```
**********************************************************
   38277 Tue Jun 11 04:01:18 2019
new/usr/src/Makefile.master
9996 use GCC 7 as default 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.
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
# Copyright 2015, OmniTI Computer Consulting, Inc. All rights reserved.
# Copyright 2015 Gary Mills
# Copyright 2015 Igor Kozhukhov <ikozhukhov@gmail.com>
# Copyright 2016 Toomas Soome <tsoome@me.com>
# Copyright 2018 OmniOS Community Edition (OmniOSce) Association.
# Copyright 2019, Joyent, Inc.
# Copyright (c) 2019, Joyent, Inc.
#

#
# 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.
# __GNUC and GCC 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)
__GNUC64=               $(__GNUC)

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

# 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

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

```
127 BLD_JAVA_8=     $(POUND_SIGN)

129 GNUC_ROOT=      /usr/gcc/7
129 GNUC_ROOT=      /opt/gcc/4.4.4
130 GCCLIBDIR=      $(GNUC_ROOT)/lib
131 GCCLIBDIR64=    $(GNUC_ROOT)/lib/$(MACH64)

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

135 RPCGEN=         /usr/bin/rpcgen
136 STABS=          $(ONBLD_TOOLS)/bin/$(MACH)/stabs
137 ELFEXTRACT=     $(ONBLD_TOOLS)/bin/$(MACH)/elfextract
138 MBH_PATCH=      $(ONBLD_TOOLS)/bin/$(MACH)/mbh_patch
139 BTXLD=          $(ONBLD_TOOLS)/bin/$(MACH)/btxld
140 VTFONTCVT=      $(ONBLD_TOOLS)/bin/$(MACH)/vtfontcvt
141 # echo(1) and true(1) are specified without absolute paths, so that the shell
142 # spawned by make(1) may use the built-in versions.  This is minimally
143 # problematic, as the shell spawned by make(1) is known and under control, the
144 # only risk being if the shell falls back to $PATH.
145 #
146 # We specifically want an echo(1) that does interpolation of escape sequences,
147 # which ksh93, /bin/sh, and bash will all provide.
148 ECHO=           echo
149 TRUE=           true
150 INS=            $(ONBLD_TOOLS)/bin/$(MACH)/install
151 SYMLINK=        /usr/bin/ln -s
152 LN=             /usr/bin/ln
153 MKDIR=          /usr/bin/mkdir
154 CHMOD=          /usr/bin/chmod
155 MV=             /usr/bin/mv -f
156 RM=             /usr/bin/rm -f
157 CUT=            /usr/bin/cut
158 NM=             /usr/ccs/bin/nm
159 DIFF=           /usr/bin/diff
160 GREP=           /usr/bin/grep
161 EGREP=          /usr/bin/egrep
162 ELFWRAP=        /usr/bin/elfwrap
163 KSH93=          /usr/bin/ksh93
164 SED=            /usr/bin/sed
165 AWK=            /usr/bin/nawk
166 CP=             /usr/bin/cp -f
167 MCS=            /usr/ccs/bin/mcs
168 CAT=            /usr/bin/cat
169 ELFDUMP=        /usr/ccs/bin/elfdump
170 M4=             /usr/bin/m4
171 GM4=            /usr/bin/gm4
172 STRIP=          /usr/ccs/bin/strip
173 LEX=            /usr/ccs/bin/lex
174 FLEX=           /usr/bin/flex
175 YACC=           /usr/ccs/bin/yacc
176 BISON=          /usr/bin/bison
177 CPP=            /usr/lib/cpp
178 ANSI_CPP=       $(GNUC_ROOT)/bin/cpp
179 JAVAC=          $(JAVA_ROOT)/bin/javac
180 JAVAH=          $(JAVA_ROOT)/bin/javah
181 JAVADOC=        $(JAVA_ROOT)/bin/javadoc
182 RMIC=           $(JAVA_ROOT)/bin/rmic
183 JAR=            $(JAVA_ROOT)/bin/jar
184 CTFCONVERT=     $(ONBLD_TOOLS)/bin/$(MACH)/ctfconvert
185 CTFDIFF=        $(ONBLD_TOOLS)/bin/$(MACH)/ctfdiff
186 CTFMERGE=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfmerge
187 CTFSTABS=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfstabs
188 CTFSTRIP=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfstrip
189 NDRGEN=         $(ONBLD_TOOLS)/bin/$(MACH)/ndrgen
190 GENOFFSETS=     $(ONBLD_TOOLS)/bin/genoffsets
191 XREF=           $(ONBLD_TOOLS)/bin/xref
```



```
192 FIND=           /usr/bin/find
193 PERL=           /usr/bin/perl
194 PERL_VERSION=   5.10.0
195 PERL_PKGVERS=   -510
196 PERL_ARCH =             i86pc-solaris-64int
197 $(SPARC_BLD)PERL_ARCH = sun4-solaris-64int
198 PYTHON_VERSION= 2.7
199 PYTHON_PKGVERS= -27
200 PYTHON_SUFFIX=
201 PYTHON=         /usr/bin/python$(PYTHON_VERSION)
202 PYTHON3_VERSION=        3.5
203 PYTHON3_PKGVERS=        -35
204 PYTHON3_SUFFIX=         m
205 PYTHON3=                /usr/bin/python$(PYTHON3_VERSION)
206 $(BUILDPY3TOOLS)TOOLS_PYTHON=           $(PYTHON3)
207 $(BUILDPY2TOOLS)TOOLS_PYTHON=           $(PYTHON)
208 SORT=           /usr/bin/sort
209 TR=             /usr/bin/tr
210 TOUCH=          /usr/bin/touch
211 WC=             /usr/bin/wc
212 XARGS=          /usr/bin/xargs
213 ELFEDIT=        /usr/bin/elfedit
214 DTRACE=         /usr/sbin/dtrace -xnolibs
215 UNIQ=           /usr/bin/uniq
216 TAR=            /usr/bin/tar
217 ASTBINDIR=      /usr/ast/bin
218 MSGCC=          $(ASTBINDIR)/msgcc
219 MSGFMT=         /usr/bin/msgfmt -s
220 LCDEF=          $(ONBLD_TOOLS)/bin/$(MACH)/localedef
221 TIC=            $(ONBLD_TOOLS)/bin/$(MACH)/tic
222 ZIC=            $(ONBLD_TOOLS)/bin/$(MACH)/zic
223 OPENSSL=        /usr/bin/openssl
224 CPCGEN=         $(ONBLD_TOOLS)/bin/$(MACH)/cpcgen

226 DEFAULT_CONSOLE_COLOR=                                  \
227         -DDEFAULT_ANSI_FOREGROUND=ANSI_COLOR_WHITE      \
228         -DDEFAULT_ANSI_BACKGROUND=ANSI_COLOR_BLACK

230 FILEMODE=       644
231 DIRMODE=        755

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

237 # For stylistic checks
238 #
239 # Note that the X and C checks are not used at this time and may need
240 # modification when they are actually used.
241 #
242 CSTYLE=         $(ONBLD_TOOLS)/bin/cstyle
243 CSTYLE_TAIL=
244 HDRCHK=         $(ONBLD_TOOLS)/bin/hdrchk
245 HDRCHK_TAIL=
246 JSTYLE=         $(ONBLD_TOOLS)/bin/jstyle

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

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

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

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

266 INS.file=       $(RM) $@; $(INS) -s -m $(FILEMODE) -f $(@D) $<
267 INS.dir=        $(INS) -s -d -m $(DIRMODE) $@
268 # installs and renames at once
269 #
270 INS.rename=     $(INS.file); $(MV) $(@D)/$(<F) $@

272 # install a link
273 INSLINKTARGET=  $<
274 INS.link=       $(RM) $@; $(LN) $(INSLINKTARGET) $@
275 INS.symlink=    $(RM) $@; $(SYMLINK) $(INSLINKTARGET) $@

277 # The path to python that will be used for the shebang line when installing
278 # python scripts to the proto area. This is overridden by makefiles to
279 # select to the correct version.
280 PYSHEBANG=      $(PYTHON)

282 #
283 # Python bakes the mtime of the .py file into the compiled .pyc and
284 # rebuilds if the baked-in mtime != the mtime of the source file
285 # (rather than only if it's less than), thus when installing python
286 # files we must make certain to not adjust the mtime of the source
287 # (.py) file.
288 #
289 INS.pyfile=     $(RM) $@; $(SED) \
290                 -e "1s:^\#!@PYTHON@:\#!$(PYSHEBANG):" \
291                 -e "1s:^\#!@TOOLS_PYTHON@:\#!$(TOOLS_PYTHON):" \
292                 < $< > $@; $(CHMOD) $(FILEMODE) $@; $(TOUCH) -r $< $@

294 # MACH must be set in the shell environment per uname -p on the build host
295 # More specific architecture variables should be set in lower makefiles.
296 #
297 # MACH64 is derived from MACH, and BUILD64 is set to '#' for
298 # architectures on which we do not build 64-bit versions.
299 # (There are no such architectures at the moment.)
300 #
301 # Set BUILD64=# in the environment to disable 64-bit amd64
302 # builds on i386 machines.

304 MACH64_1=       $(MACH:sparc=sparcv9)
305 MACH64=         $(MACH64_1:i386=amd64)

307 MACH32_1=       $(MACH:sparc=sparcv7)
308 MACH32=         $(MACH32_1:i386=i86)

310 sparc_BUILD64=
311 i386_BUILD64=
312 BUILD64=        $($(MACH)_BUILD64)

314 #
315 # C compiler mode. Future compilers may change the default on us,
316 # so force extended ANSI mode globally. Lower level makefiles can
317 # override this by setting CCMODE.
318 #
319 CCMODE=                 -Xa
320 CCMODE64=               -Xa

322 #
323 # C compiler verbose mode. This is so we can enable it globally,
```



```
324 # but turn it off in the lower level makefiles of things we cannot
325 # (or aren't going to) fix.
326 #
327 CCVERBOSE=              -v

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

333 # set this to the secret flag "-Wc,-Qiselect-regsym=0" to disable register
334 # symbols (used to detect conflicts between objects that use global registers)
335 # we disable this now for safety, and because genunix doesn't link with
336 # this feature (the v9 default) enabled.
337 #
338 # REGSYM is separate since the C++ driver syntax is different.
339 CCREGSYM=               -Wc,-Qiselect-regsym=0
340 CCCREGSYM=              -Qoption cg -Qiselect-regsym=0

342 # Prevent the removal of static symbols by the SPARC code generator (cg).
343 # The x86 code generator (ube) does not remove such symbols and as such
344 # using this workaround is not applicable for x86.
345 #
346 CCSTATICSYM=            -Wc,-Qassembler-ounrefsym=0
347 #
348 # generate 32-bit addresses in the v9 kernel. Saves memory.
349 CCABS32=                -Wc,-xcode=abs32
350 #
351 # generate v9 code which tolerates callers using the v7 ABI, for the sake of
352 # system calls.
353 CC32BITCALLERS=         -_gcc=-massume-32bit-callers

355 # GCC, especially, is increasingly beginning to auto-inline functions and
356 # sadly does so separately not under the general -fno-inline-functions
357 # Additionally, we wish to prevent optimisations which cause GCC to clone
358 # functions -- in particular, these may cause unhelpful symbols to be
359 # emitted instead of function names
360 CCNOAUTOINLINE= \
361         -_gcc=-fno-inline-small-functions \
362         -_gcc=-fno-inline-functions-called-once \
363         -_gcc=-fno-ipa-cp \
364         -_gcc7=-fno-ipa-icf \
365         -_gcc8=-fno-ipa-icf \
366         -_gcc7=-fno-clone-functions \
367         -_gcc8=-fno-clone-functions

369 # GCC may put functions in different named sub-sections of .text based on
370 # their presumed calling frequency.  At least in the kernel, where we actually
371 # deliver relocatable objects, we don't want this to happen.
372 #
373 # Since at present we don't benefit from this even in userland, we disable it gl
374 # but the application of this may move into usr/src/uts/ in future.
375 CCNOREORDER=    \
376         -_gcc7=-fno-reorder-functions \
377         -_gcc8=-fno-reorder-functions

379 #
380 # gcc has a rather aggressive optimization on by default that infers loop
381 # bounds based on undefined behavior (!!).  This can lead to some VERY
382 # surprising optimizations -- ones that may be technically correct in the
383 # strictest sense but also result in incorrect program behavior.  We turn
384 # this optimization off, with extreme prejudice.
385 #
386 CCNOAGGRESSIVELOOPS= \
387         -_gcc7=-fno-aggressive-loop-optimizations \
388         -_gcc8=-fno-aggressive-loop-optimizations
```

```
390 # One optimization the compiler might perform is to turn this:
391 #       #pragma weak foo
392 #       extern int foo;
393 #       if (&foo)
394 #               foo = 5;
395 # into
396 #       foo = 5;
397 # Since we do some of this (foo might be referenced in common kernel code
398 # but provided only for some cpu modules or platforms), we disable this
399 # optimization.
400 #
401 sparc_CCUNBOUND = -Wd,-xsafe=unboundsym
402 i386_CCUNBOUND  =
403 CCUNBOUND       = $($(MACH)_CCUNBOUND)

405 #
406 # compiler '-xarch' flag. This is here to centralize it and make it
407 # overridable for testing.
408 sparc_XARCH=    -m32
409 sparcv9_XARCH=  -m64
410 i386_XARCH=     -m32
411 amd64_XARCH=    -m64 -Ui386 -U__i386

413 # assembler '-xarch' flag.  Different from compiler '-xarch' flag.
414 sparc_AS_XARCH=         -xarch=v8plus
415 sparcv9_AS_XARCH=       -xarch=v9
416 i386_AS_XARCH=
417 amd64_AS_XARCH=         -xarch=amd64 -P -Ui386 -U__i386

419 #
420 # These flags define what we need to be 'standalone' i.e. -not- part
421 # of the rather more cosy userland environment.  This basically means
422 # the kernel.
423 #
424 # XX64  future versions of gcc will make -mcmodel=kernel imply -mno-red-zone
425 #
426 sparc_STAND_FLAGS=      -_gcc=-ffreestanding
427 sparcv9_STAND_FLAGS=    -_gcc=-ffreestanding
428 # Disabling MMX also disables 3DNow, disabling SSE also disables all later
429 # additions to SSE (SSE2, AVX ,etc.)
430 NO_SIMD=                -_gcc=-mno-mmx -_gcc=-mno-sse
431 i386_STAND_FLAGS=       -_gcc=-ffreestanding $(NO_SIMD)
432 amd64_STAND_FLAGS=      -xmodel=kernel $(NO_SIMD)

434 SAVEARGS=               -Wu,-save_args
435 amd64_STAND_FLAGS       += $(SAVEARGS)

437 STAND_FLAGS_32 = $($(MACH)_STAND_FLAGS)
438 STAND_FLAGS_64 = $($(MACH64)_STAND_FLAGS)

440 #
441 # disable the incremental linker
442 ILDOFF=                 -xildoff
443 #
444 XFFLAG=                 -xF=%all
445 XESS=                   -xs
446 XSTRCONST=              -xstrconst

448 #
449 # turn warnings into errors (C)
450 CERRWARN = -errtags=yes -errwarn=%all
451 CERRWARN += -erroff=E_EMPTY_TRANSLATION_UNIT
452 CERRWARN += -erroff=E_STATEMENT_NOT_REACHED

454 CERRWARN += -_gcc=-Wno-missing-braces
455 CERRWARN += -_gcc=-Wno-sign-compare
```



```
456 CERRWARN += -_gcc=-Wno-unknown-pragmas
457 CERRWARN += -_gcc=-Wno-unused-parameter
458 CERRWARN += -_gcc=-Wno-missing-field-initializers

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


464 CERRWARN += -_smatch=-p=illumos_user
465 include $(SRC)/Makefile.smatch

467 #
468 # turn warnings into errors (C++)
469 CCERRWARN=              -xwe

471 # C standard.  Keep Studio flags until we get rid of lint.
472 CSTD_GNU89=     -xc99=%none
473 CSTD_GNU99=     -xc99=%all
474 CSTD=           $(CSTD_GNU89)
475 C99LMODE=       $(CSTD:-xc99%=-Xc99%)

477 # In most places, assignments to these macros should be appended with +=
478 # (CPPFLAGS.first allows values to be prepended to CPPFLAGS).
479 sparc_CFLAGS=   $(sparc_XARCH) $(CCSTATICSYM)
480 sparcv9_CFLAGS= $(sparcv9_XARCH) -dalign $(CCVERBOSE) $(V9ABIWARN) $(CCREGSYM) \
481                 $(CCSTATICSYM)
482 i386_CFLAGS=    $(i386_XARCH)
483 amd64_CFLAGS=   $(amd64_XARCH)

485 sparc_ASFLAGS=  $(sparc_AS_XARCH)
486 sparcv9_ASFLAGS=$(sparcv9_AS_XARCH)
487 i386_ASFLAGS=   $(i386_AS_XARCH)
488 amd64_ASFLAGS=  $(amd64_AS_XARCH)

490 #
491 sparc_COPTFLAG=         -xO3
492 sparcv9_COPTFLAG=       -xO3
493 i386_COPTFLAG=          -O
494 amd64_COPTFLAG=         -xO3

496 COPTFLAG= $($(MACH)_COPTFLAG)
497 COPTFLAG64= $($(MACH64)_COPTFLAG)

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

503 # Direct the Sun Studio compiler to use a static globalization prefix based on t
504 # name of the module rather than something unique. Otherwise, objects
505 # will not build deterministically, as subsequent compilations of identical
506 # source will yeild objects that always look different.
507 #
508 # In the same spirit, this will also remove the date from the N_OPT stab.
509 CGLOBALSTATIC= -W0,-xglobalstatic

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

515 #
516 # We force the compilers to generate the debugging information best understood
517 # by the CTF tools.  With Sun Studio this is stabs due to bugs in the Studio
518 # compilers.  With GCC this is DWARF v2.
519 #
520 DEBUGFORMAT=    -_cc=-xdebugformat=stabs -_gcc=-gdwarf-2
```

```
522 #
523 # Ask the compiler to include debugging information
524 #
525 CCGDEBUG= -g $(DEBUGFORMAT)

527 #
528 # Flags used to build in debug mode for ctf generation.
529 #
530 CTF_FLAGS_sparc = $(CCGDEBUG) -Wc,-Qiselect-T1 $(CSTD) $(CNOGLOBAL)
531 CTF_FLAGS_i386  = $(CCGDEBUG) $(CSTD) $(CNOGLOBAL)

533 CTF_FLAGS_sparcv9       = $(CTF_FLAGS_sparc)
534 CTF_FLAGS_amd64         = $(CTF_FLAGS_i386)

536 # Sun Studio produces broken userland code when saving arguments.
537 $(__GNUC)CTF_FLAGS_amd64 += $(SAVEARGS)

539 CTF_FLAGS_32    = $(CTF_FLAGS_$(MACH))
540 CTF_FLAGS_64    = $(CTF_FLAGS_$(MACH64))
541 CTF_FLAGS       = $(CTF_FLAGS_32)

543 #
544 # Flags used with genoffsets
545 #
546 GENOFFSETS_FLAGS = $(CALLSYMS)

548 OFFSETS_CREATE = $(GENOFFSETS) -s $(CTFSTABS) -r $(CTFCONVERT) \
549         $(CW) --noecho $(CW_CC_COMPILERS) -- $(GENOFFSETS_FLAGS) \
550         $(CFLAGS) $(CPPFLAGS)

552 OFFSETS_CREATE64 = $(GENOFFSETS) -s $(CTFSTABS) -r $(CTFCONVERT) \
553         $(CW) --noecho $(CW_CC_COMPILERS) -- $(GENOFFSETS_FLAGS) \
554         $(CFLAGS64) $(CPPFLAGS)

556 #
557 # tradeoff time for space (smaller is better)
558 #
559 sparc_SPACEFLAG         = -xspace -W0,-Lt
560 sparcv9_SPACEFLAG       = -xspace -W0,-Lt
561 i386_SPACEFLAG          = -xspace
562 amd64_SPACEFLAG         =

564 SPACEFLAG               = $($(MACH)_SPACEFLAG)
565 SPACEFLAG64             = $($(MACH64)_SPACEFLAG)

567 #
568 # The Sun Studio 11 compiler has changed the behaviour of integer
569 # wrap arounds and so a flag is needed to use the legacy behaviour
570 # (without this flag panics/hangs could be exposed within the source).
571 #
572 sparc_IROPTFLAG         = -W2,-xwrap_int
573 sparcv9_IROPTFLAG       = -W2,-xwrap_int
574 i386_IROPTFLAG          =
575 amd64_IROPTFLAG         =

577 IROPTFLAG               = $($(MACH)_IROPTFLAG)
578 IROPTFLAG64             = $($(MACH64)_IROPTFLAG)

580 sparc_XREGSFLAG         = -xregs=no%appl
581 sparcv9_XREGSFLAG       = -xregs=no%appl
582 i386_XREGSFLAG          =
583 amd64_XREGSFLAG         =

585 XREGSFLAG               = $($(MACH)_XREGSFLAG)
586 XREGSFLAG64             = $($(MACH64)_XREGSFLAG)
```



```
588 # dmake SOURCEDEBUG=yes ... enables source-level debugging information, and
589 # avoids stripping it.
590 SOURCEDEBUG     = $(POUND_SIGN)
591 SRCDBGBLD       = $(SOURCEDEBUG:yes=)

593 #
594 # These variables are intended ONLY for use by developers to safely pass extra
595 # flags to the compilers without unintentionally overriding Makefile-set
596 # flags.  They should NEVER be set to any value in a Makefile.
597 #
598 # They come last in the associated FLAGS variable such that they can
599 # explicitly override things if necessary, there are gaps in this, but it's
600 # the best we can manage.
601 #
602 CUSERFLAGS              =
603 CUSERFLAGS64            = $(CUSERFLAGS)
604 CCUSERFLAGS             =
605 CCUSERFLAGS64           = $(CCUSERFLAGS)

607 CSOURCEDEBUGFLAGS       =
608 CCSOURCEDEBUGFLAGS      =
609 $(SRCDBGBLD)CSOURCEDEBUGFLAGS   = $(CCGDEBUG) -xs
610 $(SRCDBGBLD)CCSOURCEDEBUGFLAGS  = $(CCGDEBUG) -xs

612 CFLAGS=         $(COPTFLAG) $($(MACH)_CFLAGS) $(SPACEFLAG) $(CCMODE) \
613                 $(ILDOFF) $(CERRWARN) $(CSTD) $(CCUNBOUND) $(IROPTFLAG) \
614                 $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CCNOREORDER) \
615                 $(CCNOAGGRESSIVELOOPS) \
616                 $(CSOURCEDEBUGFLAGS) $(CUSERFLAGS)
617 CFLAGS64=       $(COPTFLAG64) $($(MACH64)_CFLAGS) $(SPACEFLAG64) $(CCMODE64) \
618                 $(ILDOFF) $(CERRWARN) $(CSTD) $(CCUNBOUND) $(IROPTFLAG64) \
619                 $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CCNOREORDER) \
620                 $(CCNOAGGRESSIVELOOPS) \
621                 $(CSOURCEDEBUGFLAGS) $(CUSERFLAGS64)
622 #
623 # Flags that are used to build parts of the code that are subsequently
624 # run on the build machine (also known as the NATIVE_BUILD).
625 #
626 NATIVE_CFLAGS=  $(COPTFLAG) $($(NATIVE_MACH)_CFLAGS) $(CCMODE) \
627                 $(ILDOFF) $(CERRWARN) $(CSTD) $($(NATIVE_MACH)_CCUNBOUND) \
628                 $(IROPTFLAG) $(CGLOBALSTATIC) $(CCNOAUTOINLINE) \
629                 $(CCNOREORDER) $(CSOURCEDEBUGFLAGS) $(CUSERFLAGS)

631 DTEXTDOM=-DTEXT_DOMAIN=\"$(TEXT_DOMAIN)\"       # For messaging.
632 DTS_ERRNO=-D_TS_ERRNO
633 CPPFLAGS.first= # Please keep empty.  Only lower makefiles should set this.
634 CPPFLAGS.master=$(DTEXTDOM) $(DTS_ERRNO) \
635         $(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) $(ENVCPPFLAGS4) \
636         $(ADJUNCT_PROTO:%=-I%/usr/include)
637 CPPFLAGS.native=$(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) \
638                 $(ENVCPPFLAGS4) -I$(NATIVE_ADJUNCT)/include
639 CPPFLAGS=       $(CPPFLAGS.first) $(CPPFLAGS.master)
640 AS_CPPFLAGS=    $(CPPFLAGS.first) $(CPPFLAGS.master)
641 JAVAFLAGS=      -source 1.6 -target 1.6 -Xlint:deprecation,-options

643 #
644 # For source message catalogue
645 #
646 .SUFFIXES: $(SUFFIXES) .i .po
647 MSGROOT= $(ROOT)/catalog
648 MSGDOMAIN= $(MSGROOT)/$(TEXT_DOMAIN)
649 MSGDOMAINPOFILE = $(MSGDOMAIN)/$(POFILE)
650 DCMSGDOMAIN= $(MSGROOT)/LC_TIME/$(TEXT_DOMAIN)
651 DCMSGDOMAINPOFILE = $(DCMSGDOMAIN)/$(DCFILE:.dc=.po)

653 CLOBBERFILES += $(POFILE) $(POFILES)
```

```
654 COMPILE.cpp= $(CC) -E -C $(CFLAGS) $(CPPFLAGS)
655 XGETTEXT= /usr/bin/xgettext
656 XGETFLAGS= -c TRANSLATION_NOTE
657 GNUXGETTEXT= /usr/gnu/bin/xgettext
658 GNUXGETFLAGS= --add-comments=TRANSLATION_NOTE --keyword=_ \
659         --strict --no-location --omit-header
660 BUILD.po= $(XGETTEXT) $(XGETFLAGS) -d $(<F) $<.i ;\
661         $(RM)   $@ ;\
662         $(SED) "/^domain/d" < $(<F).po > $@ ;\
663         $(RM) $(<F).po $<.i

665 #
666 # This is overwritten by local Makefile when PROG is a list.
667 #
668 POFILE= $(PROG).po

670 sparc_CCFLAGS=          -cg92 -compat=4 \
671                         -Qoption ccfe -messages=no%anachronism \
672                         $(CCERRWARN)
673 sparcv9_CCFLAGS=        $(sparcv9_XARCH) -dalign -compat=5 \
674                         -Qoption ccfe -messages=no%anachronism \
675                         -Qoption ccfe -features=no%conststrings \
676                         $(CCCREGSYM) \
677                         $(CCERRWARN)
678 i386_CCFLAGS=           -compat=4 \
679                         -Qoption ccfe -messages=no%anachronism \
680                         -Qoption ccfe -features=no%conststrings \
681                         $(CCERRWARN)
682 amd64_CCFLAGS=          $(amd64_XARCH) -compat=5 \
683                         -Qoption ccfe -messages=no%anachronism \
684                         -Qoption ccfe -features=no%conststrings \
685                         $(CCERRWARN)

687 sparc_CCOPTFLAG=        -O
688 sparcv9_CCOPTFLAG=      -O
689 i386_CCOPTFLAG=         -O
690 amd64_CCOPTFLAG=        -O

692 CCOPTFLAG=      $($(MACH)_CCOPTFLAG)
693 CCOPTFLAG64=    $($(MACH64)_CCOPTFLAG)
694 CCFLAGS=        $(CCOPTFLAG) $($(MACH)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
695                 $(CCUSERFLAGS)
696 CCFLAGS64=      $(CCOPTFLAG64) $($(MACH64)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
697                 $(CCUSERFLAGS64)

699 #
700 #
701 #
702 ELFWRAP_FLAGS   =
703 ELFWRAP_FLAGS64 =       -64

705 #
706 # Various mapfiles that are used throughout the build, and delivered to
707 # /usr/lib/ld.
708 #
709 MAPFILE.NED_i386 =      $(SRC)/common/mapfiles/common/map.noexdata
710 MAPFILE.NED_sparc =
711 MAPFILE.NED =           $(MAPFILE.NED_$(MACH))
712 MAPFILE.PGA =           $(SRC)/common/mapfiles/common/map.pagealign
713 MAPFILE.NES =           $(SRC)/common/mapfiles/common/map.noexstk
714 MAPFILE.FLT =           $(SRC)/common/mapfiles/common/map.filter
715 MAPFILE.LEX =           $(SRC)/common/mapfiles/common/map.lex.yy

717 #
718 # Generated mapfiles that are compiler specific, and used throughout the
719 # build.  These mapfiles are not delivered in /usr/lib/ld.
```



```
720 #
721 MAPFILE.NGB_sparc=      $(SRC)/common/mapfiles/gen/sparc_cc_map.noexeglobs
722 $(__GNUC64)MAPFILE.NGB_sparc= \
723                         $(SRC)/common/mapfiles/gen/sparc_gcc_map.noexeglobs
724 MAPFILE.NGB_sparcv9=    $(SRC)/common/mapfiles/gen/sparcv9_cc_map.noexeglobs
725 $(__GNUC64)MAPFILE.NGB_sparcv9= \
726                         $(SRC)/common/mapfiles/gen/sparcv9_gcc_map.noexeglobs
727 MAPFILE.NGB_i386=       $(SRC)/common/mapfiles/gen/i386_cc_map.noexeglobs
728 $(__GNUC64)MAPFILE.NGB_i386= \
729                         $(SRC)/common/mapfiles/gen/i386_gcc_map.noexeglobs
730 MAPFILE.NGB_amd64=      $(SRC)/common/mapfiles/gen/amd64_cc_map.noexeglobs
731 $(__GNUC64)MAPFILE.NGB_amd64= \
732                         $(SRC)/common/mapfiles/gen/amd64_gcc_map.noexeglobs
733 MAPFILE.NGB =           $(MAPFILE.NGB_$(MACH))

735 #
736 # A generic interface mapfile name, used by various dynamic objects to define
737 # the interfaces and interposers the object must export.
738 #
739 MAPFILE.INT =           mapfile-intf

741 #
742 # LDLIBS32 and LDLIBS64 can be set in the environment to override the following
743 # assignments.
744 #
745 # These environment settings make sure that no libraries are searched outside
746 # of the local workspace proto area:
747 #       LDLIBS32=-YP,$ROOT/lib:$ROOT/usr/lib
748 #       LDLIBS64=-YP,$ROOT/lib/$MACH64:$ROOT/usr/lib/$MACH64
749 #
750 LDLIBS32 =      $(ENVLDLIBS1) $(ENVLDLIBS2) $(ENVLDLIBS3)
751 LDLIBS32 +=     $(ADJUNCT_PROTO:%=-L%/usr/lib -L%/lib)
752 LDLIBS.cmd =    $(LDLIBS32)
753 LDLIBS.lib =    $(LDLIBS32)

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

760 #
761 # Define compilation macros.
762 #
763 COMPILE.c=      $(CC) $(CFLAGS) $(CPPFLAGS) -c
764 COMPILE64.c=    $(CC) $(CFLAGS64) $(CPPFLAGS) -c
765 COMPILE.cc=     $(CCC) $(CCFLAGS) $(CPPFLAGS) -c
766 COMPILE64.cc=   $(CCC) $(CCFLAGS64) $(CPPFLAGS) -c
767 COMPILE.s=      $(AS) $(ASFLAGS) $(AS_CPPFLAGS)
768 COMPILE64.s=    $(AS) $(ASFLAGS) $($(MACH64)_AS_XARCH) $(AS_CPPFLAGS)
769 COMPILE.d=      $(DTRACE) -G -32
770 COMPILE64.d=    $(DTRACE) -G -64
771 COMPILE.b=      $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))
772 COMPILE64.b=    $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))

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

777 #
778 # Link time macros
779 #
780 CCNEEDED                = -lC
781 CCEXTNEEDED             = -lCrun -lCstd
782 $(__GNUC)CCNEEDED       = -L$(GCCLIBDIR) -lstdc++ -lgcc_s
783 $(__GNUC)CCEXTNEEDED    = $(CCNEEDED)

785 LINK.c=         $(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS)
```

```
786 LINK64.c=       $(CC) $(CFLAGS64) $(CPPFLAGS) $(LDFLAGS)
787 NORUNPATH=      -norunpath -nolib
788 LINK.cc=        $(CCC) $(CCFLAGS) $(CPPFLAGS) $(NORUNPATH) \
789                 $(LDFLAGS) $(CCNEEDED)
790 LINK64.cc=      $(CCC) $(CCFLAGS64) $(CPPFLAGS) $(NORUNPATH) \
791                 $(LDFLAGS) $(CCNEEDED)

793 #
794 # lint macros
795 #
796 # Note that the undefine of __PRAGMA_REDEFINE_EXTNAME can be removed once
797 # ON is built with a version of lint that has the fix for 4484186.
798 #
799 ALWAYS_LINT_DEFS =      -errtags=yes -s
800 ALWAYS_LINT_DEFS +=     -erroff=E_PTRDIFF_OVERFLOW
801 ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_NARROW_CONV
802 ALWAYS_LINT_DEFS +=     -U__PRAGMA_REDEFINE_EXTNAME
803 ALWAYS_LINT_DEFS +=     $(C99LMODE)
804 ALWAYS_LINT_DEFS +=     -errsecurity=$(SECLEVEL)
805 ALWAYS_LINT_DEFS +=     -erroff=E_SEC_CREAT_WITHOUT_EXCL
806 ALWAYS_LINT_DEFS +=     -erroff=E_SEC_FORBIDDEN_WARN_CREAT
807 # XX64 -- really only needed for amd64 lint
808 ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_INT_TO_SMALL_INT
809 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_CONST_TO_SMALL_INT
810 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_TO_SMALL_INT
811 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_TO_PTR_FROM_INT
812 ALWAYS_LINT_DEFS +=     -erroff=E_COMP_INT_WITH_LARGE_INT
813 ALWAYS_LINT_DEFS +=     -erroff=E_INTEGRAL_CONST_EXP_EXPECTED
814 ALWAYS_LINT_DEFS +=     -erroff=E_PASS_INT_TO_SMALL_INT
815 ALWAYS_LINT_DEFS +=     -erroff=E_PTR_CONV_LOSES_BITS

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

821 SECLEVEL=       core
822 LINT.c=         $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS) $(CPPFLAGS) \
823                 $(ALWAYS_LINT_DEFS)
824 LINT64.c=       $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS64) $(CPPFLAGS) \
825                 $(ALWAYS_LINT_DEFS)
826 LINT.s=         $(LINT.c)

828 # For some future builds, NATIVE_MACH and MACH might be different.
829 # Therefore, NATIVE_MACH needs to be redefined in the
830 # environment as ‘uname -p‘ to override this macro.
831 #
832 # For now at least, we cross-compile amd64 on i386 machines.
833 NATIVE_MACH=    $(MACH:amd64=i386)

835 # Define native compilation macros
836 #

838 # Base directory where compilers are loaded.
839 # Defined here so it can be overridden by developer.
840 #
841 SPRO_ROOT=              $(BUILD_TOOLS)/SUNWspro
842 SPRO_VROOT=             $(SPRO_ROOT)/SS12
843 GNU_ROOT=               /usr

845 $(__GNUC)PRIMARY_CC=    gcc7,$(GNUC_ROOT)/bin/gcc,gnu
845 $(__GNUC)PRIMARY_CC=    gcc4,$(GNUC_ROOT)/bin/gcc,gnu
846 $(__SUNC)PRIMARY_CC=    studio12,$(SPRO_VROOT)/bin/cc,sun
847 $(__GNUC)PRIMARY_CCC=   gcc7,$(GNUC_ROOT)/bin/g++,gnu
847 $(__GNUC)PRIMARY_CCC=   gcc4,$(GNUC_ROOT)/bin/g++,gnu
848 $(__SUNC)PRIMARY_CCC=   studio12,$(SPRO_VROOT)/bin/CC,sun
```



```
850 CW_CC_COMPILERS=        $(PRIMARY_CC:%=--primary %) $(SHADOW_CCS:%=--shadow %)
851 CW_CCC_COMPILERS=       $(PRIMARY_CCC:%=--primary %) $(SHADOW_CCCS:%=--shadow %)


854 # Till SS12u1 formally becomes the NV CBE, LINT is hard
855 # coded to be picked up from the $SPRO_ROOT/sunstudio12.1/
856 # location. Impacted variables are sparc_LINT, sparcv9_LINT,
857 # i386_LINT, amd64_LINT.
858 # Reset them when SS12u1 is rolled out.
859 #

861 # Specify platform compiler versions for languages
862 # that we use (currently only c and c++).
863 #
864 CW=                     $(ONBLD_TOOLS)/bin/$(MACH)/cw

866 BUILD_CC=               $(CW) $(CW_CC_COMPILERS) --
867 BUILD_CCC=              $(CW) -C $(CW_CCC_COMPILERS) --
868 BUILD_CPP=              /usr/ccs/lib/cpp
869 BUILD_LD=               /usr/ccs/bin/ld
870 BUILD_LINT=             $(SPRO_ROOT)/sunstudio12.1/bin/lint

872 $(MACH)_CC=             $(BUILD_CC)
873 $(MACH)_CCC=            $(BUILD_CCC)
874 $(MACH)_CPP=            $(BUILD_CPP)
875 $(MACH)_LD=             $(BUILD_LD)
876 $(MACH)_LINT=           $(BUILD_LINT)
877 $(MACH64)_CC=           $(BUILD_CC)
878 $(MACH64)_CCC=          $(BUILD_CCC)
879 $(MACH64)_CPP=          $(BUILD_CPP)
880 $(MACH64)_LD=           $(BUILD_LD)
881 $(MACH64)_LINT=         $(BUILD_LINT)

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

886 i386_AS=                /usr/ccs/bin/as
887 $(__GNUC)i386_AS=       $(ONBLD_TOOLS)/bin/$(MACH)/aw
888 amd64_AS=               $(ONBLD_TOOLS)/bin/$(MACH)/aw

890 NATIVECC=               $($(NATIVE_MACH)_CC)
891 NATIVECCC=              $($(NATIVE_MACH)_CCC)
892 NATIVECPP=              $($(NATIVE_MACH)_CPP)
893 NATIVEAS=               $($(NATIVE_MACH)_AS)
894 NATIVELD=               $($(NATIVE_MACH)_LD)
895 NATIVELINT=             $($(NATIVE_MACH)_LINT)

897 #
898 # Makefile.master.64 overrides these settings
899 #
900 CC=                     $(NATIVECC)
901 CCC=                    $(NATIVECCC)
902 CPP=                    $(NATIVECPP)
903 AS=                     $(NATIVEAS)
904 LD=                     $(NATIVELD)
905 LINT=                   $(NATIVELINT)

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

910 BDIRECT=        -Bdirect
911 BDYNAMIC=       -Bdynamic
912 BLOCAL=         -Blocal
913 BNODIRECT=      -Bnodirect
914 BREDUCE=        -Breduce
915 BSTATIC=        -Bstatic
```

```
ZDEFS=          -zdefs
ZDIRECT=        -zdirect
ZIGNORE=        -zignore
ZINITFIRST=     -zinitfirst
ZINTERPOSE=     -zinterpose
ZLAZYLOAD=      -zlazyload
ZLOADFLTR=      -zloadfltr
ZMULDEFS=       -zmuldefs
ZNODEFAULTLIB=  -znodefaultlib
ZNODEFS=        -znodefs
ZNODELETE=      -znodelete
ZNODLOPEN=      -znodlopen
ZNODUMP=        -znodump
ZNOLAZYLOAD=    -znolazyload
ZNOLDYNSYM=     -znoldynsym
ZNORELOC=       -znoreloc
ZNOVERSION=     -znoversion
ZRECORD=        -zrecord
ZREDLOCSYM=     -zredlocsym
ZTEXT=          -ztext
ZVERBOSE=       -zverbose

GSHARED=        -G
CCMT=           -mt

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

sparc_C_PICFLAGS =      -fpic
sparcv9_C_PICFLAGS =    -fpic
i386_C_PICFLAGS =       -fpic
amd64_C_PICFLAGS =      -fpic
C_PICFLAGS =            $($(MACH)_C_PICFLAGS)
C_PICFLAGS64 =          $($(MACH64)_C_PICFLAGS)

sparc_C_BIGPICFLAGS =   -fPIC
sparcv9_C_BIGPICFLAGS = -fPIC
i386_C_BIGPICFLAGS =    -fPIC
amd64_C_BIGPICFLAGS =   -fPIC
C_BIGPICFLAGS =         $($(MACH)_C_BIGPICFLAGS)
C_BIGPICFLAGS64 =       $($(MACH64)_C_BIGPICFLAGS)

# CC requires there to be no space between '-K' and 'pic' or 'PIC'.
# and does not support -f
sparc_CC_PICFLAGS =     -_cc=-Kpic -_gcc=-fpic
sparcv9_CC_PICFLAGS =   -_cc=-KPIC -_gcc=-fPIC
i386_CC_PICFLAGS =      -_cc=-Kpic -_gcc=-fpic
amd64_CC_PICFLAGS =     -_cc=-Kpic -_gcc=-fpic
CC_PICFLAGS =           $($(MACH)_CC_PICFLAGS)
CC_PICFLAGS64 =         $($(MACH64)_CC_PICFLAGS)

AS_PICFLAGS=            -K pic
AS_BIGPICFLAGS=         -K PIC

#
# Default label for CTF sections
#
CTFCVTFLAGS=            -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=
```



```
#
# Make the transition between old and new CTF Tools. The new ctf tools
# do not support stabs (eg. Sun Studio). By setting BUILD_OLD_CTF_TOOLS
# here or in the environment file, the old ones will be built.
#
BUILD_NEW_CTF_TOOLS=
BUILD_OLD_CTF_TOOLS=$(POUND_SIGN)
$(BUILD_OLD_CTF_TOOLS)BUILD_NEW_CTF_TOOLS= $(POUND_SIGN)

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 edited to reflect the release.
#    POST_PROCESS_O:    Post-processing for '.o' files (typically C source)
#    POST_PROCESS_S_O:  Post-processing for '.o' files built from asssembly
#    POST_PROCESS_CC_O: Post-processing for '.o' files built from C++
#    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"
```

```
1049 PROCESS_COMMENT=  @?${MCS} -d -a $(RELEASE_CM) -a $(DEV_CM)
1050 $(RELEASE_BUILD)PROCESS_COMMENT=   @?${MCS} -d -a $(RELEASE_CM)

1052 STRIP_STABS=                       $(STRIP) -x $@
1053 $(SRCDBGBLD)STRIP_STABS=           :

1055 POST_PROCESS_O=
1056 POST_PROCESS_S_O=
1057 POST_PROCESS_CC_O=
1058 POST_PROCESS_A=
1059 POST_PROCESS_SO=        $(PROCESS_COMMENT) $@ ; $(STRIP_STABS) ; \
1060                         $(ELFSIGN_OBJECT)
1061 POST_PROCESS=           $(PROCESS_COMMENT) $@ ; $(STRIP_STABS) ; \
1062                         $(ELFSIGN_OBJECT)

1064 #
1065 # chk4ubin is a tool that inspects a module for a symbol table
1066 # ELF section size which can trigger an OBP bug on older platforms.
1067 # This problem affects only specific sun4u bootable modules.
1068 #
1069 CHK4UBIN=       $(ONBLD_TOOLS)/bin/$(MACH)/chk4ubin
1070 CHK4UBINFLAGS=
1071 CHK4UBINARY=    $(CHK4UBIN) $(CHK4UBINFLAGS) $@

1073 #
1074 # PKGARCHIVE specifies the default location where packages should be
1075 # placed if built.
1076 #
1077 $(RELEASE_BUILD)PKGARCHIVESUFFIX=       -nd
1078 PKGARCHIVE=$(SRC)/../../packages/$(MACH)/nightly$(PKGARCHIVESUFFIX)

1080 #
1081 # The repositories will be created with these publisher settings.  To
1082 # update an image to the resulting repositories, this must match the
1083 # publisher name provided to "pkg set-publisher."
1084 #
1085 PKGPUBLISHER_REDIST=    on-nightly
1086 PKGPUBLISHER_NONREDIST= on-extra

1088 #       Default build rules which perform comment section post-processing.
1089 #
1090 .c:
1091         $(LINK.c) -o $@ $< $(LDLIBS)
1092         $(POST_PROCESS)
1093 .c.o:
1094         $(COMPILE.c) $(OUTPUT_OPTION) $< $(CTFCONVERT_HOOK)
1095         $(POST_PROCESS_O)
1096 .c.a:
1097         $(COMPILE.c) -o $% $<
1098         $(PROCESS_COMMENT) $%
1099         $(AR) $(ARFLAGS) $@ $%
1100         $(RM) $%
1101 .s.o:
1102         $(COMPILE.s) -o $@ $<
1103         $(POST_PROCESS_S_O)
1104 .s.a:
1105         $(COMPILE.s) -o $% $<
1106         $(PROCESS_COMMENT) $%
1107         $(AR) $(ARFLAGS) $@ $%
1108         $(RM) $%
1109 .cc:
1110         $(LINK.cc) -o $@ $< $(LDLIBS)
1111         $(POST_PROCESS)
1112 .cc.o:
1113         $(COMPILE.cc) $(OUTPUT_OPTION) $<
```



```
1114         $(POST_PROCESS_CC_O)
1115 .cc.a:
1116         $(COMPILE.cc) -o $% $<
1117         $(AR) $(ARFLAGS) $@ $%
1118         $(PROCESS_COMMENT) $%
1119         $(RM) $%
1120 .y:
1121         $(YACC.y) $<
1122         $(LINK.c) -o $@ y.tab.c $(LDLIBS)
1123         $(POST_PROCESS)
1124         $(RM) y.tab.c
1125 .y.o:
1126         $(YACC.y) $<
1127         $(COMPILE.c) -o $@ y.tab.c $(CTFCONVERT_HOOK)
1128         $(POST_PROCESS_O)
1129         $(RM) y.tab.c
1130 .l:
1131         $(RM) $*.c
1132         $(LEX.l) $< > $*.c
1133         $(LINK.c) -o $@ $*.c -ll $(LDLIBS)
1134         $(POST_PROCESS)
1135         $(RM) $*.c
1136 .l.o:
1137         $(RM) $*.c
1138         $(LEX.l) $< > $*.c
1139         $(COMPILE.c) -o $@ $*.c $(CTFCONVERT_HOOK)
1140         $(POST_PROCESS_O)
1141         $(RM) $*.c

1143 .bin.o:
1144         $(COMPILE.b) -o $@ $<
1145         $(POST_PROCESS_O)

1147 .java.class:
1148         $(COMPILE.java) $<

1150 # Bourne and Korn shell script message catalog build rules.
1151 # We extract all gettext strings with sed(1) (being careful to permit
1152 # multiple gettext strings on the same line), weed out the dups, and
1153 # build the catalogue with awk(1).

1155 .sh.po .ksh.po:
1156         $(SED) -n -e ":a"                                       \
1157                   -e "h"                                        \
1158                   -e "s/.*gettext *\(\"[^\"]*\"\).*/\1/p"       \
1159                   -e "x"                                        \
1160                   -e "s/\(.*\)gettext *\"[^\"]*\"\(.*\)/\1\2/"  \
1161                   -e "t a"                                      \
1162                 $< | sort -u | $(AWK) '{ print "msgid\t" $$0 "\nmsgstr" }' > $@

1164 #
1165 # Python and Perl executable and message catalog build rules.
1166 #
1167 .SUFFIXES: .pl .pm .py .pyc

1169 .pl:
1170         $(RM) $@;
1171         $(SED) -e "s@TEXT_DOMAIN@\"$(TEXT_DOMAIN)\"@" $< > $@;
1172         $(CHMOD) +x $@

1174 .py:
1175         $(RM) $@; $(SED) \
1176                 -e "1s:^\#!@PYTHON@:\#!$(PYSHEBANG):" \
1177                 -e "1s:^\#!@TOOLS_PYTHON@:\#!$(TOOLS_PYTHON):" \
1178                 < $< > $@; $(CHMOD) +x $@
```

```
1180 .py.pyc:
1181         $(RM) $@
1182         $(PYTHON) -mpy_compile $<
1183         @[ $(<)c = $@ ] || $(MV) $(<)c $@

1185 .py.po:
1186         $(GNUXGETTEXT) $(GNUXGETFLAGS) -d $(<F:%.py=%) $< ;

1188 .pl.po .pm.po:
1189         $(XGETTEXT) $(XGETFLAGS) -d $(<F) $< ;
1190         $(RM)   $@ ;
1191         $(SED) "/^domain/d" < $(<F).po > $@ ;
1192         $(RM) $(<F).po

1194 #
1195 # When using xgettext, we want messages to go to the default domain,
1196 # rather than the specified one.  This special version of the
1197 # COMPILE.cpp macro effectively prevents expansion of TEXT_DOMAIN,
1198 # causing xgettext to put all messages into the default domain.
1199 #
1200 CPPFORPO=$(COMPILE.cpp:\"$(TEXT_DOMAIN)\"=TEXT_DOMAIN)

1202 .c.i:
1203         $(CPPFORPO) $< > $@

1205 .h.i:
1206         $(CPPFORPO) $< > $@

1208 .y.i:
1209         $(YACC) -d $<
1210         $(CPPFORPO) y.tab.c  > $@
1211         $(RM) y.tab.c

1213 .l.i:
1214         $(LEX) $<
1215         $(CPPFORPO) lex.yy.c  > $@
1216         $(RM) lex.yy.c

1218 .c.po:
1219         $(CPPFORPO) $< > $<.i
1220         $(BUILD.po)

1222 .cc.po:
1223         $(CPPFORPO) $< > $<.i
1224         $(BUILD.po)

1226 .y.po:
1227         $(YACC) -d $<
1228         $(CPPFORPO) y.tab.c  > $<.i
1229         $(BUILD.po)
1230         $(RM) y.tab.c

1232 .l.po:
1233         $(LEX) $<
1234         $(CPPFORPO) lex.yy.c  > $<.i
1235         $(BUILD.po)
1236         $(RM) lex.yy.c

1238 #
1239 # Rules to perform stylistic checks
1240 #
1241 .SUFFIXES: .x .xml .check .xmlchk

1243 .h.check:
1244         $(DOT_H_CHECK)
```



```
1246 .x.check:
1247         $(DOT_X_CHECK)

1249 .xml.xmlchk:
1250         $(MANIFEST_CHECK)

1252 #
1253 # Include rules to render automated sccs get rules "safe".
1254 #
1255 include $(SRC)/Makefile.noget
```

```
**********************************************************
   11197 Tue Jun 11 04:01:18 2019
new/usr/src/tools/env/illumos.sh
9996 use GCC 7 as default 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) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
# Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
# Copyright 2015, OmniTI Computer Consulting, Inc. All rights reserved.
# Copyright 2018 OmniOS Community Edition (OmniOSce) Association.
# Copyright 2019, Joyent, Inc.
# Copyright (c) 2019, Joyent, Inc.
#
# - This file is sourced by "bldenv.sh" and "nightly.sh" and should not
#   be executed directly.
# - This script is only interpreted by ksh93 and explicitly allows the
#   use of ksh93 language extensions.


# ------------------------------------------------------------------------------
# Parameters you are likely to want to change
# ------------------------------------------------------------------------------

#       DEBUG build only (-D, -F)
#       do not bringover from the parent (-n)
#       runs 'make check' (-C)
#       checks for new interfaces in libraries (-A)
#       sends mail on completion (-m and the MAILTO variable)
#       creates packages for PIT/RE (-p)
#       checks for changes in ELF runpaths (-r)
#       build and use this workspace's tools in $SRC/tools (-t)
export NIGHTLY_OPTIONS='-FnCDAmprt'

# Some scripts optionally send mail messages to MAILTO.
#export MAILTO=

# CODEMGR_WS - where is your workspace at
export CODEMGR_WS="'git rev-parse --show-toplevel'"

# Compilers may be specified using the following variables:
# PRIMARY_CC    - primary C compiler
# PRIMARY_CCC   - primary C++ compiler
#
# SHADOW_CCS    - list of shadow C compilers
# SHADOW_CCCS   - list of shadow C++ compilers
#
```



```
# Each entry has the form <name>,<path to binary>,<style> where name is a
# free-form name (possibly used in the makefiles to guard options), path is
# the path to the executable.  style is the 'style' of command line taken by
# the compiler, currently either gnu (or gcc) or sun (or cc), which is also
# used by Makefiles to guard options.
#
# __SUNC and __GNUC must still be set to reflect the style of the primary
# compiler (and to influence the default primary, otherwise)
#
# for example:
# export PRIMARY_CC=gcc4,/opt/gcc/4.4.4/bin/gcc,gnu
# export PRIMARY_CCC=gcc4,/opt/gcc/4.4.4/bin/g++,gnu
# export SHADOW_CCS=studio12,/opt/SUNWspro/bin/cc,sun
# export SHADOW_CCCS=studio12,/opt/SUNWspro/bin/CC,sun
#
# There can be several space-separated entries in SHADOW_* to run multiple
# shadow compilers.
#
# To disable shadow compilation, unset SHADOW_* or set them to the empty string.
#
export GNUC_ROOT=/usr/gcc/7
export PRIMARY_CCS=gcc7,$GNUC_ROOT/bin/gcc,gnu
export PRIMARY_CCCS=gcc7,$GNUC_ROOT/bin/g++,gnu
export SHADOW_CCS=gcc4,/opt/gcc/4.4.4/bin/gcc,gnu
export SHADOW_CCCS=gcc4,/opt/gcc/4.4.4/bin/g++,gnu
export SHADOW_CCS=gcc7,/usr/gcc/7/bin/gcc,gnu
export SHADOW_CCCS=gcc7,/usr/gcc/7/bin/g++,gnu

# uncomment to enable smatch
#export ENABLE_SMATCH=1

# Comment this out to disable support for SMB printing, i.e. if you
# don't want to bother providing the CUPS headers this needs.
export ENABLE_SMB_PRINTING=

# If your distro uses certain versions of Perl, make sure either Makefile.master
# contains your new defaults OR your .env file sets them.
# These are how you would override for building on OmniOS r151028, for example.
#export PERL_VERSION=5.28
#export PERL_ARCH=i86pc-solaris-thread-multi-64int
#export PERL_PKGVERS=

# If your distro uses certain versions of Python, make sure either
# Makefile.master contains your new defaults OR your .env file sets them.
#export PYTHON_VERSION=2.7
#export PYTHON_PKGVERS=-27
#export PYTHON_SUFFIX=
#export PYTHON3_VERSION=3.5
#export PYTHON3_PKGVERS=-35
#export PYTHON3_SUFFIX=m

# To disable building with either Python2 or Python 3 (or both), uncomment
# these lines:
#export BUILDPY2='#'
#export BUILDPY3='#'

# Set console color scheme either by build type:
#
#export RELEASE_CONSOLE_COLOR="-DDEFAULT_ANSI_FOREGROUND=ANSI_COLOR_BLACK \
#       -DDEFAULT_ANSI_BACKGROUND=ANSI_COLOR_WHITE"
#
#export DEBUG_CONSOLE_COLOR="-DDEFAULT_ANSI_FOREGROUND=ANSI_COLOR_RED \
#       -DDEFAULT_ANSI_BACKGROUND=ANSI_COLOR_WHITE"
#
# or just one for any build type:
#
```

```
#export DEFAULT_CONSOLE_COLOR="-DDEFAULT_ANSI_FOREGROUND=ANSI_COLOR_BLACK \
#       -DDEFAULT_ANSI_BACKGROUND=ANSI_COLOR_WHITE"

# Set if your distribution has different package versioning
#export PKGVERS_BRANCH=2018.0.0.17900

# Skip Java 8 builds on distributions that don't support it
#export BLD_JAVA_8=

# POST_NIGHTLY can be any command to be run at the end of nightly.  See
# nightly(1) for interactions between environment variables and this command.
#POST_NIGHTLY=

# ----------------------------------------------------------------------------
# You are less likely to need to modify parameters below.
# ----------------------------------------------------------------------------

# Maximum number of dmake jobs.  The recommended number is 2 + NCPUS,
# where NCPUS is the number of logical CPUs on your build system.
function maxjobs
{
        nameref maxjobs=$1
        integer ncpu
        integer -r min_mem_per_job=512 # minimum amount of memory for a job

        ncpu=$(builtin getconf ; getconf 'NPROCESSORS_ONLN')
        (( maxjobs=ncpu + 2 ))

        # Throttle number of parallel jobs launched by dmake to a value which
        # gurantees that all jobs have enough memory. This was added to avoid
        # excessive paging/swapping in cases of virtual machine installations
        # which have lots of CPUs but not enough memory assigned to handle
        # that many parallel jobs
        if [[ $(/usr/sbin/prtconf 2>'/dev/null') == ~(E)Memory\ size:\ ([[:digit
                integer max_jobs_per_memory # parallel jobs which fit into physi
                integer physical_memory # physical memory installed

                # The array ".sh.match" contains the contents of capturing
                # brackets in the last regex, .sh.match[1] will contain
                # the value matched by ([[:digit:]]+), i.e. the amount of
                # memory installed
                physical_memory="10#${.sh.match[1]}"

                ((
                        max_jobs_per_memory=round(physical_memory/min_mem_per_jo
                        maxjobs=fmax(2, fmin(maxjobs, max_jobs_per_memory))
                ))
        fi

        return 0
}
```

_____***unchanged_portion_omitted_***