```
**********************************************************
   34599 Wed Jan 22 16:20:23 2014
new/usr/src/Makefile.master
4506 GCC should be the primary compiler
**********************************************************
```

```
#
# 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.
#
# __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=
RELEASE_BUILD=                          $(POUND_SIGN)
$(RELEASE_BUILD)NOT_RELEASE_BUILD=      $(POUND_SIGN)
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=)

# 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.
#

__SUNC=                 $(POUND_SIGN)
$(__SUNC)__GNUC=        $(POUND_SIGN)
__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)

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

RPCGEN=         /usr/bin/rpcgen
STABS=          $(ONBLD_TOOLS)/bin/$(MACH)/stabs
ELFEXTRACT=     $(ONBLD_TOOLS)/bin/$(MACH)/elfextract
MBH_PATCH=      $(ONBLD_TOOLS)/bin/$(MACH)/mbh_patch
ECHO=           echo
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 = -erttags=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              =
```

```
CUSERFLAGS64            = $(CUSERFLAGS)
CCUSERFLAGS             =
CCUSERFLAGS64           = $(CCUSERFLAGS)

CSOURCEDEBUGFLAGS       =
CCSOURCEDEBUGFLAGS      =
$(SRCDBGBLD)CSOURCEDEBUGFLAGS   = -g -xs
$(SRCDBGBLD)CCSOURCEDEBUGFLAGS  = -g -xs

CFLAGS=         $(COPTFLAG) $($(MACH)_CFLAGS) $(SPACEFLAG) $(CCMODE) \
                $(ILDOFF) $(CERRWARN) $(C99MODE) $(CCUNBOUND) $(IROPTFLAG) \
                $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CSOURCEDEBUGFLAGS) \
                $(CUSERFLAGS)
CFLAGS64=       $(COPTFLAG64) $($(MACH64)_CFLAGS) $(SPACEFLAG64) $(CCMODE64) \
                $(ILDOFF) $(CERRWARN) $(C99MODE) $(CCUNBOUND) $(IROPTFLAG64) \
                $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CSOURCEDEBUGFLAGS) \
                $(CUSERFLAGS64)
#
# Flags that are used to build parts of the code that are subsequently
# run on the build machine (also known as the NATIVE_BUILD).
#
NATIVE_CFLAGS=  $(COPTFLAG) $($(NATIVE_MACH)_CFLAGS) $(CCMODE) \
                $(ILDOFF) $(CERRWARN) $(C99MODE) $($(NATIVE_MACH)_CCUNBOUND) \
                $(IROPTFLAG) $(CGLOBALSTATIC) $(CCNOAUTOINLINE) \
                $(CSOURCEDEBUGFLAGS) $(CUSERFLAGS)

DTEXTDOM=-DTEXT_DOMAIN=\"$(TEXT_DOMAIN)\"       # For messaging.
DTS_ERRNO=-D_TS_ERRNO
CPPFLAGS.master=$(DTEXTDOM) $(DTS_ERRNO) \
        $(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) $(ENVCPPFLAGS4) \
        $(ADJUNCT_PROTO:%=-I%/usr/include)
CPPFLAGS.native=$(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) \
                $(ENVCPPFLAGS4) -I$(NATIVE_ADJUNCT)/include
CPPFLAGS=       $(CPPFLAGS.master)
AS_CPPFLAGS=    $(CPPFLAGS.master)
JAVAFLAGS=      -deprecation

#
# For source message catalogue
#
.SUFFIXES: $(SUFFIXES) .i .po
MSGROOT= $(ROOT)/catalog
MSGDOMAIN= $(MSGROOT)/$(TEXT_DOMAIN)
MSGDOMAINPOFILE = $(MSGDOMAIN)/$(POFILE)
DCMSGDOMAIN= $(MSGROOT)/LC_TIME/$(TEXT_DOMAIN)
DCMSGDOMAINPOFILE = $(DCMSGDOMAIN)/$(DCFILE:.dc=.po)

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

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

sparc_CCFLAGS=          -cg92 -compat=4 \
                        -Qoption ccfe -messages=no%anachronism \
```



```
                        $(CCERRWARN)
sparcv9_CCFLAGS=        $(sparcv9_XARCH) -dalign -compat=5 \
                        -Qoption ccfe -messages=no%anachronism \
                        -Qoption ccfe -features=no%conststrings \
                        $(CCCREGSYM) \
                        $(CCERRWARN)
i386_CCFLAGS=           -compat=4 \
                        -Qoption ccfe -messages=no%anachronism \
                        -Qoption ccfe -features=no%conststrings \
                        $(CCERRWARN)
amd64_CCFLAGS=          $(amd64_XARCH) -compat=5 \
                        -Qoption ccfe -messages=no%anachronism \
                        -Qoption ccfe -features=no%conststrings \
                        $(CCERRWARN)

sparc_CCOPTFLAG=        -O
sparcv9_CCOPTFLAG=      -O
i386_CCOPTFLAG=         -O
amd64_CCOPTFLAG=        -O

CCOPTFLAG=      $($(MACH)_CCOPTFLAG)
CCOPTFLAG64=    $($(MACH64)_CCOPTFLAG)
CCFLAGS=        $(CCOPTFLAG) $($(MACH)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
                $(CCUSERFLAGS)
CCFLAGS64=      $(CCOPTFLAG64) $($(MACH64)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
                $(CCUSERFLAGS64)

#
#
#
ELFWRAP_FLAGS   =
ELFWRAP_FLAGS64 =       -64

#
# Various mapfiles that are used throughout the build, and delivered to
# /usr/lib/ld.
#
MAPFILE.NED_i386 =      $(SRC)/common/mapfiles/common/map.noexdata
MAPFILE.NED_sparc =
MAPFILE.NED =           $(MAPFILE.NED_$(MACH))
MAPFILE.PGA =           $(SRC)/common/mapfiles/common/map.pagealign
MAPFILE.NES =           $(SRC)/common/mapfiles/common/map.noexstk
MAPFILE.FLT =           $(SRC)/common/mapfiles/common/map.filter
MAPFILE.LEX =           $(SRC)/common/mapfiles/common/map.lex.yy

#
# Generated mapfiles that are compiler specific, and used throughout the
# build.  These mapfiles are not delivered in /usr/lib/ld.
#
MAPFILE.NGB_sparc=      $(SRC)/common/mapfiles/gen/sparc_cc_map.noexeglobs
$(__GNUC64)MAPFILE.NGB_sparc= \
                        $(SRC)/common/mapfiles/gen/sparc_gcc_map.noexeglobs
MAPFILE.NGB_sparcv9=    $(SRC)/common/mapfiles/gen/sparcv9_cc_map.noexeglobs
$(__GNUC64)MAPFILE.NGB_sparcv9= \
                        $(SRC)/common/mapfiles/gen/sparcv9_gcc_map.noexeglobs
MAPFILE.NGB_i386=       $(SRC)/common/mapfiles/gen/i386_cc_map.noexeglobs
$(__GNUC64)MAPFILE.NGB_i386= \
                        $(SRC)/common/mapfiles/gen/i386_gcc_map.noexeglobs
MAPFILE.NGB_amd64=      $(SRC)/common/mapfiles/gen/amd64_cc_map.noexeglobs
$(__GNUC64)MAPFILE.NGB_amd64= \
                        $(SRC)/common/mapfiles/gen/amd64_gcc_map.noexeglobs
MAPFILE.NGB =           $(MAPFILE.NGB_$(MACH))

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

```
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)
 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
```

```
**********************************************************
   34973 Wed Jan 22 16:20:23 2014
new/usr/src/tools/scripts/check_rtime.pl
4505 check_rtime should always check search paths
**********************************************************
```

```
   1 #!/usr/perl5/bin/perl -w
   2 #
   3 # CDDL HEADER START
   4 #
   5 # The contents of this file are subject to the terms of the
   6 # Common Development and Distribution License (the "License").
   7 # You may not use this file except in compliance with the License.
   8 #
   9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  10 # or http://www.opensolaris.org/os/licensing.
  11 # See the License for the specific language governing permissions
  12 # and limitations under the License.
  13 #
  14 # When distributing Covered Code, include this CDDL HEADER in each
  15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  16 # If applicable, add the following below this CDDL HEADER, with the
  17 # fields enclosed by brackets "[]" replaced with your own identifying
  18 # information: Portions Copyright [yyyy] [name of copyright owner]
  19 #
  20 # CDDL HEADER END
  21 #

  23 #
  24 # Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
  25 #

  27 #
  28 # Check ELF information.
  29 #
  30 # This script descends a directory hierarchy inspecting ELF dynamic executables
  31 # and shared objects.  The general theme is to verify that common Makefile rules
  32 # have been used to build these objects.  Typical failures occur when Makefile
  33 # rules are re-invented rather than being inherited from "cmd/lib" Makefiles.
  34 #
  35 # As always, a number of components don't follow the rules, and these are
  36 # excluded to reduce this scripts output.
  37 #
  38 # By default any file that has conditions that should be reported is first
  39 # listed and then each condition follows.  The -o (one-line) option produces a
  40 # more terse output which is better for sorting/diffing with "nightly".
  41 #
  42 # NOTE: missing dependencies, symbols or versions are reported by running the
  43 # file through ldd(1).  As objects within a proto area are built to exist in a
  44 # base system, standard use of ldd(1) will bind any objects to dependencies
  45 # that exist in the base system.  It is frequently the case that newer objects
  46 # exist in the proto area that are required to satisfy other objects
  47 # dependencies, and without using these newer objects an ldd(1) will produce
  48 # misleading error messages.  To compensate for this, the -D/-d options, or the
  49 # existence of the CODEMSG_WS/ROOT environment variables, cause the creation of
  50 # alternative dependency mappings via crle(1) configuration files that establish
  51 # any proto shared objects as alternatives to their base system location.  Thus
  52 # ldd(1) can be executed against these configuration files so that objects in a
  53 # proto area bind to their dependencies in the same proto area.


  56 # Define all global variables (required for strict)
  57 use vars  qw($Prog $Env $Ena64 $Tmpdir);
  57 use vars  qw($Prog $Env $Ena64 $Tmpdir $Gnuc);
  58 use vars  qw($LddNoU $Conf32 $Conf64);
  59 use vars  qw(%opt);
  60 use vars  qw($ErrFH $ErrTtl $InfoFH $InfoTtl $OutCnt1 $OutCnt2);
```



```
  62 # An exception file is used to specify regular expressions to match
  63 # objects. These directives specify special attributes of the object.
  64 # The regular expressions are read from the file and compiled into the
  65 # regular expression variables.
  66 #
  67 # The name of each regular expression variable is of the form
  68 #
  69 #       $EXRE_xxx
  70 #
  71 # where xxx is the name of the exception in lower case. For example,
  72 # the regular expression variable for EXEC_STACK is $EXRE_exec_stack.
  73 #
  74 # onbld_elfmod::LoadExceptionsToEXRE() depends on this naming convention
  75 # to initialize the regular expression variables, and to detect invalid
  76 # exception names.
  77 #
  78 # If a given exception is not used in the exception file, its regular
  79 # expression variable will be undefined. Users of these variables must
  80 # test the variable with defined() prior to use:
  81 #
  82 #       defined($EXRE_exec_stack) && ($foo =~ $EXRE_exec_stack)
  83 #
  84 # or if the test is to make sure the item is not specified:
  85 #
  86 #       !defined($EXRE_exec_stack) || ($foo !~ $EXRE_exec_stack)
  87 #
  88 # ----
  89 #
  90 # The exceptions are:
  91 #
  92 #   EXEC_DATA
  93 #       Objects that are not required to have non-executable writable
  94 #       data segments.
  95 #
  96 #   EXEC_STACK
  97 #       Objects that are not required to have a non-executable stack
  98 #
  99 #   NOCRLEALT
 100 #       Objects that should be skipped by AltObjectConfig() when building
 101 #       the crle script that maps objects to the proto area.
 102 #
 103 #    NODIRECT
 104 #       Objects that are not required to use direct bindings
 105 #
 106 #    NOSYMSORT
 107 #       Objects we should not check for duplicate addresses in
 108 #       the symbol sort sections.
 109 #
 110 #    OLDDEP
 111 #       Objects that are no longer needed because their functionalty
 112 #       has migrated elsewhere. These are usually pure filters that
 113 #       point at libc.
 114 #
 115 #    SKIP
 116 #       Files and directories that should be excluded from analysis.
 117 #
 118 #    STAB
 119 #       Objects that are allowed to contain stab debugging sections
 120 #
 121 #    TEXTREL
 122 #       Object for which relocations are allowed to the text segment
 123 #
 124 #    UNDEF_REF
 125 #       Objects that are allowed undefined references
 126 #
```

```
127 #    UNREF_OBJ
128 #       "unreferenced object=" ldd(1) diagnostics.
129 #
130 #    UNUSED_DEPS
131 #       Objects that are allowed to have unused dependencies
132 #
133 #    UNUSED_OBJ
134 #       Objects that are allowed to be unused dependencies
135 #
136 #    UNUSED_RPATH
137 #       Objects with unused runpaths
138 #

140 use vars  qw($EXRE_exec_data $EXRE_exec_stack $EXRE_nocrlealt);
141 use vars  qw($EXRE_nodirect $EXRE_nosymsort);
142 use vars  qw($EXRE_olddep $EXRE_skip $EXRE_stab $EXRE_textrel $EXRE_undef_ref);
143 use vars  qw($EXRE_unref_obj $EXRE_unused_deps $EXRE_unused_obj);
144 use vars  qw($EXRE_unused_rpath);

146 use strict;
147 use Getopt::Std;
148 use File::Basename;


151 # Reliably compare two OS revisions.  Arguments are <ver1> <op> <ver2>.
152 # <op> is the string form of a normal numeric comparison operator.
153 sub cmp_os_ver {
154         my @ver1 = split(/\./, $_[0]);
155         my $op = $_[1];
156         my @ver2 = split(/\./, $_[2]);

158         push @ver2, ("0") x $#ver1 - $#ver2;
159         push @ver1, ("0") x $#ver2 - $#ver1;

161         my $diff = 0;
162         while (@ver1 || @ver2) {
163                 if (($diff = shift(@ver1) - shift(@ver2)) != 0) {
164                         last;
165                 }
166         }
167         return (eval "$diff $op 0" ? 1 : 0);
168 }

170 ## ProcFile(FullPath, RelPath, File, Class, Type, Verdef)
171 #
172 # Determine whether this a ELF dynamic object and if so investigate its runtime
173 # attributes.
174 #
175 sub ProcFile {
176         my($FullPath, $RelPath, $Class, $Type, $Verdef) = @_;
177         my(@Elf, @Ldd, $Dyn, $Sym, $Stack);
178         my($Sun, $Relsz, $Pltsz, $Tex, $Stab, $Strip, $Lddopt, $SymSort);
179         my($Val, $Header, $IsX86, $RWX, $UnDep);
180         my($HasDirectBinding);

182         # Only look at executables and sharable objects
183         return if ($Type ne 'EXEC') && ($Type ne 'DYN');

185         # Ignore symbolic links
186         return if -l $FullPath;

188         # Is this an object or directory hierarchy we don't care about?
189         return if (defined($EXRE_skip) && ($RelPath =~ $EXRE_skip));

191         # Bail if we can't stat the file. Otherwise, note if it is SUID/SGID.
192         return if !stat($FullPath);
```



```
193         my $Secure = (-u _ || -g _) ? 1 : 0;

195         # Reset output message counts for new input file
196         $$ErrTtl = $$InfoTtl = 0;

198         @Ldd = 0;

200         # Determine whether we have access to inspect the file.
201         if (!(-r $FullPath)) {
202                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
203                     "unable to inspect file: permission denied");
204                 return;
205         }

207         # Determine whether we have a executable (static or dynamic) or a
208         # shared object.
209         @Elf = split(/\n/, 'elfdump -epdcy $FullPath 2>&1');

211         $Dyn = $Stack = $IsX86 = $RWX = 0;
212         $Header = 'None';
213         foreach my $Line (@Elf) {
214                 # If we have an invalid file type (which we can tell from the
215                 # first line), or we're processing an archive, bail.
216                 if ($Header eq 'None') {
217                         if (($Line =~ /invalid file/) ||
218                             ($Line =~ /\Q$FullPath\E(.*):/)) {
219                                 return;
220                         }
221                 }

223                 if ($Line =~ /^ELF Header/) {
224                         $Header = 'Ehdr';
225                         next;
226                 }

228                 if ($Line =~ /^Program Header/) {
229                         $Header = 'Phdr';
230                         $RWX = 0;
231                         next;
232                 }

234                 if ($Line =~ /^Dynamic Section/) {
235                         # A dynamic section indicates we're a dynamic object
236                         # (this makes sure we don't check static executables).
237                         $Dyn = 1;
238                         next;
239                 }

241                 if (($Header eq 'Ehdr') && ($Line =~ /e_machine:/)) {
242                         # If it's a X86 object, we need to enforce RW- data.
243                         $IsX86 = 1 if $Line =~ /(EM_AMD64|EM_386)/;
244                         next;
245                 }

247                 if (($Header eq 'Phdr') &&
248                     ($Line =~ /\[ PF_X\s+PF_W\s+PF_R \]/)) {
249                         # RWX segment seen.
250                         $RWX = 1;
251                         next;
252                 }

254                 if (($Header eq 'Phdr') &&
255                     ($Line =~ /\[ PT_LOAD \]/ && $RWX && $IsX86)) {
256                         # Seen an RWX PT_LOAD segment.
257                         if (!defined($EXRE_exec_data) ||
258                             ($RelPath !~ $EXRE_exec_data)) {
```

```
259                                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
260                                     "application requires non-executable " .
261                                     "data\t<no -Mmapfile_noexdata?>");
262                         }
263                         next;
264                 }

266                 if (($Header eq 'Phdr') && ($Line =~ /\[ PT_SUNWSTACK \]/)) {
267                         # This object defines a non-executable stack.
268                         $Stack = 1;
269                         next;
270                 }
271         }

273         # Determine whether this ELF executable or shared object has a
274         # conforming mcs(1) comment section.  If the correct $(POST_PROCESS)
275         # macros are used, only a 3 or 4 line .comment section should exist
276         # containing one or two "@(#)SunOS" identifying comments (one comment
277         # for a non-debug build, and two for a debug build). The results of
278         # the following split should be three or four lines, the last empty
279         # line being discarded by the split.
280         if ($opt{m}) {
281                 my(@Mcs, $Con, $Dev);

283                 @Mcs = split(/\n/, 'mcs -p $FullPath 2>&1');

285                 $Con = $Dev = $Val = 0;
286                 foreach my $Line (@Mcs) {
287                         $Val++;

289                         if (($Val == 3) && ($Line !~ /^@\(#\)SunOS/)) {
290                                 $Con = 1;
291                                 last;
292                         }
293                         if (($Val == 4) && ($Line =~ /^@\(#\)SunOS/)) {
294                                 $Dev = 1;
295                                 next;
296                         }
297                         if (($Dev == 0) && ($Val == 4)) {
298                                 $Con = 1;
299                                 last;
300                         }
301                         if (($Dev == 1) && ($Val == 5)) {
302                                 $Con = 1;
303                                 last;
304                         }
305                 }
306                 if ($opt{m} && ($Con == 1)) {
307                         onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
308                             "non-conforming mcs(1) comment\t<no \$(POST_PROCESS)?>");
309                 }
310         }

312         # Applications should contain a non-executable stack definition.
313         if (($Type eq 'EXEC') && ($Stack == 0) &&
314             (!defined($EXRE_exec_stack) || ($RelPath !~ $EXRE_exec_stack))) {
315                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
316                     "non-executable stack required\t<no -Mmapfile_noexstk?>");
317         }

319         # Having caught any static executables in the mcs(1) check and non-
320         # executable stack definition check, continue with dynamic objects
321         # from now on.
322         if ($Dyn eq 0) {
323                 return;
324         }
```



```
326         # Use ldd unless its a 64-bit object and we lack the hardware.
327         if (($Class == 32) || $Ena64) {
328                 my $LDDFullPath = $FullPath;

330                 if ($Secure) {
331                         # The execution of a secure application over an nfs file
332                         # system mounted nosuid will result in warning messages
333                         # being sent to /var/adm/messages.  As this type of
334                         # environment can occur with root builds, move the file
335                         # being investigated to a safe place first.  In addition
336                         # remove its secure permission so that it can be
337                         # influenced by any alternative dependency mappings.
338
339                         my $File = $RelPath;
340                         $File =~ s!^.*/!!;      # basename

342                         my($TmpPath) = "$Tmpdir/$File";

344                         system('cp', $LDDFullPath, $TmpPath);
345                         chmod 0777, $TmpPath;
346                         $LDDFullPath = $TmpPath;
347                 }

349                 # Use ldd(1) to determine the objects relocatability and use.
350                 # By default look for all unreferenced dependencies.  However,
351                 # some objects have legitimate dependencies that they do not
352                 # reference.
353                 if ($LddNoU) {
354                         $Lddopt = "-ru";
355                 } else {
356                         $Lddopt = "-rU";
357                 }
358                 @Ldd = split(/\n/, 'ldd $Lddopt $Env $LDDFullPath 2>&1');
359                 if ($Secure) {
360                         unlink $LDDFullPath;
361                 }
362         }

364         $Val = 0;
365         $Sym = 5;
366         $UnDep = 1;

368         foreach my $Line (@Ldd) {

370                 if ($Val == 0) {
371                         $Val = 1;
372                         # Make sure ldd(1) worked.  One possible failure is that
373                         # this is an old ldd(1) prior to -e addition (4390308).
374                         if ($Line =~ /usage:/) {
375                                 $Line =~ s/$/\t<old ldd(1)?>/;
376                                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl,
377                                     $RelPath, $Line);
378                                 last;
379                         } elsif ($Line =~ /execution failed/) {
380                                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl,
381                                     $RelPath, $Line);
382                                 last;
383                         }

385                         # It's possible this binary can't be executed, ie. we've
386                         # found a sparc binary while running on an intel system,
387                         # or a sparcv9 binary on a sparcv7/8 system.
388                         if ($Line =~ /wrong class/) {
389                                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
390                                     "has wrong class or data encoding");
```

```
391                             next;
392                     }
393
394                     # Historically, ldd(1) likes executable objects to have
395                     # their execute bit set.
396                     if ($Line =~ /not executable/) {
397                             onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
398                                 "is not executable");
399                             next;
400                     }
401             }
402
403             # Look for "file" or "versions" that aren't found.  Note that
404             # these lines will occur before we find any symbol referencing
405             # errors.
406             if (($Sym == 5) && ($Line =~ /not found\)/)) {
407                     if ($Line =~ /file not found\)/) {
408                             $Line =~ s/$/\t<no -zdefs?>/;
409                     }
410                     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath, $Line);
411                     next;
412             }
413             # Look for relocations whose symbols can't be found.  Note, we
414             # only print out the first 5 relocations for any file as this
415             # output can be excessive.
416             if ($Sym && ($Line =~ /symbol not found/)) {
417                     # Determine if this file is allowed undefined
418                     # references.
419                     if (($Sym == 5) && defined($EXRE_undef_ref) &&
420                         ($RelPath =~ $EXRE_undef_ref)) {
421                             $Sym = 0;
422                             next;
423                     }
424                     if ($Sym-- == 1) {
425                             onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
426                                 "continued ...") if !$opt{o};
427                             next;
428                     }
429                     # Just print the symbol name.
430                     $Line =~ s/$/\t<no -zdefs?>/;
431                     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath, $Line);
432                     next;
433             }
434             # Look for any unused search paths.
435             if ($Line =~ /unused search path=/) {
436                     # Note, skip this comparison for __GNUC builds, as the
437                     # gnu compilers insert numerous unused search paths.
438                     if ($Gnuc == 1) {
439                             next;
440                     }
436                     next if defined($EXRE_unused_rpath) &&
437                         ($Line =~ $EXRE_unused_rpath);
438
439                     if ($Secure) {
440                             $Line =~ s!$Tmpdir/!!;
441                     }
442                     $Line =~ s/^[ \t]*(.*)/\t$1\t<remove search path?>/;
443                     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath, $Line);
444                     next;
445             }
446             # Look for unreferenced dependencies.  Note, if any unreferenced
447             # objects are ignored, then set $UnDep so as to suppress any
448             # associated unused-object messages.
449             if ($Line =~ /unreferenced object=/) {
450                     if (defined($EXRE_unref_obj) &&
451                         ($Line =~ $EXRE_unref_obj)) {
```



```
452                             $UnDep = 0;
453                             next;
454                     }
455                     if ($Secure) {
456                             $Line =~ s!$Tmpdir/!!;
457                     }
458                     $Line =~ s/^[ \t]*(.*)/$1\t<remove lib or -zignore?>/;
459                     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath, $Line);
460                     next;
461             }
462             # Look for any unused dependencies.
463             if ($UnDep && ($Line =~ /unused/)) {
464                     # Skip if object is allowed to have unused dependencies
465                     next if defined($EXRE_unused_deps) &&
466                         ($RelPath =~ $EXRE_unused_deps);
467
468                     # Skip if dependency is always allowed to be unused
469                     next if defined($EXRE_unused_obj) &&
470                         ($Line =~ $EXRE_unused_obj);
471
472                     $Line =~ s!$Tmpdir/!! if $Secure;
473                     $Line =~ s/^[ \t]*(.*)/$1\t<remove lib or -zignore?>/;
474                     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath, $Line);
475                     next;
476             }
477     }
478
479     # Reuse the elfdump(1) data to investigate additional dynamic linking
480     # information.
481
482     $Sun = $Relsz = $Pltsz = $Dyn = $Stab = $SymSort = 0;
483     $Tex = $Strip = 1;
484     $HasDirectBinding = 0;
485
486     $Header = 'None';
487 ELF:    foreach my $Line (@Elf) {
488             # We're only interested in the section headers and the dynamic
489             # section.
490             if ($Line =~ /^Section Header/) {
491                     $Header = 'Shdr';
492
493                     if (($Sun == 0) && ($Line =~ /\.SUNW_reloc/)) {
494                             # This object has a combined relocation section.
495                             $Sun = 1;
496
497                     } elsif (($Stab == 0) && ($Line =~ /\.stab/)) {
498                             # This object contain .stabs sections
499                             $Stab = 1;
500                     } elsif (($SymSort == 0) &&
501                              ($Line =~ /\.SUNW_dyn(sym)|(tls)sort/)) {
502                             # This object contains a symbol sort section
503                             $SymSort = 1;
504                     }
505
506                     if (($Strip == 1) && ($Line =~ /\.symtab/)) {
507                             # This object contains a complete symbol table.
508                             $Strip = 0;
509                     }
510                     next;
511
512             } elsif ($Line =~ /^Dynamic Section/) {
513                     $Header = 'Dyn';
514                     next;
515             } elsif ($Line =~ /^Syminfo Section/) {
516                     $Header = 'Syminfo';
517                     next;
```

```
518                 } elsif (($Header ne 'Dyn') && ($Header ne 'Syminfo')) {
519                         next;
520                 }

522                 # Look into the Syminfo section.
523                 # Does this object have at least one Directly Bound symbol?
524                 if (($Header eq 'Syminfo')) {
525                         my(@Symword);

527                         if ($HasDirectBinding == 1) {
528                                 next;
529                         }

531                         @Symword = split(' ', $Line);

533                         if (!defined($Symword[1])) {
534                                 next;
535                         }
536                         if ($Symword[1] =~ /B/) {
537                                 $HasDirectBinding = 1;
538                         }
539                         next;
540                 }

542                 # Does this object contain text relocations.
543                 if ($Tex && ($Line =~ /TEXTREL/)) {
544                         # Determine if this file is allowed text relocations.
545                         if (defined($EXRE_textrel) &&
546                             ($RelPath =~ $EXRE_textrel)) {
547                                 $Tex = 0;
548                                 next ELF;
549                         }
550                         onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
551                             "TEXTREL .dynamic tag\t\t\t<no -Kpic?>");
552                         $Tex = 0;
553                         next;
554                 }

556                 # Does this file have any relocation sections (there are a few
557                 # psr libraries with no relocations at all, thus a .SUNW_reloc
558                 # section won't exist either).
559                 if (($Relsz == 0) && ($Line =~ / RELA?SZ/)) {
560                         $Relsz = hex((split(' ', $Line))[2]);
561                         next;
562                 }

564                 # Does this file have any plt relocations.  If the plt size is
565                 # equivalent to the total relocation size then we don't have
566                 # any relocations suitable for combining into a .SUNW_reloc
567                 # section.
568                 if (($Pltsz == 0) && ($Line =~ / PLTRELSZ/)) {
569                         $Pltsz = hex((split(' ', $Line))[2]);
570                         next;
571                 }

573                 # Does this object have any dependencies.
574                 if ($Line =~ /NEEDED/) {
575                         my($Need) = (split(' ', $Line))[3];

577                         if (defined($EXRE_olddep) && ($Need =~ $EXRE_olddep)) {
578                                 # Catch any old (unnecessary) dependencies.
579                                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
580                         "NEEDED=$Need\t<dependency no longer necessary>");
581                         } elsif ($opt{i}) {
582                                 # Under the -i (information) option print out
583                                 # any useful dynamic entries.
```



```
584                                 onbld_elfmod::OutMsg($InfoFH, $InfoTtl, $RelPath
585                                     "NEEDED=$Need");
586                         }
587                         next;
588                 }

590                 # Is this object built with -B direct flag on?
591                 if ($Line =~ / DIRECT /) {
592                         $HasDirectBinding = 1;
593                 }

595                 # Does this object specify a runpath.
596                 if ($opt{i} && ($Line =~ /RPATH/)) {
597                         my($Rpath) = (split(' ', $Line))[3];
598                         onbld_elfmod::OutMsg($InfoFH, $InfoTtl,
599                             $RelPath, "RPATH=$Rpath");
600                         next;
601                 }
602         }

604         # A shared object, that contains non-plt relocations, should have a
605         # combined relocation section indicating it was built with -z combreloc.
606         if (($Type eq 'DYN') && $Relsz && ($Relsz != $Pltsz) && ($Sun == 0)) {
607                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
608                     ".SUNW_reloc section missing\t\t<no -zcombreloc?>");
609         }

611         # No objects released to a customer should have any .stabs sections
612         # remaining, they should be stripped.
613         if ($opt{s} && $Stab) {
614                 goto DONESTAB if defined($EXRE_stab) && ($RelPath =~ $EXRE_stab)

616                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
617                     "debugging sections should be deleted\t<no strip -x?>");
618         }

620         # Identify an object that is not built with either -B direct or
621         # -z direct.
622         goto DONESTAB
623             if (defined($EXRE_nodirect) && ($RelPath =~ $EXRE_nodirect));

625         if ($Relsz && ($HasDirectBinding == 0)) {
626                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
627                  "object has no direct bindings\t<no -B direct or -z direct?>");
628         }

630 DONESTAB:

632         # All objects should have a full symbol table to provide complete
633         # debugging stack traces.
634         onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
635             "symbol table should not be stripped\t<remove -s?>") if $Strip;

637         # If there are symbol sort sections in this object, report on
638         # any that have duplicate addresses.
639         ProcSymSort($FullPath, $RelPath) if $SymSort;

641         # If -v was specified, and the object has a version definition
642         # section, generate output showing each public symbol and the
643         # version it belongs to.
644         ProcVerdef($FullPath, $RelPath)
645             if ($Verdef eq 'VERDEF') && $opt{v};
646 }
______unchanged_portion_omitted_

1075 die "$Prog: -D and -d options are mutually exclusive\n" if ($opt{D} && $opt{d});
```

```
1077 $Tmpdir = "/tmp" if (!($Tmpdir = $ENV{TMPDIR}) || (! -d $Tmpdir));

1084 # Determine whether this is a __GNUC build.  If so, unused search path
1085 # processing is disabled.
1086 $Gnuc = defined $ENV{__GNUC} ? 1 : 0;

1079 # If -w, change working directory to given location
1080 !$opt{w} || chdir($opt{w}) || die "$Prog: can't cd to $opt{w}";

1082 # Locate and process the exceptions file
1083 onbld_elfmod::LoadExceptionsToEXRE('check_rtime');

1085 # Is there a proto area available, either via the -d option, or because
1086 # we are part of an activated workspace?
1087 my $Proto;
1088 if ($opt{d}) {
1089         # User specified dependency directory - make sure it exists.
1090         -d $opt{d} || die "$Prog: $opt{d} is not a directory\n";
1091         $Proto = $opt{d};
1092 } elsif ($ENV{CODEMGR_WS}) {
```

_____***unchanged_portion_omitted_***