```
**********************************************************
   34599 Sat Dec 14 21:34:46 2013
new/usr/src/Makefile.master
4265 remove INTERNAL_RELEASE_BUILD
**********************************************************
```

```
#
# 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 (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012 by Delphix. All rights reserved.
#

#
# Makefile.master, global definitions for system source
#
ROOT=           /proto

#
# Adjunct root, containing an additional proto area to be used for headers
# and libraries.
#
ADJUNCT_PROTO=

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

#
# RELEASE_BUILD should be cleared for final release builds.
# NOT_RELEASE_BUILD is exactly what the name implies.
#
# INTERNAL_RELEASE_BUILD is a subset of RELEASE_BUILD. It mostly controls
# identification strings. Enabling RELEASE_BUILD automatically enables
# INTERNAL_RELEASE_BUILD.
#
# STRIP_COMMENTS toggles comment section striping. Generally the same setting
# as INTERNAL_RELEASE_BUILD.
#
# __GNUC toggles the building of ON components using gcc and related tools.
# Normally set to '#', set it to '' to do gcc build.
#
# The declaration POUND_SIGN is always '#'. This is needed to get around the
# make feature that '#' is always a comment delimiter, even when escaped or
# quoted. We use this macro expansion method to get POUND_SIGN rather than
# always breaking out a shell because the general case can cause a noticable
# slowdown in build times when so many Makefiles include Makefile.master.
```



```
#
# While the majority of users are expected to override the setting below
# with an env file (via nightly or bldenv), if you aren't building that way
# (ie, you're using "ws" or some other bootstrapping method) then you need
# this definition in order to avoid the subshell invocation mentioned above.
#

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

NOT_RELEASE_BUILD=
INTERNAL_RELEASE_BUILD=                 $(POUND_SIGN)
RELEASE_BUILD=                          $(POUND_SIGN)
$(RELEASE_BUILD)NOT_RELEASE_BUILD=      $(POUND_SIGN)
$(RELEASE_BUILD)INTERNAL_RELEASE_BUILD=
PATCH_BUILD=                            $(POUND_SIGN)

# SPARC_BLD is '#' for an Intel build.
# INTEL_BLD is '#' for a Sparc build.
SPARC_BLD_1=    $(MACH:i386=$(POUND_SIGN))
SPARC_BLD=      $(SPARC_BLD_1:sparc=)
INTEL_BLD_1=    $(MACH:sparc=$(POUND_SIGN))
INTEL_BLD=      $(INTEL_BLD_1:i386=)

STRIP_COMMENTS= $(INTERNAL_RELEASE_BUILD)

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

__GNUC=                 $(POUND_SIGN)
$(__GNUC)__SUNC=        $(POUND_SIGN)
__GNUC64=               $(__GNUC)

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

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

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

JAVA_ROOT=      /usr/java

SFW_ROOT=       /usr/sfw
SFWINCDIR=      $(SFW_ROOT)/include
SFWLIBDIR=      $(SFW_ROOT)/lib
SFWLIBDIR64=    $(SFW_ROOT)/lib/$(MACH64)

GCC_ROOT=       /opt/gcc/4.4.4
GCCLIBDIR=      $(GCC_ROOT)/lib
GCCLIBDIR64=    $(GCC_ROOT)/lib/$(MACH64)
```

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

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

181 FILEMODE=       644
182 DIRMODE=        755
```



```
184 #
185 # The version of the patch makeup table optimized for build-time use.  Used
186 # during patch builds only.
187 $(PATCH_BUILD)PMTMO_FILE=$(SRC)/patch_makeup_table.mo

189 # Declare that nothing should be built in parallel.
190 # Individual Makefiles can use the .PARALLEL target to declare otherwise.
191 .NO_PARALLEL:

193 # For stylistic checks
194 #
195 # Note that the X and C checks are not used at this time and may need
196 # modification when they are actually used.
197 #
198 CSTYLE=         $(ONBLD_TOOLS)/bin/cstyle
199 CSTYLE_TAIL=
200 HDRCHK=         $(ONBLD_TOOLS)/bin/hdrchk
201 HDRCHK_TAIL=
202 JSTYLE=         $(ONBLD_TOOLS)/bin/jstyle

204 DOT_H_CHECK=    \
205         @$(ECHO) "checking $<"; $(CSTYLE) $< $(CSTYLE_TAIL); \
206         $(HDRCHK) $< $(HDRCHK_TAIL)

208 DOT_X_CHECK=    \
209         @$(ECHO) "checking $<"; $(RPCGEN) -C -h $< | $(CSTYLE) $(CSTYLE_TAIL); \
210         $(RPCGEN) -C -h $< | $(HDRCHK) $< $(HDRCHK_TAIL)

212 DOT_C_CHECK=    \
213         @$(ECHO) "checking $<"; $(CSTYLE) $< $(CSTYLE_TAIL)

215 MANIFEST_CHECK= \
216         @$(ECHO) "checking $<"; \
217         SVCCFG_DTD=$(SRC)/cmd/svc/dtd/service_bundle.dtd.1 \
218         SVCCFG_REPOSITORY=$(SRC)/cmd/svc/seed/global.db \
219         SVCCFG_CONFIGD_PATH=$(SRC)/cmd/svc/configd/svc.configd-native \
220         $(SRC)/cmd/svc/svccfg/svccfg-native validate $<

222 INS.file=       $(RM) $@; $(INS) -s -m $(FILEMODE) -f $(@D) $<
223 INS.dir=        $(INS) -s -d -m $(DIRMODE) $@
224 # installs and renames at once
225 #
226 INS.rename=     $(INS.file); $(MV) $(@D)/$(<F) $@

228 # install a link
229 INSLINKTARGET=  $<
230 INS.link=       $(RM) $@; $(LN) $(INSLINKTARGET) $@
231 INS.symlink=    $(RM) $@; $(SYMLINK) $(INSLINKTARGET) $@

233 #
234 # Python bakes the mtime of the .py file into the compiled .pyc and
235 # rebuilds if the baked-in mtime != the mtime of the source file
236 # (rather than only if it’s less than), thus when installing python
237 # files we must make certain to not adjust the mtime of the source
238 # (.py) file.
239 #
240 INS.pyfile=     $(INS.file); $(TOUCH) -r $< $@

242 # MACH must be set in the shell environment per uname -p on the build host
243 # More specific architecture variables should be set in lower makefiles.
244 #
245 # MACH64 is derived from MACH, and BUILD64 is set to ‘#’ for
246 # architectures on which we do not build 64-bit versions.
247 # (There are no such architectures at the moment.)
248 #
```

```
249 # Set BUILD64=# in the environment to disable 64-bit amd64
250 # builds on i386 machines.

252 MACH64_1=       $(MACH:sparc=sparcv9)
253 MACH64=         $(MACH64_1:i386=amd64)

255 MACH32_1=       $(MACH:sparc=sparcv7)
256 MACH32=         $(MACH32_1:i386=i86)

258 sparc_BUILD64=
259 i386_BUILD64=
260 BUILD64=        $($(MACH)_BUILD64)

262 #
263 # C compiler mode. Future compilers may change the default on us,
264 # so force extended ANSI mode globally. Lower level makefiles can
265 # override this by setting CCMODE.
266 #
267 CCMODE=                 -Xa
268 CCMODE64=               -Xa

270 #
271 # C compiler verbose mode. This is so we can enable it globally,
272 # but turn it off in the lower level makefiles of things we cannot
273 # (or aren't going to) fix.
274 #
275 CCVERBOSE=              -v

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

281 # set this to the secret flag "-Wc,-Qiselect-regsym=0" to disable register
282 # symbols (used to detect conflicts between objects that use global registers)
283 # we disable this now for safety, and because genunix doesn't link with
284 # this feature (the v9 default) enabled.
285 #
286 # REGSYM is separate since the C++ driver syntax is different.
287 CCREGSYM=               -Wc,-Qiselect-regsym=0
288 CCCREGSYM=              -Qoption cg -Qiselect-regsym=0

290 # Prevent the removal of static symbols by the SPARC code generator (cg).
291 # The x86 code generator (ube) does not remove such symbols and as such
292 # using this workaround is not applicable for x86.
293 #
294 CCSTATICSYM=            -Wc,-Qassembler-ounrefsym=0
295 #
296 # generate 32-bit addresses in the v9 kernel. Saves memory.
297 CCABS32=                -Wc,-xcode=abs32
298 #
299 # generate v9 code which tolerates callers using the v7 ABI, for the sake of
300 # system calls.
301 CC32BITCALLERS=         -_gcc=-massume-32bit-callers

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

312 # One optimization the compiler might perform is to turn this:
313 #       #pragma weak foo
314 #       extern int foo;
```



```
315 #       if (&foo)
316 #               foo = 5;
317 # into
318 #       foo = 5;
319 # Since we do some of this (foo might be referenced in common kernel code
320 # but provided only for some cpu modules or platforms), we disable this
321 # optimization.
322 #
323 sparc_CCUNBOUND = -Wd,-xsafe=unboundsym
324 i386_CCUNBOUND  =
325 CCUNBOUND       = $($(MACH)_CCUNBOUND)

327 #
328 # compiler '-xarch' flag. This is here to centralize it and make it
329 # overridable for testing.
330 sparc_XARCH=    -m32
331 sparcv9_XARCH=  -m64
332 i386_XARCH=
333 amd64_XARCH=    -m64 -Ui386 -U__i386

335 # assembler '-xarch' flag.  Different from compiler '-xarch' flag.
336 sparc_AS_XARCH=         -xarch=v8plus
337 sparcv9_AS_XARCH=       -xarch=v9
338 i386_AS_XARCH=
339 amd64_AS_XARCH=         -xarch=amd64 -P -Ui386 -U__i386

341 #
342 # These flags define what we need to be 'standalone' i.e. -not- part
343 # of the rather more cosy userland environment.  This basically means
344 # the kernel.
345 #
346 # XX64  future versions of gcc will make -mcmodel=kernel imply -mno-red-zone
347 #
348 sparc_STAND_FLAGS=      -_gcc=-ffreestanding
349 sparcv9_STAND_FLAGS=    -_gcc=-ffreestanding
350 # Disabling MMX also disables 3DNow, disabling SSE also disables all later
351 # additions to SSE (SSE2, AVX ,etc.)
352 NO_SIMD=                -_gcc=-mno-mmx -_gcc=-mno-sse
353 i386_STAND_FLAGS=       -_gcc=-ffreestanding $(NO_SIMD)
354 amd64_STAND_FLAGS=      -xmodel=kernel $(NO_SIMD)

356 SAVEARGS=               -Wu,-save_args
357 amd64_STAND_FLAGS       += $(SAVEARGS)

359 STAND_FLAGS_32 = $($(MACH)_STAND_FLAGS)
360 STAND_FLAGS_64 = $($(MACH64)_STAND_FLAGS)

362 #
363 # disable the incremental linker
364 ILDOFF=                 -xildoff
365 #
366 XDEPEND=                -xdepend
367 XFFLAG=                 -xF=%all
368 XESS=                   -xs
369 XSTRCONST=              -xstrconst

371 #
372 # turn warnings into errors (C)
373 CERRWARN = -ertags=yes -errwarn=%all
374 CERRWARN += -erroff=E_EMPTY_TRANSLATION_UNIT
375 CERRWARN += -erroff=E_STATEMENT_NOT_REACHED

377 CERRWARN += -_gcc=-Wno-missing-braces
378 CERRWARN += -_gcc=-Wno-sign-compare
379 CERRWARN += -_gcc=-Wno-unknown-pragmas
380 CERRWARN += -_gcc=-Wno-unused-parameter
```

```
381 CERRWARN += -_gcc=-Wno-missing-field-initializers

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

386 # DEBUG v. -nd make for frequent unused variables, empty conditions, etc. in
387 # -nd builds
388 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-unused
389 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-empty-body

391 #
392 # turn warnings into errors (C++)
393 CCERRWARN=              -xwe

395 # C99 mode
396 C99_ENABLE=     -xc99=%all
397 C99_DISABLE=    -xc99=%none
398 C99MODE=        $(C99_DISABLE)
399 C99LMODE=       $(C99MODE:-xc99%=-Xc99%)

401 # In most places, assignments to these macros should be appended with +=
402 # (CPPFLAGS.master allows values to be prepended to CPPFLAGS).
403 sparc_CFLAGS=   $(sparc_XARCH) $(CCSTATICSYM)
404 sparcv9_CFLAGS= $(sparcv9_XARCH) -dalign $(CCVERBOSE) $(V9ABIWARN) $(CCREGSYM) \
405                 $(CCSTATICSYM)
406 i386_CFLAGS=    $(i386_XARCH)
407 amd64_CFLAGS=   $(amd64_XARCH)

409 sparc_ASFLAGS=  $(sparc_AS_XARCH)
410 sparcv9_ASFLAGS=$(sparcv9_AS_XARCH)
411 i386_ASFLAGS=   $(i386_AS_XARCH)
412 amd64_ASFLAGS=  $(amd64_AS_XARCH)

414 #
415 sparc_COPTFLAG=         -xO3
416 sparcv9_COPTFLAG=       -xO3
417 i386_COPTFLAG=          -O
418 amd64_COPTFLAG=         -xO3

420 COPTFLAG= $($(MACH)_COPTFLAG)
421 COPTFLAG64= $($(MACH64)_COPTFLAG)

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

427 # Direct the Sun Studio compiler to use a static globalization prefix based on t
428 # name of the module rather than something unique. Otherwise, objects
429 # will not build deterministically, as subsequent compilations of identical
430 # source will yeild objects that always look different.
431 #
432 # In the same spirit, this will also remove the date from the N_OPT stab.
433 CGLOBALSTATIC= -W0,-xglobalstatic

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

439 #
440 # Default debug format for Sun Studio 11 is dwarf, so force it to
441 # generate stabs.
442 #
443 DEBUGFORMAT=    -xdebugformat=stabs

445 #
446 # Flags used to build in debug mode for ctf generation.  Bugs in the Devpro
```



```
447 # compilers currently prevent us from building with cc-emitted DWARF.
448 #
449 CTF_FLAGS_sparc = -g -Wc,-Qiselect-T1 $(C99MODE) $(CNOGLOBAL) $(CDWARFSTR)
450 CTF_FLAGS_i386  = -g $(C99MODE) $(CNOGLOBAL) $(CDWARFSTR)

452 CTF_FLAGS_sparcv9       = $(CTF_FLAGS_sparc)
453 CTF_FLAGS_amd64         = $(CTF_FLAGS_i386)

455 # Sun Studio produces broken userland code when saving arguments.
456 $(__GNUC)CTF_FLAGS_amd64 += $(SAVEARGS)

458 CTF_FLAGS_32    = $(CTF_FLAGS_$(MACH)) $(DEBUGFORMAT)
459 CTF_FLAGS_64    = $(CTF_FLAGS_$(MACH64)) $(DEBUGFORMAT)
460 CTF_FLAGS       = $(CTF_FLAGS_32)

462 #
463 # Flags used with genoffsets
464 #
465 GOFLAGS = -_noecho \
466         $(CALLSYMS) \
467         $(CDWARFSTR)

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

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

475 #
476 # tradeoff time for space (smaller is better)
477 #
478 sparc_SPACEFLAG         = -xspace -W0,-Lt
479 sparcv9_SPACEFLAG       = -xspace -W0,-Lt
480 i386_SPACEFLAG          = -xspace
481 amd64_SPACEFLAG         =

483 SPACEFLAG               = $($(MACH)_SPACEFLAG)
484 SPACEFLAG64             = $($(MACH64)_SPACEFLAG)

486 #
487 # The Sun Studio 11 compiler has changed the behaviour of integer
488 # wrap arounds and so a flag is needed to use the legacy behaviour
489 # (without this flag panics/hangs could be exposed within the source).
490 #
491 sparc_IROPTFLAG         = -W2,-xwrap_int
492 sparcv9_IROPTFLAG       = -W2,-xwrap_int
493 i386_IROPTFLAG          =
494 amd64_IROPTFLAG         =

496 IROPTFLAG               = $($(MACH)_IROPTFLAG)
497 IROPTFLAG64             = $($(MACH64)_IROPTFLAG)

499 sparc_XREGSFLAG         = -xregs=no%appl
500 sparcv9_XREGSFLAG       = -xregs=no%appl
501 i386_XREGSFLAG          =
502 amd64_XREGSFLAG         =

504 XREGSFLAG               = $($(MACH)_XREGSFLAG)
505 XREGSFLAG64             = $($(MACH64)_XREGSFLAG)

507 # dmake SOURCEDEBUG=yes ... enables source-level debugging information, and
508 # avoids stripping it.
509 SOURCEDEBUG     = $(POUND_SIGN)
510 SRCDBGBLD       = $(SOURCEDEBUG:yes=)

512 #
```

```
513 # These variables are intended ONLY for use by developers to safely pass extra
514 # flags to the compilers without unintentionally overriding Makefile-set
515 # flags.  They should NEVER be set to any value in a Makefile.
516 #
517 # They come last in the associated FLAGS variable such that they can
518 # explicitly override things if necessary, there are gaps in this, but it's
519 # the best we can manage.
520 #
521 CUSERFLAGS              =
522 CUSERFLAGS64            = $(CUSERFLAGS)
523 CCUSERFLAGS             =
524 CCUSERFLAGS64           = $(CCUSERFLAGS)

526 CSOURCEDEBUGFLAGS       =
527 CCSOURCEDEBUGFLAGS      =
528 $(SRCDBGBLD)CSOURCEDEBUGFLAGS   = -g -xs
529 $(SRCDBGBLD)CCSOURCEDEBUGFLAGS  = -g -xs

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

548 DTEXTDOM=-DTEXT_DOMAIN=\"$(TEXT_DOMAIN)\"       # For messaging.
549 DTS_ERRNO=-D_TS_ERRNO
550 CPPFLAGS.master=$(DTEXTDOM) $(DTS_ERRNO) \
551         $(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) $(ENVCPPFLAGS4) \
552         $(ADJUNCT_PROTO:%=-I%/usr/include)
553 CPPFLAGS.native=$(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) \
554                 $(ENVCPPFLAGS4) -I$(NATIVE_ADJUNCT)/include
555 CPPFLAGS=       $(CPPFLAGS.master)
556 AS_CPPFLAGS=    $(CPPFLAGS.master)
557 JAVAFLAGS=      -deprecation

559 #
560 # For source message catalogue
561 #
562 .SUFFIXES: $(SUFFIXES) .i .po
563 MSGROOT= $(ROOT)/catalog
564 MSGDOMAIN= $(MSGROOT)/$(TEXT_DOMAIN)
565 MSGDOMAINPOFILE = $(MSGDOMAIN)/$(POFILE)
566 DCMSGDOMAIN= $(MSGROOT)/LC_TIME/$(TEXT_DOMAIN)
567 DCMSGDOMAINPOFILE = $(DCMSGDOMAIN)/$(DCFILE:.dc=.po)

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



```
579         $(RM) $(<F).po $<.i

581 #
582 # This is overwritten by local Makefile when PROG is a list.
583 #
584 POFILE= $(PROG).po

586 sparc_CCFLAGS=          -cg92 -compat=4 \
587                         -Qoption ccfe -messages=no%anachronism \
588                         $(CCERRWARN)
589 sparcv9_CCFLAGS=        $(sparcv9_XARCH) -dalign -compat=5 \
590                         -Qoption ccfe -messages=no%anachronism \
591                         -Qoption ccfe -features=no%conststrings \
592                         $(CCCREGSYM) \
593                         $(CCERRWARN)
594 i386_CCFLAGS=           -compat=4 \
595                         -Qoption ccfe -messages=no%anachronism \
596                         -Qoption ccfe -features=no%conststrings \
597                         $(CCERRWARN)
598 amd64_CCFLAGS=          $(amd64_XARCH) -compat=5 \
599                         -Qoption ccfe -messages=no%anachronism \
600                         -Qoption ccfe -features=no%conststrings \
601                         $(CCERRWARN)

603 sparc_CCOPTFLAG=        -O
604 sparcv9_CCOPTFLAG=      -O
605 i386_CCOPTFLAG=         -O
606 amd64_CCOPTFLAG=        -O

608 CCOPTFLAG=      $($(MACH)_CCOPTFLAG)
609 CCOPTFLAG64=    $($(MACH64)_CCOPTFLAG)
610 CCFLAGS=        $(CCOPTFLAG) $($(MACH)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
611                 $(CCUSERFLAGS)
612 CCFLAGS64=      $(CCOPTFLAG64) $($(MACH64)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
613                 $(CCUSERFLAGS64)

615 #
616 #
617 #
618 ELFWRAP_FLAGS   =
619 ELFWRAP_FLAGS64 =       -64

621 #
622 # Various mapfiles that are used throughout the build, and delivered to
623 # /usr/lib/ld.
624 #
625 MAPFILE.NED_i386 =      $(SRC)/common/mapfiles/common/map.noexdata
626 MAPFILE.NED_sparc =
627 MAPFILE.NED =           $(MAPFILE.NED_$(MACH))
628 MAPFILE.PGA =           $(SRC)/common/mapfiles/common/map.pagealign
629 MAPFILE.NES =           $(SRC)/common/mapfiles/common/map.noexstk
630 MAPFILE.FLT =           $(SRC)/common/mapfiles/common/map.filter
631 MAPFILE.LEX =           $(SRC)/common/mapfiles/common/map.lex.yy

633 #
634 # Generated mapfiles that are compiler specific, and used throughout the
635 # build.  These mapfiles are not delivered in /usr/lib/ld.
636 #
637 MAPFILE.NGB_sparc=      $(SRC)/common/mapfiles/gen/sparc_cc_map.noexeglobs
638 $(__GNUC64)MAPFILE.NGB_sparc= \
639                         $(SRC)/common/mapfiles/gen/sparc_gcc_map.noexeglobs
640 MAPFILE.NGB_sparcv9=    $(SRC)/common/mapfiles/gen/sparcv9_cc_map.noexeglobs
641 $(__GNUC64)MAPFILE.NGB_sparcv9= \
642                         $(SRC)/common/mapfiles/gen/sparcv9_gcc_map.noexeglobs
643 MAPFILE.NGB_i386=       $(SRC)/common/mapfiles/gen/i386_cc_map.noexeglobs
644 $(__GNUC64)MAPFILE.NGB_i386= \
```

```
645                         $(SRC)/common/mapfiles/gen/i386_gcc_map.noexeglobs
646 MAPFILE.NGB_amd64=      $(SRC)/common/mapfiles/gen/amd64_cc_map.noexeglobs
647 $(__GNUC64)MAPFILE.NGB_amd64= \
648                         $(SRC)/common/mapfiles/gen/amd64_gcc_map.noexeglobs
649 MAPFILE.NGB =           $(MAPFILE.NGB_$(MACH))

651 #
652 # A generic interface mapfile name, used by various dynamic objects to define
653 # the interfaces and interposers the object must export.
654 #
655 MAPFILE.INT =           mapfile-intf

657 #
658 # LDLIBS32 can be set in the environment to override the following assignment.
659 # LDLIBS64 can be set to override the assignment made in Makefile.master.64.
660 # These environment settings make sure that no libraries are searched outside
661 # of the local workspace proto area:
662 #       LDLIBS32=-YP,$ROOT/lib:$ROOT/usr/lib
663 #       LDLIBS64=-YP,$ROOT/lib/$MACH64:$ROOT/usr/lib/$MACH64
664 #
665 LDLIBS32 =      $(ENVLDLIBS1) $(ENVLDLIBS2) $(ENVLDLIBS3)
666 LDLIBS32 +=     $(ADJUNCT_PROTO:%=-L%/usr/lib -L%/lib)
667 LDLIBS.cmd =    $(LDLIBS32)
668 LDLIBS.lib =    $(LDLIBS32)
669 #
670 # Define compilation macros.
671 #
672 COMPILE.c=      $(CC) $(CFLAGS) $(CPPFLAGS) -c
673 COMPILE64.c=    $(CC) $(CFLAGS64) $(CPPFLAGS) -c
674 COMPILE.cc=     $(CCC) $(CCFLAGS) $(CPPFLAGS) -c
675 COMPILE64.cc=   $(CCC) $(CCFLAGS64) $(CPPFLAGS) -c
676 COMPILE.s=      $(AS) $(ASFLAGS) $(AS_CPPFLAGS)
677 COMPILE64.s=    $(AS) $(ASFLAGS) $($(MACH64)_AS_XARCH) $(AS_CPPFLAGS)
678 COMPILE.d=      $(DTRACE) -G -32
679 COMPILE64.d=    $(DTRACE) -G -64
680 COMPILE.b=      $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))
681 COMPILE64.b=    $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))

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

686 #
687 # Link time macros
688 #
689 CCNEEDED                = -lC
690 CCEXTNEEDED             = -lCrun -lCstd
691 $(__GNUC)CCNEEDED       = -L$(GCCLIBDIR) -lstdc++ -lgcc_s
692 $(__GNUC)CCEXTNEEDED    = $(CCNEEDED)

694 LINK.c=         $(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS)
695 LINK64.c=       $(CC) $(CFLAGS64) $(CPPFLAGS) $(LDFLAGS)
696 NORUNPATH=      -norunpath -nolib
697 LINK.cc=        $(CCC) $(CCFLAGS) $(CPPFLAGS) $(NORUNPATH) \
698                 $(LDFLAGS) $(CCNEEDED)
699 LINK64.cc=      $(CCC) $(CCFLAGS64) $(CPPFLAGS) $(NORUNPATH) \
700                 $(LDFLAGS) $(CCNEEDED)

702 #
703 # lint macros
704 #
705 # Note that the undefine of __PRAGMA_REDEFINE_EXTNAME can be removed once
706 # ON is built with a version of lint that has the fix for 4484186.
707 #
708 ALWAYS_LINT_DEFS =      -errtags=yes -s
709 ALWAYS_LINT_DEFS +=     -erroff=E_PTRDIFF_OVERFLOW
710 ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_NARROW_CONV
```



```
711 ALWAYS_LINT_DEFS +=     -U__PRAGMA_REDEFINE_EXTNAME
712 ALWAYS_LINT_DEFS +=     $(C99LMODE)
713 ALWAYS_LINT_DEFS +=     -errsecurity=$(SECLEVEL)
714 ALWAYS_LINT_DEFS +=     -erroff=E_SEC_CREAT_WITHOUT_EXCL
715 ALWAYS_LINT_DEFS +=     -erroff=E_SEC_FORBIDDEN_WARN_CREAT
716 # XX64 -- really only needed for amd64 lint
717 ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_INT_TO_SMALL_INT
718 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_CONST_TO_SMALL_INT
719 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_TO_SMALL_INT
720 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_TO_PTR_FROM_INT
721 ALWAYS_LINT_DEFS +=     -erroff=E_COMP_INT_WITH_LARGE_INT
722 ALWAYS_LINT_DEFS +=     -erroff=E_INTEGRAL_CONST_EXP_EXPECTED
723 ALWAYS_LINT_DEFS +=     -erroff=E_PASS_INT_TO_SMALL_INT
724 ALWAYS_LINT_DEFS +=     -erroff=E_PTR_CONV_LOSES_BITS

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

730 SECLEVEL=       core
731 LINT.c=         $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS) $(CPPFLAGS) \
732                 $(ALWAYS_LINT_DEFS)
733 LINT64.c=       $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS64) $(CPPFLAGS) \
734                 $(ALWAYS_LINT_DEFS)
735 LINT.s=         $(LINT.c)

737 # For some future builds, NATIVE_MACH and MACH might be different.
738 # Therefore, NATIVE_MACH needs to be redefined in the
739 # environment as ‘uname -p‘ to override this macro.
740 #
741 # For now at least, we cross-compile amd64 on i386 machines.
742 NATIVE_MACH=    $(MACH:amd64=i386)

744 # Define native compilation macros
745 #

747 # Base directory where compilers are loaded.
748 # Defined here so it can be overridden by developer.
749 #
750 SPRO_ROOT=              $(BUILD_TOOLS)/SUNWspro
751 SPRO_VROOT=             $(SPRO_ROOT)/SS12
752 GNU_ROOT=               $(SFW_ROOT)

754 # Till SS12u1 formally becomes the NV CBE, LINT is hard
755 # coded to be picked up from the $SPRO_ROOT/sunstudio12.1/
756 # location. Impacted variables are sparc_LINT, sparcv9_LINT,
757 # i386_LINT, amd64_LINT.
758 # Reset them when SS12u1 is rolled out.
759 #

761 # Specify platform compiler versions for languages
762 # that we use (currently only c and c++).
763 #
764 sparc_CC=               $(ONBLD_TOOLS)/bin/$(MACH)/cw -_cc
765 $(__GNUC)sparc_CC=      $(ONBLD_TOOLS)/bin/$(MACH)/cw -_gcc
766 sparc_CCC=              $(ONBLD_TOOLS)/bin/$(MACH)/cw -_CC
767 $(__GNUC)sparc_CCC=     $(ONBLD_TOOLS)/bin/$(MACH)/cw -_g++
768 sparc_CPP=              /usr/ccs/lib/cpp
769 sparc_AS=               /usr/ccs/bin/as -xregsym=no
770 sparc_LD=               /usr/ccs/bin/ld
771 sparc_LINT=             $(SPRO_ROOT)/sunstudio12.1/bin/lint

773 sparcv9_CC=             $(ONBLD_TOOLS)/bin/$(MACH)/cw -_cc
774 $(__GNUC64)sparcv9_CC=  $(ONBLD_TOOLS)/bin/$(MACH)/cw -_gcc
775 sparcv9_CCC=            $(ONBLD_TOOLS)/bin/$(MACH)/cw -_CC
776 $(__GNUC64)sparcv9_CCC= $(ONBLD_TOOLS)/bin/$(MACH)/cw -_g++
```

```
777 sparcv9_CPP=            /usr/ccs/lib/cpp
778 sparcv9_AS=             /usr/ccs/bin/as -xregsym=no
779 sparcv9_LD=             /usr/ccs/bin/ld
780 sparcv9_LINT=           $(SPRO_ROOT)/sunstudio12.1/bin/lint

782 i386_CC=               $(ONBLD_TOOLS)/bin/$(MACH)/cw -_cc
783 $(__GNUC)i386_CC=      $(ONBLD_TOOLS)/bin/$(MACH)/cw -_gcc
784 i386_CCC=              $(ONBLD_TOOLS)/bin/$(MACH)/cw -_CC
785 $(__GNUC)i386_CCC=     $(ONBLD_TOOLS)/bin/$(MACH)/cw -_g++
786 i386_CPP=              /usr/ccs/lib/cpp
787 i386_AS=               /usr/ccs/bin/as
788 $(__GNUC)i386_AS=      $(ONBLD_TOOLS)/bin/$(MACH)/aw
789 i386_LD=               /usr/ccs/bin/ld
790 i386_LINT=             $(SPRO_ROOT)/sunstudio12.1/bin/lint

792 amd64_CC=              $(ONBLD_TOOLS)/bin/$(MACH)/cw -_cc
793 $(__GNUC64)amd64_CC=   $(ONBLD_TOOLS)/bin/$(MACH)/cw -_gcc
794 amd64_CCC=             $(ONBLD_TOOLS)/bin/$(MACH)/cw -_CC
795 $(__GNUC64)amd64_CCC=  $(ONBLD_TOOLS)/bin/$(MACH)/cw -_g++
796 amd64_CPP=             /usr/ccs/lib/cpp
797 amd64_AS=              $(ONBLD_TOOLS)/bin/$(MACH)/aw
798 amd64_LD=              /usr/ccs/bin/ld
799 amd64_LINT=            $(SPRO_ROOT)/sunstudio12.1/bin/lint

801 NATIVECC=              $($(NATIVE_MACH)_CC)
802 NATIVECCC=             $($(NATIVE_MACH)_CCC)
803 NATIVECPP=             $($(NATIVE_MACH)_CPP)
804 NATIVEAS=              $($(NATIVE_MACH)_AS)
805 NATIVELD=              $($(NATIVE_MACH)_LD)
806 NATIVELINT=            $($(NATIVE_MACH)_LINT)

808 #
809 # Makefile.master.64 overrides these settings
810 #
811 CC=                    $(NATIVECC)
812 CCC=                   $(NATIVECCC)
813 CPP=                   $(NATIVECPP)
814 AS=                    $(NATIVEAS)
815 LD=                    $(NATIVELD)
816 LINT=                  $(NATIVELINT)

818 # The real compilers used for this build
819 CW_CC_CMD=             $(CC) -_compiler
820 CW_CCC_CMD=            $(CCC) -_compiler
821 REAL_CC=               $(CW_CC_CMD:sh)
822 REAL_CCC=              $(CW_CCC_CMD:sh)

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

827 BDIRECT=        -Bdirect
828 BDYNAMIC=       -Bdynamic
829 BLOCAL=         -Blocal
830 BNODIRECT=      -Bnodirect
831 BREDUCE=        -Breduce
832 BSTATIC=        -Bstatic

834 ZDEFS=          -zdefs
835 ZDIRECT=        -zdirect
836 ZIGNORE=        -zignore
837 ZINITFIRST=     -zinitfirst
838 ZINTERPOSE=     -zinterpose
839 ZLAZYLOAD=      -zlazyload
840 ZLOADFLTR=      -zloadfltr
841 ZMULDEFS=       -zmuldefs
842 ZNODEFAULTLIB=  -znodefaultlib
```



```
843 ZNODEFS=        -znodefs
844 ZNODELETE=      -znodelete
845 ZNODLOPEN=      -znodlopen
846 ZNODUMP=        -znodump
847 ZNOLAZYLOAD=    -znolazyload
848 ZNOLDYNSYM=     -znoldynsym
849 ZNORELOC=       -znoreloc
850 ZNOVERSION=     -znoversion
851 ZRECORD=        -zrecord
852 ZREDLOCSYM=     -zredlocsym
853 ZTEXT=          -ztext
854 ZVERBOSE=       -zverbose

856 GSHARED=        -G
857 CCMT=           -mt

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

862 sparc_C_PICFLAGS =      -K pic
863 sparcv9_C_PICFLAGS =    -K pic
864 i386_C_PICFLAGS =       -K pic
865 amd64_C_PICFLAGS =      -K pic
866 C_PICFLAGS =            $($(MACH)_C_PICFLAGS)
867 C_PICFLAGS64 =          $($(MACH64)_C_PICFLAGS)

869 sparc_C_BIGPICFLAGS =   -K PIC
870 sparcv9_C_BIGPICFLAGS = -K PIC
871 i386_C_BIGPICFLAGS =    -K PIC
872 amd64_C_BIGPICFLAGS =   -K PIC
873 C_BIGPICFLAGS =         $($(MACH)_C_BIGPICFLAGS)
874 C_BIGPICFLAGS64 =       $($(MACH64)_C_BIGPICFLAGS)

876 # CC requires there to be no space between '-K' and 'pic' or 'PIC'.
877 sparc_CC_PICFLAGS =     -Kpic
878 sparcv9_CC_PICFLAGS =   -KPIC
879 i386_CC_PICFLAGS =      -Kpic
880 amd64_CC_PICFLAGS =     -Kpic
881 CC_PICFLAGS =           $($(MACH)_CC_PICFLAGS)
882 CC_PICFLAGS64 =         $($(MACH64)_CC_PICFLAGS)

884 AS_PICFLAGS=            $(C_PICFLAGS)
885 AS_BIGPICFLAGS=         $(C_BIGPICFLAGS)

887 #
888 # Default label for CTF sections
889 #
890 CTFCVTFLAGS=            -i -L VERSION
891 $(SRCDBGBLD)CTFCVTFLAGS         += -g

893 #
894 # Override to pass module-specific flags to ctfmerge.  Currently used only by
895 # krtld to turn on fuzzy matching, and source-level debugging to inhibit
896 # stripping.
897 #
898 CTFMRGFLAGS=
899 $(SRCDBGBLD)CTFMRGFLAGS         += -g


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

904 ELFSIGN_O=      $(TRUE)
905 ELFSIGN_CRYPTO= $(ELFSIGN_O)
906 ELFSIGN_OBJECT= $(ELFSIGN_O)

908 # Rules (normally from make.rules) and macros which are used for post
```

```
909 # processing files. Normally, these do stripping of the comment section
910 # automatically.
911 #    RELEASE_CM:        Should be editted to reflect the release.
912 #    POST_PROCESS_O:    Post-processing for '.o' files.
913 #    POST_PROCESS_A:    Post-processing for '.a' files (currently null).
914 #    POST_PROCESS_SO:   Post-processing for '.so' files.
915 #    POST_PROCESS:      Post-processing for executable files (no suffix).
916 # Note that these macros are not completely generalized as they are to be
917 # used with the file name to be processed following.
918 #
919 # It is left as an exercise to Release Engineering to embellish the generation
920 # of the release comment string.
921 #
922 #       If this is a standard development build:
923 #               compress the comment section (mcs -c)
924 #               add the standard comment (mcs -a $(RELEASE_CM))
925 #               add the development specific comment (mcs -a $(DEV_CM))
926 #
927 #       If this is an installation build:
928 #               delete the comment section (mcs -d)
929 #               add the standard comment (mcs -a $(RELEASE_CM))
930 #               add the development specific comment (mcs -a $(DEV_CM))
931 #
932 #       If this is an release build:
933 #               delete the comment section (mcs -d)
934 #               add the standard comment (mcs -a $(RELEASE_CM))
935 #
936 # The following list of macros are used in the definition of RELEASE_CM
937 # which is used to label all binaries in the build:
938 #
939 #       RELEASE         Specific release of the build, eg: 5.2
940 #       RELEASE_MAJOR   Major version number part of $(RELEASE)
941 #       RELEASE_MINOR   Minor version number part of $(RELEASE)
942 #       VERSION         Version of the build (alpha, beta, Generic)
943 #       PATCHID         If this is a patch this value should contain
944 #                       the patchid value (eg: "Generic 100832-01"), otherwise
945 #                       it will be set to $(VERSION)
946 #       RELEASE_DATE    Date of the Release Build
947 #       PATCH_DATE      Date the patch was created, if this is blank it
948 #                       will default to the RELEASE_DATE
949 #
950 RELEASE_MAJOR=  5
951 RELEASE_MINOR=  11
952 RELEASE=        $(RELEASE_MAJOR).$(RELEASE_MINOR)
953 VERSION=        SunOS Development
954 PATCHID=        $(VERSION)
955 RELEASE_DATE=   release date not set
956 PATCH_DATE=     $(RELEASE_DATE)
957 RELEASE_CM=     "@($(POUND_SIGN))SunOS $(RELEASE) $(PATCHID) $(PATCH_DATE)"
958 DEV_CM=         "@($(POUND_SIGN))SunOS Internal Development: non-nightly build"

960 PROCESS_COMMENT=  @?${MCS} -d -a $(RELEASE_CM) -a $(DEV_CM)
971 PROCESS_COMMENT=                  @?${MCS} -c -a $(RELEASE_CM) -a $(DEV_CM)
972 $(STRIP_COMMENTS)PROCESS_COMMENT=  @?${MCS} -d -a $(RELEASE_CM) -a $(DEV_CM)
961 $(RELEASE_BUILD)PROCESS_COMMENT=   @?${MCS} -d -a $(RELEASE_CM)

963 STRIP_STABS=                       :
964 $(RELEASE_BUILD)STRIP_STABS=       $(STRIP) -x $@
965 $(SRCDBGBLD)STRIP_STABS=           :

967 POST_PROCESS_O=         $(PROCESS_COMMENT) $@
968 POST_PROCESS_A=
969 POST_PROCESS_SO=        $(PROCESS_COMMENT) $@ ; $(STRIP_STABS) ; \
970                         $(ELFSIGN_OBJECT)
971 POST_PROCESS=           $(PROCESS_COMMENT) $@ ; $(STRIP_STABS) ; \
972                         $(ELFSIGN_OBJECT)
```



```
974 #
975 # chk4ubin is a tool that inspects a module for a symbol table
976 # ELF section size which can trigger an OBP bug on older platforms.
977 # This problem affects only specific sun4u bootable modules.
978 #
979 CHK4UBIN=       $(ONBLD_TOOLS)/bin/$(MACH)/chk4ubin
980 CHK4UBINFLAGS=
981 CHK4UBINARY=    $(CHK4UBIN) $(CHK4UBINFLAGS) $@

983 #
984 # PKGARCHIVE specifies the default location where packages should be
985 # placed if built.
986 #
987 $(RELEASE_BUILD)PKGARCHIVESUFFIX=       -nd
988 PKGARCHIVE=$(SRC)/../../packages/$(MACH)/nightly$(PKGARCHIVESUFFIX)

990 #
991 # The repositories will be created with these publisher settings.  To
992 # update an image to the resulting repositories, this must match the
993 # publisher name provided to "pkg set-publisher."
994 #
995 PKGPUBLISHER_REDIST=    on-nightly
996 PKGPUBLISHER_NONREDIST= on-extra

998 #       Default build rules which perform comment section post-processing.
999 #
1000 .c:
1001         $(LINK.c) -o $@ $< $(LDLIBS)
1002         $(POST_PROCESS)
1003 .c.o:
1004         $(COMPILE.c) $(OUTPUT_OPTION) $< $(CTFCONVERT_HOOK)
1005         $(POST_PROCESS_O)
1006 .c.a:
1007         $(COMPILE.c) -o $% $<
1008         $(PROCESS_COMMENT) $%
1009         $(AR) $(ARFLAGS) $@ $%
1010         $(RM) $%
1011 .s.o:
1012         $(COMPILE.s) -o $@ $<
1013         $(POST_PROCESS_O)
1014 .s.a:
1015         $(COMPILE.s) -o $% $<
1016         $(PROCESS_COMMENT) $%
1017         $(AR) $(ARFLAGS) $@ $%
1018         $(RM) $%
1019 .cc:
1020         $(LINK.cc) -o $@ $< $(LDLIBS)
1021         $(POST_PROCESS)
1022 .cc.o:
1023         $(COMPILE.cc) $(OUTPUT_OPTION) $<
1024         $(POST_PROCESS_O)
1025 .cc.a:
1026         $(COMPILE.cc) -o $% $<
1027         $(AR) $(ARFLAGS) $@ $%
1028         $(PROCESS_COMMENT) $%
1029         $(RM) $%
1030 .y:
1031         $(YACC.y) $<
1032         $(LINK.c) -o $@ y.tab.c $(LDLIBS)
1033         $(POST_PROCESS)
1034         $(RM) y.tab.c
1035 .y.o:
1036         $(YACC.y) $<
1037         $(COMPILE.c) -o $@ y.tab.c $(CTFCONVERT_HOOK)
1038         $(POST_PROCESS_O)
```

```
1039         $(RM) y.tab.c
1040 .l:
1041         $(RM) $*.c
1042         $(LEX.l) $< > $*.c
1043         $(LINK.c) -o $@ $*.c -ll $(LDLIBS)
1044         $(POST_PROCESS)
1045         $(RM) $*.c
1046 .l.o:
1047         $(RM) $*.c
1048         $(LEX.l) $< > $*.c
1049         $(COMPILE.c) -o $@ $*.c $(CTFCONVERT_HOOK)
1050         $(POST_PROCESS_O)
1051         $(RM) $*.c

1053 .bin.o:
1054         $(COMPILE.b) -o $@ $<
1055         $(POST_PROCESS_O)

1057 .java.class:
1058         $(COMPILE.java) $<

1060 # Bourne and Korn shell script message catalog build rules.
1061 # We extract all gettext strings with sed(1) (being careful to permit
1062 # multiple gettext strings on the same line), weed out the dups, and
1063 # build the catalogue with awk(1).

1065 .sh.po .ksh.po:
1066         $(SED) -n -e ":a"                                   \
1067                   -e "h"                                        \
1068                   -e "s/.*gettext *\(\"[^\"]*\"\).*/\1/p"       \
1069                   -e "x"                                        \
1070                   -e "s/\(.*\)gettext *\"[^\"]*\"\(.*\)/\1\2/"  \
1071                   -e "t a"                                      \
1072                 $< | sort -u | awk '{ print "msgid\t" $$0 "\nmsgstr" }' > $@

1074 #
1075 # Python and Perl executable and message catalog build rules.
1076 #
1077 .SUFFIXES: .pl .pm .py .pyc

1079 .pl:
1080         $(RM) $@;
1081         $(SED) -e "s@TEXT_DOMAIN@\"$(TEXT_DOMAIN)\"@" $< > $@;
1082         $(CHMOD) +x $@

1084 .py:
1085         $(RM) $@; $(CAT) $< > $@; $(CHMOD) +x $@

1087 .py.pyc:
1088         $(RM) $@
1089         $(PYTHON) -mpy_compile $<
1090         @[ $(<)c = $@ ] || $(MV) $(<)c $@

1092 .py.po:
1093         $(GNUXGETTEXT) $(GNUXGETFLAGS) -d $(<F:%.py=%) $< ;

1095 .pl.po .pm.po:
1096         $(XGETTEXT) $(XGETFLAGS) -d $(<F) $< ;
1097         $(RM)   $@ ;
1098         $(SED) "/^domain/d" < $(<F).po > $@ ;
1099         $(RM) $(<F).po

1101 #
1102 # When using xgettext, we want messages to go to the default domain,
1103 # rather than the specified one.  This special version of the
1104 # COMPILE.cpp macro effectively prevents expansion of TEXT_DOMAIN,
```



```
1105 # causing xgettext to put all messages into the default domain.
1106 #
1107 CPPFORPO=$(COMPILE.cpp:\"$(TEXT_DOMAIN)\"=TEXT_DOMAIN)

1109 .c.i:
1110         $(CPPFORPO) $< > $@

1112 .h.i:
1113         $(CPPFORPO) $< > $@

1115 .y.i:
1116         $(YACC) -d $<
1117         $(CPPFORPO) y.tab.c  > $@
1118         $(RM) y.tab.c

1120 .l.i:
1121         $(LEX) $<
1122         $(CPPFORPO) lex.yy.c  > $@
1123         $(RM) lex.yy.c

1125 .c.po:
1126         $(CPPFORPO) $< > $<.i
1127         $(BUILD.po)

1129 .y.po:
1130         $(YACC) -d $<
1131         $(CPPFORPO) y.tab.c  > $<.i
1132         $(BUILD.po)
1133         $(RM) y.tab.c

1135 .l.po:
1136         $(LEX) $<
1137         $(CPPFORPO) lex.yy.c  > $<.i
1138         $(BUILD.po)
1139         $(RM) lex.yy.c

1141 #
1142 # Rules to perform stylistic checks
1143 #
1144 .SUFFIXES: .x .xml .check .xmlchk

1146 .h.check:
1147         $(DOT_H_CHECK)

1149 .x.check:
1150         $(DOT_X_CHECK)

1152 .xml.xmlchk:
1153         $(MANIFEST_CHECK)

1155 #
1156 # Include rules to render automated sccs get rules "safe".
1157 #
1158 include $(SRC)/Makefile.noget
```

```
**********************************************************
    3414 Sat Dec 14 21:34:46 2013
new/usr/src/cmd/sgs/libconv/Makefile.com
4265 remove INTERNAL_RELEASE_BUILD
**********************************************************
#
# 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 (c) 1994, 2010, Oracle and/or its affiliates. All rights reserved.
#

LIBRARY =       libconv.a

COMOBJS32 =     cap_machelf32.o         dynamic_machelf32.o \
                globals_machelf32.o     sections_machelf32.o \
                symbols_machelf32.o     symbols_sparc_machelf32.o

COMOBJS64 =     cap_machelf64.o         dynamic_machelf64.o \
                globals_machelf64.o     sections_machelf64.o \
                symbols_machelf64.o     symbols_sparc_machelf64.o

COMOBJS=        arch.o                  audit.o \
                c_literal.o \
                cap.o                   config.o \
                corenote.o              data.o \
                deftag.o                demangle.o \
                dl.o                    dwarf.o \
                dwarf_ehe.o             dynamic.o \
                elf.o                   entry.o \
                globals.o               group.o \
                lddstub.o               map.o \
                phdr.o                  relocate.o \
                relocate_i386.o         relocate_amd64.o \
                relocate_sparc.o        sections.o \
                segments.o              strproc.o \
                symbols.o               syminfo.o \
                tokens.o                time.o \
                version.o

ELFCAP_OBJS=    elfcap.o

ASOBJS=         vernote.o

BLTOBJS=        arch_msg.o              audit_msg.o \
                c_literal_msg.o \
                cap_msg.o               config_msg.o \
                corenote_msg.o          data_msg.o \
```



```
                deftag_msg.o            demangle_msg.o \
                dl_msg.o                dwarf_msg.o \
                dwarf_ehe_msg.o         dynamic_msg.o \
                elf_msg.o               entry_msg.o \
                globals_msg.o           group_msg.o \
                map_msg.o               lddstub_msg.o \
                phdr_msg.o              relocate_amd64_msg.o \
                relocate_i386_msg.o     relocate_sparc_msg.o \
                sections_msg.o          segments_msg.o \
                symbols_msg.o           symbols_sparc_msg.o \
                syminfo_msg.o           time_msg.o \
                version_msg.o


OBJECTS =       $(COMOBJS) $(COMOBJS32) $(COMOBJS64) $(ELFCAP_OBJS) \
                $(ASOBJS) $(BLTOBJS)

#
# This library is unusual since it's a static archive of PIC objects.
# Since static archives should never contain CTF data (regardless of
# whether the object code is position-independent), we disable CTF.
#
NOCTFOBJS =     $(OBJECTS)
CTFMERGE_LIB =  :

include         $(SRC)/lib/Makefile.lib
include         $(SRC)/cmd/sgs/Makefile.com

CERRWARN        += -_gcc=-Wno-type-limits
CERRWARN        += -_gcc=-Wno-switch

CTFCONVERT_O=

README_REVISION=../../packages/common/readme_revision
ONLDREADME=     ../../packages/common/SUNWonld-README

PICS=           $(OBJECTS:%=pics/%)

CPPFLAGS +=     -I$(SRCBASE)/lib/libc/inc -I$(ELFCAP) \
                -I$(SRC)/common/sgsrtcid

ARFLAGS=        cr

AS_CPPFLAGS=    -P -D_ASM $(CPPFLAGS)

BLTDATA=        $(BLTOBJS:%.o=%.c) $(BLTOBJS:%.o=%.h) report_bufsize.h

SRCS=           ../common/llib-lconv
LINTSRCS=       $(COMOBJS:%.o=../common/%.c) \
                    $(COMOBJS_NOMSG:%.o=../common/%.c) \
                    $(ELFCOM_OBJS:%.o=$(ELFCAP)/%.c) ../common/lintsup.c
LINTSRCS32 =    $(COMOBJS32:%32.o=../common/%.c)
LINTSRCS64 =    $(COMOBJS64:%64.o=../common/%.c)

# INTERNAL_RELEASE_BUILD is defined by standard full builds (nightly),
# but not for sgs builds we do for development. The result of these
# two lines is that dev builds pass -d to the readme_revision script,
# generating a more detailed version string for the linker components
# that includes the workspace, user, CR, and date. Official builds get
# a simpler uncluttered version string.
VERNOTE_DEBUG= -d
$(INTERNAL_RELEASE_BUILD)VERNOTE_DEBUG=

SGSMSGTARG=     $(BLTOBJS:%_msg.o=../common/%.msg)

LINTFLAGS +=    -u
```

```
119 LINTFLAGS64 +=  -u

121 CLEANFILES +=   $(BLTDATA) $(LINTOUTS) bld_vernote vernote.s
122 CLOBBERFILES += $(LINTLIBS)
```

```
**********************************************************
    2942 Sat Dec 14 21:34:46 2013
new/usr/src/cmd/sgs/libconv/Makefile.targ
4265 remove INTERNAL_RELEASE_BUILD
**********************************************************
#
# 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 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

pics/%.o:       ../common/%.c
                $(COMPILE.c) -o $@ $<
                $(POST_PROCESS_O)

pics/%.o:       %.s
                $(COMPILE.s) -o $@ $<
                $(POST_PROCESS_O)

pics/%32.o:     ../common/%.c
                $(COMPILE.c) -o $@ $<
                $(POST_PROCESS_O)

pics/%64.o:     ../common/%.c
                $(COMPILE.c) -D_ELF64 -o $@ $<
                $(POST_PROCESS_O)

pics/elfcap.o:  $(ELFCAP)/elfcap.c
                $(COMPILE.c) -o $@ $(ELFCAP)/elfcap.c
                $(POST_PROCESS_O)

# This rule generates the report_bufsize.h include file used by libconv
# code to ensure that their private buffer size calculations agree with
# the public values exposed in sgs/include/conv.h. The limit value
# supplied must be larger than the largest buffer used in libconv. There
# is little penalty for making it very large, because the header file is
# only included in error situations where the compilation will fail.
#
# We make this depend on Makefile.targ, because a change to Makefile.targ
# can change the limit, in which case we want to force everything to rebuild.
report_bufsize.h:       ../Makefile.targ
                perl ../../tools/libconv_mk_report_bufsize.pl 8000

../common/%.c ../common/%_machelf.c:    %_msg.h

%_msg.h %_msg.c: $(SGSMSG) ../common/%.msg report_bufsize.h
```



```
                $(SGSMSG) $(SGSMSGFLAGS) -h $*_msg.h -d $*_msg.c \
                        -n sgs_msg_libconv_$* $<

$(SGSMSG):      FRC
                @ cd $(SGSTOOLS)/$(MACH); pwd; $(MAKE) catalog
                @ pwd

vernote.s:      bld_vernote $(README_REVISION) $(ONLDREADME)
                ./bld_vernote \
                        -R 'perl $(README_REVISION) $(ONLDREADME)' \
                        -R 'perl $(README_REVISION) $(VERNOTE_DEBUG) \
                                $(ONLDREADME)' \
                        -r "$(RELEASE)" -o $@

bld_vernote:    ../common/bld_vernote.ksh
                $(RM) -f $@
                cp ../common/bld_vernote.ksh $@
                chmod a+x $@

$(LIBRARY):     pics $$(PICS)
                @ $(RM) $(LIBRARY)
                $(AR) $(ARFLAGS) $@ $(PICS)
                $(POST_PROCESS_A)

chkmsg:         $(LINTSRCS)
                sh $(CHKMSG) $(CHKMSGFLAGS) $(LINTSRCS)

.PARALLEL:      $(LINTOUT32) $(LINTOUT64) $(LINTLIB32) $(LINTLIB64)

lint:           $(LINTLIB32) $(LINTOUT32) $(LINTLIB64) $(LINTOUT64) \
                    .WAIT $(SGSLINTOUT)

catalog:

pics:
                -@mkdir -p $@

clobber:        clean
                -$(RM) $(LIBRARY) $(CLOBBERFILES)

clean:
                -$(RM) $(PICS) $(CLEANFILES)

delete:

include         $(SRC)/cmd/sgs/Makefile.targ
```

****
2873 Sat Dec 14 21:34:46 2013
new/usr/src/cmd/sgs/libconv/common/bld_vernote.ksh
4265 remove INTERNAL_RELEASE_BUILD
****
_____***unchanged_portion_omitted_***

```
107 notefile=""
108 release=""
109 revision=""

111 while getopts R:o:r: c
111 while getopts DR:o:r: c
112 do
113         case $c in
114         o)
115                 notefile=$OPTARG
116                 ;;
117         r)
118                 release=$OPTARG
119                 ;;
120         R)
121                 revision=$OPTARG
122                 ;;
123         \?)
124                 usage
125                 exit 1
126                 ;;
127         esac
128 done

130 if [[ ( -z $notefile ) || ( -z $release ) || ( -z $revision ) ]]; then
131         usage
132         exit 1
133 fi


136 if [[ $MACH = "sparc" ]]; then
137         build_sparcnote
138 elif [[ $MACH = "i386" ]]; then
139         build_i386note
140 else
141         echo "I don't know how to build a vernote.s for ${MACH}, so sorry"
142         exit 1
143 fi
```

```
**********************************************************
    1429 Sat Dec 14 21:34:46 2013
new/usr/src/cmd/sgs/packages/Makefile.com
4265 remove INTERNAL_RELEASE_BUILD
**********************************************************
#
# 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 (c) 1994, 2010, Oracle and/or its affiliates. All rights reserved.
#

include         $(SRC)/Makefile.master

LINTLOG=        ../lint.$(MACH).log

PKGARCHIVE=     .
DATAFILES=      copyright prototype_com prototype_$(MACH) postinstall \
                preremove depend checkinstall
README=         SUNWonld-README
FILES=          $(DATAFILES) pkginfo
PACKAGE=        SUNWonld
ROOTONLD=       $(ROOT)/opt/SUNWonld
ROOTREADME=     $(README:%=$(ROOTONLD)/%)

VERDEBUG=       -D
$(INTERNAL_RELEASE_BUILD)VERDEBUG=

CLEANFILES=     $(FILES) awk_pkginfo ../bld_awk_pkginfo $(LINTLOG)
CLOBBERFILES=   $(PACKAGE) $(LINTLOG).bak

../%:           ../common/%.ksh
                $(RM) $@
                cp $< $@
                chmod +x $@

$(ROOTONLD)/%:  ../common/%
                $(RM) $@
                cp $< $@
                chmod +x $@
```

```
**********************************************************
   10437 Sat Dec 14 21:34:47 2013
new/usr/src/tools/scripts/bldenv.sh
4265 remove INTERNAL_RELEASE_BUILD
**********************************************************
_____unchanged_portion_omitted_
 109 [+SEE ALSO?\bnightly\b(1)]
 110 '

 112 # main
 113 builtin basename

 115 # boolean flags (true/false)
 116 typeset flags=(
 117         typeset c=false
 118         typeset f=false
 119         typeset d=false
 120         typeset O=false
 121         typeset o=false
 122         typeset t=true
 123         typeset s=(
 124                 typeset e=false
 125                 typeset h=false
 126                 typeset d=false
 127                 typeset o=false
 128         )
 129 )

 131 typeset progname="$(basename -- "${0}")"

 133 OPTIND=1
 134 SUFFIX="-nd"

 136 while getopts -a "${progname}" "${USAGE}" OPT ; do
 137     case ${OPT} in
 138           c)    flags.c=true  ;;
 139           +c)   flags.c=false ;;
 140           f)    flags.f=true  ;;
 141           +f)   flags.f=false ;;
 142           d)    flags.d=true  SUFFIX=""    ;;
 143           +d)   flags.d=false SUFFIX="-nd" ;;
 144           t)    flags.t=true  ;;
 145           +t)   flags.t=false ;;
 146           \?)   usage ;;
 147     esac
 148 done
 149 shift $((OPTIND-1))

 151 # test that the path to the environment-setting file was given
 152 if (( $# < 1 )) ; then
 153         usage
 154 fi

 156 # force locale to C
 157 export \
 158         LC_COLLATE=C \
 159         LC_CTYPE=C \
 160         LC_MESSAGES=C \
 161         LC_MONETARY=C \
 162         LC_NUMERIC=C \
 163         LC_TIME=C

 165 # clear environment variables we know to be bad for the build
 166 unset \
 167         LD_OPTIONS \
 168         LD_LIBRARY_PATH \
```



```
 169         LD_AUDIT \
 170         LD_BIND_NOW \
 171         LD_BREADTH \
 172         LD_CONFIG \
 173         LD_DEBUG \
 174         LD_FLAGS \
 175         LD_LIBRARY_PATH_64 \
 176         LD_NOVERSION \
 177         LD_ORIGIN \
 178         LD_LOADFLTR \
 179         LD_NOAUXFLTR \
 180         LD_NOCONFIG \
 181         LD_NODIRCONFIG \
 182         LD_NOOBJALTER \
 183         LD_PRELOAD \
 184         LD_PROFILE \
 185         CONFIG \
 186         GROUP \
 187         OWNER \
 188         REMOTE \
 189         ENV \
 190         ARCH \
 191         CLASSPATH

 193 #
 194 # Setup environment variables
 195 #
 196 if [[ -f /etc/nightly.conf ]]; then
 197         source /etc/nightly.conf
 198 fi

 200 if [[ -f "$1" ]]; then
 201         if [[ "$1" == */* ]]; then
 202                 source "$1"
 203         else
 204                 source "./$1"
 205         fi
 206 else
 207         if [[ -f "/opt/onbld/env/$1" ]]; then
 208                 source "/opt/onbld/env/$1"
 209         else
 210                 printf \
 211                     'Cannot find env file as either %s or /opt/onbld/env/%s\n' \
 212                     "$1" "$1"
 213                 exit 1
 214         fi
 215 fi
 216 shift

 218 # contents of stdenv.sh inserted after next line:
 219 # STDENV_START
 220 # STDENV_END

 222 # Check if we have sufficient data to continue...
 223 [[ -v CODEMGR_WS ]] || fatal_error "Error: Variable CODEMGR_WS not set."
 224 [[ -d "${CODEMGR_WS}" ]] || fatal_error "Error: ${CODEMGR_WS} is not a directory
 225 [[ -f "${CODEMGR_WS}/usr/src/Makefile" ]] || fatal_error "Error: ${CODEMGR_WS}/u

 227 # must match the getopts in nightly.sh
 228 OPTIND=1
 229 NIGHTLY_OPTIONS="-${NIGHTLY_OPTIONS#-}"
 230 while getopts '+0aABCDdFfGIilMmNnopRrtUuWwXxz' FLAG "$NIGHTLY_OPTIONS"
 231 do
 232         case "$FLAG" in
 233           o)    flags.o=true  ;;
 234           +o)   flags.o=false ;;
```

```
235           t)    flags.t=true  ;;
236           +t)   flags.t=false ;;
237           *)    ;;
238         esac
239 done

241 POUND_SIGN="#"
242 # have we set RELEASE_DATE in our env file?
243 if [ -z "$RELEASE_DATE" ]; then
244         RELEASE_DATE=$(LC_ALL=C date +"%B %Y")
245 fi
246 BUILD_DATE=$(LC_ALL=C date +%Y-%b-%d)
247 BASEWSDIR=$(basename -- "${CODEMGR_WS}")
248 DEV_CM="\"@(#)SunOS Internal Development: $LOGNAME $BUILD_DATE [$BASEWSDIR]\""
249 export DEV_CM RELEASE_DATE POUND_SIGN

251 export INTERNAL_RELEASE_BUILD=

251 print 'Build type   is  \c'
252 if ${flags.d} ; then
253         print 'DEBUG'
254         unset RELEASE_BUILD
255         unset EXTRA_OPTIONS
256         unset EXTRA_CFLAGS
257 else
258         # default is a non-DEBUG build
259         print 'non-DEBUG'
260         export RELEASE_BUILD=
261         unset EXTRA_OPTIONS
262         unset EXTRA_CFLAGS
263 fi

265 # update build-type variables
266 PKGARCHIVE="${PKGARCHIVE}${SUFFIX}"

268 #       Set PATH for a build
269 PATH="/opt/onbld/bin:/opt/onbld/bin/${MACH}:/opt/SUNWspro/bin:/usr/ccs/bin:/usr/
270 if [[ "${SUNWSPRO}" != "" ]]; then
271         export PATH="${SUNWSPRO}/bin:$PATH"
272 fi

274 if [[ -n "${MAKE}" ]]; then
275         if [[ -x "${MAKE}" ]]; then
276                 export PATH="$(dirname -- "${MAKE}"):$PATH"
277         else
278                 print "\$MAKE (${MAKE}) is not a valid executible"
279                 exit 1
280         fi
281 fi

283 TOOLS="${SRC}/tools"
284 TOOLS_PROTO="${TOOLS}/proto/root_${MACH}-nd" ; export TOOLS_PROTO

286 if "${flags.t}" ; then
287         export ONBLD_TOOLS="${ONBLD_TOOLS:=${TOOLS_PROTO}/opt/onbld}"

289         export STABS="${TOOLS_PROTO}/opt/onbld/bin/${MACH}/stabs"
290         export CTFSTABS="${TOOLS_PROTO}/opt/onbld/bin/${MACH}/ctfstabs"
291         export GENOFFSETS="${TOOLS_PROTO}/opt/onbld/bin/genoffsets"

293         export CTFCONVERT="${TOOLS_PROTO}/opt/onbld/bin/${MACH}/ctfconvert"
294         export CTFMERGE="${TOOLS_PROTO}/opt/onbld/bin/${MACH}/ctfmerge"

296         export CTFCVTPTBL="${TOOLS_PROTO}/opt/onbld/bin/ctfcvtptbl"
297         export CTFFINDMOD="${TOOLS_PROTO}/opt/onbld/bin/ctffindmod"
```



```
299         PATH="${TOOLS_PROTO}/opt/onbld/bin/${MACH}:${PATH}"
300         PATH="${TOOLS_PROTO}/opt/onbld/bin:${PATH}"
301         export PATH
302 fi

304 export DMAKE_MODE=${DMAKE_MODE:-parallel}

306 if "${flags.o}" ; then
307         export CH=
308 else
309         unset CH
310 fi
311 DEF_STRIPFLAG="-s"

313 TMPDIR="/tmp"

315 # "o_FLAG" is used by "nightly.sh" (it may be useful to rename this
316 # variable using a more descriptive name later)
317 export o_FLAG="$(${flags.o} && print 'y' || print 'n')"

319 export \
320         PATH TMPDIR \
321         POUND_SIGN \
322         DEF_STRIPFLAG \
323         RELEASE_DATE
324 unset \
325         CFLAGS \
326         LD_LIBRARY_PATH

328 # a la ws
329 ENVLDLIBS1=
330 ENVLDLIBS2=
331 ENVLDLIBS3=
332 ENVCPPFLAGS1=
333 ENVCPPFLAGS2=
334 ENVCPPFLAGS3=
335 ENVCPPFLAGS4=
336 PARENT_ROOT=
337 PARENT_TOOLS_ROOT=

339 if [[ "$MULTI_PROTO" != "yes" && "$MULTI_PROTO" != "no" ]]; then
340         printf \
341             'WARNING: invalid value for MULTI_PROTO (%s); setting to "no".\n' \
342             "$MULTI_PROTO"
343         export MULTI_PROTO="no"
344 fi

346 [[ "$MULTI_PROTO" == "yes" ]] && export ROOT="${ROOT}${SUFFIX}"

348 ENVLDLIBS1="-L$ROOT/lib -L$ROOT/usr/lib"
349 ENVCPPFLAGS1="-I$ROOT/usr/include"
350 MAKEFLAGS=e

352 export \
353         ENVLDLIBS1 \
354         ENVLDLIBS2 \
355         ENVLDLIBS3 \
356         ENVCPPFLAGS1 \
357         ENVCPPFLAGS2 \
358         ENVCPPFLAGS3 \
359         ENVCPPFLAGS4 \
360         MAKEFLAGS \
361         PARENT_ROOT \
362         PARENT_TOOLS_ROOT

364 printf 'RELEASE      is %s\n'   "$RELEASE"
```

```
365 printf 'VERSION      is %s\n'   "$VERSION"
366 printf 'RELEASE_DATE is %s\n\n' "$RELEASE_DATE"

368 if [[ -f "$SRC/Makefile" ]] && egrep -s '^setup:' "$SRC/Makefile" ; then
369         print "The top-level 'setup' target is available \c"
370         print "to build headers and tools."
371         print ""

373 elif "${flags.t}" ; then
374         printf \
375             'The tools can be (re)built with the install target in %s.\n\n' \
376             "${TOOLS}"
377 fi

379 #
380 # place ourselves in a new task, respecting BUILD_PROJECT if set.
381 #
382 /usr/bin/newtask -c $$ ${BUILD_PROJECT:+-p$BUILD_PROJECT}

384 if [[ "${flags.c}" == "false" && -x "$SHELL" && \
385     "$(basename -- "${SHELL}")" != "csh" ]]; then
386         # $SHELL is set, and it's not csh.

388         if "${flags.f}" ; then
389                 print 'WARNING: -f is ignored when $SHELL is not csh'
390         fi

392         printf 'Using %s as shell.\n' "$SHELL"
393         exec "$SHELL" ${@:+-c "$@"}

395 elif "${flags.f}" ; then
396         print 'Using csh -f as shell.'
397         exec csh -f ${@:+-c "$@"}

399 else
400         print 'Using csh as shell.'
401         exec csh ${@:+-c "$@"}
402 fi

404 # not reached
```

```
**********************************************************
   17656 Sat Dec 14 21:34:47 2013
new/usr/src/tools/scripts/nightly.1
4265 remove INTERNAL_RELEASE_BUILD
**********************************************************
.\" "
.\" " 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 (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved
.\" "Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
.\" "
.TH nightly 1 "6 July 2010"
.SH NAME
.I nightly
\- build an OS-Net consolidation overnight
.SH SYNOPSIS
\fBnightly [-in] [-V VERS] <env_file>\fP
.LP
.SH DESCRIPTION
.IX "OS-Net build tools" "nightly" "" "\fBnightly\fP"
.LP
.I nightly,
the mother of all build scripts,
can bringover, build, archive, package, error check, and
generally do everything it takes to
turn OS/Net consolidation source code into useful stuff.
It is customizable to permit you to run anything from a
simple build to all of the cross-checking a gatekeeper
needs.  The advantage to using
.I nightly
is that you build things correctly, consistently and
automatically, with the best practices; building with
.I nightly
can mean never having to say you're sorry to your
gatekeeper.
.LP
More
specifically,
.I nightly
performs the following tasks, in order, if
all these things are desired:
.LP
.RS
.TP
\(bu
perform a "make clobber" to clean up old binaries
.TP
\(bu
bringover from the identified parent gate/clone
.TP
\(bu
```



```
perform non-DEBUG and DEBUG builds
.TP
\(bu
list proto area files and compare with previous list
.TP
\(bu
copy updated proto area to parent
.TP
\(bu
list shared lib interface and compare with previous list
.TP
\(bu
perform a "make lint" of the kernel and report errors
.TP
\(bu
perform a "make check" to report hdrchk/cstyle errors
.TP
\(bu
report the presence of any core files
.TP
\(bu
check the ELF runtime attributes of all dynamic objects
.TP
\(bu
check for unreferenced files
.TP
\(bu
report on which proto area objects have changed (since the last build)
.TP
\(bu
report the total build time
.TP
\(bu
save a detailed log file for reference
.TP
\(bu
mail the user a summary of the completed build
.RE
.LP
The actions of the script are almost completely determined by
the environment variables in the
.I env
file, the only necessary argument.  Ths only thing you really
need to use
.I nightly
is an
.I env
file that does what you want.
.LP
Like most of the other build tools in usr/src/tools, this script tends
to change on a fairly regular basis; do not expect to be able to build
OS/Net with a version of nightly significantly older than your source
tree.  It has what is effectively a Consolidation Private relationship
to other build tools and with many parts of the OS/Net makefiles,
although it may also be used to build other consolidations.
.LP
.SH NIGHTLY_OPTIONS
The environment variable NIGHTLY_OPTIONS controls the actions
.I nightly
will take as it proceeds.
The -i, -n, +t and -V options may also be used from the command
line to control the actions without editing your environment file.
The -i and -n options complete the build more quickly by bypassing
some actions. If NIGHTLY_OPTIONS is not set, then "-Bmt" build
options will be used.
```

```
.B Basic action options
.TP 10
.B \-D
Do a build with DEBUG on (non-DEBUG is built by default)
.TP
.B \-F
Do _not_ do a non-DEBUG build (use with -D to get just a DEBUG build)
.TP
.B \-M
Do not run pmodes (safe file permission checker)
.TP
.B \-i
Do an incremental build, suppressing the "make clobber" that by
default removes all existing binaries and derived files.  From the
command line, -i also suppresses the lint pass and the cstyle/hdrchk
pass
.TP
.B \-n
Suppress the bringover so that the build will start immediately with
current source code
.TP
.B \-o
Do an "old style" (pre-S10) build using root privileges to set OWNER
and GROUP from the Makefiles.
.TP
.B \-p
Create packages for regular install
.TP
.B \-U
Update proto area in the parent workspace
.TP
.B \-u
Update the parent workspace with files generated by the build, as follows.
.RS
.TP
\(bu
Copy proto_list_${MACH} and friends to usr/src in the parent.
.TP
\(bu
When used with -f, build a usr/src/unrefmaster.out in
the parent by merging all the usr/src/unref-${MACH}.out files in the
parent.
.TP
\(bu
When used with -A or -r, copy the contents of the resulting
ELF-data.${MACH} directory to usr/src/ELF-data.${MACH} in the parent
workspace.
.RE
.TP
.B \-m
Send mail to $MAILTO at end of build
.TP
.B \-t
Build and use the tools in $SRC/tools (default setting).
.TP
.B \+t
Use the build tools in "$ONBLD_TOOLS/bin".

.LP
.B Code checking options
.TP 10
.B \-A
Check for ABI discrepancies in .so files.
It is only required for shared object developers when there is an
addition, deletion or change of interface in the .so files.
.TP
```



```
.B \-C
Check for cstyle/hdrchk errors
.TP
.B \-f
Check for unreferenced files.  Since the full workspace must be built
in order to accurately identify unreferenced files, -f is ignored for
incremental (-i) builds, or builds that do not include -l, and -p.
.TP
.B \-r
Check the ELF runtime attributes of all dynamic objects
.TP
.B \-l
Do "make lint" in $LINTDIRS (default: $SRC n)
.TP
.B \-N
Do not run protocmp or checkpaths (note: this option is not
recommended, especially in conjunction with the \-p option)
.TP
.B \-W
Do not report warnings (for freeware gate ONLY)
.TP
.B \-w
Report which proto area objects differ between this and the last build.
See wsdiff(1) for details. Note that the proto areas used for comparison
are the last ones constructed as part of the build. As an example, if both
a non-debug and debug build are performed (in that order), then the debug
proto area will be used for comparison (which might not be what you want).
.LP
.B Groups of options
.TP 10
.B \-G
Gate keeper default group of options (-u)
.TP
.B \-I
Integration engineer default group of options (-mpu)
.TP
.B \-R
Default group of options for building a release (-mp)

.LP
.B Miscellaneous options
.TP 10
.B \-V VERS
set the build version string to VERS, overriding VERSION
.TP
.B \-X
Copies the proto area and packages from the IHV and IHV-bin gates into the
nightly proto and package areas.  This is only available on i386.  See
.B REALMODE ENVIRONMENT VARIABLES
and
.B BUILDING THE IHV WORKSPACE
below.

.LP
.SH ENVIRONMENT VARIABLES
.LP
Here is a list of prominent environment variables that
.I nightly
references and the meaning of each variable.
.LP
.RE
.B CODEMGR_WS
.RS 5
The root of your workspace, including whatever metadata is kept by
the source code management system.  This is the workspace in which the
build will be done.
```

```
.RE
.LP
.B PARENT_WS
.RS 5
The root of the workspace that is the parent of the
one being built.  This is particularly relevant for configurations
with a main
workspace and build workspaces underneath it; see the
\-u and \-U
options as well as the PKGARCHIVE environment variable, for more
information.
.RE
.LP
.B BRINGOVER_WS
.RS 5
This is the workspace from which
.I nightly
will fetch sources to either populate or update your workspace;
it defaults to $CLONE_WS.
.RE
.LP
.B CLOSED_BRINGOVER_WS
.RS 5
A full Mercurial workspace has two repositories: one for open source
and one for closed source.  If this variable is non-null,
.I nightly
will pull from the repository that it names to get the closed source.
It defaults to $CLOSED_CLONE_WS.
.LP
If $CODEMGR_WS already exists and contains only the open repository,
.I nightly
will ignore this variable; you'll need to pull the closed repository
by hand if you want it.
.RE
.LP
.B CLONE_WS
.RS 5
This is the workspace from which
.I nightly
will fetch sources by default.  This is
often distinct from the parent, particularly if the parent is a gate.
.RE
.LP
.B CLOSED_CLONE_WS
.RS 5
This is the default closed-source Mercurial repository that
.I nightly
might pull from (see
.B CLOSED_BRINGOVER_WS
for details).
.RE
.LP
.B SRC
.RS 5
Root of OS-Net source code, referenced by the Makefiles.  It is
the starting point of build activity.  It should be expressed
in terms of $CODEMGR_WS.
.RE
.LP
.B ROOT
.RS 5
Root of the proto area for the build.  The makefiles direct
installation of build products to this area and
direct references to these files by builds of commands and other
targets.  It should be expressed in terms of $CODEMGR_WS.
.LP
```



```
If $MULTI_PROTO is "no", $ROOT may contain a DEBUG or non-DEBUG
build.  If $MULTI_PROTO is "yes", $ROOT contains the DEBUG build and
$ROOT-nd contains the non-DEBUG build.
.RE
.LP
.B TOOLS_ROOT
.RS 5
Root of the tools proto area for the build.  The makefiles direct
installation of tools build products to this area.  Unless \fB+t\fR
is part of $NIGHTLY_OPTIONS, these tools will be used during the
build.
.LP
As built by nightly, this will always contain non-DEBUG objects.
Therefore, this will always have a -nd suffix, regardless of
$MULTI_PROTO.
.RE
.LP
.B MACH
.RS 5
The instruction set architecture of the build machine as given
by \fIuname -p\fP, e.g. sparc, i386.
.RE
.LP
.B LOCKNAME
.RS 5
The name of the file used to lock out multiple runs of
.IR nightly .
This should generally be left to the default setting.
.RE
.LP
.B ATLOG
.RS 5
The location of the log directory maintained by
.IR nightly .
This should generally be left to the default setting.
.RE
.LP
.B LOGFILE
.RS 5
The name of the log file in the $ATLOG directory maintained by
.IR nightly .
This should generally be left to the default setting.
.RE
.LP
.B STAFFER
.RS 5
The non-root account to use on the build machine for the
bringover from the clone or parent workspace.
This may not be the same identify used by the SCM.
.RE
.LP
.B MAILTO
.RS 5
The address to be used to send completion e-mail at the end of
the build (for the \-m option).
.RE
.LP
.B MAILFROM
.RS 5
The address to be used for From: in the completion e-mail at the
end of the build (for the \-m option).
.RE
.LP
.B REF_PROTO_LIST
.RS 5
Name of file used with protocmp to compare proto area contents.
```

```
.RE
.LP
.B PARENT_ROOT
.RS 5
The parent root, which is the destination for copying the proto
area(s) when using the \-U option.
.RE
.LP
.B PARENT_TOOLS_ROOT
.RS 5
The parent tools root, which is the destination for copying the tools
proto area when using the \-U option.
.RE
.LP
.B RELEASE
.RS 5
The release version number to be used; e.g., 5.10.1 (Note: this is set
in Makefile.master and should not normally be overridden).
.RE
.LP
.B VERSION
.RS 5
The version text string to be used; e.g., "onnv:‘date ’+%Y-%m-%d’’".
.RE
.LP
.B RELEASE_DATE
.RS 5
The release date text to be used; e.g., October 2009. If not set in
your environment file, then this text defaults to the output from
$(LC_ALL=C date +"%B %Y"); e.g., "October 2009".
.RE
.LP
.B INTERNAL_RELEASE_BUILD
.RS 5
See Makefile.master - but it mostly controls id strings. Generally,
let
.I nightly
set this for you.
.RE
.LP
.B RELEASE_BUILD
.RS 5
Define this to build a release with a non-DEBUG kernel.
Generally, let
.I nightly
set this for you based on its options.
.RE
.LP
.B PKGARCHIVE
.RS 5
The destination for packages.  This may be relative to
$CODEMGR_WS for private packages or relative to $PARENT_WS
if you have different workspaces for different architectures
but want one hierarchy of packages.
.RE
.LP
.B MAKEFLAGS
.RS 5
Set default flags to make; e.g., -k to build all targets regardless of errors.
.RE
.LP
.B UT_NO_USAGE_TRACKING
.RS 5
Disables usage reporting by listed Devpro tools. Otherwise it sends mail
to some Devpro machine every time the tools are used.
.RE
```



```
.LP
.B LINTDIRS
.RS 5
Directories to lint with the \-l option.
.RE
.LP
.B BUILD_TOOLS
.RS 5
BUILD_TOOLS is the root of all tools including the compilers; e.g.,
/ws/onnv-tools.  It is used by the makefile system, but not nightly.
.RE
.LP
.B ONBLD_TOOLS
.RS 5
ONBLD_TOOLS is the root of all the tools that are part of SUNWonbld; e.g.,
/ws/onnv-tools/onbld.  By default, it is derived from
.BR BUILD_TOOLS .
It is used by the makefile system, but not nightly.
.RE
.LP
.B SPRO_ROOT
.RS 5
The gate-defined default location for the Sun compilers, e.g.
/ws/onnv-tools/SUNWspro.  By default, it is derived from
.BR BUILD_TOOLS .
It is used by the makefile system, but not nightly.
.RE
.LP
.B JAVA_ROOT
.RS 5
The location for the java compilers for the build, generally /usr/java.
.RE
.LP
.B OPTHOME
.RS 5
The gate-defined default location of things formerly in /opt; e.g.,
/ws/onnv-tools.  This is used by nightly, but not the makefiles.
.RE
.LP
.B TEAMWARE
.RS 5
The gate-defined default location for the Teamware tools; e.g.,
/ws/onnv-tools/SUNWspro.  By default, it is derived from
.BR OPTHOME .
This is used by nightly, but not the makefiles.  There is no
corresponding variable for Mercurial or Subversion, which are assumed
to be installed in the default path.
.RE
.LP
.B ON_CLOSED_BINS
.RS 5
OpenSolaris builds do not contain the closed source tree.  Instead,
the developer downloads a closed binaries tree and unpacks it.
.B ON_CLOSED_BINS
tells nightly
where to find these closed binaries, so that it can add them into the
build.
.LP
.RE
.B ON_CRYPTO_BINS
.RS 5
This is the path to a compressed tarball that contains debug
cryptographic binaries that have been signed to allow execution
outside of Sun, e.g., $PARENT_WS/packages/$MACH/on-crypto.$MACH.bz2.
.I nightly
will automatically adjust the path for non-debug builds.  This tarball
```

```
516 is needed if the closed-source tree is not present.  Also, it is
517 usually needed when generating OpenSolaris deliverables from a project
518 workspace.  This is because most projects do not have access to the
519 necessary key and certificate that would let them sign their own
520 cryptographic binaries.
521 .LP
522 .RE
523 .B CHECK_PATHS
524 .RS 5
525 Normally, nightly runs the 'checkpaths' script to check for
526 discrepancies among the files that list paths to other files, such as
527 exception lists and req.flg.  Set this flag to 'n' to disable this
528 check, which appears in the nightly output as "Check lists of files."
529 .RE
530 .LP
531 .B CHECK_DMAKE
532 .RS 5
533 Nightly validates that the version of dmake encountered is known to be
534 safe to use.  Set this flag to 'n' to disable this test, allowing any
535 version of dmake to be used.
536 .RE
537 .LP
538 .B MULTI_PROTO
539 .RS 5
540 If "no" (the default),
541 .I nightly
542 will reuse $ROOT for both the DEBUG and non-DEBUG builds.  If "yes",
543 the DEBUG build will go in $ROOT and the non-DEBUG build will go in
544 $ROOT-nd.  Other values will be treated as "no".
545 .RE
546 .LP
547 .SH NIGHTLY HOOK ENVIRONMENT VARIABLES
548 .LP
549 Several optional environment variables may specify commands to run at
550 various points during the build.  Commands specified in the hook
551 variable will be run in a subshell; command output will be appended to
552 the mail message and log file.  If the hook exits with a non-zero
553 status, the build is aborted immediately.  Environment variables
554 defined in the environment file will be available.
555 .LP
556 .B SYS_PRE_NIGHTLY
557 .RS 5
558 Run just after the workspace lock is acquired.  This is reserved for
559 per-build-machine customizations and should be set only in /etc/nightly.conf
560 .RE
561 .LP
562 .B PRE_NIGHTLY
563 .RS 5
564 Run just after SYS_PRE_NIGHTLY.
565 .RE
566 .LP
567 .B PRE_BRINGOVER
568 .RS 5
569 Run just before bringover is started; not run if no bringover is done.
570 .RE
571 .LP
572 .B POST_BRINGOVER
573 .RS 5
574 Run just after bringover completes; not run if no bringover is done.
575 .RE
576 .LP
577 .B POST_NIGHTLY
578 .RS 5
579 Run after the build completes, with the return status of nightly - one
580 of "Completed", "Interrupted", or "Failed" - available in the
581 environment variable NIGHTLY_STATUS.
```



```
582 .RE
583 .LP
584 .B SYS_POST_NIGHTLY
585 .RS 5
586 This is reserved for per-build-machine customizations, and runs
587 immedately after POST_NIGHTLY.
588 .RE
589 .LP
590 .SH REALMODE ENVIRONMENT VARIABLES
591 .LP
592 The following environment variables referenced by
593 .I nightly
594 are only required when the -X option is used.
595 .LP
596 .RE
597 .B IA32_IHV_WS
598 .RS 5
599 Reference to the IHV workspace containing IHV driver binaries.
600 The IHV workspace must be fully built before starting the ON realmode build.
601 .LP
602 .RE
603 .B IA32_IHV_ROOT
604 .RS 5
605 Reference to the IHV workspace proto area.
606 The IHV workspace must be fully built before starting the ON realmode build.
607 .LP
608 .RE
609 .B IA32_IHV_PKGS
610 .RS 5
611 Reference to the IHV workspace packages.  If this is empty or the directory
612 is non-existent, then nightly will skip copying the packages.
613 .LP
614 .RE
615 .B IA32_IHV_BINARY_PKGS
616 .RS 5
617 Reference to binary-only IHV packages.  If this is empty or the directory
618 is non-existent, then nightly will skip copying the packages.
619 .LP
620 .RE
621 .B SPARC_RM_PKGARCHIVE
622 .RS 5
623 Destination for sparc realmode package SUNWrmodu.
624 Yes, this sparc package really is built on x86.
625 .SH FILES
626 .LP
627 .RS 5
628 /etc/nightly.conf
629 .RE
630 .LP
631 If present, nightly executes this file just prior to executing the
632 .I env
633 file.
634 .SH BUILDING THE IHV WORKSPACE
635 .LP
636 The IHV workspace can be built with
637 .I nightly.
638 The recommended options are:
639 .LP
640 .RS 5
641 NIGHTLY_OPTIONS="-pmWN"
642 .RE
643 .LP
644 None of the realmode environment variables needed for ON realmode builds
645 are required to build the IHV workspace.
646 .SH EXAMPLES
647 .LP
```

```
 648 Start with the example file in usr/src/tools/env/developer.sh
 649 (or gatekeeper.sh), copy to myenv and make your changes.
 650 .LP
 651 .PD 0
 652 # grep NIGHTLY_OPTIONS myenv
 653 .LP
 654 NIGHTLY_OPTIONS="-ACrlapDm"
 655 .LP
 656 export NIGHTLY_OPTIONS
 657 .LP
 658 # /opt/onbld/bin/nightly -i myenv
 659 .PD
 660 .LP
 661 .SH SEE ALSO
 662 .BR bldenv (1)
```

```
**********************************************************
   70526 Sat Dec 14 21:34:47 2013
new/usr/src/tools/scripts/nightly.sh
4265 remove INTERNAL_RELEASE_BUILD
**********************************************************
______unchanged_portion_omitted_

 756 #
 757 # Functions for setting build flags (DEBUG/non-DEBUG).  Keep them
 758 # together.
 759 #

 761 function set_non_debug_build_flags {
 762         export INTERNAL_RELEASE_BUILD ; INTERNAL_RELEASE_BUILD=
 762         export RELEASE_BUILD ; RELEASE_BUILD=
 763         unset EXTRA_OPTIONS
 764         unset EXTRA_CFLAGS
 765 }

 767 function set_debug_build_flags {
 769         export INTERNAL_RELEASE_BUILD ; INTERNAL_RELEASE_BUILD=
 768         unset RELEASE_BUILD
 769         unset EXTRA_OPTIONS
 770         unset EXTRA_CFLAGS
 771 }
_____unchanged_portion_omitted_
```

```
**********************************************************
   13794 Sat Dec 14 21:34:47 2013
new/usr/src/uts/Makefile.targ
4265 remove INTERNAL_RELEASE_BUILD
**********************************************************
```

```
#
# 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 (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
#
#       This Makefiles contains the common targets and definitions for
#       all kernels. It is to be included in the Makefiles for specific
#       implementation architectures and processor architecture dependent
#       modules: i.e.: all driving kernel Makefiles.
#


#
#       Default rule for building the lint library directory:
#
$(LINT_LIB_DIR):
        -@mkdir -p $@ 2> /dev/null

#
#       All C objects depend on inline files. However, cc(1) doesn't generate
#       the correct dependency info. Also, these Makefiles don't contain a
#       separate list of C-derived object files (but it is light weight to
#       let the assembler files think they depend upon this when they don't).
#       Fortunately, the inline files won't change very often. So, for now,
#       all objects depend on the inline files. Remove this when the inliner
#       is fixed to drop correct dependency information.
#
$(OBJECTS): $(INLINES)

#
#       Partially link .o files to generate the kmod. The fake dependency
#       on modstubs simplifies things...
#       ELFSIGN_MOD is defined in the individual KCF plug-in modules Makefiles,
#       and will sign the ELF objects using elfsign(1).
#
$(BINARY):              $(OBJECTS)
        $(LD) -r $(LDFLAGS) -o $@ $(OBJECTS)
        $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
        $(POST_PROCESS)
        $(ELFSIGN_MOD)

#
#       This target checks each kmod for undefined entry points. It does not
#       modify the kmod in any way.
```



```
#
$(MODULE).check:        FRC
        @BUILD_TYPE=DBG32 $(MAKE) $(MODULE).check.targ

$(MODULE).check.targ:   $(BINARY) $(OBJECTS) $(EXTRA_CHECK_OBJS) $(UNIX_O) $(MOD
        $(LD) -o /dev/null $(OBJECTS) $(EXTRA_CHECK_OBJS) $(UNIX_O) $(MODSTUBS_O

#
#       Module lint library construction targets.
#
MOD_LINT_LIB    = $(LINT_LIB_DIR)/llib-l$(LINT_MODULE).ln

$(MOD_LINT_LIB):        $(LINT_LIB_DIR) $(LINTS)
        @-$(ECHO) "\n$(OBJS_DIR)/$(MODULE): (library construction):"
        @($(LINT) -o $(LINT_MODULE)-$(OBJS_DIR) \
                $(LINTFLAGS) $(LINTS) $(LTAIL))
        @$(MV) llib-l$(LINT_MODULE)-$(OBJS_DIR).ln $@

$(LINT_MODULE).lint:    $(MOD_LINT_LIB) $(LINT_LIB) $(GEN_LINT_LIB)
        @-$(ECHO) "\n$(OBJS_DIR)/$(LINT_MODULE): global crosschecks:"
        @($(LINT) $(LINTFLAGS) $(MOD_LINT_LIB) \
                $(LINT_LIB) $(GEN_LINT_LIB) $(LTAIL))

#
# Since assym.h is a derived file, the dependency must be explicit for
# all files including this file. (This is only actually required in the
# instance when the .nse_depinfo file does not exist.) It may seem that
# the lint targets should also have a similar dependency, but they don't
# since only C headers are included when #defined(lint) is true. The
# actual lists are defined in */Makefile.files.
#
$(ASSYM_DEPS:%=$(OBJS_DIR)/%):  $(DSF_DIR)/$(OBJS_DIR)/assym.h

#
#       Everybody need to know how to create a modstubs.o built with the
#       appropriate flags and located in the appropriate location.
#
$(MODSTUBS_O):  $(MODSTUBS)
        $(COMPILE.s) -o $@ $(MODSTUBS)

$(LINTS_DIR)/modstubs.ln:       $(MODSTUBS)
        @($(LHEAD) $(LINT.s) $(MODSTUBS) $(LTAIL))

#
# Build the source file which contains the kernel's utsname,
# with release, version and machine set as follows:
#
#       release: contents of $(RELEASE) (Spaces replaced by '_')
#       version: contents of $(PATCHID) (Spaces replaced by '_')
#       machine: contents of $(UNAME_M)
#
# Build environment information is only contained in the comment section.
#
# The version string, normally the variable VERSION, is set to display
# environmental information temporarily while in development because
# it provides a little more useful information.
#
VERSION_STRING  = ($(ECHO) $$LOGNAME [\`basename $$CODEMGR_WS\`] \\\c; date +%D)
$(INTERNAL_RELEASE_BUILD)VERSION_STRING = $(ECHO) $(PATCHID)

$(OBJS_DIR)/vers.o: $(OBJECTS)
        $(COMPILE.c) -DUTS_RELEASE=\"'$(ECHO) $(RELEASE) | sed -e 's/ /_/g''\" \
            -DUTS_VERSION=\"'$(ECHO) $(PATCHID) | sed -e 's/ /_/g''\" \
            -DUTS_VERSION=\"'$(VERSION_STRING) | sed -e 's/ /_/g''\" \
            -DUTS_PLATFORM=\"$(UNAME_M)\" -o $@ $(SRC)/uts/common/os/vers.c
        $(CTFCONVERT_O)
```

```
121         $(POST_PROCESS_O)

123 $(LINTS_DIR)/vers.ln: $(SRC)/uts/common/os/vers.c
124         @($(LHEAD) $(LINT.c) -DUTS_RELEASE=\"\" -DUTS_VERSION=\"\" \
125             -DUTS_PLATFORM=\"\" $(SRC)/uts/common/os/vers.c $(LTAIL))

127 #
128 #       Installation targets and rules:
129 #
130 $(ROOT_MOD_DIR) $(USR_MOD_DIR):
131         -$(INS.dir)

133 $(ROOT_MOD_DIRS_32):    $(ROOT_MOD_DIR)
134         -$(INS.dir)

136 $(USR_MOD_DIRS_32):     $(USR_MOD_DIR)
137         -$(INS.dir)

139 $(ROOT_MOD_DIR)/%:      $(OBJS_DIR)/% $(ROOT_MOD_DIR) FRC
140         $(INS.file)

142 $(ROOT_CPU_DIR)/%:      $(OBJS_DIR)/% $(ROOT_CPU_DIR) FRC
143         $(INS.file)

145 $(ROOT_DRV_DIR)/%:      $(OBJS_DIR)/% $(ROOT_DRV_DIR) FRC
146         $(INS.file)

148 $(ROOT_DTRACE_DIR)/%:   $(OBJS_DIR)/% $(ROOT_DTRACE_DIR) FRC
149         $(INS.file)

151 $(ROOT_EXEC_DIR)/%:     $(OBJS_DIR)/% $(ROOT_EXEC_DIR) FRC
152         $(INS.file)

154 $(ROOT_FS_DIR)/%:       $(OBJS_DIR)/% $(ROOT_FS_DIR) FRC
155         $(INS.file)

157 $(ROOT_SCHED_DIR)/%:    $(OBJS_DIR)/% $(ROOT_SCHED_DIR) FRC
158         $(INS.file)

160 $(ROOT_SOCK_DIR)/%:     $(OBJS_DIR)/% $(ROOT_SOCK_DIR) FRC
161         $(INS.file)

163 $(ROOT_STRMOD_DIR)/%:   $(OBJS_DIR)/% $(ROOT_STRMOD_DIR) FRC
164         $(INS.file)

166 $(ROOT_IPP_DIR)/%:      $(OBJS_DIR)/% $(ROOT_IPP_DIR) FRC
167         $(INS.file)

169 $(ROOT_SYS_DIR)/%:      $(OBJS_DIR)/% $(ROOT_SYS_DIR) FRC
170         $(INS.file)

172 $(ROOT_MISC_DIR)/%:     $(OBJS_DIR)/% $(ROOT_MISC_DIR) FRC
173         $(INS.file)

175 $(ROOT_DACF_DIR)/%:     $(OBJS_DIR)/% $(ROOT_DACF_DIR) FRC
176         $(INS.file)

178 $(ROOT_BRAND_DIR)/%:    $(OBJS_DIR)/% $(ROOT_BRAND_DIR) FRC
179         $(INS.file)

181 $(ROOT_CRYPTO_DIR)/%:   $(OBJS_DIR)/% $(ROOT_CRYPTO_DIR) FRC
182         $(INS.file)

184 $(ROOT_KGSS_DIR)/%:     $(OBJS_DIR)/% $(ROOT_KGSS_DIR) FRC
185         $(INS.file)
```



```
187 $(ROOT_SCSI_VHCI_DIR)/%: $(OBJS_DIR)/% $(ROOT_SCSI_VHCI_DIR) FRC
188         $(INS.file)

190 $(ROOT_PMCS_FW_DIR)/%:  $(OBJS_DIR)/% $(ROOT_PMCS_FW_DIR) FRC
191         $(INS.file)

193 $(ROOT_QLC_FW_DIR)/%:   $(OBJS_DIR)/% $(ROOT_QLC_FW_DIR) FRC
194         $(INS.file)

196 $(ROOT_EMLXS_FW_DIR)/%: $(OBJS_DIR)/% $(ROOT_EMLXS_FW_DIR) FRC
197         $(INS.file)

199 $(ROOT_MACH_DIR)/%:     $(OBJS_DIR)/% $(ROOT_MACH_DIR) FRC
200         $(INS.file)

202 $(ROOT_FONT_DIR)/%:     $(OBJS_DIR)/% $(ROOT_MOD_DIR) $(ROOT_FONT_DIR) FRC
203         $(INS.file)

205 $(ROOT_MAC_DIR)/%:      $(OBJS_DIR)/% $(ROOT_MOD_DIR) $(ROOT_MAC_DIR) FRC
206         $(INS.file)

208 $(USR_DRV_DIR)/%:       $(OBJS_DIR)/% $(USR_DRV_DIR) FRC
209         $(INS.file)

211 $(USR_EXEC_DIR)/%:      $(OBJS_DIR)/% $(USR_EXEC_DIR) FRC
212         $(INS.file)

214 $(USR_FS_DIR)/%:        $(OBJS_DIR)/% $(USR_FS_DIR) FRC
215         $(INS.file)

217 $(USR_SCHED_DIR)/%:     $(OBJS_DIR)/% $(USR_SCHED_DIR) FRC
218         $(INS.file)

220 $(USR_SOCK_DIR)/%:      $(OBJS_DIR)/% $(USR_SOCK_DIR) FRC
221         $(INS.file)

223 $(USR_STRMOD_DIR)/%:    $(OBJS_DIR)/% $(USR_STRMOD_DIR) FRC
224         $(INS.file)

226 $(USR_SYS_DIR)/%:       $(OBJS_DIR)/% $(USR_SYS_DIR) FRC
227         $(INS.file)

229 $(USR_MISC_DIR)/%:      $(OBJS_DIR)/% $(USR_MISC_DIR) FRC
230         $(INS.file)

232 $(USR_DACF_DIR)/%:      $(OBJS_DIR)/% $(USR_DACF_DIR) FRC
233         $(INS.file)

235 $(USR_PCBE_DIR)/%:      $(OBJS_DIR)/% $(USR_PCBE_DIR) FRC
236         $(INS.file)

238 $(USR_DTRACE_DIR)/%:    $(OBJS_DIR)/% $(USR_DTRACE_DIR) FRC
239         $(INS.file)

241 $(USR_BRAND_DIR)/%:     $(OBJS_DIR)/% $(USR_BRAND_DIR) FRC
242         $(INS.file)

244 $(ROOT_KICONV_DIR)/%:   $(OBJS_DIR)/% $(ROOT_KICONV_DIR) FRC
245         $(INS.file)

247 include $(SRC)/Makefile.psm.targ

249 #
250 #       Target for 64b modules
251 #
252 $(ROOT_KERN_DIR_64):
```

```
253         -$(INS.dir)

255 $(ROOT_KERN_DIR_64)/%:  $(OBJS_DIR)/% $(ROOT_KERN_DIR_64) FRC
256         $(INS.file)

258 %/$(SUBDIR64):          %
259         -$(INS.dir)

261 #
262 #       Targets for '.conf' file installation.
263 #
264 $(ROOT_CONFFILE):       $(SRC_CONFFILE) $(ROOT_CONFFILE:%/$(CONFFILE)=%)
265         $(INS.conffile)

267 #
268 #       Targets for creating links between common platforms. ROOT_PLAT_LINKS
269 #       are are the /platform level while ROOT_PLAT_LINKS_2 are one level
270 #       down (/platform/`uname -i`/{lib|sbin|kernel}.
271 #
272 $(ROOT_PLAT_LINKS):
273         $(INS.slink1)

275 $(ROOT_PLAT_LINKS_2):
276         $(INS.slink2)

278 $(USR_PLAT_LINKS):
279         $(INS.slink1)

281 $(USR_PLAT_LINKS_2):
282         $(INS.slink2)

284 #
285 # multiple builds support
286 #
287 def $(DEF_DEPS)                 := TARGET = def
288 all $(ALL_DEPS)                 := TARGET = all
289 clean $(CLEAN_DEPS)             := TARGET = clean
290 clobber $(CLOBBER_DEPS)         := TARGET = clobber
291 lint $(LINT_DEPS)               := TARGET = lint
292 modlintlib $(MODLINTLIB_DEPS)   := TARGET = modlintlib
293 modlist $(MODLIST_DEPS)         := TARGET = modlist
294 modlist $(MODLIST_DEPS)         := NO_STATE= -K $$MODSTATE$$$$
295 clean.lint $(CLEAN_LINT_DEPS)   := TARGET = clean.lint
296 install $(INSTALL_DEPS)         := TARGET = install
297 symcheck $(SYM_DEPS)            := TARGET = symcheck

299 ALL_TARGS       = def all clean clobber lint modlintlib \
300                   clean.lint lintlib install symcheck

302 ALL_OBJ32       = $(ALL_TARGS:%=%.obj32)

304 $(ALL_OBJ32):   FRC
305         @BUILD_TYPE=OBJ32 VERSION='$(VERSION)' $(MAKE) $(NO_STATE) $(TARGET).tar

307 ALL_DEBUG32     = $(ALL_TARGS:%=%.debug32)

309 $(ALL_DEBUG32): FRC
310         @BUILD_TYPE=DBG32 VERSION='$(VERSION)' $(MAKE) $(NO_STATE) $(TARGET).tar

312 ALL_OBJ64       = $(ALL_TARGS:%=%.obj64)

314 $(ALL_OBJ64):   FRC
315         @BUILD_TYPE=OBJ64 VERSION='$(VERSION)' $(MAKE) $(NO_STATE) $(TARGET).tar

317 ALL_DEBUG64     = $(ALL_TARGS:%=%.debug64)
```



```
319 $(ALL_DEBUG64): FRC
320         @BUILD_TYPE=DBG64 VERSION='$(VERSION)' $(MAKE) $(NO_STATE) $(TARGET).tar

322 #
323 #       Currently only the IP module needs symbol checking on obj64.
324 #       Other modules have the same global-objs nm output for debug64 and obj64.
325 #
326 $(SISCHECK_DEPS):       $(DEF_DEPS)
327         @TARG='$(ECHO) $@ | $(CUT) -d'.' -f2'; \
328         MODSYMS=$(MODULE).symbols.$$TARG; \
329         if [ -f "$(MODULE).global-objs.$$TARG" ]; then \
330                 $(GREP) -v '#' $(MODULE).global-objs.$$TARG |$(GREP) . | \
331                     $(SORT) -u > $$MODSYMS.tmp; \
332                 $(NM) $$TARG/$(MODULE) |$(GREP) OBJT |$(GREP) -v UNDEF | \
333                     $(CUT) -d'|' -f8 |$(GREP) -v '^___const_' | \
334                     $(GREP) -v '\.[0-9]*$$' |$(SORT) -u \
335                     > $$MODSYMS.tmp.new; \
336                 $(DIFF) $$MODSYMS.tmp $$MODSYMS.tmp.new > $$MODSYMS.diff || \
337                     ($(ECHO) "warning: $(MODULE) symbol checking:" \
338                     "global variable(s) introduced and/or removed."; \
339                     $(CAT) $$MODSYMS.diff; exit 1) \
340         fi

342 $(SISCLEAN_DEPS):
343         -TARG='$(ECHO) $@ | $(CUT) -d'.' -f2'; \
344         MODSYMS=$(MODULE).symbols.$$TARG; \
345         $(RM) $$MODSYMS.tmp $$MODSYMS.tmp.new $$MODSYMS.diff Nothing_to_remove


348 $(OBJS_DIR):
349         -@mkdir -p $@ 2> /dev/null

351 def.targ:              $(OBJS_DIR) $(ALL_TARGET)

353 all.targ:              $(OBJS_DIR) $(ALL_TARGET)

355 lint.targ:             $(OBJS_DIR) $(LINT_TARGET)

357 modlintlib.targ:       $(OBJS_DIR) $(MOD_LINT_LIB)

359 install.targ:          $(OBJS_DIR) $(INSTALL_TARGET)

361 #
362 # Support for Install.sh.
363 #

365 modlist:        $(MODLIST_DEPS)

367 # paths relative to $(ROOT).
368 RELMODULE = $(ROOTMODULE:$(ROOT)/%=%)
369 RELCONF = $(ROOT_CONFFILE:$(ROOT)/%=%)
370 RELLINK = $(ROOTLINK:$(ROOT)/%=%)
371 RELUNIX = $(UNIX32_LINK:$(ROOT)/%=%)
372 RELSOFTLINKS = $(ROOTSOFTLINKS:$(ROOT)/%=%)

374 MODSRC:sh=              pwd

376 #
377 # Generate module information for Install.sh, i.e., specify what files
378 # Install.sh should include.  Each line looks like
379 #   <tag> <srcdir> <arg1> <arg2> ...
380 # where <tag> specifies the type of file, <srcdir> gives the source
381 # path (useful if there is an error), and <argN> is one or more
382 # additional bits of information that Install.sh needs (e.g., source
383 # directory, install directory, filtering tags).  See Install.sh for
384 # details on the arguments for each tag type, especially the functions
```

```
385 # copymod, filtmod, and filtimpl.
386 #
387 # Changes to this target may require corresponding changes to
388 # Install.sh.
389 #
390 # Don't issue a MOD entry if it's not in the install list.
391 #

393 $(MODLIST_DEPS): FRC
394         @case $@ in \
395         *32) \
396                 class=32; \
397                 [ -n "$(RELMODULE)" ] && relmodule=`dirname $(RELMODULE)`;; \
398         *64) \
399                 class=64; \
400                 [ -n "$(RELMODULE)" ] && \
401                     relmodule=`dirname $(RELMODULE)`/$(SUBDIR64);; \
402         esac; \
403         if [ -z "$(THISIMPL)" ]; then \
404                 impl=all; \
405         else \
406                 impl=$(THISIMPL); \
407         fi; \
408         if [ -n "$(ROOTMODULE)" -a -n "$(INSTALL_TARGET)" ]; then \
409                 if [ -z "$(MODULE)" ]; then \
410                         module=`basename $(ROOTMODULE)`; \
411                 else \
412                         module=$(MODULE); \
413                 fi; \
414                 tinstall="$(INSTALL_TARGET)"; \
415                 for t in $$tinstall; do \
416                         if [ "$(ROOTMODULE)" = $$t ]; then \
417                                 echo MOD $(MODSRC) $$module $$relmodule \
418                                     $$class $$impl; \
419                                 break; \
420                         fi \
421                 done \
422         fi; \
423         if [ -n "$(CONF_SRCDIR)" ]; then \
424                 tinstall="$(INSTALL_TARGET)"; \
425                 for t in $$tinstall; do \
426                         if [ $(ROOT_CONFFILE) = $$t ]; then \
427                                 echo CONF $(MODSRC) $(RELCONF) \
428                                     $(MODSRC)/$(CONF_SRCDIR) $$impl $$module; \
429                                 break; \
430                         fi \
431                 done \
432         fi; \
433         if [ -n "$(ROOTLINK)" ]; then \
434                 rellinks="$(RELLINK)"; \
435                 for r in $$rellinks; do \
436                         if [ $$class = 32 ]; then \
437                                 linkdir=`dirname $$r`; \
438                         else \
439                                 linkdir=`dirname $$r`/$(SUBDIR64); \
440                         fi; \
441                         echo LINK $(MODSRC) $$relmodule $$module \
442                                 $$linkdir `basename $$r` $$impl; \
443                 done \
444         fi; \
445         if [ -n "$(UNIX32_LINK)" ]; then \
446                 echo SYMLINK $(MODSRC) $(SUBDIR64)/$(UNIX) \
447                     `dirname $(RELUNIX)` unix $$impl $$module; \
448         fi; \
449         trelsoftlinks="$(RELSOFTLINKS)"; \
450         for t in $$trelsoftlinks; do \
```



```
451                 if [ $$class = 32 ]; then \
452                         linkdir=`dirname $$t`; \
453                 else \
454                         linkdir=`dirname $$t`/$(SUBDIR64); \
455                 fi; \
456                 linkname=`basename $$t`; \
457                 echo SYMLINK $(MODSRC) $(MODULE) $$linkdir $$linkname \
458                     $$impl $$module; \
459         done

461 #
462 #       Cleanliness is next to ...
463 #
464 clean.targ:
465         -$(RM) $(CLEANFILES) Nothing_to_remove

467 clobber.targ:
468         -$(RM) $(CLOBBERFILES) Nothing_to_remove

470 clean.lint.targ:
471         -$(RM) $(CLEANLINTFILES) Nothing_to_remove

473 #
474 #       Create fake lintlibs in the 64b dirs so
475 #       global linting works
476 #
477 lint64:
478         @$(ECHO) $(MODULE) fake lints
479         @for dir in $(LINT64_DIRS); do \
480                 if [ ! -d $$dir ]; then mkdir $$dir; fi \
481         done
482         @for file in $(LINT64_FILES); do \
483                 if [ ! -f $$file ]; then touch $$file; fi \
484         done

486 #
487 #       In some places we also need to create fake lintlibs for 32b
488 #       dirs so global linting works
489 #
490 lint32:
491         @$(ECHO) $(MODULE) fake lints
492         @for dir in $(LINT32_DIRS); do \
493                 if [ ! -d $$dir ]; then mkdir $$dir; fi \
494         done
495         @for file in $(LINT32_FILES); do \
496                 if [ ! -f $$file ]; then touch $$file; fi \
497         done

499 FRC:
```