```
**********************************************************
   35909 Thu Oct  4 19:36:11 2018
new/usr/src/Makefile.master
9420 need GCC options to disable function cloning
**********************************************************
```

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

  22 #
  23 # Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
  24 # Copyright (c) 2012 by Delphix. All rights reserved.
  25 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
  26 # Copyright 2015, OmniTI Computer Consulting, Inc. All rights reserved.
  27 # Copyright 2015 Gary Mills
  28 # Copyright 2015 Igor Kozhukhov <ikozhukhov@gmail.com>
  29 # Copyright 2016 Toomas Soome <tsoome@me.com>
  30 # Copyright 2018 OmniOS Community Edition (OmniOSce) Association.
  31 #

  33 #
  34 # Makefile.master, global definitions for system source
  35 #
  36 ROOT=           /proto

  38 #
  39 # Adjunct root, containing an additional proto area to be used for headers
  40 # and libraries.
  41 #
  42 ADJUNCT_PROTO=

  44 #
  45 # Adjunct for building things that run on the build machine.
  46 #
  47 NATIVE_ADJUNCT= /usr

  49 #
  50 # RELEASE_BUILD should be cleared for final release builds.
  51 # NOT_RELEASE_BUILD is exactly what the name implies.
  52 #
  53 # __GNUC toggles the building of ON components using gcc and related tools.
  54 # Normally set to '#', set it to '' to do gcc build.
  55 #
  56 # The declaration POUND_SIGN is always '#'. This is needed to get around the
  57 # make feature that '#' is always a comment delimiter, even when escaped or
  58 # quoted. We use this macro expansion method to get POUND_SIGN rather than
  59 # always breaking out a shell because the general case can cause a noticable
  60 # slowdown in build times when so many Makefiles include Makefile.master.
  61 #
```



```
  62 # While the majority of users are expected to override the setting below
  63 # with an env file (via nightly or bldenv), if you aren't building that way
  64 # (ie, you're using "ws" or some other bootstrapping method) then you need
  65 # this definition in order to avoid the subshell invocation mentioned above.
  66 #

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

  71 NOT_RELEASE_BUILD=
  72 RELEASE_BUILD=                          $(POUND_SIGN)
  73 $(RELEASE_BUILD)NOT_RELEASE_BUILD=      $(POUND_SIGN)
  74 PATCH_BUILD=                            $(POUND_SIGN)

  76 # SPARC_BLD is '#' for an Intel build.
  77 # INTEL_BLD is '#' for a Sparc build.
  78 SPARC_BLD_1=    $(MACH:i386=$(POUND_SIGN))
  79 SPARC_BLD=      $(SPARC_BLD_1:sparc=)
  80 INTEL_BLD_1=    $(MACH:sparc=$(POUND_SIGN))
  81 INTEL_BLD=      $(INTEL_BLD_1:i386=)

  83 # The variables below control the compilers used during the build.
  84 # There are a number of permutations.
  85 #
  86 # __GNUC and __SUNC control (and indicate) the primary compiler.  Whichever
  87 # one is not POUND_SIGN is the primary, with the other as the shadow.  They
  88 # may also be used to control entirely compiler-specific Makefile assignments.
  89 # __GNUC and GCC are the default.
  90 #
  91 # __GNUC64 indicates that the 64bit build should use the GNU C compiler.
  92 # There is no Sun C analogue.
  93 #
  94 # The following version-specific options are operative regardless of which
  95 # compiler is primary, and control the versions of the given compilers to be
  96 # used.  They also allow compiler-version specific Makefile fragments.
  97 #

  99 __SUNC=                 $(POUND_SIGN)
 100 $(__SUNC)__GNUC=        $(POUND_SIGN)
 101 __GNUC64=               $(__GNUC)

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

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

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

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

 119 # define runtime JAVA_HOME, primarily for cmd/pools/poold
 120 JAVA_HOME=      /usr/java
 121 # define buildtime JAVA_ROOT
 122 JAVA_ROOT=      /usr/java
 123 # Build uses java7 by default.  Pass one the variables below set to empty
 124 # string in the environment to override.
 125 BLD_JAVA_6=     $(POUND_SIGN)
 126 BLD_JAVA_8=     $(POUND_SIGN)
```

```
GNUC_ROOT=      /opt/gcc/4.4.4
GCCLIBDIR=      $(GNUC_ROOT)/lib
GCCLIBDIR64=    $(GNUC_ROOT)/lib/$(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
BTXLD=          $(ONBLD_TOOLS)/bin/$(MACH)/btxld
VTFONTCVT=      $(ONBLD_TOOLS)/bin/$(MACH)/vtfontcvt
# echo(1) and true(1) are specified without absolute paths, so that the shell
# spawned by make(1) may use the built-in versions.  This is minimally
# problematic, as the shell spawned by make(1) is known and under control, the
# only risk being if the shell falls back to $PATH.
#
# We specifically want an echo(1) that does interpolation of escape sequences,
# which ksh93, /bin/sh, and bash will all provide.
ECHO=           echo
TRUE=           true
INS=            $(ONBLD_TOOLS)/bin/$(MACH)/install
SYMLINK=        /usr/bin/ln -s
LN=             /usr/bin/ln
MKDIR=          /usr/bin/mkdir
CHMOD=          /usr/bin/chmod
MV=             /usr/bin/mv -f
RM=             /usr/bin/rm -f
CUT=            /usr/bin/cut
NM=             /usr/ccs/bin/nm
DIFF=           /usr/bin/diff
GREP=           /usr/bin/grep
EGREP=          /usr/bin/egrep
ELFWRAP=        /usr/bin/elfwrap
KSH93=          /usr/bin/ksh93
SED=            /usr/bin/sed
AWK=            /usr/bin/nawk
CP=             /usr/bin/cp -f
MCS=            /usr/ccs/bin/mcs
CAT=            /usr/bin/cat
ELFDUMP=        /usr/ccs/bin/elfdump
M4=             /usr/bin/m4
STRIP=          /usr/ccs/bin/strip
LEX=            /usr/ccs/bin/lex
FLEX=           /usr/bin/flex
YACC=           /usr/ccs/bin/yacc
CPP=            /usr/lib/cpp
ANSI_CPP=       $(GNUC_ROOT)/bin/cpp
JAVAC=          $(JAVA_ROOT)/bin/javac
JAVAH=          $(JAVA_ROOT)/bin/javah
JAVADOC=        $(JAVA_ROOT)/bin/javadoc
RMIC=           $(JAVA_ROOT)/bin/rmic
JAR=            $(JAVA_ROOT)/bin/jar
CTFCONVERT=     $(ONBLD_TOOLS)/bin/$(MACH)/ctfconvert
CTFMERGE=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfmerge
CTFSTABS=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfstabs
CTFSTRIP=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfstrip
NDRGEN=         $(ONBLD_TOOLS)/bin/$(MACH)/ndrgen
GENOFFSETS=     $(ONBLD_TOOLS)/bin/genoffsets
XREF=           $(ONBLD_TOOLS)/bin/xref
FIND=           /usr/bin/find
PERL=           /usr/bin/perl
PERL_VERSION=   5.10.0
PERL_PKGVERS=   -510
PERL_ARCH =             i86pc-solaris-64int
$(SPARC_BLD)PERL_ARCH = sun4-solaris-64int
```



```
PYTHON_VERSION= 2.7
PYTHON_PKGVERS= -27
PYTHON=         /usr/bin/python$(PYTHON_VERSION)
SORT=           /usr/bin/sort
TOUCH=          /usr/bin/touch
WC=             /usr/bin/wc
XARGS=          /usr/bin/xargs
ELFEDIT=        /usr/bin/elfedit
DTRACE=         /usr/sbin/dtrace -xnolibs
UNIQ=           /usr/bin/uniq
TAR=            /usr/bin/tar
ASTBINDIR=      /usr/ast/bin
MSGCC=          $(ASTBINDIR)/msgcc
MSGFMT=         /usr/bin/msgfmt -s
LCDEF=          $(ONBLD_TOOLS)/bin/$(MACH)/localedef
TIC=            $(ONBLD_TOOLS)/bin/$(MACH)/tic
ZIC=            $(ONBLD_TOOLS)/bin/$(MACH)/zic
OPENSSL=        /usr/bin/openssl

FILEMODE=       644
DIRMODE=        755

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

# For stylistic checks
#
# Note that the X and C checks are not used at this time and may need
# modification when they are actually used.
#
CSTYLE=         $(ONBLD_TOOLS)/bin/cstyle
CSTYLE_TAIL=
HDRCHK=         $(ONBLD_TOOLS)/bin/hdrchk
HDRCHK_TAIL=
JSTYLE=         $(ONBLD_TOOLS)/bin/jstyle

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

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

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

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

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

# install a link
INSLINKTARGET=  $<
INS.link=       $(RM) $@; $(LN) $(INSLINKTARGET) $@
INS.symlink=    $(RM) $@; $(SYMLINK) $(INSLINKTARGET) $@
```

```
260 #
261 # Python bakes the mtime of the .py file into the compiled .pyc and
262 # rebuilds if the baked-in mtime != the mtime of the source file
263 # (rather than only if it's less than), thus when installing python
264 # files we must make certain to not adjust the mtime of the source
265 # (.py) file.
266 #
267 INS.pyfile=     $(RM) $@; $(SED) -e "1s:^\#!@PYTHON@:\#!$(PYTHON):" < $< > $@; $

269 # MACH must be set in the shell environment per uname -p on the build host
270 # More specific architecture variables should be set in lower makefiles.
271 #
272 # MACH64 is derived from MACH, and BUILD64 is set to '#' for
273 # architectures on which we do not build 64-bit versions.
274 # (There are no such architectures at the moment.)
275 #
276 # Set BUILD64=# in the environment to disable 64-bit amd64
277 # builds on i386 machines.

279 MACH64_1=       $(MACH:sparc=sparcv9)
280 MACH64=         $(MACH64_1:i386=amd64)

282 MACH32_1=       $(MACH:sparc=sparcv7)
283 MACH32=         $(MACH32_1:i386=i86)

285 sparc_BUILD64=
286 i386_BUILD64=
287 BUILD64=        $($(MACH)_BUILD64)

289 #
290 # C compiler mode. Future compilers may change the default on us,
291 # so force extended ANSI mode globally. Lower level makefiles can
292 # override this by setting CCMODE.
293 #
294 CCMODE=                 -Xa
295 CCMODE64=               -Xa

297 #
298 # C compiler verbose mode. This is so we can enable it globally,
299 # but turn it off in the lower level makefiles of things we cannot
300 # (or aren't going to) fix.
301 #
302 CCVERBOSE=              -v

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

308 # set this to the secret flag "-Wc,-Qiselect-regsym=0" to disable register
309 # symbols (used to detect conflicts between objects that use global registers)
310 # we disable this now for safety, and because genunix doesn't link with
311 # this feature (the v9 default) enabled.
312 #
313 # REGSYM is separate since the C++ driver syntax is different.
314 CCREGSYM=               -Wc,-Qiselect-regsym=0
315 CCCREGSYM=              -Qoption cg -Qiselect-regsym=0

317 # Prevent the removal of static symbols by the SPARC code generator (cg).
318 # The x86 code generator (ube) does not remove such symbols and as such
319 # using this workaround is not applicable for x86.
320 #
321 CCSTATICSYM=            -Wc,-Qassembler-ounrefsym=0
322 #
323 # generate 32-bit addresses in the v9 kernel. Saves memory.
324 CCABS32=                -Wc,-xcode=abs32
325 #
```



```
326 # generate v9 code which tolerates callers using the v7 ABI, for the sake of
327 # system calls.
328 CC32BITCALLERS=         -_gcc=-massume-32bit-callers

330 # GCC, especially, is increasingly beginning to auto-inline functions and
331 # sadly does so separately not under the general -fno-inline-functions
332 # Additionally, we wish to prevent optimisations which cause GCC to clone
333 # functions -- in particular, these may cause unhelpful symbols to be
334 # emitted instead of function names
335 CCNOAUTOINLINE= \
336         -_gcc=-fno-inline-small-functions \
335 CCNOAUTOINLINE= -_gcc=-fno-inline-small-functions \
337         -_gcc=-fno-inline-functions-called-once \
338         -_gcc=-fno-ipa-cp \
339         -_gcc6=-fno-ipa-icf \
340         -_gcc7=-fno-ipa-icf \
341         -_gcc8=-fno-ipa-icf \
342         -_gcc6=-fno-clone-functions \
343         -_gcc7=-fno-clone-functions \
344         -_gcc8=-fno-clone-functions
337         -_gcc=-fno-ipa-cp

346 # One optimization the compiler might perform is to turn this:
347 #       #pragma weak foo
348 #       extern int foo;
349 #       if (&foo)
350 #               foo = 5;
351 # into
352 #       foo = 5;
353 # Since we do some of this (foo might be referenced in common kernel code
354 # but provided only for some cpu modules or platforms), we disable this
355 # optimization.
356 #
357 sparc_CCUNBOUND = -Wd,-xsafe=unboundsym
358 i386_CCUNBOUND  =
359 CCUNBOUND       = $($(MACH)_CCUNBOUND)

361 #
362 # compiler '-xarch' flag. This is here to centralize it and make it
363 # overridable for testing.
364 sparc_XARCH=    -m32
365 sparcv9_XARCH=  -m64
366 i386_XARCH=     -m32
367 amd64_XARCH=    -m64 -Ui386 -U__i386

369 # assembler '-xarch' flag.  Different from compiler '-xarch' flag.
370 sparc_AS_XARCH=         -xarch=v8plus
371 sparcv9_AS_XARCH=       -xarch=v9
372 i386_AS_XARCH=
373 amd64_AS_XARCH=         -xarch=amd64 -P -Ui386 -U__i386

375 #
376 # These flags define what we need to be 'standalone' i.e. -not- part
377 # of the rather more cosy userland environment.  This basically means
378 # the kernel.
379 #
380 # XX64  future versions of gcc will make -mcmodel=kernel imply -mno-red-zone
381 #
382 sparc_STAND_FLAGS=      -_gcc=-ffreestanding
383 sparcv9_STAND_FLAGS=    -_gcc=-ffreestanding
384 # Disabling MMX also disables 3DNow, disabling SSE also disables all later
385 # additions to SSE (SSE2, AVX ,etc.)
386 NO_SIMD=                -_gcc=-mno-mmx -_gcc=-mno-sse
387 i386_STAND_FLAGS=       -_gcc=-ffreestanding $(NO_SIMD)
388 amd64_STAND_FLAGS=      -xmodel=kernel $(NO_SIMD)
```

```
390 SAVEARGS=               -Wu,-save_args
391 amd64_STAND_FLAGS       += $(SAVEARGS)

393 STAND_FLAGS_32 = $($(MACH)_STAND_FLAGS)
394 STAND_FLAGS_64 = $($(MACH64)_STAND_FLAGS)

396 #
397 # disable the incremental linker
398 ILDOFF=                 -xildoff
399 #
400 XDEPEND=                -xdepend
401 XFFLAG=                 -xF=%all
402 XESS=                   -xs
403 XSTRCONST=              -xstrconst

405 #
406 # turn warnings into errors (C)
407 CERRWARN = -errtags=yes -errwarn=%all
408 CERRWARN += -erroff=E_EMPTY_TRANSLATION_UNIT
409 CERRWARN += -erroff=E_STATEMENT_NOT_REACHED

411 CERRWARN += -_gcc=-Wno-missing-braces
412 CERRWARN += -_gcc=-Wno-sign-compare
413 CERRWARN += -_gcc=-Wno-unknown-pragmas
414 CERRWARN += -_gcc=-Wno-unused-parameter
415 CERRWARN += -_gcc=-Wno-missing-field-initializers

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

420 # DEBUG v. -nd make for frequent unused variables, empty conditions, etc. in
421 # -nd builds
422 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-unused
423 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-empty-body

425 #
426 # turn warnings into errors (C++)
427 CCERRWARN=              -xwe

429 # C standard.  Keep Studio flags until we get rid of lint.
430 CSTD_GNU89=     -xc99=%none
431 CSTD_GNU99=     -xc99=%all
432 CSTD=           $(CSTD_GNU89)
433 C99LMODE=       $(CSTD:-xc99%=-Xc99%)

435 # In most places, assignments to these macros should be appended with +=
436 # (CPPFLAGS.first allows values to be prepended to CPPFLAGS).
437 sparc_CFLAGS=   $(sparc_XARCH) $(CCSTATICSYM)
438 sparcv9_CFLAGS= $(sparcv9_XARCH) -dalign $(CCVERBOSE) $(V9ABIWARN) $(CCREGSYM) \
439                 $(CCSTATICSYM)
440 i386_CFLAGS=    $(i386_XARCH)
441 amd64_CFLAGS=   $(amd64_XARCH)

443 sparc_ASFLAGS=  $(sparc_AS_XARCH)
444 sparcv9_ASFLAGS=$(sparcv9_AS_XARCH)
445 i386_ASFLAGS=   $(i386_AS_XARCH)
446 amd64_ASFLAGS=  $(amd64_AS_XARCH)

448 #
449 sparc_COPTFLAG=         -xO3
450 sparcv9_COPTFLAG=       -xO3
451 i386_COPTFLAG=          -O
452 amd64_COPTFLAG=         -xO3

454 COPTFLAG= $($(MACH)_COPTFLAG)
455 COPTFLAG64= $($(MACH64)_COPTFLAG)
```



```
457 # When -g is used, the compiler globalizes static objects
458 # (gives them a unique prefix). Disable that.
459 CNOGLOBAL= -W0,-noglobal

461 # Direct the Sun Studio compiler to use a static globalization prefix based on t
462 # name of the module rather than something unique. Otherwise, objects
463 # will not build deterministically, as subsequent compilations of identical
464 # source will yeild objects that always look different.
465 #
466 # In the same spirit, this will also remove the date from the N_OPT stab.
467 CGLOBALSTATIC= -W0,-xglobalstatic

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

473 #
474 # Default debug format for Sun Studio 11 is dwarf, so force it to
475 # generate stabs.
476 #
477 DEBUGFORMAT=    -xdebugformat=stabs

479 #
480 # Flags used to build in debug mode for ctf generation.  Bugs in the Devpro
481 # compilers currently prevent us from building with cc-emitted DWARF.
482 #
483 CTF_FLAGS_sparc = -g -Wc,-Qiselect-T1 $(CSTD) $(CNOGLOBAL) $(CDWARFSTR)
484 CTF_FLAGS_i386  = -g $(CSTD) $(CNOGLOBAL) $(CDWARFSTR)

486 CTF_FLAGS_sparcv9       = $(CTF_FLAGS_sparc)
487 CTF_FLAGS_amd64         = $(CTF_FLAGS_i386)

489 # Sun Studio produces broken userland code when saving arguments.
490 $(__GNUC)CTF_FLAGS_amd64 += $(SAVEARGS)

492 CTF_FLAGS_32    = $(CTF_FLAGS_$(MACH)) $(DEBUGFORMAT)
493 CTF_FLAGS_64    = $(CTF_FLAGS_$(MACH64)) $(DEBUGFORMAT)
494 CTF_FLAGS       = $(CTF_FLAGS_32)

496 #
497 # Flags used with genoffsets
498 #
499 GOFLAGS = $(CALLSYMS) $(CDWARFSTR)

501 OFFSETS_CREATE = $(GENOFFSETS) -s $(CTFSTABS) -r $(CTFCONVERT) \
502         $(CW) --noecho $(CW_CC_COMPILERS) -- $(GOFLAGS) $(CFLAGS) $(CPPFLAGS)

504 OFFSETS_CREATE64 = $(GENOFFSETS) -s $(CTFSTABS) -r $(CTFCONVERT) \
505         $(CW) --noecho $(CW_CC_COMPILERS) -- $(GOFLAGS) $(CFLAGS64) $(CPPFLAGS)

507 #
508 # tradeoff time for space (smaller is better)
509 #
510 sparc_SPACEFLAG         = -xspace -W0,-Lt
511 sparcv9_SPACEFLAG       = -xspace -W0,-Lt
512 i386_SPACEFLAG          = -xspace
513 amd64_SPACEFLAG         =

515 SPACEFLAG               = $($(MACH)_SPACEFLAG)
516 SPACEFLAG64             = $($(MACH64)_SPACEFLAG)

518 #
519 # The Sun Studio 11 compiler has changed the behaviour of integer
520 # wrap arounds and so a flag is needed to use the legacy behaviour
521 # (without this flag panics/hangs could be exposed within the source).
```

```
522 #
523 sparc_IROPTFLAG         = -W2,-xwrap_int
524 sparcv9_IROPTFLAG       = -W2,-xwrap_int
525 i386_IROPTFLAG          =
526 amd64_IROPTFLAG         =

528 IROPTFLAG               = $($(MACH)_IROPTFLAG)
529 IROPTFLAG64             = $($(MACH64)_IROPTFLAG)

531 sparc_XREGSFLAG         = -xregs=no%appl
532 sparcv9_XREGSFLAG       = -xregs=no%appl
533 i386_XREGSFLAG          =
534 amd64_XREGSFLAG         =

536 XREGSFLAG               = $($(MACH)_XREGSFLAG)
537 XREGSFLAG64             = $($(MACH64)_XREGSFLAG)

539 # dmake SOURCEDEBUG=yes ... enables source-level debugging information, and
540 # avoids stripping it.
541 SOURCEDEBUG     = $(POUND_SIGN)
542 SRCDBGBLD       = $(SOURCEDEBUG:yes=)

544 #
545 # These variables are intended ONLY for use by developers to safely pass extra
546 # flags to the compilers without unintentionally overriding Makefile-set
547 # flags.  They should NEVER be set to any value in a Makefile.
548 #
549 # They come last in the associated FLAGS variable such that they can
550 # explicitly override things if necessary, there are gaps in this, but it's
551 # the best we can manage.
552 #
553 CUSERFLAGS              =
554 CUSERFLAGS64            = $(CUSERFLAGS)
555 CCUSERFLAGS             =
556 CCUSERFLAGS64           = $(CCUSERFLAGS)

558 CSOURCEDEBUGFLAGS       =
559 CCSOURCEDEBUGFLAGS      =
560 $(SRCDBGBLD)CSOURCEDEBUGFLAGS   = -g -xs
561 $(SRCDBGBLD)CCSOURCEDEBUGFLAGS  = -g -xs

563 CFLAGS=         $(COPTFLAG) $($(MACH)_CFLAGS) $(SPACEFLAG) $(CCMODE) \
564                 $(ILDOFF) $(CERRWARN) $(CSTD) $(CCUNBOUND) $(IROPTFLAG) \
565                 $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CSOURCEDEBUGFLAGS) \
566                 $(CUSERFLAGS)
567 CFLAGS64=       $(COPTFLAG64) $($(MACH64)_CFLAGS) $(SPACEFLAG64) $(CCMODE64) \
568                 $(ILDOFF) $(CERRWARN) $(CSTD) $(CCUNBOUND) $(IROPTFLAG64) \
569                 $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CSOURCEDEBUGFLAGS) \
570                 $(CUSERFLAGS64)
571 #
572 # Flags that are used to build parts of the code that are subsequently
573 # run on the build machine (also known as the NATIVE_BUILD).
574 #
575 NATIVE_CFLAGS=  $(COPTFLAG) $($(NATIVE_MACH)_CFLAGS) $(CCMODE) \
576                 $(ILDOFF) $(CERRWARN) $(CSTD) $($(NATIVE_MACH)_CCUNBOUND) \
577                 $(IROPTFLAG) $(CGLOBALSTATIC) $(CCNOAUTOINLINE) \
578                 $(CSOURCEDEBUGFLAGS) $(CUSERFLAGS)

580 DTEXTDOM=-DTEXT_DOMAIN=\"$(TEXT_DOMAIN)\"       # For messaging.
581 DTS_ERRNO=-D_TS_ERRNO
582 CPPFLAGS.first= # Please keep empty.  Only lower makefiles should set this.
583 CPPFLAGS.master=$(DTEXTDOM) $(DTS_ERRNO) \
584         $(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) $(ENVCPPFLAGS4) \
585         $(ADJUNCT_PROTO:%=-I%/usr/include)
586 CPPFLAGS.native=$(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) \
587                 $(ENVCPPFLAGS4) -I$(NATIVE_ADJUNCT)/include
```



```
588 CPPFLAGS=       $(CPPFLAGS.first) $(CPPFLAGS.master)
589 AS_CPPFLAGS=    $(CPPFLAGS.first) $(CPPFLAGS.master)
590 JAVAFLAGS=      -source 1.6 -target 1.6 -Xlint:deprecation,-options

592 #
593 # For source message catalogue
594 #
595 .SUFFIXES: $(SUFFIXES) .i .po
596 MSGROOT= $(ROOT)/catalog
597 MSGDOMAIN= $(MSGROOT)/$(TEXT_DOMAIN)
598 MSGDOMAINPOFILE = $(MSGDOMAIN)/$(POFILE)
599 DCMSGDOMAIN= $(MSGROOT)/LC_TIME/$(TEXT_DOMAIN)
600 DCMSGDOMAINPOFILE = $(DCMSGDOMAIN)/$(DCFILE:.dc=.po)

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

614 #
615 # This is overwritten by local Makefile when PROG is a list.
616 #
617 POFILE= $(PROG).po

619 sparc_CCFLAGS=          -cg92 -compat=4 \
620                         -Qoption ccfe -messages=no%anachronism \
621                         $(CCERRWARN)
622 sparcv9_CCFLAGS=        $(sparcv9_XARCH) -dalign -compat=5 \
623                         -Qoption ccfe -messages=no%anachronism \
624                         -Qoption ccfe -features=no%conststrings \
625                         $(CCCREGSYM) \
626                         $(CCERRWARN)
627 i386_CCFLAGS=           -compat=4 \
628                         -Qoption ccfe -messages=no%anachronism \
629                         -Qoption ccfe -features=no%conststrings \
630                         $(CCERRWARN)
631 amd64_CCFLAGS=          $(amd64_XARCH) -compat=5 \
632                         -Qoption ccfe -messages=no%anachronism \
633                         -Qoption ccfe -features=no%conststrings \
634                         $(CCERRWARN)

636 sparc_CCOPTFLAG=        -O
637 sparcv9_CCOPTFLAG=      -O
638 i386_CCOPTFLAG=         -O
639 amd64_CCOPTFLAG=        -O

641 CCOPTFLAG=      $($(MACH)_CCOPTFLAG)
642 CCOPTFLAG64=    $($(MACH64)_CCOPTFLAG)
643 CCFLAGS=        $(CCOPTFLAG) $($(MACH)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
644                 $(CCUSERFLAGS)
645 CCFLAGS64=      $(CCOPTFLAG64) $($(MACH64)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
646                 $(CCUSERFLAGS64)

648 #
649 #
650 #
651 ELFWRAP_FLAGS   =
652 ELFWRAP_FLAGS64 =       -64
```

```
654 #
655 # Various mapfiles that are used throughout the build, and delivered to
656 # /usr/lib/ld.
657 #
658 MAPFILE.NED_i386 =      $(SRC)/common/mapfiles/common/map.noexdata
659 MAPFILE.NED_sparc =
660 MAPFILE.NED =           $(MAPFILE.NED_$(MACH))
661 MAPFILE.PGA =           $(SRC)/common/mapfiles/common/map.pagealign
662 MAPFILE.NES =           $(SRC)/common/mapfiles/common/map.noexstk
663 MAPFILE.FLT =           $(SRC)/common/mapfiles/common/map.filter
664 MAPFILE.LEX =           $(SRC)/common/mapfiles/common/map.lex.yy

666 #
667 # Generated mapfiles that are compiler specific, and used throughout the
668 # build.  These mapfiles are not delivered in /usr/lib/ld.
669 #
670 MAPFILE.NGB_sparc=      $(SRC)/common/mapfiles/gen/sparc_cc_map.noexeglobs
671 $(__GNUC64)MAPFILE.NGB_sparc= \
672                         $(SRC)/common/mapfiles/gen/sparc_gcc_map.noexeglobs
673 MAPFILE.NGB_sparcv9=    $(SRC)/common/mapfiles/gen/sparcv9_cc_map.noexeglobs
674 $(__GNUC64)MAPFILE.NGB_sparcv9= \
675                         $(SRC)/common/mapfiles/gen/sparcv9_gcc_map.noexeglobs
676 MAPFILE.NGB_i386=       $(SRC)/common/mapfiles/gen/i386_cc_map.noexeglobs
677 $(__GNUC64)MAPFILE.NGB_i386= \
678                         $(SRC)/common/mapfiles/gen/i386_gcc_map.noexeglobs
679 MAPFILE.NGB_amd64=      $(SRC)/common/mapfiles/gen/amd64_cc_map.noexeglobs
680 $(__GNUC64)MAPFILE.NGB_amd64= \
681                         $(SRC)/common/mapfiles/gen/amd64_gcc_map.noexeglobs
682 MAPFILE.NGB =           $(MAPFILE.NGB_$(MACH))

684 #
685 # A generic interface mapfile name, used by various dynamic objects to define
686 # the interfaces and interposers the object must export.
687 #
688 MAPFILE.INT =           mapfile-intf

690 #
691 # LDLIBS32 and LDLIBS64 can be set in the environment to override the following
692 # assignments.
693 #
694 # These environment settings make sure that no libraries are searched outside
695 # of the local workspace proto area:
696 #       LDLIBS32=-YP,$ROOT/lib:$ROOT/usr/lib
697 #       LDLIBS64=-YP,$ROOT/lib/$MACH64:$ROOT/usr/lib/$MACH64
698 #
699 LDLIBS32 =      $(ENVLDLIBS1) $(ENVLDLIBS2) $(ENVLDLIBS3)
700 LDLIBS32 +=     $(ADJUNCT_PROTO:%=-L%/usr/lib -L%/lib)
701 LDLIBS.cmd =    $(LDLIBS32)
702 LDLIBS.lib =    $(LDLIBS32)

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

709 #
710 # Define compilation macros.
711 #
712 COMPILE.c=      $(CC) $(CFLAGS) $(CPPFLAGS) -c
713 COMPILE64.c=    $(CC) $(CFLAGS64) $(CPPFLAGS) -c
714 COMPILE.cc=     $(CCC) $(CCFLAGS) $(CPPFLAGS) -c
715 COMPILE64.cc=   $(CCC) $(CCFLAGS64) $(CPPFLAGS) -c
716 COMPILE.s=      $(AS) $(ASFLAGS) $(AS_CPPFLAGS)
717 COMPILE64.s=    $(AS) $(ASFLAGS) $($(MACH64)_AS_XARCH) $(AS_CPPFLAGS)
718 COMPILE.d=      $(DTRACE) -G -32
719 COMPILE64.d=    $(DTRACE) -G -64
```



```
720 COMPILE.b=      $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))
721 COMPILE64.b=    $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))

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

726 #
727 # Link time macros
728 #
729 CCNEEDED                = -lC
730 CCEXTNEEDED             = -lCrun -lCstd
731 $(__GNUC)CCNEEDED       = -L$(GCCLIBDIR) -lstdc++ -lgcc_s
732 $(__GNUC)CCEXTNEEDED    = $(CCNEEDED)

734 LINK.c=         $(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS)
735 LINK64.c=       $(CC) $(CFLAGS64) $(CPPFLAGS) $(LDFLAGS)
736 NORUNPATH=      -norunpath -nolib
737 LINK.cc=        $(CCC) $(CCFLAGS) $(CPPFLAGS) $(NORUNPATH) \
738                 $(LDFLAGS) $(CCNEEDED)
739 LINK64.cc=      $(CCC) $(CCFLAGS64) $(CPPFLAGS) $(NORUNPATH) \
740                 $(LDFLAGS) $(CCNEEDED)

742 #
743 # lint macros
744 #
745 # Note that the undefine of __PRAGMA_REDEFINE_EXTNAME can be removed once
746 # ON is built with a version of lint that has the fix for 4484186.
747 #
748 ALWAYS_LINT_DEFS =      -errtags=yes -s
749 ALWAYS_LINT_DEFS +=     -erroff=E_PTRDIFF_OVERFLOW
750 ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_NARROW_CONV
751 ALWAYS_LINT_DEFS +=     -U__PRAGMA_REDEFINE_EXTNAME
752 ALWAYS_LINT_DEFS +=     $(C99LMODE)
753 ALWAYS_LINT_DEFS +=     -errsecurity=$(SECLEVEL)
754 ALWAYS_LINT_DEFS +=     -erroff=E_SEC_CREAT_WITHOUT_EXCL
755 ALWAYS_LINT_DEFS +=     -erroff=E_SEC_FORBIDDEN_WARN_CREAT
756 # XX64 -- really only needed for amd64 lint
757 ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_INT_TO_SMALL_INT
758 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_CONST_TO_SMALL_INT
759 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_TO_SMALL_INT
760 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_TO_PTR_FROM_INT
761 ALWAYS_LINT_DEFS +=     -erroff=E_COMP_INT_WITH_LARGE_INT
762 ALWAYS_LINT_DEFS +=     -erroff=E_INTEGRAL_CONST_EXP_EXPECTED
763 ALWAYS_LINT_DEFS +=     -erroff=E_PASS_INT_TO_SMALL_INT
764 ALWAYS_LINT_DEFS +=     -erroff=E_PTR_CONV_LOSES_BITS

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

770 SECLEVEL=       core
771 LINT.c=         $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS) $(CPPFLAGS) \
772                 $(ALWAYS_LINT_DEFS)
773 LINT64.c=       $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS64) $(CPPFLAGS) \
774                 $(ALWAYS_LINT_DEFS)
775 LINT.s=         $(LINT.c)

777 # For some future builds, NATIVE_MACH and MACH might be different.
778 # Therefore, NATIVE_MACH needs to be redefined in the
779 # environment as ‘uname -p‘ to override this macro.
780 #
781 # For now at least, we cross-compile amd64 on i386 machines.
782 NATIVE_MACH=    $(MACH:amd64=i386)

784 # Define native compilation macros
785 #
```

```
787 # Base directory where compilers are loaded.
788 # Defined here so it can be overridden by developer.
789 #
790 SPRO_ROOT=              $(BUILD_TOOLS)/SUNWspro
791 SPRO_VROOT=             $(SPRO_ROOT)/SS12
792 GNU_ROOT=               /usr

794 $(__GNUC)PRIMARY_CC=    gcc4,$(GNUC_ROOT)/bin/gcc,gnu
795 $(__SUNC)PRIMARY_CC=    studio12,$(SPRO_VROOT)/bin/cc,sun
796 $(__GNUC)PRIMARY_CCC=   gcc4,$(GNUC_ROOT)/bin/g++,gnu
797 $(__SUNC)PRIMARY_CCC=   studio12,$(SPRO_VROOT)/bin/CC,sun

799 CW_CC_COMPILERS=        $(PRIMARY_CC:%=--primary %) $(SHADOW_CCS:%=--shadow %)
800 CW_CCC_COMPILERS=       $(PRIMARY_CCC:%=--primary %) $(SHADOW_CCCS:%=--shadow %)


803 # Till SS12u1 formally becomes the NV CBE, LINT is hard
804 # coded to be picked up from the $SPRO_ROOT/sunstudio12.1/
805 # location. Impacted variables are sparc_LINT, sparcv9_LINT,
806 # i386_LINT, amd64_LINT.
807 # Reset them when SS12u1 is rolled out.
808 #

810 # Specify platform compiler versions for languages
811 # that we use (currently only c and c++).
812 #
813 CW=                     $(ONBLD_TOOLS)/bin/$(MACH)/cw

815 BUILD_CC=               $(CW) $(CW_CC_COMPILERS) --
816 BUILD_CCC=              $(CW) -C $(CW_CCC_COMPILERS) --
817 BUILD_CPP=              /usr/ccs/lib/cpp
818 BUILD_LD=               /usr/ccs/bin/ld
819 BUILD_LINT=             $(SPRO_ROOT)/sunstudio12.1/bin/lint

821 $(MACH)_CC=             $(BUILD_CC)
822 $(MACH)_CCC=            $(BUILD_CCC)
823 $(MACH)_CPP=            $(BUILD_CPP)
824 $(MACH)_LD=             $(BUILD_LD)
825 $(MACH)_LINT=           $(BUILD_LINT)
826 $(MACH64)_CC=           $(BUILD_CC)
827 $(MACH64)_CCC=          $(BUILD_CCC)
828 $(MACH64)_CPP=          $(BUILD_CPP)
829 $(MACH64)_LD=           $(BUILD_LD)
830 $(MACH64)_LINT=         $(BUILD_LINT)

832 sparc_AS=               /usr/ccs/bin/as -xregsym=no
833 sparcv9_AS=             $($(MACH)_AS)

835 i386_AS=                /usr/ccs/bin/as
836 $(__GNUC)i386_AS=       $(ONBLD_TOOLS)/bin/$(MACH)/aw
837 amd64_AS=               $(ONBLD_TOOLS)/bin/$(MACH)/aw

839 NATIVECC=              $($(NATIVE_MACH)_CC)
840 NATIVECCC=             $($(NATIVE_MACH)_CCC)
841 NATIVECPP=             $($(NATIVE_MACH)_CPP)
842 NATIVEAS=              $($(NATIVE_MACH)_AS)
843 NATIVELD=              $($(NATIVE_MACH)_LD)
844 NATIVELINT=            $($(NATIVE_MACH)_LINT)

846 #
847 # Makefile.master.64 overrides these settings
848 #
849 CC=                     $(NATIVECC)
850 CCC=                    $(NATIVECCC)
851 CPP=                    $(NATIVECPP)
```



```
852 AS=                     $(NATIVEAS)
853 LD=                     $(NATIVELD)
854 LINT=                   $(NATIVELINT)

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

859 BDIRECT=        -Bdirect
860 BDYNAMIC=       -Bdynamic
861 BLOCAL=         -Blocal
862 BNODIRECT=      -Bnodirect
863 BREDUCE=        -Breduce
864 BSTATIC=        -Bstatic

866 ZDEFS=          -zdefs
867 ZDIRECT=        -zdirect
868 ZIGNORE=        -zignore
869 ZINITFIRST=     -zinitfirst
870 ZINTERPOSE=     -zinterpose
871 ZLAZYLOAD=      -zlazyload
872 ZLOADFLTR=      -zloadfltr
873 ZMULDEFS=       -zmuldefs
874 ZNODEFAULTLIB=  -znodefaultlib
875 ZNODEFS=        -znodefs
876 ZNODELETE=      -znodelete
877 ZNODLOPEN=      -znodlopen
878 ZNODUMP=        -znodump
879 ZNOLAZYLOAD=    -znolazyload
880 ZNOLDYNSYM=     -znoldynsym
881 ZNORELOC=       -znoreloc
882 ZNOVERSION=     -znoversion
883 ZRECORD=        -zrecord
884 ZREDLOCSYM=     -zredlocsym
885 ZTEXT=          -ztext
886 ZVERBOSE=       -zverbose

888 GSHARED=        -G
889 CCMT=           -mt

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

894 sparc_C_PICFLAGS =      -K pic
895 sparcv9_C_PICFLAGS =    -K pic
896 i386_C_PICFLAGS =       -K pic
897 amd64_C_PICFLAGS =      -K pic
898 C_PICFLAGS =            $($(MACH)_C_PICFLAGS)
899 C_PICFLAGS64 =          $($(MACH64)_C_PICFLAGS)

901 sparc_C_BIGPICFLAGS =   -K PIC
902 sparcv9_C_BIGPICFLAGS = -K PIC
903 i386_C_BIGPICFLAGS =    -K PIC
904 amd64_C_BIGPICFLAGS =   -K PIC
905 C_BIGPICFLAGS =         $($(MACH)_C_BIGPICFLAGS)
906 C_BIGPICFLAGS64 =       $($(MACH64)_C_BIGPICFLAGS)

908 # CC requires there to be no space between '-K' and 'pic' or 'PIC'.
909 sparc_CC_PICFLAGS =     -Kpic
910 sparcv9_CC_PICFLAGS =   -KPIC
911 i386_CC_PICFLAGS =      -Kpic
912 amd64_CC_PICFLAGS =     -Kpic
913 CC_PICFLAGS =           $($(MACH)_CC_PICFLAGS)
914 CC_PICFLAGS64 =         $($(MACH64)_CC_PICFLAGS)

916 AS_PICFLAGS=            $(C_PICFLAGS)
917 AS_BIGPICFLAGS=         $(C_BIGPICFLAGS)
```

```
#
# Default label for CTF sections
#
CTFCVTFLAGS=            -i -L VERSION

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

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

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



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

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

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

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

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

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

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

```
1050         $(COMPILE.cc) -o $% $<
1051         $(AR) $(ARFLAGS) $@ $%
1052         $(PROCESS_COMMENT) $%
1053         $(RM) $%
1054 .y:
1055         $(YACC.y) $<
1056         $(LINK.c) -o $@ y.tab.c $(LDLIBS)
1057         $(POST_PROCESS)
1058         $(RM) y.tab.c
1059 .y.o:
1060         $(YACC.y) $<
1061         $(COMPILE.c) -o $@ y.tab.c $(CTFCONVERT_HOOK)
1062         $(POST_PROCESS_O)
1063         $(RM) y.tab.c
1064 .l:
1065         $(RM) $*.c
1066         $(LEX.l) $< > $*.c
1067         $(LINK.c) -o $@ $*.c -ll $(LDLIBS)
1068         $(POST_PROCESS)
1069         $(RM) $*.c
1070 .l.o:
1071         $(RM) $*.c
1072         $(LEX.l) $< > $*.c
1073         $(COMPILE.c) -o $@ $*.c $(CTFCONVERT_HOOK)
1074         $(POST_PROCESS_O)
1075         $(RM) $*.c

1077 .bin.o:
1078         $(COMPILE.b) -o $@ $<
1079         $(POST_PROCESS_O)

1081 .java.class:
1082         $(COMPILE.java) $<

1084 # Bourne and Korn shell script message catalog build rules.
1085 # We extract all gettext strings with sed(1) (being careful to permit
1086 # multiple gettext strings on the same line), weed out the dups, and
1087 # build the catalogue with awk(1).

1089 .sh.po .ksh.po:
1090         $(SED) -n -e ":a"                                       \
1091                   -e "h"                                        \
1092                   -e "s/.*gettext *\(\"[^\"]*\"\).*/\1/p"       \
1093                   -e "x"                                        \
1094                   -e "s/\(.*\)gettext *\"[^\"]*\"\(.*\)/\1\2/"  \
1095                   -e "t a"                                      \
1096                $< | sort -u | $(AWK) '{ print "msgid\t" $$0 "\nmsgstr" }' > $@

1098 #
1099 # Python and Perl executable and message catalog build rules.
1100 #
1101 .SUFFIXES: .pl .pm .py .pyc

1103 .pl:
1104         $(RM) $@;
1105         $(SED) -e "s@TEXT_DOMAIN@\"$(TEXT_DOMAIN)\"@" $< > $@;
1106         $(CHMOD) +x $@

1108 .py:
1109         $(RM) $@; $(SED) -e "1s:^\#!@PYTHON@:\#!$(PYTHON):" < $< > $@; $(CHMOD)

1111 .py.pyc:
1112         $(RM) $@
1113         $(PYTHON) -mpy_compile $<
1114         @[ $(<)c = $@ ] || $(MV) $(<)c $@
```



```
1116 .py.po:
1117         $(GNUXGETTEXT) $(GNUXGETFLAGS) -d $(<F:%.py=%) $< ;

1119 .pl.po .pm.po:
1120         $(XGETTEXT) $(XGETFLAGS) -d $(<F) $< ;
1121         $(RM)   $@ ;
1122         $(SED) "/^domain/d" < $(<F).po > $@ ;
1123         $(RM) $(<F).po

1125 #
1126 # When using xgettext, we want messages to go to the default domain,
1127 # rather than the specified one.  This special version of the
1128 # COMPILE.cpp macro effectively prevents expansion of TEXT_DOMAIN,
1129 # causing xgettext to put all messages into the default domain.
1130 #
1131 CPPFORPO=$(COMPILE.cpp:\"$(TEXT_DOMAIN)\"=TEXT_DOMAIN)

1133 .c.i:
1134         $(CPPFORPO) $< > $@

1136 .h.i:
1137         $(CPPFORPO) $< > $@

1139 .y.i:
1140         $(YACC) -d $<
1141         $(CPPFORPO) y.tab.c  > $@
1142         $(RM) y.tab.c

1144 .l.i:
1145         $(LEX) $<
1146         $(CPPFORPO) lex.yy.c  > $@
1147         $(RM) lex.yy.c

1149 .c.po:
1150         $(CPPFORPO) $< > $<.i
1151         $(BUILD.po)

1153 .cc.po:
1154         $(CPPFORPO) $< > $<.i
1155         $(BUILD.po)

1157 .y.po:
1158         $(YACC) -d $<
1159         $(CPPFORPO) y.tab.c  > $<.i
1160         $(BUILD.po)
1161         $(RM) y.tab.c

1163 .l.po:
1164         $(LEX) $<
1165         $(CPPFORPO) lex.yy.c  > $<.i
1166         $(BUILD.po)
1167         $(RM) lex.yy.c

1169 #
1170 # Rules to perform stylistic checks
1171 #
1172 .SUFFIXES: .x .xml .check .xmlchk

1174 .h.check:
1175         $(DOT_H_CHECK)

1177 .x.check:
1178         $(DOT_X_CHECK)

1180 .xml.xmlchk:
1181         $(MANIFEST_CHECK)
```

```
1183 #
1184 # Include rules to render automated sccs get rules "safe".
1185 #
1186 include $(SRC)/Makefile.noget
```