf[MAXPATHLEN];

447         while ((pid = wait(&status)) != running_pid) {
448                 if (pid == -1) {
449                         fatal_mksh(gettext("wait() failed: %s"), errmsg(errno));
406                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 98, "wait() fa
450                 }
451         }
452         (void) fflush(stdout);
453         (void) fflush(stderr);

455         if (status == 0) {

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

461                 return succeeded;
462         }

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


469         exit_status = WEXITSTATUS(status);

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

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

478         /*
479          * If the child returned an error, we now try to print a
480          * nice message about it.
481          */
482
483         tmp_buf[0] = (int) nul_char;
484         if (!silent_error) {
485                 if (exit_status != 0) {
486                         (void) fprintf(stdout,
487                                        gettext("*** Error code %d"),
444                                        catgets(libmksdmsi18n_catd, 1, 103, "***
488                                        exit_status);
489                 } else {
490                                 (void) fprintf(stdout,
491                                                gettext("*** Signal %d"),
448                                                catgets(libmksdmsi18n_catd, 1, 10
492                                                termination_signal);
493                         if (core_dumped) {
494                                 (void) fprintf(stdout,
495                                                gettext(" - core dumped"));
452                                                catgets(libmksdmsi18n_catd, 1, 10
496                         }
```

```
497                 }
498                 if (ignore_error) {
499                         (void) fprintf(stdout,
500                                        gettext(" (ignored)"));
457                                        catgets(libmksdmsi18n_catd, 1, 109, " (ig
501                 }
502                 (void) fprintf(stdout, "\n");
503                 (void) fflush(stdout);
504         }

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

510         return failed;
511 }

513 /*
514  *      sh_command2string(command, destination)
515  *
516  *      Run one sh command and capture the output from it.
517  *
518  *      Return value:
519  *
520  *      Parameters:
521  *              command         The command to run
522  *              destination     Where to deposit the output from the command
523  *
524  *      Static variables used:
525  *
526  *      Global variables used:
527  */
528 void
529 sh_command2string(register String command, register String destination)
530 {
531         register FILE           *fd;
532         register int            chr;
533         int                     status;
534         Boolean                 command_generated_output = false;

536         command->text.p = (int) nul_char;
537         WCSTOMBS(mbs_buffer, command->buffer.start);
538         if ((fd = popen(mbs_buffer, "r")) == NULL) {
539                 WCSTOMBS(mbs_buffer, command->buffer.start);
540                 fatal_mksh(gettext("Could not run command '%s' for :sh transform
497                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 111, "Could not run co
541                       mbs_buffer);
542         }
543         while ((chr = getc(fd)) != EOF) {
544                 if (chr == (int) newline_char) {
545                         chr = (int) space_char;
546                 }
547                 command_generated_output = true;
548                 append_char(chr, destination);
549         }

551         /*
552          * We don't want to keep the last LINE_FEED since usually
553          * the output of the 'sh:' command is used to evaluate
554          * some MACRO. ( /bin/sh and other shell add a line feed
555          * to the output so that the prompt appear in the right place.
556          * We don't need that
557          */
558         if (command_generated_output){
559                 if ( *(destination->text.p-1) == (int) space_char) {
```



```
560                         * (-- destination->text.p) = '\0';
561                 }
562         } else {
563                 /*
564                  * If the command didn't generate any output,
565                  * set the buffer to a null string.
566                  */
567                 *(destination->text.p) = '\0';
568         }
569
570         status = pclose(fd);
571         if (status != 0) {
572                 WCSTOMBS(mbs_buffer, command->buffer.start);
573                 fatal_mksh(gettext("The command '%s' returned status '%d'"),
530                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 112, "The command '%s'
574                       mbs_buffer,
575                       WEXITSTATUS(status));
576         }
577 }
______unchanged_portion_omitted_
```

```
**********************************************************
   36511 Wed May 20 12:19:59 2015
new/usr/src/cmd/make/lib/mksh/macro.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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


/*
 *      macro.cc
 *
 *      Handle expansion of make macros
 */

/*
 * Included files
 */
#include <mksh/dosys.h>         /* sh_command2string() */
#include <mksh/i18n.h>          /* get_char_semantics_value() */
#include <mksh/macro.h>
#include <mksh/misc.h>          /* retmem() */
#include <mksh/read.h>          /* get_next_block_fn() */
#include <mksdmsi18n/mksdmsi18n.h>      /* libmksdmsi18n_init() */

#include <widec.h>
#include <libintl.h>
#endif /* ! codereview */

/*
 * File table of contents
 */
static void     add_macro_to_global_list(Name macro_to_add);
static void     expand_value_with_daemon(Name, register Property macro, register

static void     init_arch_macros(void);
static void     init_mach_macros(void);
static Boolean  init_arch_done = false;
static Boolean  init_mach_done = false;


long env_alloc_num = 0;
long env_alloc_bytes = 0;
```



```
/*
 *      getvar(name)
 *
 *      Return expanded value of macro.
 *
 *      Return value:
 *                              The expanded value of the macro
 *
 *      Parameters:
 *              name            The name of the macro we want the value for
 *
 *      Global variables used:
 */
Name
getvar(register Name name)
{
        String_rec              destination;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        register Name           result;

        if ((name == host_arch) || (name == target_arch)) {
                if (!init_arch_done) {
                        init_arch_done = true;
                        init_arch_macros();
                }
        }
        if ((name == host_mach) || (name == target_mach)) {
                if (!init_mach_done) {
                        init_mach_done = true;
                        init_mach_macros();
                }
        }

        INIT_STRING_FROM_STACK(destination, buffer);
        expand_value(maybe_append_prop(name, macro_prop)->body.macro.value,
                     &destination,
                     false);
        result = GETNAME(destination.buffer.start, FIND_LENGTH);
        if (destination.free_after_use) {
                retmem(destination.buffer.start);
        }
        return result;
}

/*
 *      expand_value(value, destination, cmd)
 *
 *      Recursively expands all macros in the string value.
 *      destination is where the expanded value should be appended.
 *
 *      Parameters:
 *              value           The value we are expanding
 *              destination     Where to deposit the expansion
 *              cmd             If we are evaluating a command line we
 *                              turn \ quoting off
 *
 *      Global variables used:
 */
void
expand_value(Name value, register String destination, Boolean cmd)
{
        Source_rec              sourceb;
        register Source         source = &sourceb;
        register wchar_t        *source_p = NULL;
        register wchar_t        *source_end = NULL;
        wchar_t                 *block_start = NULL;
```

```
127         int                     quote_seen = 0;

129         if (value == NULL) {
130                 /*
131                  * Make sure to get a string allocated even if it
132                  * will be empty.
133                  */
134                 MBSTOWCS(wcs_buffer, "");
135                 append_string(wcs_buffer, destination, FIND_LENGTH);
136                 destination->text.end = destination->text.p;
137                 return;
138         }
139         if (!value->dollar) {
140                 /*
141                  * If the value we are expanding does not contain
142                  * any $, we don't have to parse it.
143                  */
144                 APPEND_NAME(value,
145                         destination,
146                         (int) value->hash.length
147                 );
148                 destination->text.end = destination->text.p;
149                 return;
150         }

152         if (value->being_expanded) {
153                 fatal_reader_mksh(gettext("Loop detected when expanding macro va
 44                 fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 113, "Loop dete
154                                 value->string_mb);
155         }
156         value->being_expanded = true;
157         /* Setup the structure we read from */
158         Wstring vals(value);
159         sourceb.string.text.p = sourceb.string.buffer.start = wsdup(vals.get_str
160         sourceb.string.free_after_use = true;
161         sourceb.string.text.end =
162           sourceb.string.buffer.end =
163             sourceb.string.text.p + value->hash.length;
164         sourceb.previous = NULL;
165         sourceb.fd = -1;
166         sourceb.inp_buf =
167           sourceb.inp_buf_ptr =
168             sourceb.inp_buf_end = NULL;
169         sourceb.error_converting = false;
170         /* Lift some pointers from the struct to local register variables */
171         CACHE_SOURCE(0);
172 /* We parse the string in segments */
173 /* We read chars until we find a $, then we append what we have read so far */
174 /* (since last $ processing) to the destination. When we find a $ we call */
175 /* expand_macro() and let it expand that particular $ reference into dest */
176         block_start = source_p;
177         quote_seen = 0;
178         for (; 1; source_p++) {
179                 switch (GET_CHAR()) {
180                 case backslash_char:
181                         /* Quote $ in macro value */
182                         if (!cmd) {
183                                 quote_seen = ~quote_seen;
184                         }
185                         continue;
186                 case dollar_char:
187                         /* Save the plain string we found since */
188                         /* start of string or previous $ */
189                         if (quote_seen) {
190                                 append_string(block_start,
191                                               destination,
```



```
192                                               source_p - block_start - 1);
193                                 block_start = source_p;
194                                 break;
195                         }
196                         append_string(block_start,
197                                       destination,
198                                       source_p - block_start);
199                         source->string.text.p = ++source_p;
200                         UNCACHE_SOURCE();
201                         /* Go expand the macro reference */
202                         expand_macro(source, destination, sourceb.string.buffer.
203                         CACHE_SOURCE(1);
204                         block_start = source_p + 1;
205                         break;
206                 case nul_char:
207                         /* The string ran out. Get some more */
208                         append_string(block_start,
209                                       destination,
210                                       source_p - block_start);
211                         GET_NEXT_BLOCK_NOCHK(source);
212                         if (source == NULL) {
213                                 destination->text.end = destination->text.p;
214                                 value->being_expanded = false;
215                                 return;
216                         }
217                         if (source->error_converting) {
218                                 fatal_reader_mksh("Internal error: Invalid byte
109                                 fatal_reader_mksh(NOCATGETS("Internal error: Inv
219                         }
220                         block_start = source_p;
221                         source_p--;
222                         continue;
223                 }
224                 quote_seen = 0;
225         }
226         retmem(sourceb.string.buffer.start);
227 }

229 /*
230  *      expand_macro(source, destination, current_string, cmd)
231  *
232  *      Should be called with source->string.text.p pointing to
233  *      the first char after the $ that starts a macro reference.
234  *      source->string.text.p is returned pointing to the first char after
235  *      the macro name.
236  *      It will read the macro name, expanding any macros in it,
237  *      and get the value. The value is then expanded.
238  *      destination is a String that is filled in with the expanded macro.
239  *      It may be passed in referencing a buffer to expand the macro into.
240  *      Note that most expansions are done on demand, e.g. right
241  *      before the command is executed and not while the file is
242  *      being parsed.
243  *
244  *      Parameters:
245  *              source          The source block that references the string
246  *                              to expand
247  *              destination     Where to put the result
248  *              current_string  The string we are expanding, for error msg
249  *              cmd             If we are evaluating a command line we
250  *                              turn \ quoting off
251  *
252  *      Global variables used:
253  *              funny           Vector of semantic tags for characters
254  *              is_conditional  Set if a conditional macro is refd
255  *              make_word_mentioned Set if the word "MAKE" is mentioned
256  *              makefile_type   We deliver extra msg when reading makefiles
```

```
 257  *              query           The Name "?", compared against
 258  *              query_mentioned Set if the word "?" is mentioned
 259  */
 260 void
 261 expand_macro(register Source source, register String destination, wchar_t *curre
 262 {
 263         static Name             make = (Name)NULL;
 264         static wchar_t          colon_sh[4];
 265         static wchar_t          colon_shell[7];
 266         String_rec              string;
 267         wchar_t                 buffer[STRING_BUFFER_LENGTH];
 268         register wchar_t        *source_p = source->string.text.p;
 269         register wchar_t        *source_end = source->string.text.end;
 270         register int            closer = 0;
 271         wchar_t                 *block_start = (wchar_t *)NULL;
 272         int                     quote_seen = 0;
 273         register int            closer_level = 1;
 274         Name                    name = (Name)NULL;
 275         wchar_t                 *colon = (wchar_t *)NULL;
 276         wchar_t                 *percent = (wchar_t *)NULL;
 277         wchar_t                 *eq = (wchar_t *) NULL;
 278         Property                macro = NULL;
 279         wchar_t                 *p = (wchar_t*)NULL;
 280         String_rec              extracted;
 281         wchar_t                 extracted_string[MAXPATHLEN];
 282         wchar_t                 *left_head = NULL;
 283         wchar_t                 *left_tail = NULL;
 284         wchar_t                 *right_tail = NULL;
 285         int                     left_head_len = 0;
 286         int                     left_tail_len = 0;
 287         int                     tmp_len = 0;
 288         wchar_t                 *right_hand[128];
 289         int                     i = 0;
 290         enum {
 291                 no_extract,
 292                 dir_extract,
 293                 file_extract
 294         }                       extraction = no_extract;
 295         enum {
 296                 no_replace,
 297                 suffix_replace,
 298                 pattern_replace,
 299                 sh_replace
 300         }                       replacement = no_replace;

 302         if (make == NULL) {
 303                 MBSTOWCS(wcs_buffer, "MAKE");
 194                 MBSTOWCS(wcs_buffer, NOCATGETS("MAKE"));
 304                 make = GETNAME(wcs_buffer, FIND_LENGTH);

 306                 MBSTOWCS(colon_sh, ":sh");
 307                 MBSTOWCS(colon_shell, ":shell");
 197                 MBSTOWCS(colon_sh, NOCATGETS(":sh"));
 198                 MBSTOWCS(colon_shell, NOCATGETS(":shell"));
 308         }

 310         right_hand[0] = NULL;

 312         /* First copy the (macro-expanded) macro name into string. */
 313         INIT_STRING_FROM_STACK(string, buffer);
 314 recheck_first_char:
 315         /* Check the first char of the macro name to figure out what to do. */
 316         switch (GET_CHAR()) {
 317         case nul_char:
 318                 GET_NEXT_BLOCK_NOCHK(source);
 319                 if (source == NULL) {
```



```
 320                         WCSTOMBS(mbs_buffer, current_string);
 321                         fatal_reader_mksh(gettext("'$' at end of string '%s'"),
 212                         fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 114, "'
 322                                 mbs_buffer);
 323                 }
 324                 if (source->error_converting) {
 325                         fatal_reader_mksh("Internal error: Invalid byte sequence
 216                         fatal_reader_mksh(NOCATGETS("Internal error: Invalid byt
 326                 }
 327                 goto recheck_first_char;
 328         case parenleft_char:
 329                 /* Multi char name. */
 330                 closer = (int) parenright_char;
 331                 break;
 332         case braceleft_char:
 333                 /* Multi char name. */
 334                 closer = (int) braceright_char;
 335                 break;
 336         case newline_char:
 337                 fatal_reader_mksh(gettext("'$' at end of line"));
 228                 fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 115, "'$' at en
 338         default:
 339                 /* Single char macro name. Just suck it up */
 340                 append_char(*source_p, &string);
 341                 source->string.text.p = source_p + 1;
 342                 goto get_macro_value;
 343         }

 345         /* Handle multi-char macro names */
 346         block_start = ++source_p;
 347         quote_seen = 0;
 348         for (; 1; source_p++) {
 349                 switch (GET_CHAR()) {
 350                 case nul_char:
 351                         append_string(block_start,
 352                                       &string,
 353                                       source_p - block_start);
 354                         GET_NEXT_BLOCK_NOCHK(source);
 355                         if (source == NULL) {
 356                                 if (current_string != NULL) {
 357                                         WCSTOMBS(mbs_buffer, current_string);
 358                                         fatal_reader_mksh(gettext("Unmatched '%c
 249                                         fatal_reader_mksh(catgets(libmksdmsi18n_
 359                                                           closer ==
 360                                                           (int) braceright_char ?
 361                                                           (int) braceleft_char :
 362                                                           (int) parenleft_char,
 363                                                           mbs_buffer);
 364                                 } else {
 365                                         fatal_reader_mksh(gettext("Premature EOF
 256                                         fatal_reader_mksh(catgets(libmksdmsi18n_
 366                                 }
 367                         }
 368                         if (source->error_converting) {
 369                                 fatal_reader_mksh("Internal error: Invalid byte
 260                                 fatal_reader_mksh(NOCATGETS("Internal error: Inv
 370                         }
 371                         block_start = source_p;
 372                         source_p--;
 373                         continue;
 374                 case newline_char:
 375                         fatal_reader_mksh(gettext("Unmatched '%c' on line"),
 266                         fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 118, "U
 376                                 closer == (int) braceright_char ?
 377                                 (int) braceleft_char :
 378                                 (int) parenleft_char);
```

```
379                 case backslash_char:
380                         /* Quote dollar in macro value. */
381                         if (!cmd) {
382                                 quote_seen = ~quote_seen;
383                         }
384                         continue;
385                 case dollar_char:
386                         /*
387                          * Macro names may reference macros.
388                          * This expands the value of such macros into the
389                          * macro name string.
390                          */
391                         if (quote_seen) {
392                                 append_string(block_start,
393                                               &string,
394                                               source_p - block_start - 1);
395                                 block_start = source_p;
396                                 break;
397                         }
398                         append_string(block_start,
399                                       &string,
400                                       source_p - block_start);
401                         source->string.text.p = ++source_p;
402                         UNCACHE_SOURCE();
403                         expand_macro(source, &string, current_string, cmd);
404                         CACHE_SOURCE(0);
405                         block_start = source_p;
406                         source_p--;
407                         break;
408                 case parenleft_char:
409                         /* Allow nested pairs of () in the macro name. */
410                         if (closer == (int) parenright_char) {
411                                 closer_level++;
412                         }
413                         break;
414                 case braceleft_char:
415                         /* Allow nested pairs of {} in the macro name. */
416                         if (closer == (int) braceright_char) {
417                                 closer_level++;
418                         }
419                         break;
420                 case parenright_char:
421                 case braceright_char:
422                         /*
423                          * End of the name. Save the string in the macro
424                          * name string.
425                          */
426                         if ((*source_p == closer) && (--closer_level <= 0)) {
427                                 source->string.text.p = source_p + 1;
428                                 append_string(block_start,
429                                               &string,
430                                               source_p - block_start);
431                                 goto get_macro_value;
432                         }
433                         break;
434                 }
435                 quote_seen = 0;
436         }
437         /*
438          * We got the macro name. We now inspect it to see if it
439          * specifies any translations of the value.
440          */
441 get_macro_value:
442         name = NULL;
443         /* First check if we have a $(@D) type translation. */
444         if ((get_char_semantics_value(string.buffer.start[0]) &
```



```
445              (int) special_macro_sem) &&
446             (string.text.p - string.buffer.start >= 2) &&
447             ((string.buffer.start[1] == 'D') ||
448              (string.buffer.start[1] == 'F'))) {
449                 switch (string.buffer.start[1]) {
450                 case 'D':
451                         extraction = dir_extract;
452                         break;
453                 case 'F':
454                         extraction = file_extract;
455                         break;
456                 default:
457                         WCSTOMBS(mbs_buffer, string.buffer.start);
458                         fatal_reader_mksh(gettext("Illegal macro reference '%s'"
349                         fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1, 119, "I
459                                           mbs_buffer);
460                 }
461                 /* Internalize the macro name using the first char only. */
462                 name = GETNAME(string.buffer.start, 1);
463                 (void) wscpy(string.buffer.start, string.buffer.start + 2);
464         }
465         /* Check for other kinds of translations. */
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 *) wschr(colon + 1,
481                                                          (int) percent_char)) ==
482                         while (colon != NULL) {
483                                 if ((eq = (wchar_t *) wschr(colon + 1,
484                                                             (int) equal_char)) =
485                                         fatal_reader_mksh(gettext("= missing fro
376                                         fatal_reader_mksh(catgets(libmksdmsi18n_
486                                 }
487                                 left_tail_len = eq - colon - 1;
488                                 if(left_tail) {
489                                         retmem(left_tail);
490                                 }
491                                 left_tail = ALLOC_WC(left_tail_len + 1);
492                                 (void) wsncpy(left_tail,
493                                               colon + 1,
494                                               eq - colon - 1);
495                                 left_tail[eq - colon - 1] = (int) nul_char;
496                                 replacement = suffix_replace;
497                                 if ((colon = (wchar_t *) wschr(eq + 1,
498                                                                (int) colon_char)
499                                         tmp_len = colon - eq;
500                                         if(right_tail) {
501                                                 retmem(right_tail);
502                                         }
503                                         right_tail = ALLOC_WC(tmp_len);
504                                         (void) wsncpy(right_tail,
505                                                       eq + 1,
506                                                       colon - eq - 1);
507                                         right_tail[colon - eq - 1] =
508                                           (int) nul_char;
```

```
509                                 } else {
510                                         if(right_tail) {
511                                                 retmem(right_tail);
512                                         }
513                                         right_tail = ALLOC_WC(wslen(eq) + 1);
514                                         (void) wscpy(right_tail, eq + 1);
515                                 }
516                         }
517                 } else {
518                         if ((eq = (wchar_t *) wschr(colon + 1,
519                                                     (int) equal_char)) == NULL)
520                                 fatal_reader_mksh(gettext("= missing from replac
411                                 fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1,
521                         }
522                         if ((percent = (wchar_t *) wschr(colon + 1,
523                                                          (int) percent_char)) ==
524                                 fatal_reader_mksh(gettext("%% missing from repla
415                                 fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1,
525                         }
526                         if (eq < percent) {
527                                 fatal_reader_mksh(gettext("%% missing from repla
418                                 fatal_reader_mksh(catgets(libmksdmsi18n_catd, 1,
528                         }

530                         if (percent > (colon + 1)) {
531                                 tmp_len = percent - colon;
532                                 if(left_head) {
533                                         retmem(left_head);
534                                 }
535                                 left_head = ALLOC_WC(tmp_len);
536                                 (void) wsncpy(left_head,
537                                               colon + 1,
538                                               percent - colon - 1);
539                                 left_head[percent-colon-1] = (int) nul_char;
540                                 left_head_len = percent-colon-1;
541                         } else {
542                                 left_head = NULL;
543                                 left_head_len = 0;
544                         }

546                         if (eq > percent+1) {
547                                 tmp_len = eq - percent;
548                                 if(left_tail) {
549                                         retmem(left_tail);
550                                 }
551                                 left_tail = ALLOC_WC(tmp_len);
552                                 (void) wsncpy(left_tail,
553                                               percent + 1,
554                                               eq - percent - 1);
555                                 left_tail[eq-percent-1] = (int) nul_char;
556                                 left_tail_len = eq-percent-1;
557                         } else {
558                                 left_tail = NULL;
559                                 left_tail_len = 0;
560                         }

562                         if ((percent = (wchar_t *) wschr(++eq,
563                                                          (int) percent_char)) ==

565                                 right_hand[0] = ALLOC_WC(wslen(eq) + 1);
566                                 right_hand[1] = NULL;
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) wsncpy(right_hand[i],
573                                                       eq,
574                                                       percent - eq);
575                                         right_hand[i][percent-eq] =
576                                           (int) nul_char;
577                                         if (i++ >= VSIZEOF(right_hand)) {
578                                                 fatal_mksh(gettext("Too many %%
469                                                 fatal_mksh(catgets(libmksdmsi18n
579                                         }
580                                         eq = percent + 1;
581                                         if (eq[0] == (int) nul_char) {
582                                                 MBSTOWCS(wcs_buffer, "");
583                                                 right_hand[i] = (wchar_t *) wsdu
584                                                 i++;
585                                                 break;
586                                         }
587                                 } while ((percent = (wchar_t *) wschr(eq, (int)
588                                 if (eq[0] != (int) nul_char) {
589                                         right_hand[i] = ALLOC_WC(wslen(eq) + 1);
590                                         (void) wscpy(right_hand[i], eq);
591                                         i++;
592                                 }
593                                 right_hand[i] = NULL;
594                         }
595                         replacement = pattern_replace;
596                 }
597         }
598         if (name == NULL) {
599                 /*
600                  * No translations found.
601                  * Use the whole string as the macro name.
602                  */
603                 name = GETNAME(string.buffer.start,
604                                string.text.p - string.buffer.start);
605         }
606         if (string.free_after_use) {
607                 retmem(string.buffer.start);
608         }
609         if (name == make) {
610                 make_word_mentioned = true;
611         }
612         if (name == query) {
613                 query_mentioned = true;
614         }
615         if ((name == host_arch) || (name == target_arch)) {
616                 if (!init_arch_done) {
617                         init_arch_done = true;
618                         init_arch_macros();
619                 }
620         }
621         if ((name == host_mach) || (name == target_mach)) {
622                 if (!init_mach_done) {
623                         init_mach_done = true;
624                         init_mach_macros();
625                 }
626         }
627         /* Get the macro value. */
628         macro = get_prop(name->prop, macro_prop);
629         if ((macro != NULL) && macro->body.macro.is_conditional) {
630                 conditional_macro_used = true;
631                 /*
632                  * Add this conditional macro to the beginning of the
633                  * global list.
634                  */
635                 add_macro_to_global_list(name);
636                 if (makefile_type == reading_makefile) {
```

```
637                     warning_mksh(gettext("Conditional macro `%s' referenced
528                     warning_mksh(catgets(libmksdmsi18n_catd, 1, 164, "Condit
638                                     name->string_mb, file_being_read, line_n
639             }
640         }
641         /* Macro name read and parsed. Expand the value. */
642         if ((macro == NULL) || (macro->body.macro.value == NULL)) {
643                 /* If the value is empty, we just get out of here. */
644                 goto exit;
645         }
646         if (replacement == sh_replace) {
647                 /* If we should do a :sh transform, we expand the command
648                  * and process it.
649                  */
650                 INIT_STRING_FROM_STACK(string, buffer);
651                 /* Expand the value into a local string buffer and run cmd. */
652                 expand_value_with_daemon(name, macro, &string, cmd);
653                 sh_command2string(&string, destination);
654         } else if ((replacement != no_replace) || (extraction != no_extract)) {
655                 /*
656                  * If there were any transforms specified in the macro
657                  * name, we deal with them here.
658                  */
659                 INIT_STRING_FROM_STACK(string, buffer);
660                 /* Expand the value into a local string buffer. */
661                 expand_value_with_daemon(name, macro, &string, cmd);
662                 /* Scan the expanded string. */
663                 p = string.buffer.start;
664                 while (*p != (int) nul_char) {
665                         wchar_t         chr;

667                         /*
668                          * First skip over any white space and append
669                          * that to the destination string.
670                          */
671                         block_start = p;
672                         while ((*p != (int) nul_char) && iswspace(*p)) {
673                                 p++;
674                         }
675                         append_string(block_start,
676                                       destination,
677                                       p - block_start);
678                         /* Then find the end of the next word. */
679                         block_start = p;
680                         while ((*p != (int) nul_char) && !iswspace(*p)) {
681                                 p++;
682                         }
683                         /* If we cant find another word we are done */
684                         if (block_start == p) {
685                                 break;
686                         }
687                         /* Then apply the transforms to the word */
688                         INIT_STRING_FROM_STACK(extracted, extracted_string);
689                         switch (extraction) {
690                         case dir_extract:
691                                 /*
692                                  * $(@D) type transform. Extract the
693                                  * path from the word. Deliver "." if
694                                  * none is found.
695                                  */
696                                 if (p != NULL) {
697                                         chr = *p;
698                                         *p = (int) nul_char;
699                                 }
700                                 eq = (wchar_t *) wsrchr(block_start, (int) slash
701                                 if (p != NULL) {
```



```
702                                         *p = chr;
703                                 }
704                                 if ((eq == NULL) || (eq > p)) {
705                                         MBSTOWCS(wcs_buffer, ".");
706                                         append_string(wcs_buffer, &extracted, 1)
707                                 } else {
708                                         append_string(block_start,
709                                                       &extracted,
710                                                       eq - block_start);
711                                 }
712                                 break;
713                         case file_extract:
714                                 /*
715                                  * $(@F) type transform. Remove the path
716                                  * from the word if any.
717                                  */
718                                 if (p != NULL) {
719                                         chr = *p;
720                                         *p = (int) nul_char;
721                                 }
722                                 eq = (wchar_t *) wsrchr(block_start, (int) slash
723                                 if (p != NULL) {
724                                         *p = chr;
725                                 }
726                                 if ((eq == NULL) || (eq > p)) {
727                                         append_string(block_start,
728                                                       &extracted,
729                                                       p - block_start);
730                                 } else {
731                                         append_string(eq + 1,
732                                                       &extracted,
733                                                       p - eq - 1);
734                                 }
735                                 break;
736                         case no_extract:
737                                 append_string(block_start,
738                                               &extracted,
739                                               p - block_start);
740                                 break;
741                         }
742                         switch (replacement) {
743                         case suffix_replace:
744                                 /*
745                                  * $(FOO:.o=.c) type transform.
746                                  * Maybe replace the tail of the word.
747                                  */
748                                 if (((extracted.text.p -
749                                       extracted.buffer.start) >=
750                                      left_tail_len) &&
751                                     IS_WEQUALN(extracted.text.p - left_tail_len,
752                                                left_tail,
753                                                left_tail_len)) {
754                                         append_string(extracted.buffer.start,
755                                                       destination,
756                                                       (extracted.text.p -
757                                                        extracted.buffer.start)
758                                                       - left_tail_len);
759                                         append_string(right_tail,
760                                                       destination,
761                                                       FIND_LENGTH);
762                                 } else {
763                                         append_string(extracted.buffer.start,
764                                                       destination,
765                                                       FIND_LENGTH);
766                                 }
767                                 break;
```

```
768                         case pattern_replace:
769                                 /* $(X:a%b=c%d) type transform. */
770                                 if (((extracted.text.p -
771                                       extracted.buffer.start) >=
772                                      left_head_len+left_tail_len) &&
773                                     IS_WEQUALN(left_head,
774                                             extracted.buffer.start,
775                                             left_head_len) &&
776                                     IS_WEQUALN(left_tail,
777                                             extracted.text.p - left_tail_len,
778                                             left_tail_len)) {
779                                         i = 0;
780                                         while (right_hand[i] != NULL) {
781                                           append_string(right_hand[i],
782                                                         destination,
783                                                         FIND_LENGTH);
784                                           i++;
785                                           if (right_hand[i] != NULL) {
786                                                   append_string(extracted.
787                                                                 start +
788                                                                 left_head_
789                                                                 destinatio
790                                                                 (extracted
791                                           }
792                                         }
793                                 } else {
794                                         append_string(extracted.buffer.start,
795                                                       destination,
796                                                       FIND_LENGTH);
797                                 }
798                                 break;
799                         case no_replace:
800                                 append_string(extracted.buffer.start,
801                                               destination,
802                                               FIND_LENGTH);
803                                 break;
804                         case sh_replace:
805                                 break;
806                             }
807                 }
808                 if (string.free_after_use) {
809                         retmem(string.buffer.start);
810                 }
811         } else {
812                 /*
813                  * This is for the case when the macro name did not
814                  * specify transforms.
815                  */
816                 if (!strncmp(name->string_mb, "GET", 3)) {
707                 if (!strncmp(name->string_mb, NOCATGETS("GET"), 3)) {
817                         dollarget_seen = true;
818                 }
819                 dollarless_flag = false;
820                 if (!strncmp(name->string_mb, "<", 1) &&
821                     dollarget_seen) {
822                         dollarless_flag = true;
823                         dollarget_seen = false;
824                 }
825                 expand_value_with_daemon(name, macro, destination, cmd);
826         }
827 exit:
828         if(left_tail) {
829                 retmem(left_tail);
830         }
831         if(right_tail) {
832                 retmem(right_tail);
```



```
833         }
834         if(left_head) {
835                 retmem(left_head);
836         }
837         i = 0;
838         while (right_hand[i] != NULL) {
839                 retmem(right_hand[i]);
840                 i++;
841         }
842         *destination->text.p = (int) nul_char;
843         destination->text.end = destination->text.p;
844 }
```

_____***unchanged_portion_omitted_***

```
885 /*
886  *      init_arch_macros(void)
887  *
888  *      Set the magic macros TARGET_ARCH, HOST_ARCH,
889  *
890  *      Parameters:
891  *
892  *      Global variables used:
893  *                              host_arch   Property for magic macro HOST_ARCH
894  *                              target_arch Property for magic macro TARGET_ARCH
895  *
896  *      Return value:
897  *                              The function does not return a value, but can
898  *                              call fatal() in case of error.
899  */
900 static void
901 init_arch_macros(void)
902 {
903         String_rec      result_string;
904         wchar_t         wc_buf[STRING_BUFFER_LENGTH];
905         char            mb_buf[STRING_BUFFER_LENGTH];
906         FILE            *pipe;
907         Name            value;
908         int             set_host, set_target;
909         const char      *mach_command = "/bin/mach";
800         const char      *mach_command = NOCATGETS("/bin/mach");

911         set_host = (get_prop(host_arch->prop, macro_prop) == NULL);
912         set_target = (get_prop(target_arch->prop, macro_prop) == NULL);

914         if (set_host || set_target) {
915                 INIT_STRING_FROM_STACK(result_string, wc_buf);
916                 append_char((int) hyphen_char, &result_string);

918                 if ((pipe = popen(mach_command, "r")) == NULL) {
919                         fatal_mksh(gettext("Execute of %s failed"), mach_command
810                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 185, "Execute
920                 }
921                 while (fgets(mb_buf, sizeof(mb_buf), pipe) != NULL) {
922                         MBSTOWCS(wcs_buffer, mb_buf);
923                         append_string(wcs_buffer, &result_string, wslen(wcs_buff
924                 }
925                 if (pclose(pipe) != 0) {
926                         fatal_mksh(gettext("Execute of %s failed"), mach_command
817                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 186, "Execute
927                 }

929                 value = GETNAME(result_string.buffer.start, wslen(result_string.

931                 if (set_host) {
932                         (void) setvar_daemon(host_arch, value, false, no_daemon,
933                 }
```

```
 934                 if (set_target) {
 935                         (void) setvar_daemon(target_arch, value, false, no_daemo
 936                 }
 937         }
 938 }

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

 966         set_host = (get_prop(host_mach->prop, macro_prop) == NULL);
 967         set_target = (get_prop(target_mach->prop, macro_prop) == NULL);

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

 973                 if ((pipe = popen(arch_command, "r")) == NULL) {
 974                         fatal_mksh(gettext("Execute of %s failed"), arch_command
 865                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 183, "Execute
 975                 }
 976                 while (fgets(mb_buf, sizeof(mb_buf), pipe) != NULL) {
 977                         MBSTOWCS(wcs_buffer, mb_buf);
 978                         append_string(wcs_buffer, &result_string, wslen(wcs_buff
 979                 }
 980                 if (pclose(pipe) != 0) {
 981                         fatal_mksh(gettext("Execute of %s failed"), arch_command
 872                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 184, "Execute
 982                 }

 984                 value = GETNAME(result_string.buffer.start, wslen(result_string.

 986                 if (set_host) {
 987                         (void) setvar_daemon(host_mach, value, false, no_daemon,
 988                 }
 989                 if (set_target) {
 990                         (void) setvar_daemon(target_mach, value, false, no_daemo
 991                 }
 992         }
 993 }
_____unchanged_portion_omitted_

1046 /*
```



```
1047  * We use a permanent buffer to reset SUNPRO_DEPENDENCIES value.
1048  */
1049 char    *sunpro_dependencies_buf = NULL;
1050 char    *sunpro_dependencies_oldbuf = NULL;
1051 int     sunpro_dependencies_buf_size = 0;

1053 /*
1054  *      setvar_daemon(name, value, append, daemon, strip_trailing_spaces)
1055  *
1056  *      Set a macro value, possibly supplying a daemon to be used
1057  *      when referencing the value.
1058  *
1059  *      Return value:
1060  *                              The property block with the new value
1061  *
1062  *      Parameters:
1063  *              name            Name of the macro to set
1064  *              value           The value to set
1065  *              append          Should we reset or append to the current value?
1066  *              daemon          Special treatment when reading the value
1067  *              strip_trailing_spaces from the end of value->string
1068  *              debug_level     Indicates how much tracing we should do
1069  *
1070  *      Global variables used:
1071  *              makefile_type   Used to check if we should enforce read only
1072  *              path_name       The Name "PATH", compared against
1073  *              virtual_root    The Name "VIRTUAL_ROOT", compared against
1074  *              vpath_defined   Set if the macro VPATH is set
1075  *              vpath_name      The Name "VPATH", compared against
1076  *              envvar          A list of environment vars with $ in value
1077  */
1078 Property
1079 setvar_daemon(register Name name, register Name value, Boolean append, Daemon da
1080 {
1081         register Property       macro = maybe_append_prop(name, macro_prop);
1082         register Property       macro_apx = get_prop(name->prop, macro_append_pr
1083         int                     length = 0;
1084         String_rec              destination;
1085         wchar_t                 buffer[STRING_BUFFER_LENGTH];
1086         register Chain          chain;
1087         Name                    val;
1088         wchar_t                 *val_string = (wchar_t*)NULL;
1089         Wstring                 wcb;


1092         if ((makefile_type != reading_nothing) &&
1093             macro->body.macro.read_only) {
1094                 return macro;
1095         }
1096         /* Strip spaces from the end of the value */
1097         if (daemon == no_daemon) {
1098                 if(value != NULL) {
1099                         wcb.init(value);
1100                         length = wcb.length();
1101                         val_string = wcb.get_string();
1102                 }
1103                 if ((length > 0) && iswspace(val_string[length-1])) {
1104                         INIT_STRING_FROM_STACK(destination, buffer);
1105                         buffer[0] = 0;
1106                         append_string(val_string, &destination, length);
1107                         if (strip_trailing_spaces) {
1108                                 while ((length > 0) &&
1109                                        iswspace(destination.buffer.start[length-
1110                                         destination.buffer.start[--length] = 0;
1111                                 }
1112                         }
```

```
1113                        value = GETNAME(destination.buffer.start, FIND_LENGTH);
1114                }
1115        }
1116
1117        if(macro_apx != NULL) {
1118                val = macro_apx->body.macro_appendix.value;
1119        } else {
1120                val = macro->body.macro.value;
1121        }

1123        if (append) {
1124                /*
1125                 * If we are appending, we just tack the new value after
1126                 * the old one with a space in between.
1127                 */
1128                INIT_STRING_FROM_STACK(destination, buffer);
1129                buffer[0] = 0;
1130                if ((macro != NULL) && (val != NULL)) {
1131                        APPEND_NAME(val,
1132                                    &destination,
1133                                    (int) val->hash.length);
1134                        if (value != NULL) {
1135                                wcb.init(value);
1136                                if(wcb.length() > 0) {
1137                                        MBTOWC(wcs_buffer, " ");
1138                                        append_char(wcs_buffer[0], &destination)
1139                                }
1140                        }
1141                }
1142                if (value != NULL) {
1143                        APPEND_NAME(value,
1144                                    &destination,
1145                                    (int) value->hash.length);
1146                }
1147                value = GETNAME(destination.buffer.start, FIND_LENGTH);
1148                wcb.init(value);
1149                if (destination.free_after_use) {
1150                        retmem(destination.buffer.start);
1151                }
1152        }

1154        /* Debugging trace */
1155        if (debug_level > 1) {
1156                if (value != NULL) {
1157                        switch (daemon) {
1158                        case chain_daemon:
1159                                (void) printf("%s =", name->string_mb);
1160                                for (chain = (Chain) value;
1161                                     chain != NULL;
1162                                     chain = chain->next) {
1163                                        (void) printf(" %s", chain->name->string
1164                                }
1165                                (void) printf("\n");
1166                                break;
1167                        case no_daemon:
1168                                (void) printf("%s= %s\n",
1169                                              name->string_mb,
1170                                              value->string_mb);
1171                                break;
1172                        }
1173                } else {
1174                        (void) printf("%s =\n", name->string_mb);
1175                }
1176        }
1177        /* Set the new values in the macro property block */
1178 /**/
```



```
1179        if(macro_apx != NULL) {
1180                macro_apx->body.macro_appendix.value = value;
1181                INIT_STRING_FROM_STACK(destination, buffer);
1182                buffer[0] = 0;
1183                if (value != NULL) {
1184                        APPEND_NAME(value,
1185                                    &destination,
1186                                    (int) value->hash.length);
1187                        if (macro_apx->body.macro_appendix.value_to_append != NU
1188                                MBTOWC(wcs_buffer, " ");
1189                                append_char(wcs_buffer[0], &destination);
1190                        }
1191                }
1192                if (macro_apx->body.macro_appendix.value_to_append != NULL) {
1193                        APPEND_NAME(macro_apx->body.macro_appendix.value_to_appe
1194                                    &destination,
1195                                    (int) macro_apx->body.macro_appendix.value
1196                }
1197                value = GETNAME(destination.buffer.start, FIND_LENGTH);
1198                if (destination.free_after_use) {
1199                        retmem(destination.buffer.start);
1200                }
1201        }
1202 /**/
1203        macro->body.macro.value = value;
1204        macro->body.macro.daemon = daemon;
1205        /*
1206         * If the user changes the VIRTUAL_ROOT, we need to flush
1207         * the vroot package cache.
1208         */
1209        if (name == path_name) {
1210                flush_path_cache();
1211        }
1212        if (name == virtual_root) {
1213                flush_vroot_cache();
1214        }
1215        /* If this sets the VPATH we remember that */
1216        if ((name == vpath_name) &&
1217            (value != NULL) &&
1218            (value->hash.length > 0)) {
1219                vpath_defined = true;
1220        }
1221        /*
1222         * For environment variables we also set the
1223         * environment value each time.
1224         */
1225        if (macro->body.macro.exported) {
1226                static char     *env;

1228                if (!reading_environment && (value != NULL)) {
1229                        Envvar  p;

1231                        for (p = envvar; p != NULL; p = p->next) {
1232                                if (p->name == name) {
1233                                        p->value = value;
1234                                        p->already_put = false;
1235                                        goto found_it;
1236                                }
1237                        }
1238                        p = ALLOC(Envvar);
1239                        p->name = name;
1240                        p->value = value;
1241                        p->next = envvar;
1242                        p->env_string = NULL;
1243                        p->already_put = false;
1244                        envvar = p;
```

```
found_it:;
                } if (reading_environment || (value == NULL) || !value->dollar)
                        length = 2 + strlen(name->string_mb);
                        if (value != NULL) {
                                length += strlen(value->string_mb);
                        }
                        Property env_prop = maybe_append_prop(name, env_mem_prop
                        /*
                         * We use a permanent buffer to reset SUNPRO_DEPENDENCIE
                         */
                        if (!strncmp(name->string_mb, "SUNPRO_DEPENDENCIES", 19)
                        if (!strncmp(name->string_mb, NOCATGETS("SUNPRO_DEPENDEN
                                if (length >= sunpro_dependencies_buf_size) {
                                        sunpro_dependencies_buf_size=length*2;
                                        if (sunpro_dependencies_buf_size < 4096)
                                                sunpro_dependencies_buf_size = 4
                                        if (sunpro_dependencies_buf)
                                                sunpro_dependencies_oldbuf = sun
                                        sunpro_dependencies_buf=getmem(sunpro_de
                                }
                                env = sunpro_dependencies_buf;
                        } else {
                                env = getmem(length);
                        }
                        env_alloc_num++;
                        env_alloc_bytes += length;
                        (void) sprintf(env,
                                        "%s=%s",
                                        name->string_mb,
                                        value == NULL ?
                                          "" : value->string_mb);
                        (void) putenv(env);
                        env_prop->body.env_mem.value = env;
                        if (sunpro_dependencies_oldbuf) {
                                /* Return old buffer */
                                retmem_mb(sunpro_dependencies_oldbuf);
                                sunpro_dependencies_oldbuf = NULL;
                        }
                }
        }
        if (name == target_arch) {
                Name            ha = getvar(host_arch);
                Name            ta = getvar(target_arch);
                Name            vr = getvar(virtual_root);
                int             length;
                wchar_t         *new_value;
                wchar_t         *old_vr;
                Boolean         new_value_allocated = false;

                Wstring         ha_str(ha);
                Wstring         ta_str(ta);
                Wstring         vr_str(vr);

                wchar_t * wcb_ha = ha_str.get_string();
                wchar_t * wcb_ta = ta_str.get_string();
                wchar_t * wcb_vr = vr_str.get_string();

                length = 32 +
                  wslen(wcb_ha) +
                    wslen(wcb_ta) +
                      wslen(wcb_vr);
                old_vr = wcb_vr;
                MBSTOWCS(wcs_buffer, "/usr/arch/");
                MBSTOWCS(wcs_buffer, NOCATGETS("/usr/arch/"));
                if (IS_WEQUALN(old_vr,
                                wcs_buffer,
```



```
                                wslen(wcs_buffer))) {
                        old_vr = (wchar_t *) wschr(old_vr, (int) colon_char) + 1
                }
                if ( (ha == ta) || (wslen(wcb_ta) == 0) ) {
                        new_value = old_vr;
                } else {
                        new_value = ALLOC_WC(length);
                        new_value_allocated = true;
                        WCSTOMBS(mbs_buffer, old_vr);
                        (void) wsprintf(new_value,
                                        "/usr/arch/%s/%s:%s",
                                        NOCATGETS("/usr/arch/%s/%s:%s"),
                                        ha->string_mb + 1,
                                        ta->string_mb + 1,
                                        mbs_buffer);
                }
                if (new_value[0] != 0) {
                        (void) setvar_daemon(virtual_root,
                                             GETNAME(new_value, FIND_LENGTH),
                                             false,
                                             no_daemon,
                                             true,
                                             debug_level);
                }
                if (new_value_allocated) {
                        retmem(new_value);
                }
        }
        return macro;
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   24131 Wed May 20 12:19:59 2015
new/usr/src/cmd/make/lib/mksh/misc.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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


/*
 *      misc.cc
 *
 *      This file contains various unclassified routines. Some main groups:
 *              getname
 *              Memory allocation
 *              String handling
 *              Property handling
 *              Error message handling
 *              Make internal state dumping
 *              main routine support
 */

/*
 * Included files
 */
#include <bsd/bsd.h>            /* bsd_signal() */
#include <mksh/i18n.h>          /* get_char_semantics_value() */
#include <mksh/misc.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <stdarg.h>             /* va_list, va_start(), va_end() */
#include <stdlib.h>             /* mbstowcs() */
#include <sys/signal.h>         /* SIG_DFL */
#include <sys/wait.h>           /* wait() */

#include <string.h>             /* strerror() */
#include <libintl.h>
#endif /* ! codereview */


/*
 * Defined macros
 */

/*
```



```
 * typedefs & structs
 */

/*
 * Static variables
 */
extern "C" {
        void            (*sigivalue)(int) = SIG_DFL;
        void            (*sigqvalue)(int) = SIG_DFL;
        void            (*sigtvalue)(int) = SIG_DFL;
        void            (*sighvalue)(int) = SIG_DFL;
}

long    getname_bytes_count = 0;
long    getname_names_count = 0;
long    getname_struct_count = 0;

long    freename_bytes_count = 0;
long    freename_names_count = 0;
long    freename_struct_count = 0;

long    expandstring_count = 0;
long    getwstring_count = 0;

/*
 * File table of contents
 */
static void     expand_string(register String string, register int length);

#define FATAL_ERROR_MSG_SIZE 200

/*
 *      getmem(size)
 *
 *      malloc() version that checks the returned value.
 *
 *      Return value:
 *                              The memory chunk we allocated
 *
 *      Parameters:
 *              size            The size of the chunk we need
 *
 *      Global variables used:
 */
char *
getmem(register int size)
{
        register char           *result = (char *) malloc((unsigned) size);
        if (result == NULL) {
                char buf[FATAL_ERROR_MSG_SIZE];
                sprintf(buf, "*** Error: malloc(%d) failed: %s\n", size, strerro
                strcat(buf, gettext("mksh: Fatal error: Out of memory\n"));
                sprintf(buf, NOCATGETS("*** Error: malloc(%d) failed: %s\n"), si
                strcat(buf, catgets(libmksdmsi18n_catd, 1, 126, "mksh: Fatal err
                fputs(buf, stderr);
                exit_status = 1;
                exit(1);
        }
        return result;
}
_____unchanged_portion_omitted_

/*
 *      errmsg(errnum)
 *
 *      Return the error message for a system call error
```

```
 *
 *      Return value:
 *                              An error message string
 *
 *      Parameters:
 *              errnum          The number of the error we want to describe
 *
 *      Global variables used:
 *              sys_errlist     A vector of error messages
 *              sys_nerr        The size of sys_errlist
 */
char *
errmsg(int errnum)
{

        extern int              sys_nerr;
        char                    *errbuf;

        if ((errnum < 0) || (errnum > sys_nerr)) {
                errbuf = getmem(6+1+11+1);
                (void) sprintf(errbuf, gettext("Error %d"), errnum);
                (void) sprintf(errbuf, catgets(libmksdmsi18n_catd, 1, 127, "Erro
                return errbuf;
        } else {
                return strerror(errnum);

        }
}

static char static_buf[MAXPATHLEN*3];

/*
 *      fatal_mksh(format, args...)
 *
 *      Print a message and die
 *
 *      Parameters:
 *              format          printf type format string
 *              args            Arguments to match the format
 */
/*VARARGS*/
void
fatal_mksh(const char *message, ...)
{
        va_list args;
        char    *buf = static_buf;
        char    *mksh_fat_err = gettext("mksh: Fatal error: ");
        char    *cur_wrk_dir = gettext("Current working directory: ");
        char    *mksh_fat_err = catgets(libmksdmsi18n_catd, 1, 128, "mksh: Fatal
        char    *cur_wrk_dir = catgets(libmksdmsi18n_catd, 1, 129, "Current work
        int     mksh_fat_err_len = strlen(mksh_fat_err);

        va_start(args, message);
        (void) fflush(stdout);
        (void) strcpy(buf, mksh_fat_err);
        size_t buf_len = vsnprintf(static_buf + mksh_fat_err_len,
                                   sizeof(static_buf) - mksh_fat_err_len,
                                   message, args)
                         + mksh_fat_err_len
                         + strlen(cur_wrk_dir)
                         + strlen(get_current_path_mksh())
                         + 3; // "\n\n"
        va_end(args);
        if (buf_len >= sizeof(static_buf)) {
                buf = getmem(buf_len);
                (void) strcpy(buf, mksh_fat_err);
```



```
                va_start(args, message);
                (void) vsprintf(buf + mksh_fat_err_len, message, args);
                va_end(args);
        }
        (void) strcat(buf, "\n");
/*
        if (report_pwd) {
 */
        if (1) {
                (void) strcat(buf, cur_wrk_dir);
                (void) strcat(buf, get_current_path_mksh());
                (void) strcat(buf, "\n");
        }
        (void) fputs(buf, stderr);
        (void) fflush(stderr);
        if (buf != static_buf) {
                retmem_mb(buf);
        }
        exit_status = 1;
        exit(1);
}

/*
 *      fatal_reader_mksh(format, args...)
 *
 *      Parameters:
 *              format          printf style format string
 *              args            arguments to match the format
 */
/*VARARGS*/
void
fatal_reader_mksh(const char * pattern, ...)
{
        va_list args;
        char    message[1000];

        va_start(args, pattern);
/*
        if (file_being_read != NULL) {
                WCSTOMBS(mbs_buffer, file_being_read);
                if (line_number != 0) {
                        (void) sprintf(message,
                                       gettext("%s, line %d: %s"),
                                       catgets(libmksdmsi18n_catd, 1, 130, "%s,
                                       mbs_buffer,
                                       line_number,
                                       pattern);
                } else {
                        (void) sprintf(message,
                                       "%s: %s",
                                       mbs_buffer,
                                       pattern);
                }
                pattern = message;
        }
 */

        (void) fflush(stdout);
        (void) fprintf(stderr, gettext("mksh: Fatal error in reader: "));
        (void) fprintf(stderr, catgets(libmksdmsi18n_catd, 1, 131, "mksh: Fatal
        (void) vfprintf(stderr, pattern, args);
        (void) fprintf(stderr, "\n");
        va_end(args);

/*
        if (temp_file_name != NULL) {
```

```
                (void) fprintf(stderr,
                                gettext("mksh: Temp-file %s not removed\n"),
                                catgets(libmksdmsi18n_catd, 1, 132, "mksh: Temp-f
                                temp_file_name->string_mb);
                temp_file_name = NULL;
        }
 */

/*
        if (report_pwd) {
 */
        if (1) {
                (void) fprintf(stderr,
                                gettext("Current working directory %s\n"),
                                catgets(libmksdmsi18n_catd, 1, 133, "Current work
                                get_current_path_mksh());
        }
        (void) fflush(stderr);
        exit_status = 1;
        exit(1);
}

/*
 *      warning_mksh(format, args...)
 *
 *      Print a message and continue.
 *
 *      Parameters:
 *              format          printf type format string
 *              args            Arguments to match the format
 */
/*VARARGS*/
void
warning_mksh(char * message, ...)
{
        va_list args;

        va_start(args, message);
        (void) fflush(stdout);
        (void) fprintf(stderr, gettext("mksh: Warning: "));
        (void) fprintf(stderr, catgets(libmksdmsi18n_catd, 1, 134, "mksh: Warnin
        (void) vfprintf(stderr, message, args);
        (void) fprintf(stderr, "\n");
        va_end(args);
/*
        if (report_pwd) {
 */
        if (1) {
                (void) fprintf(stderr,
                                gettext("Current working directory %s\n"),
                                catgets(libmksdmsi18n_catd, 1, 135, "Current work
                                get_current_path_mksh());
        }
        (void) fflush(stderr);
}
```

*unchanged_portion_omitted_*

```
/*
 *      append_prop(target, type)
 *
 *      Create a new property and append it to the property list of a Name.
 *
 *      Return value:
 *                              A new property block for the target
 *
 *      Parameters:
```



```
 *              target          The target that wants a new property
 *              type            The type of property being requested
 *
 *      Global variables used:
 */
Property
append_prop(register Name target, register Property_id type)
{
        register Property       *insert = &target->prop;
        register Property       prop = *insert;
        register int            size;

        switch (type) {
        case conditional_prop:
                size = sizeof (struct Conditional);
                break;
        case line_prop:
                size = sizeof (struct Line);
                break;
        case macro_prop:
                size = sizeof (struct _Macro);
                break;
        case makefile_prop:
                size = sizeof (struct Makefile);
                break;
        case member_prop:
                size = sizeof (struct Member);
                break;
        case recursive_prop:
                size = sizeof (struct Recursive);
                break;
        case sccs_prop:
                size = sizeof (struct Sccs);
                break;
        case suffix_prop:
                size = sizeof (struct Suffix);
                break;
        case target_prop:
                size = sizeof (struct Target);
                break;
        case time_prop:
                size = sizeof (struct STime);
                break;
        case vpath_alias_prop:
                size = sizeof (struct Vpath_alias);
                break;
        case long_member_name_prop:
                size = sizeof (struct Long_member_name);
                break;
        case macro_append_prop:
                size = sizeof (struct _Macro_appendix);
                break;
        case env_mem_prop:
                size = sizeof (struct _Env_mem);
                break;
        default:
                fatal_mksh(gettext("Internal error. Unknown prop type %d"), type
                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 136, "Internal error.
        }
        for (; prop != NULL; insert = &prop->next, prop = *insert);
        size += PROPERTY_HEAD_SIZE;
        *insert = prop = (Property) getmem(size);
        memset((char *) prop, 0, size);
        prop->type = type;
        prop->next = NULL;
        return prop;
```

```
 615 }
______unchanged_portion_omitted_


 849 void
 850 mbstowcs_with_check(wchar_t *pwcs, const char *s, size_t n)
 851 {
 852         if(mbstowcs(pwcs, s, n) == -1) {
 853                 fatal_mksh(gettext("The string ‘%s’ is not valid in current loca
 795                 fatal_mksh(catgets(libmksdmsi18n_catd, 1, 143, "The string ‘%s’
 854         }
 855 }
______unchanged_portion_omitted_
```

```
**********************************************************
    3236 Wed May 20 12:20:00 2015
new/usr/src/cmd/make/lib/mksh/mksh.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

```
   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  *      mksh.cc
  29  *
  30  *      Execute the command(s) of one Make or DMake rule
  31  */

  33 /*
  34  * Included files
  35  */
  36 #include <mksh/dosys.h>         /* redirect_io() */
  37 #include <mksh/misc.h>          /* retmem() */
  38 #include <mksh/mksh.h>
  39 #include <mksdmsi18n/mksdmsi18n.h>
  39 #include <errno.h>
  40 #include <signal.h>


  43 /*
  44  * Workaround for NFS bug. Sometimes, when running 'chdir' on a remote
  45  * dmake server, it fails with "Stale NFS file handle" error.
  46  * The second attempt seems to work.
  47  */
  48 int
  49 my_chdir(char * dir) {
  50         int res = chdir(dir);
  51         if (res != 0 && (errno == ESTALE || errno == EAGAIN)) {
  52                 /* Stale NFS file handle. Try again */
  53                 res = chdir(dir);
  54         }
  55         return res;
  56 }
```

_____*unchanged_portion_omitted_*

```
  79 static void
  80 init_mksh_globals(char *shell)
```



```
  81 {
  82 /*
  83         MBSTOWCS(wcs_buffer, "SHELL");
  84         MBSTOWCS(wcs_buffer, NOCATGETS("SHELL"));
  84         shell_name = GETNAME(wcs_buffer, FIND_LENGTH);
  85         MBSTOWCS(wcs_buffer, shell);
  86         (void) SETVAR(shell_name, GETNAME(wcs_buffer, FIND_LENGTH), false);
  87  */
  88         char * dmake_shell;
  89         if ((dmake_shell = getenv("DMAKE_SHELL")) == NULL) {
  90         if ((dmake_shell = getenv(NOCATGETS("DMAKE_SHELL"))) == NULL) {
  90                 dmake_shell = shell;
  91         }
  92         MBSTOWCS(wcs_buffer, dmake_shell);
  93         shell_name = GETNAME(wcs_buffer, FIND_LENGTH);
  94 }

  96 /*
  97  * Change the pathname in the value of the SUNPRO_DEPENDENCIES env variable
  98  * from oldpath to newpath.
  99  */
 100 static void
 101 change_sunpro_dependencies_value(char *oldpath, char *newpath)
 102 {
 103         char            buf[MAXPATHLEN];
 104         static char     *env;
 105         int             length;
 106         int             oldpathlen;
 107         char            *sp_dep_value;

 109         /* check if SUNPRO_DEPENDENCIES is set in the environment */
 110         if ((sp_dep_value = getenv("SUNPRO_DEPENDENCIES")) != NULL) {
 111         if ((sp_dep_value = getenv(NOCATGETS("SUNPRO_DEPENDENCIES"))) != NULL) {
 111                 oldpathlen = strlen(oldpath);
 112                 /* check if oldpath is indeed in the value of SUNPRO_DEPENDENCIE
 113                 if (strncmp(oldpath, sp_dep_value, oldpathlen) == 0) {
 114                         (void) sprintf(buf,
 115                                         "%s%s",
 116                                         newpath,
 117                                         sp_dep_value + oldpathlen);
 118                         length = 2 +
 119                                 strlen("SUNPRO_DEPENDENCIES") +
 120                                 strlen(NOCATGETS("SUNPRO_DEPENDENCIES")) +
 120                                 strlen(buf);
 121                         env = getmem(length);
 122                         (void) sprintf(env,
 123                                         "%s=%s",
 124                                         "SUNPRO_DEPENDENCIES",
 125                                         NOCATGETS("SUNPRO_DEPENDENCIES"),
 125                                         buf);
 126                         (void) putenv(env);
 127                 }
 128         }
 129 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    4555 Wed May 20 12:20:00 2015
new/usr/src/cmd/make/lib/mksh/read.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */


/*
 *      read.c
 *
 *      This file contains the makefile reader.
 */

/*
 * Included files
 */
#include <mksh/misc.h>          /* retmem() */
#include <mksh/read.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <sys/uio.h>            /* read() */
#include <unistd.h>             /* close(), unlink(), read() */
#include <libintl.h>
#endif /* ! codereview */

#define STRING_LEN_TO_CONVERT   (8*1024)

/*
 *      get_next_block_fn(source)
 *
 *      Will get the next block of text to read either
 *      by popping one source bVSIZEOFlock of the stack of Sources
 *      or by reading some more from the makefile.
 *
 *      Return value:
 *                              The new source block to read from
 *
 *      Parameters:
 *              source          The old source block
 *
 *      Global variables used:
 *              file_being_read The name of the current file, error msg
 */
```



```
Boolean         make_state_locked;
Source
get_next_block_fn(register Source source)
{
        register off_t          to_read;
        register int            length;
        register size_t         num_wc_chars;
        char                    ch_save;
        char                    *ptr;

        if (source == NULL) {
                return NULL;
        }
        if ((source->fd < 0) ||
                ((source->bytes_left_in_file <= 0) &&
                        (source->inp_buf_ptr >= source->inp_buf_end))) {
                /* We can't read from the makefile, so pop the source block */
                if (source->fd > 2) {
                        (void) close(source->fd);
                        if (make_state_lockfile != NULL) {
                                (void) unlink(make_state_lockfile);
                                retmem_mb(make_state_lockfile);
                                make_state_lockfile = NULL;
                                make_state_locked = false;
                        }
                }
                if (source->string.free_after_use &&
                    (source->string.buffer.start != NULL)) {
                        retmem(source->string.buffer.start);
                        source->string.buffer.start = NULL;
                }
                if (source->inp_buf != NULL) {
                        retmem_mb(source->inp_buf);
                        source->inp_buf = NULL;
                }
                source = source->previous;
                if (source != NULL) {
                        source->error_converting = false;
                }
                return source;
        }
        if (source->bytes_left_in_file > 0) {
        /*
         * Read the whole makefile.
         * Hopefully the kernel managed to prefetch the stuff.
         */
                to_read = source->bytes_left_in_file;
                source->inp_buf_ptr = source->inp_buf = getmem(to_read + 1);
                source->inp_buf_end = source->inp_buf + to_read;
                length = read(source->fd, source->inp_buf, (unsigned int) to_rea
                if (length != to_read) {
                        WCSTOMBS(mbs_buffer, file_being_read);
                        if (length == 0) {
                                fatal_mksh(gettext("Error reading '%s': Prematur
 41                             fatal_mksh(catgets(libmksdmsi18n_catd, 1, 140, "
                                        mbs_buffer);
                        } else {
                                fatal_mksh(gettext("Error reading '%s': %s"),
 44                             fatal_mksh(catgets(libmksdmsi18n_catd, 1, 141, "
                                        mbs_buffer,
                                        errmsg(errno));
                        }
                }
                *source->inp_buf_end = nul_char;
                source->bytes_left_in_file = 0;
        }
```

```
125         /*
126          * Try to convert the next piece.
127          */
128         ptr = source->inp_buf_ptr + STRING_LEN_TO_CONVERT;
129         if (ptr > source->inp_buf_end) {
130                 ptr = source->inp_buf_end;
131         }
132         for (num_wc_chars = 0; ptr > source->inp_buf_ptr; ptr--) {
133                 ch_save = *ptr;
134                 *ptr = nul_char;
135                 num_wc_chars = mbstowcs(source->string.text.end,
136                                         source->inp_buf_ptr,
137                                         STRING_LEN_TO_CONVERT);
138                 *ptr = ch_save;
139                 if (num_wc_chars != (size_t)-1) {
140                         break;
141                 }
142         }
143
144         if ((int) num_wc_chars == (size_t)-1) {
145                 source->error_converting = true;
146                 return source;
147         }

149         source->error_converting = false;
150         source->inp_buf_ptr = ptr;
151         source->string.text.end += num_wc_chars;
152         *source->string.text.end = 0;

154         if (source->inp_buf_ptr >= source->inp_buf_end) {
155                 if (*(source->string.text.end - 1) != (int) newline_char) {
156                         WCSTOMBS(mbs_buffer, file_being_read);
157                         warning_mksh(gettext("newline is not last character in f
 84                      warning_mksh(catgets(libmksdmsi18n_catd, 1, 142, "newlin
158                                              mbs_buffer);
159                         *source->string.text.end++ = (int) newline_char;
160                         *source->string.text.end = (int) nul_char;
161                         *source->string.buffer.end++;
162                 }
163                 if (source->inp_buf != NULL) {
164                         retmem_mb(source->inp_buf);
165                         source->inp_buf = NULL;
166                 }
167         }
168         return source;
169 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1155 Wed May 20 12:20:01 2015
new/usr/src/cmd/make/lib/vroot/Makefile
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

# Copyright 2015, Richard Lowe.

LIBRARY =       libvroot.a
VERS =          .1
OBJECTS =       access.o        \
                args.o          \
                chdir.o         \
                chmod.o         \
                chown.o         \
                chroot.o        \
                creat.o         \
                execve.o        \
                lock.o          \
                lstat.o         \
                mkdir.o         \
                mount.o         \
                open.o          \
                readlink.o      \
                report.o        \
                rmdir.o         \
                stat.o          \
                truncate.o      \
                unlink.o        \
                utimes.o        \
                vroot.o         \
                setenv.o

include $(SRC)/lib/Makefile.lib
include ../../Makefile.com

POFILE = libvroot.po
POFILES = $(OBJECTS:%.o=%.po)

#endif /* ! codereview */
LIBS = $(LIBRARY)
SRCDIR = ../
MAPFILES=
CPPFLAGS += -D_FILE_OFFSET_BITS=64

all: $(LIBS)

install: all

lint:

$(POFILE): $(POFILES)
        $(CAT) $(POFILES) > $@

_msg: $(MSGDOMAIN) $(POFILE)
        $(RM) $(MSGDOMAIN)/$(POFILE)
```



```
        $(CP) $(POFILE) $(MSGDOMAIN)

#endif /* ! codereview */
include $(SRC)/lib/Makefile.targ
```

```
**********************************************************
    5041 Wed May 20 12:20:01 2015
new/usr/src/cmd/make/lib/vroot/lock.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

#include <avo/intl.h>   /* for NOCATGETS */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/errno.h>
#include <sys/param.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <vroot/vroot.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <signal.h>
#include <errno.h>                      /* errno */
#include <libintl.h>
#endif /* ! codereview */

extern  char            *sys_errlist[];
extern  int             sys_nerr;

static  void            file_lock_error(char *msg, char *file, char *str, int ar

#define BLOCK_INTERUPTS sigfillset(&newset) ; \
        sigprocmask(SIG_SETMASK, &newset, &oldset)

#define UNBLOCK_INTERUPTS \
        sigprocmask(SIG_SETMASK, &oldset, &newset)

/*
 * This code stolen from the NSE library and changed to not depend
 * upon any NSE routines or header files.
 *
 * Simple file locking.
 * Create a symlink to a file.  The "test and set" will be
 * atomic as creating the symlink provides both functions.
 *
 * The timeout value specifies how long to wait for stale locks
```



```
 * to disappear.  If the lock is more than 'timeout' seconds old
 * then it is ok to blow it away.  This part has a small window
 * of vunerability as the operations of testing the time,
 * removing the lock and creating a new one are not atomic.
 * It would be possible for two processes to both decide to blow
 * away the lock and then have process A remove the lock and establish
 * its own, and then have process B remove the lock which accidentily
 * removes A's lock rather than the stale one.
 *
 * A further complication is with the NFS.  If the file in question is
 * being served by an NFS server, then its time is set by that server.
 * We can not use the time on the client machine to check for a stale
 * lock.  Therefore, a temp file on the server is created to get
 * the servers current time.
 *
 * Returns an error message.  NULL return means the lock was obtained.
 *
 * 12/6/91 Added the parameter "file_locked".  Before this parameter
 * was added, the calling procedure would have to wait for file_lock()
 * to return before it sets the flag. If the user interrupted "make"
 * between the time the lock was acquired and the time file_lock()
 * returns, make wouldn't know that the file has been locked, and therefore
 * it wouldn' remove the lock. Setting the flag right after locking the file
 * makes this window much smaller.
 */

int
file_lock(char *name, char *lockname, int *file_locked, int timeout)
{
        int             counter = 0;
        static char     msg[MAXPATHLEN+1];
        int             printed_warning = 0;
        int             r;
        struct stat     statb;
        sigset_t newset;
        sigset_t oldset;

        *file_locked = 0;
        if (timeout <= 0) {
                timeout = 120;
        }
        for (;;) {
                BLOCK_INTERUPTS;
                r = symlink(name, lockname);
                if (r == 0) {
                        *file_locked = 1;
                        UNBLOCK_INTERUPTS;
                        return 0; /* success */
                }
                UNBLOCK_INTERUPTS;

                if (errno != EEXIST) {
                        file_lock_error(msg, name, (char *)"symlink(%s, %s)",
                        file_lock_error(msg, name, (char *)NOCATGETS("symlink(%s
                            (int) name, (int) lockname);
                        fprintf(stderr, "%s", msg);
                        return errno;
                }

                counter = 0;
                for (;;) {
                        sleep(1);
                        r = lstat(lockname, &statb);
                        if (r == -1) {
                                /*
                                 * The lock must have just gone away - try
```

```
125                                      * again.
126                                      */
127                                     break;
128                             }
129
130                             if ((counter > 5) && (!printed_warning)) {
131                                     /* Print waiting message after 5 secs */
132                                     (void) getcwd(msg, MAXPATHLEN);
133                                     fprintf(stderr,
134                                             gettext("file_lock: file %s is already l
 61                                             catgets(libmksdmsi18n_catd, 1, 162, "fil
135                                             name);
136                                     fprintf(stderr,
137                                             gettext("file_lock: will periodically ch
 64                                             catgets(libmksdmsi18n_catd, 1, 163, "fil
138                                             lockname);
139                                     fprintf(stderr,
140                                             gettext("Current working directory %s\n"
 67                                             catgets(libmksdmsi18n_catd, 1, 144, "Cur
141                                             msg);
142
143                                     printed_warning = 1;
144                             }
145
146                             if (++counter > timeout ) {
147                                     /*
148                                      * Waited enough - return an error..
149                                      */
150                                     return EEXIST;
151                             }
152                     }
153             }
154             /* NOTREACHED */
155 }
156
157 /*
158  * Format a message telling why the lock could not be created.
159  */
160 static  void
161 file_lock_error(char *msg, char *file, char *str, int arg1, int arg2)
162 {
163             int             len;
164
165             sprintf(msg, gettext("Could not lock file `%s'; "), file);
 92             sprintf(msg, catgets(libmksdmsi18n_catd, 1, 145, "Could not lock file `%
166             len = strlen(msg);
167             sprintf(&msg[len], str, arg1, arg2);
168             strcat(msg, gettext(" failed - "));
 95             strcat(msg, catgets(libmksdmsi18n_catd, 1, 146, " failed - "));
169             if (errno < sys_nerr) {
170                     strcat(msg, strerror(errno));
171             } else {
172                     len = strlen(msg);
173                     sprintf(&msg[len], "errno %d", errno);
100                     sprintf(&msg[len], NOCATGETS("errno %d"), errno);
174             }
175 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7997 Wed May 20 12:20:01 2015
new/usr/src/cmd/make/lib/vroot/report.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/param.h>
#include <sys/wait.h>
#include <unistd.h>
#include <libintl.h>
#endif /* ! codereview */

#include <vroot/report.h>
#include <vroot/vroot.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <avo/intl.h>   /* for NOCATGETS */
#include <mk/defs.h>    /* for tmpdir */

static  FILE    *report_file;
static  FILE    *command_output_fp;
static  char    *target_being_reported_for;
static  char    *search_dir;
static  char    command_output_tmpfile[30];
static  int     is_path = 0;
static  char    sfile[MAXPATHLEN];
extern "C" {
static  void    (*warning_ptr) (char *, ...) = (void (*) (char *, ...)) NULL;
}
```

*unchanged_portion_omitted_*

```
/*
 *  Update the file, if necessary.  We don't want to rewrite
 *  the file if we don't have to because we don't want the time of the file
 *  to change in that case.
 */

extern "C" {
static void
```



```
close_file(void)
{
        char            line[MAXPATHLEN+2];
        char            buf[MAXPATHLEN+2];
        FILE            *nse_depinfo_fp;
        FILE            *merge_fp;
        char            nse_depinfo_file[MAXPATHLEN];
        char            merge_file[MAXPATHLEN];
        char            lock_file[MAXPATHLEN];
        int             err;
        int             len;
        int             changed = 0;
        int             file_locked;

        fprintf(command_output_fp, "\n");
        fclose(command_output_fp);
        if ((command_output_fp = fopen(command_output_tmpfile, "r")) == NULL) {
                return;
        }
        sprintf(nse_depinfo_file, "%s/%s", search_dir, NSE_DEPINFO);
        sprintf(merge_file, "%s/.tmp%s.%d", search_dir, NSE_DEPINFO, getpid());
        sprintf(merge_file, NOCATGETS("%s/.tmp%s.%d"), search_dir, NSE_DEPINFO,
        sprintf(lock_file, "%s/%s", search_dir, NSE_DEPINFO_LOCK);
        err = file_lock(nse_depinfo_file, lock_file, &file_locked, 0);
        if (err) {
                if (warning_ptr != (void (*) (char *, ...)) NULL) {
                        (*warning_ptr)(gettext("Couldn't write to %s"), nse_depi
                        (*warning_ptr)(catgets(libmksdmsi18n_catd, 1, 147, "Coul
                      }
                unlink(command_output_tmpfile);
                return;
        }
        /* If .nse_depinfo file doesn't exist */
        if ((nse_depinfo_fp = fopen(nse_depinfo_file, "r+")) == NULL) {
                if (is_path) {
                        if ((nse_depinfo_fp =
                             fopen(nse_depinfo_file, "w")) == NULL) {
                                fprintf(stderr, gettext("Cannot open '%s' for wr
                                fprintf(stderr, catgets(libmksdmsi18n_catd, 1, 1
                                    nse_depinfo_file);
                                unlink(command_output_tmpfile);

                                unlink(lock_file);
                                return;
                        }
                        while (fgets(line, MAXPATHLEN+2, command_output_fp)
                               != NULL) {
                                fprintf(nse_depinfo_fp, "%s", line);
                        }
                        fclose(command_output_fp);
                }
                fclose(nse_depinfo_fp);
                if (file_locked) {
                        unlink(lock_file);
                }
                unlink(command_output_tmpfile);
                return;
        }
        if ((merge_fp = fopen(merge_file, "w")) == NULL) {
                fprintf(stderr, gettext("Cannot open %s for writing\n"), merge_f
                fprintf(stderr, catgets(libmksdmsi18n_catd, 1, 149, "Cannot open
                if (file_locked) {
                        unlink(lock_file);
                }
                unlink(command_output_tmpfile);
                return;
```

```
155         }
156         len = strlen(sfile);
157         while (fgets(line, MAXPATHLEN+2, nse_depinfo_fp) != NULL) {
158                 if (strncmp(line, sfile, len) == 0 && line[len] == ':') {
159                         while (fgets(buf, MAXPATHLEN+2, command_output_fp)
160                                != NULL) {
161                                 if (is_path) {
162                                         fprintf(merge_fp, "%s", buf);
163                                         if (strcmp(line, buf)) {
164                                                 /* changed */
165                                                 changed = 1;
166                                         }
167                                 }
168                                 if (buf[strlen(buf)-1] == '\n') {
169                                         break;
170                                 }
171                         }
172                         if (changed || !is_path) {
173                                 while (fgets(line, MAXPATHLEN, nse_depinfo_fp)
174                                        != NULL) {
175                                         fputs(line, merge_fp);
176                                 }
177                                 clean_up(nse_depinfo_fp, merge_fp,
178                                          nse_depinfo_file, merge_file, 0);
179                         } else {
180                                 clean_up(nse_depinfo_fp, merge_fp,
181                                          nse_depinfo_file, merge_file, 1);
182                         }
183                         if (file_locked) {
184                                 unlink(lock_file);
185                         }
186                         unlink(command_output_tmpfile);
187                         return;
188                 } /* entry found */
189                 fputs(line, merge_fp);
190         }
191         /* Entry never found.  Add it if there is a search path */
192         if (is_path) {
193                 while (fgets(line, MAXPATHLEN+2, command_output_fp) != NULL) {
194                         fprintf(nse_depinfo_fp, "%s", line);
195                 }
196         }
197         clean_up(nse_depinfo_fp, merge_fp, nse_depinfo_file, merge_file, 1);
198         if (file_locked) {
199                 unlink(lock_file);
200         }
201 }
```

*unchanged_portion_omitted_*

```
205 static void
206 report_dep(char *iflag, char *filename)
207 {

209         if (command_output_fp == NULL) {
210                 sprintf(command_output_tmpfile,
211                         "%s/%s.%d.XXXXXX", tmpdir, NSE_DEPINFO, getpid());
208                         NOCATGETS("%s/%s.%d.XXXXXX"), tmpdir, NSE_DEPINFO, getpi
212                 int fd = mkstemp(command_output_tmpfile);
213                 if ((fd < 0) || (command_output_fp = fdopen(fd, "w")) == NULL) {
214                         return;
215                 }
216                 if ((search_dir = getenv("NSE_DEP")) == NULL) {
213                 if ((search_dir = getenv(NOCATGETS("NSE_DEP"))) == NULL) {
217                         return;
218                 }
219                 atexit(close_file);
```



```
220                 strcpy(sfile, filename);
221                 if (iflag == NULL || *iflag == '\0') {
222                         return;
223                 }
224                 fprintf(command_output_fp, "%s:", sfile);
225         }
226         fprintf(command_output_fp, " ");
227         fprintf(command_output_fp, iflag);
228         if (iflag != NULL) {
229                 is_path = 1;
230         }
231 }
```

*unchanged_portion_omitted_*

```
251 void
252 report_search_path(char *iflag)
253 {
254         char            curdir[MAXPATHLEN];
255         char            *sdir;
256         char            *newiflag;
257         char            filename[MAXPATHLEN];
258         char            *p, *ptr;

260         if ((sdir = getenv("NSE_DEP")) == NULL) {
257         if ((sdir = getenv(NOCATGETS("NSE_DEP"))) == NULL) {
261                 return;
262         }
263         if ((p= getenv(SUNPRO_DEPENDENCIES)) == NULL) {
264                 return;
265         }
266         ptr = strchr(p, ' ');
267         if( ! ptr ) {
268                 return;
269         }
270         sprintf(filename, "%s-CPP", ptr+1);
267         sprintf(filename, NOCATGETS("%s-CPP"), ptr+1);
271         getcwd(curdir, sizeof(curdir));
272         if (strcmp(curdir, sdir) != 0 && strlen(iflag) > 2 &&
273             iflag[2] != '/') {
274                 /* Makefile must have had an "cd xx; cc ..." */
275                 /* Modify the -I path to be relative to the cd */
276                 newiflag = (char *)malloc(strlen(iflag) + strlen(curdir) + 2);
277                 sprintf(newiflag, "-%c%s/%s", iflag[1], curdir, &iflag[2]);
278                 report_dep(newiflag, filename);
279         } else {
280                 report_dep(iflag, filename);
281         }
282 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    9112 Wed May 20 12:20:02 2015
new/usr/src/cmd/make/lib/vroot/vroot.cc
make: translate using gettext, rather than the unmaintainable catgets
**********************************************************
```

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


#include <stdlib.h>
#include <string.h>

#include <vroot/vroot.h>
#include <vroot/args.h>

#include <string.h>
#include <sys/param.h>
#include <sys/file.h>

#include <avo/intl.h>   /* for NOCATGETS */

typedef struct {
        short           init;
        pathpt          vector;
        const char      *env_var;
} vroot_patht;
```

*unchanged_portion_omitted_*

```
static vroot_datat      vroot_data= {
        { 0, NULL, "VIRTUAL_ROOT"},
        { 0, NULL, "PATH"},
        { 0, NULL, NOCATGETS("VIRTUAL_ROOT")},
        { 0, NULL, NOCATGETS("PATH")},
        "", NULL, NULL, NULL, 0, 1};

void
add_dir_to_path(const char *path, register pathpt *pointer, register int positio
{
        register int            size= 0;
        register int            length;
        register char           *name;
        register pathcellpt     p;
        pathpt                  new_path;
```



```
        if (*pointer != NULL) {
                for (p= &((*pointer)[0]); p->path != NULL; p++, size++);
                if (position < 0)
                        position= size;}
        else
                if (position < 0)
                        position= 0;
        if (position >= size) {
                new_path= (pathpt)calloc((unsigned)(position+2), sizeof(pathcell
                if (*pointer != NULL) {
                        memcpy((char *)new_path,(char *)(*pointer),  size*sizeof
                        free((char *)(*pointer));};
                *pointer= new_path;};
        length= strlen(path);
        name= (char *)malloc((unsigned)(length+1));
        (void)strcpy(name, path);
        if ((*pointer)[position].path != NULL)
                free((*pointer)[position].path);
        (*pointer)[position].path= name;
        (*pointer)[position].length= length;
}
```

*unchanged_portion_omitted_*