```
**********************************************************
    3908 Tue Jan 28 13:01:14 2014
new/exception_lists/interface_check
4519 ABI checking needs to adapt to modern times, run by default
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Reviewed by: Jens Elkner <jel+illumos@cs.uni-magdeburg.de>
**********************************************************
#
# 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) 2009, 2010, Oracle and/or its affiliates. All rights reserved.

# This file provides exceptions to the usual rules applied to shared
# objects by interface_check. All strings are Perl regular expressions
# that are compared to file names. In addition to the standard Perl
# syntax, there is one extension:
#
#       MACH(dir)
#
# is expanded into a regular expression that matches the given
# directory, or a 64-bit subdirectory of the directory with the
# name of a 64-bit architecture. For example, MACH(lib) will match
# any of the following:
#
#       lib
#       lib/amd64
#       lib/sparcv9


# Shared objects underneath these parts of the tree are taken to be plugins.
# Plugins are not required to have versioned file names, and are not required
# to be internally versioned.
#
PLUGIN          ^usr/apache/libexec
PLUGIN          ^usr/lib/devfsadm
PLUGIN          ^usr/lib/efcode/.*\.so$
PLUGIN          ^usr/lib/elfedit
PLUGIN          ^usr/lib/fm/fmd/plugins
PLUGIN          ^usr/lib/fm/fmd/schemes
PLUGIN          ^usr/lib/fm/topo/plugins
PLUGIN          ^usr/lib/fwflash
PLUGIN          ^usr/lib/iconv
PLUGIN          ^usr/lib/inet/ppp
PLUGIN          ^usr/lib/mdb
PLUGIN          ^usr/lib/pci
PLUGIN          ^usr/lib/picl/plugins
```



```
PLUGIN          ^usr/lib/python2.[46]
PLUGIN          ^usr/lib/rcm/modules
PLUGIN          ^usr/lib/scsi/plugins
PLUGIN          ^usr/lib/sysevent/modules
PLUGIN          ^usr/perl5/5\.[^\\]*/lib
PLUGIN          ^usr/platform
PLUGIN          ^usr/sadm/lib/wbem
# We unfortunately can't use MACH() here, since .../64/ is literal, and not a
# link to to amd64 or sparcv9
PLUGIN          ^usr/lib/dtrace/libdtrace_forceload\.so$
PLUGIN          ^usr/lib/dtrace/64/libdtrace_forceload\.so$

# sbcp is a special case, and not a plugin. However, it does not have a
# versioned name, and does not contain versioning, so the PLUGIN exemptions fit.
PLUGIN  ^usr/4lib/sbcp$


# Objects that are not expected to contain versioning information.
# Note that PLUGIN objects are automatically exempt from this,
# so these directives are generally applied to non-plugin objects
NOVERDEF        ^usr/4lib/libc\.so\.
NOVERDEF        ^usr/MACH(lib)/0\@0\.so\.1$
NOVERDEF        ^usr/lib/MACH(abi)/apptrace\.so\.1$
NOVERDEF        ^usr/MACH(lib)/libfru.*\.so\.1$
NOVERDEF        ^usr/MACH(lib)/libkrb5\.so\.1$
NOVERDEF        ^usr/MACH(lib)/libzpool\.so\.1$
NOVERDEF        ^usr/MACH(lib)/madv\.so\.1$
NOVERDEF        ^usr/MACH(lib)/mpss\.so\.1$
NOVERDEF        ^usr/MACH(lib)/s10_brand\.so\.1$
NOVERDEF        ^usr/MACH(lib)/s10_npreload\.so\.1$
NOVERDEF        ^usr/MACH(lib)/sn1_brand\.so\.1$
NOVERDEF        ^usr/lib/fs/[^/]*/fstyp\.so\.1$
NOVERDEF        ^usr/lib/libmilter\.so\.1$
NOVERDEF        ^usr/lib/libwrap\.so\.1\.0$
NOVERDEF        ^usr/lib/locale/MACH(iso_8859_1)/iso_8859_1\.so\.3$
NOVERDEF        ^usr/lib/picl/plugins$
NOVERDEF        ^usr/sadm/admin/dhcpmgr/dhcpmgr\.so\.1$
NOVERDEF        ^usr/sadm/admin/printmgr/lib/libpmgr\.so\.1$


# Objects that are allowed to deviate from our standard version
# names.
NONSTD_VERNAME  ^usr/MACH(lib)/libtecla\.so\.1$


# These libc variants have an SONAME of libc\.so\.1$
NONSTD_VERNAME  ^usr/MACH(lib)/libc/libc_hwcap[1-3]+\.so\.1$


# The ABI requires the SONAME for libsys.so.1 to be /usr/lib/ld.so.1
# That means that the base version will also be /usr/lib/ld.so.1, which
# is non-standard.
NONSTD_VERNAME  ^usr/lib/libsys\.so\.1$
```

```
**********************************************************
    8469 Tue Jan 28 13:01:14 2014
new/usr/src/tools/env/illumos.sh
4519 ABI checking needs to adapt to modern times, run by default
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Reviewed by: Jens Elkner <jel+illumos@cs.uni-magdeburg.de>
**********************************************************
#
# 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 2010, 2011 Nexenta Systems, Inc.  All rights reserved.
# Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
#

# Configuration variables for the runtime environment of the nightly
# build script and other tools for construction and packaging of
# releases.
# This example is suitable for building an illumos workspace, which
# will contain the resulting archives. It is based off the onnv
# release. It sets NIGHTLY_OPTIONS to make nightly do:
#       DEBUG build only (-D, -F)
#       do not bringover from the parent (-n)
#       runs 'make check' (-C)
#       checks for new interfaces in libraries (-A)
#endif /* ! codereview */
#       runs lint in usr/src (-l plus the LINTDIRS variable)
#       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)
#
# - 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.
#
export NIGHTLY_OPTIONS='-FnCDAlmprt'
export NIGHTLY_OPTIONS='-FnCDlmprt'

#
# -- PLEASE READ THIS --
#
# The variables  GATE and CODEMGR_WS must always be customised to
# match your workspace/gate location!!
#
# -- PLEASE READ THIS --
#
```



```
# This is a variable for the rest of the script - GATE doesn't matter to
# nightly itself
export GATE='testws'

# CODEMGR_WS - where is your workspace at (or what should nightly name it)
export CODEMGR_WS="$HOME/ws/$GATE"

# 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_
```

```
**********************************************************
   60387 Tue Jan 28 13:01:15 2014
new/usr/src/tools/scripts/nightly.sh
4522 the build doesn't fail nearly often enough
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Reviewed by: Jens Elkner <jel+illumos@cs.uni-magdeburg.de>
**********************************************************
______unchanged_portion_omitted_

#
# Function to do the build, including package generation.
# usage: build LABEL SUFFIX ND MULTIPROTO
# - LABEL is used to tag build output.
# - SUFFIX is used to distinguish files (e.g., DEBUG vs non-DEBUG,
#   open-only vs full tree).
# - ND is "-nd" (non-DEBUG builds) or "" (DEBUG builds).
# - If MULTIPROTO is "yes", it means to name the proto area according to
#   SUFFIX.  Otherwise ("no"), (re)use the standard proto area.
#
function build {
        LABEL=$1
        SUFFIX=$2
        ND=$3
        MULTIPROTO=$4
        INSTALLOG=install${SUFFIX}-${MACH}
        NOISE=noise${SUFFIX}-${MACH}
        PKGARCHIVE=${PKGARCHIVE_ORIG}${SUFFIX}

        ORIGROOT=$ROOT
        [ $MULTIPROTO = no ] || export ROOT=$ROOT$SUFFIX

        export ENVLDLIBS1='myldlibs $ROOT'
        export ENVCPPFLAGS1='myheaders $ROOT'

        this_build_ok=y
        #
        #       Build OS-Networking source
        #
        echo "\n==== Building OS-Net source at 'date' ($LABEL) ====\n" \
                >> $LOGFILE

        rm -f $SRC/${INSTALLOG}.out
        cd $SRC
        /bin/time $MAKE -e install 2>&1 | \
            tee -a $SRC/${INSTALLOG}.out >> $LOGFILE

        echo "\n==== Build errors ($LABEL) ====\n" >> $mail_msg_file
        egrep ":" $SRC/${INSTALLOG}.out |
                egrep -e "(^${MAKE}:|[  ]error[:        \n])" | \
                egrep -v "Ignoring unknown host" | \
                egrep -v "cc .* -o error " | \
                egrep -v "warning" | tee $TMPDIR/build_errs${SUFFIX} \
                >> $mail_msg_file
        if [[ -s $TMPDIR/build_errs${SUFFIX} ]]; then
                egrep -v "warning" >> $mail_msg_file
        if [ "$?" = "0" ]; then
                build_ok=n
                this_build_ok=n
        fi
        grep "bootblock image is .* bytes too big" $SRC/${INSTALLOG}.out \
                >> $mail_msg_file
        if [ "$?" = "0" ]; then
                build_ok=n
                this_build_ok=n
        fi
```



```
        echo "\n==== Build warnings ($LABEL) ====\n" >>$mail_msg_file
        egrep -i warning: $SRC/${INSTALLOG}.out \
                | egrep -v '^tic:' \
                | egrep -v "symbol (\'|')timezone' has differing types:" \
                | egrep -v "parameter <PSTAMP> set to" \
                | egrep -v "Ignoring unknown host" \
                | egrep -v "redefining segment flags attribute for" \
                | tee $TMPDIR/build_warnings${SUFFIX} >> $mail_msg_file
        if [[ -s $TMPDIR/build_warnings${SUFFIX} ]]; then
                build_ok=n
                this_build_ok=n
        fi
                >> $mail_msg_file

        echo "\n==== Ended OS-Net source build at 'date' ($LABEL) ====\n" \
                >> $LOGFILE

        echo "\n==== Elapsed build time ($LABEL) ====\n" >>$mail_msg_file
        tail -3  $SRC/${INSTALLOG}.out >>$mail_msg_file

        if [ "$i_FLAG" = "n" ]; then
                rm -f $SRC/${NOISE}.ref
                if [ -f $SRC/${NOISE}.out ]; then
                        mv $SRC/${NOISE}.out $SRC/${NOISE}.ref
                fi
                grep : $SRC/${INSTALLOG}.out \
                        | egrep -v '^/' \
                        | egrep -v '^(Start|Finish|real|user|sys|./bld_awk)' \
                        | egrep -v '^tic:' \
                        | egrep -v '^mcs' \
                        | egrep -v '^LD_LIBRARY_PATH=' \
                        | egrep -v 'ar: creating' \
                        | egrep -v 'ar: writing' \
                        | egrep -v 'conflicts:' \
                        | egrep -v ':saved created' \
                        | egrep -v '^stty.*c:' \
                        | egrep -v '^mfgname.c:' \
                        | egrep -v '^uname-i.c:' \
                        | egrep -v '^volumes.c:' \
                        | egrep -v '^lint library construction:' \
                        | egrep -v 'tsort: INFORM:' \
                        | egrep -v 'stripalign:' \
                        | egrep -v 'chars, width' \
                        | egrep -v "symbol (\'|')timezone' has differing types:"
                        | egrep -v 'PSTAMP' \
                        | egrep -v '|%WHOANDWHERE%|' \
                        | egrep -v '^Manifying' \
                        | egrep -v 'Ignoring unknown host' \
                        | egrep -v 'Processing method:' \
                        | egrep -v '^Writing' \
                        | egrep -v 'spellin1:' \
                        | egrep -v '^adding:' \
                        | egrep -v "^echo 'msgid" \
                        | egrep -v '^echo ' \
                        | egrep -v '\.c:$' \
                        | egrep -v '^Adding file:' \
                        | egrep -v 'CLASSPATH=' \
                        | egrep -v '\/var\/mail\/:saved' \
                        | egrep -v -- '-DUTS_VERSION=' \
                        | egrep -v '^Running Mkbootstrap' \
                        | egrep -v '^Applet length read:' \
                        | egrep -v 'bytes written:' \
                        | egrep -v '^File:SolarisAuthApplet.bin' \
                        | egrep -v -i 'jibversion' \
                        | egrep -v '^Output size:' \
```

```
279                         | egrep -v '^Solo size statistics:' \
280                         | egrep -v '^Using ROM API Version' \
281                         | egrep -v '^Zero Signature length:' \
282                         | egrep -v '^Note \(probably harmless\):' \
283                         | egrep -v '::' \
284                         | egrep -v -- '-xcache' \
285                         | egrep -v '^\+' \
286                         | egrep -v '^cc1: note: -fwritable-strings' \
287                         | egrep -v 'svccfg-native -s svc:/' \
288                         | sort | uniq >$SRC/${NOISE}.out
289                 if [ ! -f $SRC/${NOISE}.ref ]; then
290                         cp $SRC/${NOISE}.out $SRC/${NOISE}.ref
291                 fi
292                 echo "\n==== Build noise differences ($LABEL) ====\n" \
293                         >>$mail_msg_file
294                 diff $SRC/${NOISE}.ref $SRC/${NOISE}.out >>$mail_msg_file
295         fi

297         #
298         #       Re-sign selected binaries using signing server
299         #       (gatekeeper builds only)
300         #
301         if [ -n "$CODESIGN_USER" -a "$this_build_ok" = "y" ]; then
302                 echo "\n==== Signing proto area at 'date' ====\n" >> $LOGFILE
303                 signing_file="${TMPDIR}/signing"
304                 rm -f ${signing_file}
305                 export CODESIGN_USER
306                 signproto $SRC/tools/codesign/creds 2>&1 | \
307                         tee -a ${signing_file} >> $LOGFILE
308                 echo "\n==== Finished signing proto area at 'date' ====\n" \
309                     >> $LOGFILE
310                 echo "\n==== Crypto module signing errors ($LABEL) ====\n" \
311                     >> $mail_msg_file
312                 egrep 'WARNING|ERROR' ${signing_file} >> $mail_msg_file
313                 if (( $? == 0 )) ; then
314                         build_ok=n
315                         this_build_ok=n
316                 fi
317         fi

319         #
320         #       Building Packages
321         #
322         if [ "$p_FLAG" = "y" -a "$this_build_ok" = "y" ]; then
323                 if [ -d $SRC/pkg ]; then
324                         echo "\n==== Creating $LABEL packages at 'date' ====\n"
325                                 >> $LOGFILE
326                         echo "Clearing out $PKGARCHIVE ..." >> $LOGFILE
327                         rm -rf $PKGARCHIVE >> "$LOGFILE" 2>&1
328                         mkdir -p $PKGARCHIVE >> "$LOGFILE" 2>&1

330                         rm -f $SRC/pkg/${INSTALLOG}.out
331                         cd $SRC/pkg
332                         /bin/time $MAKE -e install 2>&1 | \
333                             tee -a $SRC/pkg/${INSTALLOG}.out >> $LOGFILE

335                         echo "\n==== package build errors ($LABEL) ====\n" \
336                                 >> $mail_msg_file

338                         egrep "${MAKE}|ERROR|WARNING" $SRC/pkg/${INSTALLOG}.out
339                                 grep ':' | \
340                                 grep -v PSTAMP | \
341                                 egrep -v "Ignoring unknown host" | \
342                                 tee $TMPDIR/package >> $mail_msg_file
343                         if [[ -s $TMPDIR/package ]]; then
344                                 build_extras_ok=n
```



```
345                                 this_build_ok=n
346                         fi
336                                 egrep -v "Ignoring unknown host" \
337                                 >> $mail_msg_file
347                 else
348                         #
349                         # Handle it gracefully if -p was set but there so
350                         # no pkg directory.
351                         #
352                         echo "\n==== No $LABEL packages to build ====\n" \
353                                 >> $LOGFILE
354                 fi
355         else
356                 echo "\n==== Not creating $LABEL packages ====\n" >> $LOGFILE
357         fi

359         ROOT=$ORIGROOT
360 }

362 # Usage: dolint /dir y|n
363 # Arg. 2 is a flag to turn on/off the lint diff output
364 function dolint {
365         if [ ! -d "$1" ]; then
366                 echo "dolint error: $1 is not a directory"
367                 exit 1
368         fi

370         if [ "$2" != "y" -a "$2" != "n" ]; then
371                 echo "dolint internal error: $2 should be 'y' or 'n'"
372                 exit 1
373         fi

375         lintdir=$1
376         dodiff=$2
377         base=`basename $lintdir`
378         LINTOUT=$lintdir/lint-${MACH}.out
379         LINTNOISE=$lintdir/lint-noise-${MACH}
380         export ENVLDLIBS1=`myldlibs $ROOT`
381         export ENVCPPFLAGS1='myheaders $ROOT'

383         set_debug_build_flags

385         #
386         #       '$MAKE lint' in $lintdir
387         #
388         echo "\n==== Begin '$MAKE lint' of $base at 'date' ====\n" >> $LOGFILE

390         # remove old lint.out
391         rm -f $lintdir/lint.out $lintdir/lint-noise.out
392         if [ -f $lintdir/lint-noise.ref ]; then
393                 mv $lintdir/lint-noise.ref ${LINTNOISE}.ref
394         fi

396         rm -f $LINTOUT
397         cd $lintdir
398         #
399         # Remove all .ln files to ensure a full reference file
400         #
401         rm -f Nothing_to_remove \
402             `find . \( -name SCCS -o -name .hg -o -name .svn -o -name .git \) \
403                 -prune -o -type f -name '*.ln' -print `

405         /bin/time $MAKE -ek lint 2>&1 | \
406             tee -a $LINTOUT >> $LOGFILE

408 #endif /* ! codereview */
```

```
409         echo "\n==== '$MAKE lint' of $base ERRORS ====\n" >> $mail_msg_file

411 #endif /* ! codereview */
412         grep "$MAKE:" $LINTOUT |
413                 egrep -v "Ignoring unknown host" | \
414                 tee $TMPDIR/lint_errs >> $mail_msg_file
415         if [[ -s $TMPDIR/lint_errs ]]; then
416                 build_extras_ok=n
417         fi
398                 egrep -v "Ignoring unknown host" \
399                 >> $mail_msg_file

419         echo "\n==== Ended '$MAKE lint' of $base at 'date' ====\n" >> $LOGFILE

421         echo "\n==== Elapsed time of '$MAKE lint' of $base ====\n" \
422                 >>$mail_msg_file
423         tail -3  $LINTOUT >>$mail_msg_file

425         rm -f ${LINTNOISE}.ref
426         if [ -f ${LINTNOISE}.out ]; then
427                 mv ${LINTNOISE}.out ${LINTNOISE}.ref
428         fi
429         grep : $LINTOUT | \
430                 egrep -v '^(real|user|sys)' |
431                 egrep -v '(library construction)' | \
432                 egrep -v ': global crosschecks' | \
433                 egrep -v 'Ignoring unknown host' | \
434                 egrep -v '\.c:$' | \
435                 sort | uniq > ${LINTNOISE}.out
436         if [ ! -f ${LINTNOISE}.ref ]; then
437                 cp ${LINTNOISE}.out ${LINTNOISE}.ref
438         fi

440 #endif /* ! codereview */
441         if [ "$dodiff" != "n" ]; then
442                 echo "\n==== lint warnings $base ====\n" \
443                         >>$mail_msg_file
444                 # should be none, though there are a few that were filtered out
445                 # above
446                 egrep -i '(warning|lint):' ${LINTNOISE}.out \
447                         | sort | uniq | tee $TMPDIR/lint_warns >> $mail_msg_file
448                 if [[ -s $TMPDIR/lint_warns ]]; then
449                         build_extras_ok=n
450                 fi
421                         | sort | uniq >> $mail_msg_file
451                 echo "\n==== lint noise differences $base ====\n" \
452                         >> $mail_msg_file
453                 diff ${LINTNOISE}.ref ${LINTNOISE}.out \
454                         >> $mail_msg_file
455         fi
456 }

458 #
459 # Build and install the onbld tools.
460 #
461 # usage: build_tools DESTROOT
462 #
463 # returns non-zero status if the build was successful.
464 #
465 function build_tools {
466         DESTROOT=$1

468         INSTALLOG=install-${MACH}

470         echo "\n==== Building tools at 'date' ====\n" \
471                 >> $LOGFILE
```



```
473         rm -f ${TOOLS}/${INSTALLOG}.out
474         cd ${TOOLS}
475         /bin/time $MAKE TOOLS_PROTO=${DESTROOT} -e install 2>&1 | \
476             tee -a ${TOOLS}/${INSTALLOG}.out >> $LOGFILE

478         echo "\n==== Tools build errors ====\n" >> $mail_msg_file

480         egrep ":" ${TOOLS}/${INSTALLOG}.out |
481                 egrep -e "(${MAKE}:|[   ]error[:        \n])" | \
482                 egrep -v "Ignoring unknown host" | \
483                 egrep -v warning | tee $TMPDIR/tools_errors >> $mail_msg_file

485         if [[ -s $TMPDIR/tools_errors ]]; then
486                 return 1
487         fi
488         return 0
454                 egrep -v warning >> $mail_msg_file
455         return $?
489 }
```

_____*unchanged_portion_omitted_*

```
624 MACH='uname -p'

626 if [ "$OPTHOME" = "" ]; then
627         OPTHOME=/opt
628         export OPTHOME
629 fi

631 USAGE='Usage: nightly [-in] [+t] [-V VERS ] <env_file>

633 Where:
634         -i      Fast incremental options (no clobber, lint, check)
635         -n      Do not do a bringover
636         +t      Use the build tools in $ONBLD_TOOLS/bin
637         -V VERS set the build version string to VERS

639         <env_file>  file in Bourne shell syntax that sets and exports
640         variables that configure the operation of this script and many of
641         the scripts this one calls. If <env_file> does not exist,
642         it will be looked for in $OPTHOME/onbld/env.

644 non-DEBUG is the default build type. Build options can be set in the
645 NIGHTLY_OPTIONS variable in the <env_file> as follows:

647         -A      check for ABI differences in .so files
648         -C      check for cstyle/hdrchk errors
649         -D      do a build with DEBUG on
650         -F      do _not_ do a non-DEBUG build
651         -G      gate keeper default group of options (-au)
652         -I      integration engineer default group of options (-ampu)
653         -M      do not run pmodes (safe file permission checker)
654         -N      do not run protocmp
655         -R      default group of options for building a release (-mp)
656         -U      update proto area in the parent
657         -V VERS set the build version string to VERS
658         -f      find unreferenced files
659         -i      do an incremental build (no "make clobber")
660         -l      do "make lint" in $LINTDIRS (default: $SRC y)
661         -m      send mail to $MAILTO at end of build
662         -n      do not do a bringover
663         -p      create packages
664         -r      check ELF runtime attributes in the proto area
665         -t      build and use the tools in $SRC/tools (default setting)
666         +t      Use the build tools in $ONBLD_TOOLS/bin
```

```
667         -u      update proto_list_$MACH and friends in the parent workspace;
668                 when used with -f, also build an unrefmaster.out in the parent
669         -w      report on differences between previous and current proto areas
670 '
671 #
672 #       A log file will be generated under the name $LOGFILE
673 #       for partially completed build and log.`date '+%F'`
674 #       in the same directory for fully completed builds.
675 #

677 # default values for low-level FLAGS; G I R are group FLAGS
678 A_FLAG=n
679 C_FLAG=n
680 D_FLAG=n
681 F_FLAG=n
682 f_FLAG=n
683 i_FLAG=n; i_CMD_LINE_FLAG=n
684 l_FLAG=n
685 M_FLAG=n
686 m_FLAG=n
687 N_FLAG=n
688 n_FLAG=n
689 p_FLAG=n
690 r_FLAG=n
691 t_FLAG=y
692 U_FLAG=n
693 u_FLAG=n
694 V_FLAG=n
695 w_FLAG=n
696 #
697 build_ok=y
698 build_extras_ok=y
699 #endif /* ! codereview */

701 #
702 # examine arguments
703 #

705 OPTIND=1
706 while getopts +intV: FLAG
707 do
708         case $FLAG in
709           i )   i_FLAG=y; i_CMD_LINE_FLAG=y
710                 ;;
711           n )   n_FLAG=y
712                 ;;
713          +t )   t_FLAG=n
714                 ;;
715           V )   V_FLAG=y
716                 V_ARG="$OPTARG"
717                 ;;
718          \? )   echo "$USAGE"
719                 exit 1
720                 ;;
721         esac
722 done

724 # correct argument count after options
725 shift `expr $OPTIND - 1`

727 # test that the path to the environment-setting file was given
728 if [ $# -ne 1 ]; then
729         echo "$USAGE"
730         exit 1
731 fi
```



```
733 # check if user is running nightly as root
734 # ISUSER is set non-zero if an ordinary user runs nightly, or is zero
735 # when root invokes nightly.
736 /usr/bin/id | grep '^uid=0(' >/dev/null 2>&1
737 ISUSER=$?;      export ISUSER

739 #
740 # force locale to C
741 LC_COLLATE=C;   export LC_COLLATE
742 LC_CTYPE=C;     export LC_CTYPE
743 LC_MESSAGES=C;  export LC_MESSAGES
744 LC_MONETARY=C;  export LC_MONETARY
745 LC_NUMERIC=C;   export LC_NUMERIC
746 LC_TIME=C;      export LC_TIME

748 # clear environment variables we know to be bad for the build
749 unset LD_OPTIONS
750 unset LD_AUDIT          LD_AUDIT_32             LD_AUDIT_64
751 unset LD_BIND_NOW       LD_BIND_NOW_32          LD_BIND_NOW_64
752 unset LD_BREADTH        LD_BREADTH_32           LD_BREADTH_64
753 unset LD_CONFIG         LD_CONFIG_32            LD_CONFIG_64
754 unset LD_DEBUG          LD_DEBUG_32             LD_DEBUG_64
755 unset LD_DEMANGLE       LD_DEMANGLE_32          LD_DEMANGLE_64
756 unset LD_FLAGS          LD_FLAGS_32             LD_FLAGS_64
757 unset LD_LIBRARY_PATH   LD_LIBRARY_PATH_32      LD_LIBRARY_PATH_64
758 unset LD_LOADFLTR       LD_LOADFLTR_32          LD_LOADFLTR_64
759 unset LD_NOAUDIT        LD_NOAUDIT_32           LD_NOAUDIT_64
760 unset LD_NOAUXFLTR      LD_NOAUXFLTR_32         LD_NOAUXFLTR_64
761 unset LD_NOCONFIG       LD_NOCONFIG_32          LD_NOCONFIG_64
762 unset LD_NODIRCONFIG    LD_NODIRCONFIG_32       LD_NODIRCONFIG_64
763 unset LD_NODIRECT       LD_NODIRECT_32          LD_NODIRECT_64
764 unset LD_NOLAZYLOAD     LD_NOLAZYLOAD_32        LD_NOLAZYLOAD_64
765 unset LD_NOOBJALTER     LD_NOOBJALTER_32        LD_NOOBJALTER_64
766 unset LD_NOVERSION      LD_NOVERSION_32         LD_NOVERSION_64
767 unset LD_ORIGIN         LD_ORIGIN_32            LD_ORIGIN_64
768 unset LD_PRELOAD        LD_PRELOAD_32           LD_PRELOAD_64
769 unset LD_PROFILE        LD_PROFILE_32           LD_PROFILE_64

771 unset CONFIG
772 unset GROUP
773 unset OWNER
774 unset REMOTE
775 unset ENV
776 unset ARCH
777 unset CLASSPATH
778 unset NAME

780 #
781 # To get ONBLD_TOOLS from the environment, it must come from the env file.
782 # If it comes interactively, it is generally TOOLS_PROTO, which will be
783 # clobbered before the compiler version checks, which will therefore fail.
784 #
785 unset ONBLD_TOOLS

787 #
788 #       Setup environmental variables
789 #
790 if [ -f /etc/nightly.conf ]; then
791         . /etc/nightly.conf
792 fi

794 if [ -f $1 ]; then
795         if [[ $1 = */* ]]; then
796                 . $1
797         else
798                 . ./$1
```

```
799         fi
800 else
801         if [ -f $OPTHOME/onbld/env/$1 ]; then
802                 . $OPTHOME/onbld/env/$1
803         else
804                 echo "Cannot find env file as either $1 or $OPTHOME/onbld/env/$1
805                 exit 1
806         fi
807 fi

809 # contents of stdenv.sh inserted after next line:
810 # STDENV_START
811 # STDENV_END

813 # Check if we have sufficient data to continue...
814 [[ -v CODEMGR_WS ]] || fatal_error "Error: Variable CODEMGR_WS not set."
815 if  [[ "${NIGHTLY_OPTIONS}" == ~(F)n ]] ; then
816         # Check if the gate data are valid if we don't do a "bringover" below
817         [[ -d "${CODEMGR_WS}" ]] || \
818                 fatal_error "Error: ${CODEMGR_WS} is not a directory."
819         [[ -f "${CODEMGR_WS}/usr/src/Makefile" ]] || \
820                 fatal_error "Error: ${CODEMGR_WS}/usr/src/Makefile not found."
821 fi

823 #
824 # place ourselves in a new task, respecting BUILD_PROJECT if set.
825 #
826 if [ -z "$BUILD_PROJECT" ]; then
827         /usr/bin/newtask -c $$
828 else
829         /usr/bin/newtask -c $$ -p $BUILD_PROJECT
830 fi

832 ps -o taskid= -p $$ | read build_taskid
833 ps -o project= -p $$ | read build_project

835 #
836 # See if NIGHTLY_OPTIONS is set
837 #
838 if [ "$NIGHTLY_OPTIONS" = "" ]; then
839         NIGHTLY_OPTIONS="-aBm"
840 fi

842 #
843 # If BRINGOVER_WS was not specified, let it default to CLONE_WS
844 #
845 if [ "$BRINGOVER_WS" = "" ]; then
846         BRINGOVER_WS=$CLONE_WS
847 fi

849 #
850 # If BRINGOVER_FILES was not specified, default to usr
851 #
852 if [ "$BRINGOVER_FILES" = "" ]; then
853         BRINGOVER_FILES="usr"
854 fi

856 check_closed_bins

858 #
859 # Note: changes to the option letters here should also be applied to the
860 #       bldenv script.  'd' is listed for backward compatibility.
861 #
862 NIGHTLY_OPTIONS=-${NIGHTLY_OPTIONS#-}
863 OPTIND=1
864 while getopts +ABCDdFfGIilMmNnpRrtUuw FLAG $NIGHTLY_OPTIONS
```



```
865 do
866         case $FLAG in
867           A )   A_FLAG=y
868                 ;;
869           B )   D_FLAG=y
870                 ;; # old version of D
871           C )   C_FLAG=y
872                 ;;
873           D )   D_FLAG=y
874                 ;;
875           F )   F_FLAG=y
876                 ;;
877           f )   f_FLAG=y
878                 ;;
879           G )   u_FLAG=y
880                 ;;
881           I )   m_FLAG=y
882                 p_FLAG=y
883                 u_FLAG=y
884                 ;;
885           i )   i_FLAG=y
886                 ;;
887           l )   l_FLAG=y
888                 ;;
889           M )   M_FLAG=y
890                 ;;
891           m )   m_FLAG=y
892                 ;;
893           N )   N_FLAG=y
894                 ;;
895           n )   n_FLAG=y
896                 ;;
897           p )   p_FLAG=y
898                 ;;
899           R )   m_FLAG=y
900                 p_FLAG=y
901                 ;;
902           r )   r_FLAG=y
903                 ;;
904          +t )   t_FLAG=n
905                 ;;
906           U )   if [ -z "${PARENT_ROOT}" ]; then
907                         echo "PARENT_ROOT must be set if the U flag is" \
908                             "present in NIGHTLY_OPTIONS."
909                         exit 1
910                 fi
911                 NIGHTLY_PARENT_ROOT=$PARENT_ROOT
912                 if [ -n "${PARENT_TOOLS_ROOT}" ]; then
913                         NIGHTLY_PARENT_TOOLS_ROOT=$PARENT_TOOLS_ROOT
914                 fi
915                 U_FLAG=y
916                 ;;
917           u )   u_FLAG=y
918                 ;;
919           w )   w_FLAG=y
920                 ;;
921          \? )   echo "$USAGE"
922                 exit 1
923                 ;;
924         esac
925 done

927 if [ $ISUSER -ne 0 ]; then
928         # Set default value for STAFFER, if needed.
929         if [ -z "$STAFFER" -o "$STAFFER" = "nobody" ]; then
930                 STAFFER=`/usr/xpg4/bin/id -un`
```

```
931                 export STAFFER
932         fi
933 fi

935 if [ -z "$MAILTO" -o "$MAILTO" = "nobody" ]; then
936         MAILTO=$STAFFER
937         export MAILTO
938 fi

940 PATH="$OPTHOME/onbld/bin:$OPTHOME/onbld/bin/${MACH}:/usr/ccs/bin"
941 PATH="$PATH:$OPTHOME/SUNWspro/bin:/usr/bin:/usr/sbin:/usr/ucb"
942 PATH="$PATH:/usr/openwin/bin:/usr/sfw/bin:/opt/sfw/bin:."
943 export PATH

945 # roots of source trees, both relative to $SRC and absolute.
946 relsrcdirs="."
947 abssrcdirs="$SRC"

949 PROTOCMPTERSE="protocmp.terse -gu"
950 POUND_SIGN="#"
951 # have we set RELEASE_DATE in our env file?
952 if [ -z "$RELEASE_DATE" ]; then
953         RELEASE_DATE=$(LC_ALL=C date +"%B %Y")
954 fi
955 BUILD_DATE=$(LC_ALL=C date +%Y-%b-%d)
956 BASEWSDIR=$(basename $CODEMGR_WS)
957 DEV_CM="\"@(#)SunOS Internal Development: $LOGNAME $BUILD_DATE [$BASEWSDIR]\""

959 # we export POUND_SIGN, RELEASE_DATE and DEV_CM to speed up the build process
960 # by avoiding repeated shell invocations to evaluate Makefile.master
961 # definitions.
962 export POUND_SIGN RELEASE_DATE DEV_CM

964 maketype="distributed"
965 if [[ -z "$MAKE" ]]; then
966         MAKE=dmake
967 elif [[ ! -x "$MAKE" ]]; then
968         echo "\$MAKE is set to garbage in the environment"
969         exit 1
970 fi
971 # get the dmake version string alone
972 DMAKE_VERSION=$( $MAKE -v )
973 DMAKE_VERSION=${DMAKE_VERSION#*: }
974 # focus in on just the dotted version number alone
975 DMAKE_MAJOR=$( echo $DMAKE_VERSION | \
976         sed -e 's/.*\<\([^.]*\.[^   ]*\).*$/\1/' )
977 # extract the second (or final) integer
978 DMAKE_MINOR=${DMAKE_MAJOR#*.}
979 DMAKE_MINOR=${DMAKE_MINOR%%.*}
980 # extract the first integer
981 DMAKE_MAJOR=${DMAKE_MAJOR%%.*}
982 CHECK_DMAKE=${CHECK_DMAKE:-y}
983 # x86 was built on the 12th, sparc on the 13th.
984 if [ "$CHECK_DMAKE" = "y" -a \
985      "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/12" -a \
986      "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/13" -a \( \
987      "$DMAKE_MAJOR" -lt 7 -o \
988      "$DMAKE_MAJOR" -eq 7 -a "$DMAKE_MINOR" -lt 4 \) ]; then
989         if [ -z "$DMAKE_VERSION" ]; then
990                 echo "$MAKE is missing."
991                 exit 1
992         fi
993         echo 'whence $MAKE'" version is:"
994         echo "  ${DMAKE_VERSION}"
995         cat <<EOF
```



```
997 This version may not be safe for use, if you really want to use this version
998 anyway add the following to your environment to disable this check:

1000   CHECK_DMAKE=n
1001 EOF
1002         exit 1
1003 fi
1004 export PATH
1005 export MAKE

1007 if [ "${SUNWSPRO}" != "" ]; then
1008         PATH="${SUNWSPRO}/bin:$PATH"
1009         export PATH
1010 fi

1012 hostname=$(uname -n)
1013 if [[ $DMAKE_MAX_JOBS != +([0-9]) || $DMAKE_MAX_JOBS -eq 0 ]]
1014 then
1015         maxjobs=
1016         if [[ -f $HOME/.make.machines ]]
1017         then
1018                 # Note: there is a hard tab and space character in the []s
1019                 # below.
1020                 egrep -i "^[    ]*$hostname[    \.]" \
1021                         $HOME/.make.machines | read host jobs
1022                 maxjobs=${jobs##*=}
1023         fi

1025         if [[ $maxjobs != +([0-9]) || $maxjobs -eq 0 ]]
1026         then
1027                 # default
1028                 maxjobs=4
1029         fi

1031         export DMAKE_MAX_JOBS=$maxjobs
1032 fi

1034 DMAKE_MODE=parallel;
1035 export DMAKE_MODE

1037 if [ -z "${ROOT}" ]; then
1038         echo "ROOT must be set."
1039         exit 1
1040 fi

1042 #
1043 # if -V flag was given, reset VERSION to V_ARG
1044 #
1045 if [ "$V_FLAG" = "y" ]; then
1046         VERSION=$V_ARG
1047 fi

1049 TMPDIR="/tmp/nightly.tmpdir.$$"
1050 export TMPDIR
1051 rm -rf ${TMPDIR}
1052 mkdir -p $TMPDIR || exit 1
1053 chmod 777 $TMPDIR

1055 #
1056 # Keep elfsign's use of pkcs11_softtoken from looking in the user home
1057 # directory, which doesn't always work.   Needed until all build machines
1058 # have the fix for 6271754
1059 #
1060 SOFTTOKEN_DIR=$TMPDIR
1061 export SOFTTOKEN_DIR
```

```
1063 #
1064 # Tools should only be built non-DEBUG.  Keep track of the tools proto
1065 # area path relative to $TOOLS, because the latter changes in an
1066 # export build.
1067 #
1068 # TOOLS_PROTO is included below for builds other than usr/src/tools
1069 # that look for this location.  For usr/src/tools, this will be
1070 # overridden on the $MAKE command line in build_tools().
1071 #
1072 TOOLS=${SRC}/tools
1073 TOOLS_PROTO_REL=proto/root_${MACH}-nd
1074 TOOLS_PROTO=${TOOLS}/${TOOLS_PROTO_REL};        export TOOLS_PROTO

1076 unset   CFLAGS LD_LIBRARY_PATH LDFLAGS

1078 # create directories that are automatically removed if the nightly script
1079 # fails to start correctly
1080 function newdir {
1081         dir=$1
1082         toadd=
1083         while [ ! -d $dir ]; do
1084                 toadd="$dir $toadd"
1085                 dir=‘dirname $dir‘
1086         done
1087         torm=
1088         newlist=
1089         for dir in $toadd; do
1090                 if staffer mkdir $dir; then
1091                         newlist="$ISUSER $dir $newlist"
1092                         torm="$dir $torm"
1093                 else
1094                         [ -z "$torm" ] || staffer rmdir $torm
1095                         return 1
1096                 fi
1097         done
1098         newdirlist="$newlist $newdirlist"
1099         return 0
1100 }
1101 newdirlist=

1103 [ -d $CODEMGR_WS ] || newdir $CODEMGR_WS || exit 1

1105 # since this script assumes the build is from full source, it nullifies
1106 # variables likely to have been set by a "ws" script; nullification
1107 # confines the search space for headers and libraries to the proto area
1108 # built from this immediate source.
1109 ENVLDLIBS1=
1110 ENVLDLIBS2=
1111 ENVLDLIBS3=
1112 ENVCPPFLAGS1=
1113 ENVCPPFLAGS2=
1114 ENVCPPFLAGS3=
1115 ENVCPPFLAGS4=
1116 PARENT_ROOT=

1118 export ENVLDLIBS3 ENVCPPFLAGS1 ENVCPPFLAGS2 ENVCPPFLAGS3 ENVCPPFLAGS4 \
1119         ENVLDLIBS1 ENVLDLIBS2 PARENT_ROOT

1121 PKGARCHIVE_ORIG=$PKGARCHIVE

1123 #
1124 # Juggle the logs and optionally send mail on completion.
1125 #

1127 function logshuffle {
1128         LLOG="$ATLOG/log.‘date ’+%F.%H:%M’‘"
```



```
1129         if [ -f $LLOG -o -d $LLOG ]; then
1130                 LLOG=$LLOG.$$
1131         fi
1132         mkdir $LLOG
1133         export LLOG

1135         if [ "$build_ok" = "y" ]; then
1136                 mv $ATLOG/proto_list_${MACH} $LLOG

1138                 if [ -f $ATLOG/proto_list_tools_${MACH} ]; then
1139                         mv $ATLOG/proto_list_tools_${MACH} $LLOG
1140                 fi

1142                 if [ -f $TMPDIR/wsdiff.results ]; then
1143                         mv $TMPDIR/wsdiff.results $LLOG
1144                 fi

1146                 if [ -f $TMPDIR/wsdiff-nd.results ]; then
1147                         mv $TMPDIR/wsdiff-nd.results $LLOG
1148                 fi
1149         fi

1151         #
1152         # Now that we’re about to send mail, it’s time to check the noise
1153         # file.  In the event that an error occurs beyond this point, it will
1154         # be recorded in the nightly.log file, but nowhere else.  This would
1155         # include only errors that cause the copying of the noise log to fail
1156         # or the mail itself not to be sent.
1157         #

1159         exec >>$LOGFILE 2>&1
1160         if [ -s $build_noise_file ]; then
1161                 echo "\n==== Nightly build noise ====\n" |
1162                     tee -a $LOGFILE >>$mail_msg_file
1163                 cat $build_noise_file >>$LOGFILE
1164                 cat $build_noise_file >>$mail_msg_file
1165                 echo | tee -a $LOGFILE >>$mail_msg_file
1166         fi
1167         rm -f $build_noise_file

1169         case "$build_ok" in
1170                 y)
1171                         state=Completed
1172                         ;;
1173                 i)
1174                         state=Interrupted
1175                         ;;
1176                 *)
1177                         state=Failed
1178                         ;;
1179         esac

1181         if [[ $state != "Interrupted" && $build_extras_ok != "y" ]]; then
1182                 state=Failed
1183         fi

1185 #endif /* ! codereview */
1186         NIGHTLY_STATUS=$state
1187         export NIGHTLY_STATUS

1189         run_hook POST_NIGHTLY $state
1190         run_hook SYS_POST_NIGHTLY $state

1192         #
1193         # mailx(1) sets From: based on the -r flag
1194         # if it is given.
```

```
1195         #
1196         mailx_r=
1197         if [[ -n "${MAILFROM}" ]]; then
1198                 mailx_r="-r ${MAILFROM}"
1199         fi

1201         cat $build_time_file $build_environ_file $mail_msg_file \
1202             > ${LLOG}/mail_msg
1203         if [ "$m_FLAG" = "y" ]; then
1204                 cat ${LLOG}/mail_msg | /usr/bin/mailx ${mailx_r} -s \
1205         "Nightly ${MACH} Build of `basename ${CODEMGR_WS}` ${state}." \
1206                         ${MAILTO}
1207         fi

1209         if [ "$u_FLAG" = "y" -a "$build_ok" = "y" ]; then
1210                 staffer cp ${LLOG}/mail_msg $PARENT_WS/usr/src/mail_msg-${MACH}
1211                 staffer cp $LOGFILE $PARENT_WS/usr/src/nightly-${MACH}.log
1212         fi

1214         mv $LOGFILE $LLOG
1215 }

1217 #
1218 #       Remove the locks and temporary files on any exit
1219 #
1220 function cleanup {
1221         logshuffle

1223         [ -z "$lockfile" ] || staffer rm -f $lockfile
1224         [ -z "$atloglockfile" ] || rm -f $atloglockfile
1225         [ -z "$ulockfile" ] || staffer rm -f $ulockfile
1226         [ -z "$Ulockfile" ] || rm -f $Ulockfile

1228         set -- $newdirlist
1229         while [ $# -gt 0 ]; do
1230                 ISUSER=$1 staffer rmdir $2
1231                 shift; shift
1232         done
1233         rm -rf $TMPDIR
1234 }

1236 function cleanup_signal {
1237         build_ok=i
1238         # this will trigger cleanup(), above.
1239         exit 1
1240 }

1242 trap cleanup 0
1243 trap cleanup_signal 1 2 3 15

1245 #
1246 # Generic lock file processing -- make sure that the lock file doesn't
1247 # exist.  If it does, it should name the build host and PID.  If it
1248 # doesn't, then make sure we can create it.  Clean up locks that are
1249 # known to be stale (assumes host name is unique among build systems
1250 # for the workspace).
1251 #
1252 function create_lock {
1253         lockf=$1
1254         lockvar=$2

1256         ldir=`dirname $lockf`
1257         [ -d $ldir ] || newdir $ldir || exit 1
1258         eval $lockvar=$lockf

1260         while ! staffer ln -s $hostname.$STAFFER.$$ $lockf 2> /dev/null; do
```



```
1261                 basews=`basename $CODEMGR_WS`
1262                 ls -l $lockf | nawk '{print $NF}' | IFS=. read host user pid
1263                 if [ "$host" != "$hostname" ]; then
1264                         echo "$MACH build of $basews apparently" \
1265                             "already started by $user on $host as $pid."
1266                         exit 1
1267                 elif kill -s 0 $pid 2>/dev/null; then
1268                         echo "$MACH build of $basews already started" \
1269                             "by $user as $pid."
1270                         exit 1
1271                 else
1272                         # stale lock; clear it out and try again
1273                         rm -f $lockf
1274                 fi
1275         done
1276 }

1278 #
1279 # Return the list of interesting proto areas, depending on the current
1280 # options.
1281 #
1282 function allprotos {
1283         typeset roots="$ROOT"

1285         if [[ "$F_FLAG" = n && "$MULTI_PROTO" = yes ]]; then
1286                 roots="$roots $ROOT-nd"
1287         fi

1289         echo $roots
1290 }

1292 # Ensure no other instance of this script is running on this host.
1293 # LOCKNAME can be set in <env_file>, and is by default, but is not
1294 # required due to the use of $ATLOG below.
1295 if [ -n "$LOCKNAME" ]; then
1296         create_lock /tmp/$LOCKNAME "lockfile"
1297 fi
1298 #
1299 # Create from one, two, or three other locks:
1300 #       $ATLOG/nightly.lock
1301 #               - protects against multiple builds in same workspace
1302 #       $PARENT_WS/usr/src/nightly.$MACH.lock
1303 #               - protects against multiple 'u' copy-backs
1304 #       $NIGHTLY_PARENT_ROOT/nightly.lock
1305 #               - protects against multiple 'U' copy-backs
1306 #
1307 # Overriding ISUSER to 1 causes the lock to be created as root if the
1308 # script is run as root.  The default is to create it as $STAFFER.
1309 ISUSER=1 create_lock $ATLOG/nightly.lock "atloglockfile"
1310 if [ "$u_FLAG" = "y" ]; then
1311         create_lock $PARENT_WS/usr/src/nightly.$MACH.lock "ulockfile"
1312 fi
1313 if [ "$U_FLAG" = "y" ]; then
1314         # NIGHTLY_PARENT_ROOT is written as root if script invoked as root.
1315         ISUSER=1 create_lock $NIGHTLY_PARENT_ROOT/nightly.lock "Ulockfile"
1316 fi

1318 # Locks have been taken, so we're doing a build and we're committed to
1319 # the directories we may have created so far.
1320 newdirlist=

1322 #
1323 # Create mail_msg_file
1324 #
1325 mail_msg_file="${TMPDIR}/mail_msg"
1326 touch $mail_msg_file
```

```
1327 build_time_file="${TMPDIR}/build_time"
1328 build_environ_file="${TMPDIR}/build_environ"
1329 touch $build_environ_file
1330 #
1331 #       Move old LOGFILE aside
1332 #       ATLOG directory already made by 'create_lock' above
1333 #
1334 if [ -f $LOGFILE ]; then
1335         mv -f $LOGFILE ${LOGFILE}-
1336 fi
1337 #
1338 #       Build OsNet source
1339 #
1340 START_DATE=`date`
1341 SECONDS=0
1342 echo "\n==== Nightly $maketype build started:   $START_DATE ====" \
1343     | tee -a $LOGFILE > $build_time_file

1345 echo "\nBuild project:  $build_project\nBuild taskid:   $build_taskid" | \
1346     tee -a $mail_msg_file >> $LOGFILE

1348 # make sure we log only to the nightly build file
1349 build_noise_file="${TMPDIR}/build_noise"
1350 exec </dev/null >$build_noise_file 2>&1

1352 run_hook SYS_PRE_NIGHTLY
1353 run_hook PRE_NIGHTLY

1355 echo "\n==== list of environment variables ====\n" >> $LOGFILE
1356 env >> $LOGFILE

1358 echo "\n==== Nightly argument issues ====\n" | tee -a $mail_msg_file >> $LOGFILE

1360 if [ "$N_FLAG" = "y" ]; then
1361         if [ "$p_FLAG" = "y" ]; then
1362                 cat <<EOF | tee -a $mail_msg_file >> $LOGFILE
1363 WARNING: the p option (create packages) is set, but so is the N option (do
1364          not run protocmp); this is dangerous; you should unset the N option
1365 EOF
1366         else
1367                 cat <<EOF | tee -a $mail_msg_file >> $LOGFILE
1368 Warning: the N option (do not run protocmp) is set; it probably shouldn't be
1369 EOF
1370         fi
1371         echo "" | tee -a $mail_msg_file >> $LOGFILE
1372 fi

1374 if [ "$D_FLAG" = "n" -a "$l_FLAG" = "y" ]; then
1375         #
1376         # In the past we just complained but went ahead with the lint
1377         # pass, even though the proto area was built non-DEBUG.  It's
1378         # unlikely that non-DEBUG headers will make a difference, but
1379         # rather than assuming it's a safe combination, force the user
1380         # to specify a DEBUG build.
1381         #
1382         echo "WARNING: DEBUG build not requested; disabling lint.\n" \
1383             | tee -a $mail_msg_file >> $LOGFILE
1384         l_FLAG=n
1385 fi

1387 if [ "$f_FLAG" = "y" ]; then
1388         if [ "$i_FLAG" = "y" ]; then
1389                 echo "WARNING: the -f flag cannot be used during incremental" \
1390                     "builds; ignoring -f\n" | tee -a $mail_msg_file >> $LOGFILE
1391                 f_FLAG=n
1392         fi
```



```
1393         if [ "${l_FLAG}${p_FLAG}" != "yy" ]; then
1394                 echo "WARNING: the -f flag requires -l, and -p;" \
1395                     "ignoring -f\n" | tee -a $mail_msg_file >> $LOGFILE
1396                 f_FLAG=n
1397         fi
1398 fi

1400 if [ "$w_FLAG" = "y" -a ! -d $ROOT ]; then
1401         echo "WARNING: -w specified, but $ROOT does not exist;" \
1402             "ignoring -w\n" | tee -a $mail_msg_file >> $LOGFILE
1403         w_FLAG=n
1404 fi

1406 if [ "$t_FLAG" = "n" ]; then
1407         #
1408         # We're not doing a tools build, so make sure elfsign(1) is
1409         # new enough to safely sign non-crypto binaries.  We test
1410         # debugging output from elfsign to detect the old version.
1411         #
1412         newelfsigntest='SUNW_CRYPTO_DEBUG=stderr /usr/bin/elfsign verify \
1413             -e /usr/lib/security/pkcs11_softtoken.so.1 2>&1 \
1414             | egrep algorithmOID'
1415         if [ -z "$newelfsigntest" ]; then
1416                 echo "WARNING: /usr/bin/elfsign out of date;" \
1417                     "will only sign crypto modules\n" | \
1418                     tee -a $mail_msg_file >> $LOGFILE
1419                 export ELFSIGN_OBJECT=true
1420         elif [ "$VERIFY_ELFSIGN" = "y" ]; then
1421                 echo "WARNING: VERIFY_ELFSIGN=y requires" \
1422                     "the -t flag; ignoring VERIFY_ELFSIGN\n" | \
1423                     tee -a $mail_msg_file >> $LOGFILE
1424         fi
1425 fi

1427 case $MULTI_PROTO in
1428 yes|no) ;;
1429 *)
1430         echo "WARNING: MULTI_PROTO is \"$MULTI_PROTO\"; " \
1431             "should be \"yes\" or \"no\"." | tee -a $mail_msg_file >> $LOGFILE
1432         echo "Setting MULTI_PROTO to \"no\".\n" | \
1433             tee -a $mail_msg_file >> $LOGFILE
1434         export MULTI_PROTO=no
1435         ;;
1436 esac

1438 echo "\n==== Build version ====\n" | tee -a $mail_msg_file >> $LOGFILE
1439 echo $VERSION | tee -a $mail_msg_file >> $LOGFILE

1441 # Save the current proto area if we're comparing against the last build
1442 if [ "$w_FLAG" = "y" -a -d "$ROOT" ]; then
1443     if [ -d "$ROOT.prev" ]; then
1444         rm -rf $ROOT.prev
1445     fi
1446     mv $ROOT $ROOT.prev
1447 fi

1449 # Same for non-DEBUG proto area
1450 if [ "$w_FLAG" = "y" -a "$MULTI_PROTO" = yes -a -d "$ROOT-nd" ]; then
1451         if [ -d "$ROOT-nd.prev" ]; then
1452                 rm -rf $ROOT-nd.prev
1453         fi
1454         mv $ROOT-nd $ROOT-nd.prev
1455 fi

1457 #
1458 # Echo the SCM type of the parent workspace, this can't just be which_scm
```

```
1459 # as that does not know how to identify various network repositories.
1460 #
1461 function parent_wstype {
1462         typeset scm_type junk

1464         CODEMGR_WS="$BRINGOVER_WS" "$WHICH_SCM" 2>/dev/null \
1465             | read scm_type junk
1466         if [[ -z "$scm_type" || "$scm_type" == unknown ]]; then
1467                 # Probe BRINGOVER_WS to determine its type
1468                 if [[ $BRINGOVER_WS == ssh://* ]]; then
1469                         scm_type="mercurial"
1470                 elif [[ $BRINGOVER_WS == http://* ]] && \
1471                     wget -q -O- --save-headers "$BRINGOVER_WS/?cmd=heads" | \
1472                     egrep -s "application/mercurial" 2> /dev/null; then
1473                         scm_type="mercurial"
1474                 else
1475                         scm_type="none"
1476                 fi
1477         fi

1479         # fold both unsupported and unrecognized results into "none"
1480         case "$scm_type" in
1481         mercurial)
1482                 ;;
1483         *)      scm_type=none
1484                 ;;
1485         esac

1487         echo $scm_type
1488 }

1490 # Echo the SCM types of $CODEMGR_WS and $BRINGOVER_WS
1491 function child_wstype {
1492         typeset scm_type junk

1494         # Probe CODEMGR_WS to determine its type
1495         if [[ -d $CODEMGR_WS ]]; then
1496                 $WHICH_SCM | read scm_type junk || exit 1
1497         fi

1499         case "$scm_type" in
1500         none|git|mercurial)
1501                 ;;
1502         *)      scm_type=none
1503                 ;;
1504         esac

1506         echo $scm_type
1507 }

1509 SCM_TYPE=$(child_wstype)

1511 #
1512 #       Decide whether to clobber
1513 #
1514 if [ "$i_FLAG" = "n" -a -d "$SRC" ]; then
1515         echo "\n==== Make clobber at 'date' ====\n" >> $LOGFILE

1517         cd $SRC
1518         # remove old clobber file
1519         rm -f $SRC/clobber.out
1520         rm -f $SRC/clobber-${MACH}.out

1522         # Remove all .make.state* files, just in case we are restarting
1523         # the build after having interrupted a previous 'make clobber'.
1524         find . \( -name SCCS -o -name .hg -o -name .svn -o -name .git \
```



```
1525                 -o -name 'interfaces.*' \) -prune \
1526                 -o -name '.make.*' -print | xargs rm -f

1528         $MAKE -ek clobber 2>&1 | tee -a $SRC/clobber-${MACH}.out >> $LOGFILE
1529         echo "\n==== Make clobber ERRORS ====\n" >> $mail_msg_file
1530         grep "$MAKE:" $SRC/clobber-${MACH}.out |
1531                 egrep -v "Ignoring unknown host" | \
1532                 tee $TMPDIR/clobber_errs >> $mail_msg_file

1534         if [[ -s $TMPDIR/clobber_errs ]]; then
1535                 build_extras_ok=n
1536         fi
 665                 egrep -v "Ignoring unknown host" \
 666                 >> $mail_msg_file

1538         if [[ "$t_FLAG" = "y" ]]; then
1539                 echo "\n==== Make tools clobber at 'date' ====\n" >> $LOGFILE
1540                 cd ${TOOLS}
1541                 rm -f ${TOOLS}/clobber-${MACH}.out
1542                 $MAKE TOOLS_PROTO=$TOOLS_PROTO -ek clobber 2>&1 | \
1543                         tee -a ${TOOLS}/clobber-${MACH}.out >> $LOGFILE
1544                 echo "\n==== Make tools clobber ERRORS ====\n" \
1545                         >> $mail_msg_file
1546                 grep "$MAKE:" ${TOOLS}/clobber-${MACH}.out \
1547                         >> $mail_msg_file
1548                 if (( $? == 0 )); then
1549                         build_extras_ok=n
1550                 fi
1551 #endif /* ! codereview */
1552                 rm -rf ${TOOLS_PROTO}
1553                 mkdir -p ${TOOLS_PROTO}
1554         fi

1556         typeset roots=$(allprotos)
1557         echo "\n\nClearing $roots" >> "$LOGFILE"
1558         rm -rf $roots

1560         # Get back to a clean workspace as much as possible to catch
1561         # problems that only occur on fresh workspaces.
1562         # Remove all .make.state* files, libraries, and .o's that may
1563         # have been omitted from clobber.  A couple of libraries are
1564         # under source code control, so leave them alone.
1565         # We should probably blow away temporary directories too.
1566         cd $SRC
1567         find $relsrcdirs \( -name SCCS -o -name .hg -o -name .svn \
1568             -o -name .git -o -name 'interfaces.*' \) -prune -o \
1569             \( -name '.make.*' -o -name 'lib*.a' -o -name 'lib*.so*' -o \
1570                -name '*.o' \) -print | \
1571             grep -v 'tools/ctf/dwarf/.*/libdwarf' | xargs rm -f
1572 else
1573         echo "\n==== No clobber at 'date' ====\n" >> $LOGFILE
1574 fi

1576 type bringover_mercurial > /dev/null 2>&1 || function bringover_mercurial {
1577         typeset -x PATH=$PATH

1579         # If the repository doesn't exist yet, then we want to populate it.
1580         if [[ ! -d $CODEMGR_WS/.hg ]]; then
1581                 staffer hg init $CODEMGR_WS
1582                 staffer echo "[paths]" > $CODEMGR_WS/.hg/hgrc
1583                 staffer echo "default=$BRINGOVER_WS" >> $CODEMGR_WS/.hg/hgrc
1584                 touch $TMPDIR/new_repository
1585         fi

1587         typeset -x HGMERGE="/bin/false"
```

```
1589         #
1590         # If the user has changes, regardless of whether those changes are
1591         # committed, and regardless of whether those changes conflict, then
1592         # we'll attempt to merge them either implicitly (uncommitted) or
1593         # explicitly (committed).
1594         #
1595         # These are the messages we'll use to help clarify mercurial output
1596         # in those cases.
1597         #
1598         typeset mergefailmsg="\
1599 ***\n\
1600 *** nightly was unable to automatically merge your changes.  You should\n\
1601 *** redo the full merge manually, following the steps outlined by mercurial\n\
1602 *** above, then restart nightly.\n\
1603 ***\n"
1604         typeset mergepassmsg="\
1605 ***\n\
1606 *** nightly successfully merged your changes.  This means that your working\n\
1607 *** directory has been updated, but those changes are not yet committed.\n\
1608 *** After nightly completes, you should validate the results of the merge,\n\
1609 *** then use hg commit manually.\n\
1610 ***\n"

1612         #
1613         # For each repository in turn:
1614         #
1615         # 1. Do the pull.  If this fails, dump the output and bail out.
1616         #
1617         # 2. If the pull resulted in an extra head, do an explicit merge.
1618         #    If this fails, dump the output and bail out.
1619         #
1620         # Because we can't rely on Mercurial to exit with a failure code
1621         # when a merge fails (Mercurial issue #186), we must grep the
1622         # output of pull/merge to check for attempted and/or failed merges.
1623         #
1624         # 3. If a merge failed, set the message and fail the bringover.
1625         #
1626         # 4. Otherwise, if a merge succeeded, set the message
1627         #
1628         # 5. Dump the output, and any message from step 3 or 4.
1629         #

1631         typeset HG_SOURCE=$BRINGOVER_WS
1632         if [ ! -f $TMPDIR/new_repository ]; then
1633                 HG_SOURCE=$TMPDIR/open_bundle.hg
1634                 staffer hg --cwd $CODEMGR_WS incoming --bundle $HG_SOURCE \
1635                     -v $BRINGOVER_WS > $TMPDIR/incoming_open.out

1637                 #
1638                 # If there are no incoming changesets, then incoming will
1639                 # fail, and there will be no bundle file.  Reset the source,
1640                 # to allow the remaining logic to complete with no false
1641                 # negatives.  (Unlike incoming, pull will return success
1642                 # for the no-change case.)
1643                 #
1644                 if (( $? != 0 )); then
1645                         HG_SOURCE=$BRINGOVER_WS
1646                 fi
1647         fi

1649         staffer hg --cwd $CODEMGR_WS pull -u $HG_SOURCE \
1650             > $TMPDIR/pull_open.out 2>&1
1651         if (( $? != 0 )); then
1652                 printf "%s: pull failed as follows:\n\n" "$CODEMGR_WS"
1653                 cat $TMPDIR/pull_open.out
1654                 if grep "^merging.*failed" $TMPDIR/pull_open.out > /dev/null 2>&
```



```
1655                         printf "$mergefailmsg"
1656                 fi
1657                 touch $TMPDIR/bringover_failed
1658                 return
1659         fi

1661         if grep "not updating" $TMPDIR/pull_open.out > /dev/null 2>&1; then
1662                 staffer hg --cwd $CODEMGR_WS merge \
1663                     >> $TMPDIR/pull_open.out 2>&1
1664                 if (( $? != 0 )); then
1665                         printf "%s: merge failed as follows:\n\n" \
1666                             "$CODEMGR_WS"
1667                         cat $TMPDIR/pull_open.out
1668                         if grep "^merging.*failed" $TMPDIR/pull_open.out \
1669                             > /dev/null 2>&1; then
1670                                 printf "$mergefailmsg"
1671                         fi
1672                         touch $TMPDIR/bringover_failed
1673                         return
1674                 fi
1675         fi

1677         printf "updated %s with the following results:\n" "$CODEMGR_WS"
1678         cat $TMPDIR/pull_open.out
1679         if grep "^merging" $TMPDIR/pull_open.out >/dev/null 2>&1; then
1680                 printf "$mergepassmsg"
1681         fi
1682         printf "\n"

1684         #
1685         # Per-changeset output is neither useful nor manageable for a
1686         # newly-created repository.
1687         #
1688         if [ -f $TMPDIR/new_repository ]; then
1689                 return
1690         fi

1692         printf "\nadded the following changesets to open repository:\n"
1693         cat $TMPDIR/incoming_open.out
1694 }

1696 type bringover_none > /dev/null 2>&1 || function bringover_none {
1697         echo "Couldn't figure out what kind of SCM to use for $BRINGOVER_WS."
1698         touch $TMPDIR/bringover_failed
1699 }

1701 #
1702 #       Decide whether to bringover to the codemgr workspace
1703 #
1704 if [ "$n_FLAG" = "n" ]; then
1705         PARENT_SCM_TYPE=$(parent_wstype)

1707         if [[ $SCM_TYPE != none && $SCM_TYPE != $PARENT_SCM_TYPE ]]; then
1708                 echo "cannot bringover from $PARENT_SCM_TYPE to $SCM_TYPE, " \
1709                     "quitting at 'date'." | tee -a $mail_msg_file >> $LOGFILE
1710                 exit 1
1711         fi

1713         run_hook PRE_BRINGOVER

1715         echo "\n==== bringover to $CODEMGR_WS at 'date' ====\n" >> $LOGFILE
1716         echo "\n==== BRINGOVER LOG ====\n" >> $mail_msg_file

1718         eval "bringover_${PARENT_SCM_TYPE}" 2>&1 |
1719                 tee -a $mail_msg_file >> $LOGFILE
```

```
1721         if [ -f $TMPDIR/bringover_failed ]; then
1722                 rm -f $TMPDIR/bringover_failed
1723                 build_ok=n
1724                 echo "trouble with bringover, quitting at ‘date‘." |
1725                         tee -a $mail_msg_file >> $LOGFILE
1726                 exit 1
1727         fi

1729         #
1730         # It’s possible that we used the bringover above to create
1731         # $CODEMGR_WS.  If so, then SCM_TYPE was previously "none,"
1732         # but should now be the same as $BRINGOVER_WS.
1733         #
1734         [[ $SCM_TYPE = none ]] && SCM_TYPE=$PARENT_SCM_TYPE

1736         run_hook POST_BRINGOVER

1738         check_closed_bins

1740 else
1741         echo "\n==== No bringover to $CODEMGR_WS ====\n" >> $LOGFILE
1742 fi

1744 # Safeguards
1745 [[ -v CODEMGR_WS ]] || fatal_error "Error: Variable CODEMGR_WS not set."
1746 [[ -d "${CODEMGR_WS}" ]] || fatal_error "Error: ${CODEMGR_WS} is not a directory
1747 [[ -f "${CODEMGR_WS}/usr/src/Makefile" ]] || fatal_error "Error: ${CODEMGR_WS}/u

1749 echo "\n==== Build environment ====\n" | tee -a $build_environ_file >> $LOGFILE

1751 # System
1752 whence uname | tee -a $build_environ_file >> $LOGFILE
1753 uname -a 2>&1 | tee -a $build_environ_file >> $LOGFILE
1754 echo | tee -a $build_environ_file >> $LOGFILE

1756 # make
1757 whence $MAKE | tee -a $build_environ_file >> $LOGFILE
1758 $MAKE -v | tee -a $build_environ_file >> $LOGFILE
1759 echo "number of concurrent jobs = $DMAKE_MAX_JOBS" |
1760     tee -a $build_environ_file >> $LOGFILE

1762 #
1763 # Report the compiler versions.
1764 #

1766 if [[ ! -f $SRC/Makefile ]]; then
1767         build_ok=n
1768         echo "\nUnable to find \"Makefile\" in $SRC." | \
1769             tee -a $build_environ_file >> $LOGFILE
1770         exit 1
1771 fi

1773 ( cd $SRC
1774   for target in cc-version cc64-version java-version; do
1775         echo
1776         #
1777         # Put statefile somewhere we know we can write to rather than trip
1778         # over a read-only $srcroot.
1779         #
1780         rm -f $TMPDIR/make-state
1781         export SRC
1782         if $MAKE -K $TMPDIR/make-state -e $target 2>/dev/null; then
1783                 continue
1784         fi
1785         touch $TMPDIR/nocompiler
1786   done
```



```
1787   echo
1788 ) | tee -a $build_environ_file >> $LOGFILE

1790 if [ -f $TMPDIR/nocompiler ]; then
1791         rm -f $TMPDIR/nocompiler
1792         build_ok=n
1793         echo "Aborting due to missing compiler." |
1794                 tee -a $build_environ_file >> $LOGFILE
1795         exit 1
1796 fi

1798 # as
1799 whence as | tee -a $build_environ_file >> $LOGFILE
1800 as -V 2>&1 | head -1 | tee -a $build_environ_file >> $LOGFILE
1801 echo | tee -a $build_environ_file >> $LOGFILE

1803 # Check that we’re running a capable link-editor
1804 whence ld | tee -a $build_environ_file >> $LOGFILE
1805 LDVER=‘ld -V 2>&1‘
1806 echo $LDVER | tee -a $build_environ_file >> $LOGFILE
1807 LDVER=‘echo $LDVER | sed -e "s/.*-1\.\([0-9]*\).*/\1/"‘
1808 if [ ‘expr $LDVER \< 422‘ -eq 1 ]; then
1809         echo "The link-editor needs to be at version 422 or higher to build" | \
1810             tee -a $build_environ_file >> $LOGFILE
1811         echo "the latest stuff.  Hope your build works." | \
1812             tee -a $build_environ_file >> $LOGFILE
1813 fi

1815 #
1816 # Build and use the workspace’s tools if requested
1817 #
1818 if [[ "$t_FLAG" = "y" ]]; then
1819         set_non_debug_build_flags

1821         build_tools ${TOOLS_PROTO}
1822         if (( $? != 0 )); then
1823                 build_ok=n
1824         else
 678         if [[ $? != 0  && "$t_FLAG" = y ]]; then
1825                 use_tools $TOOLS_PROTO
1826         fi
1827 fi

1829 # timestamp the start of the normal build; the findunref tool uses it.
1830 touch $SRC/.build.tstamp

1832 normal_build

1834 ORIG_SRC=$SRC
1835 BINARCHIVE=${CODEMGR_WS}/bin-${MACH}.cpio.Z


1838 #
1839 # There are several checks that need to look at the proto area, but
1840 # they only need to look at one, and they don’t care whether it’s
1841 # DEBUG or non-DEBUG.
1842 #
1843 if [[ "$MULTI_PROTO" = yes && "$D_FLAG" = n ]]; then
1844         checkroot=$ROOT-nd
1845 else
1846         checkroot=$ROOT
1847 fi

1849 if [ "$build_ok" = "y" ]; then
1850         echo "\n==== Creating protolist system file at ‘date‘ ====" \
1851                 >> $LOGFILE
```

```
1852         protolist $checkroot > $ATLOG/proto_list_${MACH}
1853         echo "==== protolist system file created at ‘date‘ ====\n" \
1854                 >> $LOGFILE

1856         if [ "$N_FLAG" != "y" ]; then

1858                 E1=
1859                 f1=
1860                 for f in $f1; do
1861                         if [ -f "$f" ]; then
1862                                 E1="$E1 -e $f"
1863                         fi
1864                 done

1866                 E2=
1867                 f2=
1868                 if [ -d "$SRC/pkg" ]; then
1869                         f2="$f2 exceptions/packaging"
1870                 fi

1872                 for f in $f2; do
1873                         if [ -f "$f" ]; then
1874                                 E2="$E2 -e $f"
1875                         fi
1876                 done
1877         fi

1879         if [ "$N_FLAG" != "y" -a -d $SRC/pkg ]; then
1880                 echo "\n==== Validating manifests against proto area ====\n" \
1881                     >> $mail_msg_file
1882                 ( cd $SRC/pkg ; $MAKE -e protocmp ROOT="$checkroot" ) | \
1883                     tee $TMPDIR/protocmp_noise >> $mail_msg_file
1884                 if [[ -s $TMPDIR/protocmp_noise ]]; then
1885                         build_extras_ok=n
1886                 fi
 736                 ( cd $SRC/pkg ; $MAKE -e protocmp ROOT="$checkroot" ) \
 737                     >> $mail_msg_file

1887         fi

1889         if [ "$N_FLAG" != "y" -a -f "$REF_PROTO_LIST" ]; then
1890                 echo "\n==== Impact on proto area ====\n" >> $mail_msg_file
1891                 if [ -n "$E2" ]; then
1892                         ELIST=$E2
1893                 else
1894                         ELIST=$E1
1895                 fi
1896                 $PROTOCMPTERSE \
1897                         "Files in yesterday’s proto area, but not today’s:" \
1898                         "Files in today’s proto area, but not yesterday’s:" \
1899                         "Files that changed between yesterday and today:" \
1900                         ${ELIST} \
1901                         -d $REF_PROTO_LIST \
1902                         $ATLOG/proto_list_${MACH} \
1903                         >> $mail_msg_file
1904         fi
1905 fi

1907 if [[ "$u_FLAG" == "y" && "$build_ok" == "y" && \
1908     "$build_extras_ok" == "y" ]]; then
 759 if [ "$u_FLAG" = "y"  -a "$build_ok" = "y" ]; then
1909         staffer cp $ATLOG/proto_list_${MACH} \
1910                 $PARENT_WS/usr/src/proto_list_${MACH}
1911 fi

1913 # Update parent proto area if necessary. This is done now
```



```
1914 # so that the proto area has either DEBUG or non-DEBUG kernels.
1915 # Note that this clears out the lock file, so we can dispense with
1916 # the variable now.
1917 if [ "$U_FLAG" = "y" -a "$build_ok" = "y" ]; then
1918         echo "\n==== Copying proto area to $NIGHTLY_PARENT_ROOT ====\n" | \
1919             tee -a $LOGFILE >> $mail_msg_file
1920         rm -rf $NIGHTLY_PARENT_ROOT/*
1921         unset Ulockfile
1922         mkdir -p $NIGHTLY_PARENT_ROOT
1923         if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
1924                 ( cd $ROOT; tar cf - . |
1925                     ( cd $NIGHTLY_PARENT_ROOT;  umask 0; tar xpf - ) ) 2>&1 |
1926                     tee -a $mail_msg_file >> $LOGFILE
1927         fi
1928         if [[ "$MULTI_PROTO" = yes && "$F_FLAG" = n ]]; then
1929                 rm -rf $NIGHTLY_PARENT_ROOT-nd/*
1930                 mkdir -p $NIGHTLY_PARENT_ROOT-nd
1931                 cd $ROOT-nd
1932                 ( tar cf - . |
1933                     ( cd $NIGHTLY_PARENT_ROOT-nd; umask 0; tar xpf - ) ) 2>&1 |
1934                     tee -a $mail_msg_file >> $LOGFILE
1935         fi
1936         if [ -n "${NIGHTLY_PARENT_TOOLS_ROOT}" ]; then
1937                 echo "\n==== Copying tools proto area to $NIGHTLY_PARENT_TOOLS_R
1938                     tee -a $LOGFILE >> $mail_msg_file
1939                 rm -rf $NIGHTLY_PARENT_TOOLS_ROOT/*
1940                 mkdir -p $NIGHTLY_PARENT_TOOLS_ROOT
1941                 if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
1942                         ( cd $TOOLS_PROTO; tar cf - . |
1943                             ( cd $NIGHTLY_PARENT_TOOLS_ROOT;
1944                             umask 0; tar xpf - ) ) 2>&1 |
1945                             tee -a $mail_msg_file >> $LOGFILE
1946                 fi
1947         fi
1948 fi

1950 #
1951 # ELF verification: ABI (-A) and runtime (-r) checks
1952 #
1953 if [[ ($build_ok = y) && (($A_FLAG = y) || ($r_FLAG = y)) ]]; then
 804 if [[ ($build_ok = y) && ( ($A_FLAG = y) || ($r_FLAG = y) ) ]]; then
1954         # Directory ELF-data.$MACH holds the files produced by these tests.
1955         elf_ddir=$SRC/ELF-data.$MACH

1957         # If there is a previous ELF-data backup directory, remove it. Then,
1958         # rotate current ELF-data directory into its place and create a new
1959         # empty directory
1960         rm -rf $elf_ddir.ref
1961         if [[ -d $elf_ddir ]]; then
1962                 mv $elf_ddir $elf_ddir.ref
1963         fi
1964         mkdir -p $elf_ddir

1966         # Call find_elf to produce a list of the ELF objects in the proto area.
1967         # This list is passed to check_rtime and interface_check, preventing
1968         # them from separately calling find_elf to do the same work twice.
1969         find_elf -fr $checkroot > $elf_ddir/object_list

1971         if [[ $A_FLAG = y ]]; then
1972                 echo "\n==== Check versioning and ABI information ====\n"  | \
1973                     tee -a $LOGFILE >> $mail_msg_file

1975                 # Produce interface description for the proto. Report errors.
1976                 interface_check -o -w $elf_ddir -f object_list \
1977                         -i interface -E interface.err
1978                 if [[ -s $elf_ddir/interface.err ]]; then
```

```
1979                        tee -a $LOGFILE < $elf_ddir/interface.err \
1980                            >> $mail_msg_file
1981                        build_extras_ok=n
1982 #endif /* ! codereview */
1983                fi

1985                # If ELF_DATA_BASELINE_DIR is defined, compare the new interface
1986                # description file to that from the baseline gate. Issue a
1987                # warning if the baseline is not present, and keep going.
1988                if [[ "$ELF_DATA_BASELINE_DIR" != '' ]]; then
1989                        base_ifile="$ELF_DATA_BASELINE_DIR/interface"

1991                        echo "\n==== Compare versioning and ABI information" \
1992                            "to baseline ====\n"  | \
1993                            tee -a $LOGFILE >> $mail_msg_file
1994                        echo "Baseline:  $base_ifile\n" >> $LOGFILE

1996                        if [[ -f $base_ifile ]]; then
1997                                interface_cmp -d -o $base_ifile \
1998                                    $elf_ddir/interface > $elf_ddir/interface.cm
1999                                if [[ -s $elf_ddir/interface.cmp ]]; then
2000                                        echo | tee -a $LOGFILE >> $mail_msg_file
2001                                        tee -a $LOGFILE < \
2002                                            $elf_ddir/interface.cmp \
2003                                            >> $mail_msg_file
2004                                        build_extras_ok=n
2005 #endif /* ! codereview */
2006                                fi
2007                        else
2008                                echo "baseline not available. comparison" \
2009                                    "skipped" | \
2010                                    tee -a $LOGFILE >> $mail_msg_file
2011                        fi

2013                fi
2014        fi

2016        if [[ $r_FLAG = y ]]; then
2017                echo "\n==== Check ELF runtime attributes ====\n" | \
2018                    tee -a $LOGFILE >> $mail_msg_file

2020                # If we're doing a DEBUG build the proto area will be left
2021                # with debuggable objects, thus don't assert -s.
2022                if [[ $D_FLAG = y ]]; then
2023                        rtime_sflag=""
2024                else
2025                        rtime_sflag="-s"
2026                fi
2027                check_rtime -i -m -v $rtime_sflag -o -w $elf_ddir \
2028                        -D object_list  -f object_list -E runtime.err \
2029                        -I runtime.attr.raw
2030                if (( $? != 0 )); then
2031                        build_extras_ok=n
2032                fi
2033 #endif /* ! codereview */

2035                # check_rtime -I output needs to be sorted in order to
2036                # compare it to that from previous builds.
2037                sort $elf_ddir/runtime.attr.raw > $elf_ddir/runtime.attr
2038                rm $elf_ddir/runtime.attr.raw

2040                # Report errors
2041                if [[ -s $elf_ddir/runtime.err ]]; then
2042                        tee -a $LOGFILE < $elf_ddir/runtime.err \
2043                                >> $mail_msg_file
2044                        build_extras_ok=n
```



```
2045 #endif /* ! codereview */
2046                fi

2048                # If there is an ELF-data directory from a previous build,
2049                # then diff the attr files. These files contain information
2050                # about dependencies, versioning, and runpaths. There is some
2051                # overlap with the ABI checking done above, but this also
2052                # flushes out non-ABI interface differences along with the
2053                # other information.
2054                echo "\n==== Diff ELF runtime attributes" \
2055                    "(since last build) ====\n" | \
2056                    tee -a $LOGFILE >> $mail_msg_file >> $mail_msg_file

2058                if [[ -f $elf_ddir.ref/runtime.attr ]]; then
2059                        diff $elf_ddir.ref/runtime.attr \
2060                                $elf_ddir/runtime.attr \
2061                                >> $mail_msg_file
2062                fi
2063        fi

2065        # If -u set, copy contents of ELF-data.$MACH to the parent workspace.
2066        if [[ "$u_FLAG" = "y" ]]; then
2067                p_elf_ddir=$PARENT_WS/usr/src/ELF-data.$MACH

2069                # If parent lacks the ELF-data.$MACH directory, create it
2070                if [[ ! -d $p_elf_ddir ]]; then
2071                        staffer mkdir -p $p_elf_ddir
2072                fi

2074                # These files are used asynchronously by other builds for ABI
2075                # verification, as above for the -A option. As such, we require
2076                # the file replacement to be atomic. Copy the data to a temp
2077                # file in the same filesystem and then rename into place.
2078                (
2079                        cd $elf_ddir
2080                        for elf_dfile in *; do
2081                                staffer cp $elf_dfile \
2082                                    ${p_elf_ddir}/${elf_dfile}.new
2083                                staffer mv -f ${p_elf_ddir}/${elf_dfile}.new \
2084                                    ${p_elf_ddir}/${elf_dfile}
2085                        done
2086                )
2087        fi
2088 fi

2090 # DEBUG lint of kernel begins

2092 if [ "$i_CMD_LINE_FLAG" = "n" -a "$l_FLAG" = "y" ]; then
2093        if [ "$LINTDIRS" = "" ]; then
2094                # LINTDIRS="$SRC/uts y $SRC/stand y $SRC/psm y"
2095                LINTDIRS="$SRC y"
2096        fi
2097        set $LINTDIRS
2098        while [ $# -gt 0 ]; do
2099                dolint $1 $2; shift; shift
2100        done
2101 else
2102        echo "\n==== No '$MAKE lint' ====\n" >> $LOGFILE
2103 fi

2105 # "make check" begins

2107 if [ "$i_CMD_LINE_FLAG" = "n" -a "$C_FLAG" = "y" ]; then
2108        # remove old check.out
2109        rm -f $SRC/check.out
```

```
2111         rm -f $SRC/check-${MACH}.out
2112         cd $SRC
2113         $MAKE -ek check ROOT="$checkroot" 2>&1 | tee -a $SRC/check-${MACH}.out \
2114             >> $LOGFILE
2115         echo "\n==== cstyle/hdrchk errors ====\n" >> $mail_msg_file

2117         grep ":" $SRC/check-${MACH}.out |
2118                 egrep -v "Ignoring unknown host" | \
2119                 sort | uniq | tee $TMPDIR/check_errors >> $mail_msg_file

2121         if [[ -s $TMPDIR/check_errors ]]; then
2122                 build_extras_ok=n
2123         fi
 832                 sort | uniq >> $mail_msg_file
2124 else
2125         echo "\n==== No 'MAKE check' ====\n" >> $LOGFILE
2126 fi

2128 echo "\n==== Find core files ====\n" | \
2129     tee -a $LOGFILE >> $mail_msg_file

2131 find $abssrcdirs -name core -a -type f -exec file {} \; | \
2132         tee -a $LOGFILE >> $mail_msg_file

2134 if [ "$f_FLAG" = "y" -a "$build_ok" = "y" ]; then
2135         echo "\n==== Diff unreferenced files (since last build) ====\n" \
2136             | tee -a $LOGFILE >>$mail_msg_file
2137         rm -f $SRC/unref-${MACH}.ref
2138         if [ -f $SRC/unref-${MACH}.out ]; then
2139                 mv $SRC/unref-${MACH}.out $SRC/unref-${MACH}.ref
2140         fi

2142         findunref -S $SCM_TYPE -t $SRC/.build.tstamp -s usr $CODEMGR_WS \
2143             ${TOOLS}/findunref/exception_list 2>> $mail_msg_file | \
2144             sort > $SRC/unref-${MACH}.out

2146         if [ ! -f $SRC/unref-${MACH}.ref ]; then
2147                 cp $SRC/unref-${MACH}.out $SRC/unref-${MACH}.ref
2148         fi

2150         diff $SRC/unref-${MACH}.ref $SRC/unref-${MACH}.out >>$mail_msg_file
2151 fi

2153 # Verify that the usual lists of files, such as exception lists,
2154 # contain only valid references to files.  If the build has failed,
2155 # then don't check the proto area.
2156 CHECK_PATHS=${CHECK_PATHS:-y}
2157 if [ "$CHECK_PATHS" = y -a "$N_FLAG" != y ]; then
2158         echo "\n==== Check lists of files ====\n" | tee -a $LOGFILE \
2159                 >>$mail_msg_file
2160         arg=-b
2161         [ "$build_ok" = y ] && arg=
2162         checkpaths $arg $checkroot > $SRC/checkpaths.out 2>&1
2163         if [[ -s $SRC/checkpaths.out ]]; then
2164                 tee -a $LOGFILE < $SRC/checkpaths.out >> $mail_msg_file
2165                 build_extras_ok=n
2166         fi
 871         checkpaths $arg $checkroot 2>&1 | tee -a $LOGFILE >>$mail_msg_file
2167 fi

2169 if [ "$M_FLAG" != "y" -a "$build_ok" = y ]; then
2170         echo "\n==== Impact on file permissions ====\n" \
2171                 >> $mail_msg_file

2173         abspkg=
2174         for d in $abssrcdirs; do
```



```
2175                 if [ -d "$d/pkg" ]; then
2176                         abspkg="$abspkg $d"
2177                 fi
2178         done

2180         if [ -n "$abspkg" ]; then
2181                 for d in "$abspkg"; do
2182                         ( cd $d/pkg ; $MAKE -e pmodes ) >> $mail_msg_file
2183                 done
2184         fi
2185 fi

2187 if [ "$w_FLAG" = "y" -a "$build_ok" = "y" ]; then
2188         if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
2189                 do_wsdiff DEBUG $ROOT.prev $ROOT
2190         fi

2192         if [[ "$MULTI_PROTO" = yes && "$F_FLAG" = n ]]; then
2193                 do_wsdiff non-DEBUG $ROOT-nd.prev $ROOT-nd
2194         fi
2195 fi

2197 END_DATE=`date`
2198 echo "==== Nightly $maketype build completed: $END_DATE ====" | \
2199     tee -a $LOGFILE >> $build_time_file

2201 typeset -i10 hours
2202 typeset -Z2 minutes
2203 typeset -Z2 seconds

2205 elapsed_time=$SECONDS
2206 ((hours = elapsed_time / 3600 ))
2207 ((minutes = elapsed_time / 60  % 60))
2208 ((seconds = elapsed_time % 60))

2210 echo "\n==== Total build time ====" | \
2211     tee -a $LOGFILE >> $build_time_file
2212 echo "\nreal    ${hours}:${minutes}:${seconds}" | \
2213     tee -a $LOGFILE >> $build_time_file

2215 if [ "$u_FLAG" = "y" -a "$f_FLAG" = "y" -a "$build_ok" = "y" ]; then
2216         staffer cp ${SRC}/unref-${MACH}.out $PARENT_WS/usr/src/

2218         #
2219         # Produce a master list of unreferenced files -- ideally, we'd
2220         # generate the master just once after all of the nightlies
2221         # have finished, but there's no simple way to know when that
2222         # will be.  Instead, we assume that we're the last nightly to
2223         # finish and merge all of the unref-${MACH}.out files in
2224         # $PARENT_WS/usr/src/.  If we are in fact the final ${MACH} to
2225         # finish, then this file will be the authoritative master
2226         # list.  Otherwise, another ${MACH}'s nightly will eventually
2227         # overwrite ours with its own master, but in the meantime our
2228         # temporary "master" will be no worse than any older master
2229         # which was already on the parent.
2230         #

2232         set -- $PARENT_WS/usr/src/unref-*.out
2233         cp "$1" ${TMPDIR}/unref.merge
2234         shift

2236         for unreffile; do
2237                 comm -12 ${TMPDIR}/unref.merge "$unreffile" > ${TMPDIR}/unref.$$
2238                 mv ${TMPDIR}/unref.$$ ${TMPDIR}/unref.merge
2239         done
```

```
2241         staffer cp ${TMPDIR}/unref.merge $PARENT_WS/usr/src/unrefmaster.out
2242 fi

2244 #
2245 # All done save for the sweeping up.
2246 # (whichever exit we hit here will trigger the "cleanup" trap which
2247 # optionally sends mail on completion).
2248 #
2249 if [[ "$build_ok" == "y" && "$build_extras_ok" == "y" ]]; then
 954 if [ "$build_ok" = "y" ]; then
2250         exit 0
2251 fi
2252 exit 1
```

```
**********************************************************
    4986 Tue Jan 28 13:01:15 2014
new/usr/src/tools/scripts/onbld_elfmod_vertype.pm
4519 ABI checking needs to adapt to modern times, run by default
Reviewed by: Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
Reviewed by: Yuri Pankov <yuri.pankov@nexenta.com>
Reviewed by: Jens Elkner <jel+illumos@cs.uni-magdeburg.de>
**********************************************************
```

```
   1 package onbld_elfmod_vertype;

   3 #
   4 # CDDL HEADER START
   5 #
   6 # The contents of this file are subject to the terms of the
   7 # Common Development and Distribution License (the "License").
   8 # You may not use this file except in compliance with the License.
   9 #
  10 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  11 # or http://www.opensolaris.org/os/licensing.
  12 # See the License for the specific language governing permissions
  13 # and limitations under the License.
  14 #
  15 # When distributing Covered Code, include this CDDL HEADER in each
  16 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  17 # If applicable, add the following below this CDDL HEADER, with the
  18 # fields enclosed by brackets "[]" replaced with your own identifying
  19 # information: Portions Copyright [yyyy] [name of copyright owner]
  20 #
  21 # CDDL HEADER END
  22 #

  24 #
  25 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
  26 #

  28 #
  29 # This perl module implements the rules used to categorize ELF versions
  30 # for the core Solaris OS and related code repositories. Although this
  31 # code fits logically into the onbld_elfmod module, it is maintained as
  32 # a separate module in order to allow maintainers of other code to provide
  33 # an implementation appropriate to their local conventions.
  34 #
  35 # By isolating the codebase specific details of ELF version names in this
  36 # module and reporting the results via a fixed interface, we allow
  37 # interface_check and interface_cmp to be written in a way that isolates
  38 # them from the specific names that apply to a given body of code.
  39 # Those tools allow you to substitute your own module in place of this one
  40 # to customize their behavior.
  41 #
  42 # The types of versions understood by interface_check and interface_cmp
  43 # fall into the following categories:
  44 #
  45 #       NUMBERED:       A public version that follows the standard numbering
  46 #                       convention of a known prefix (e.g. ILLUMOS_),
  47 #                       followed by 2 or 3 dot separated numeric values:
  46 #                       convention of a known prefix (e.g. SUNW_), followed
  47 #                       by 2 or 3 dot separated numeric values:
  48 #
  49 #                               <PREFIX>major.minor[.micro]
  50 #
  51 #       PLAIN:          A public version that may or may not contain
  52 #                       numeric characters, but for which numeric characters
  53 #                       are not treated as such.
  54 #
  55 #       SONAME:         Base version with the same name as the object SONAME
  56 #
```



```
  57 #       PRIVATE:        A private version that follows the same rules as PLAIN.
  58 #
  59 #       UNKNOWN:        A version string that does not fit any of the
  60 #                       above categories
  61 #
  62 # The above categories are generic, in the sense that they apply to any
  63 # code base. However, each code base will have different well known prefix
  64 # and name strings that map to these categories. The purpose of this module
  65 # is to map these special well known strings to the category they represent
  66 # for the code base in question.
  67 #

  69 use strict;


  72 ## Category(Version, Soname)
  73 #
  74 # Return an array containing the category of ELF version represented
  75 # by the given Version, and other category dependent information.
  76 #
  77 # entry:
  78 #       Version - Version string to examine
  79 #       Soname - Empty ('') string, or SONAME of object that contains the
  80 #                given version if it is available. In some environments,
  81 #                the valid versions depend on the particular object in
  82 #                question. This argument can be used to customize the
  83 #                results of this routine based on the object being analyzed.
  84 #
  85 # exit:
  86 #       This routine returns an array to describe the type of version
  87 #       encountered. Element [0] is always a string token that gives one
  88 #       of the version categories described in the module header comment.
  89 #       For types other than NUMBERED, this is the only element in the
  90 #       return array.
  91 #
  92 #       NUMBERED versions receive a return array with additional values
  93 #       describing the version:
  94 #
  95 #                ( 'NUMBERED', cnt, prefix, major, minor[, micro])
  96 #
  97 #       If the version has 3 numberic values, cnt is 3, and micro
  98 #       is present. If there are 2 numeric values, cnt is 2, and micro
  99 #       is omitted.
 100 #
 101 sub Category {
 102         my ($Ver, $Soname) = @_;

 104         # For illumos, the SUNW_ or ILLUMOS_ prefix is used for numbered
 105         # public versions.
 106         if ($Ver =~ /^((?:SUNW|ILLUMOS)_)(\d+)\.(\d+)(\.(\d+))?/) {
 104         # For Solaris and related products, the SUNW_ prefix is
 105         # used for numbered public versions.
 106         if ($Ver =~ /^(SUNW_)(\d+)\.(\d+)(\.(\d+))?/) {
 107                 return ('NUMBERED', 3, $1, $2, $3, $5) if defined($5);
 108                 return ('NUMBERED', 2, $1, $2, $3);
 109         }

 111         # Well known plain versions. In Solaris, these names were used
 112         # to tag symbols that come from the SVR4 underpinnings to Solaris.
 113         # Later additions are all in the NUMBERED form.
 113         # Later Sun-specific additions are all tagged SUNW_xxx.
 114         return ('PLAIN')
 115             if (($Ver =~ /^SYSVABI_1.[23]$/) || ($Ver =~ /^SISCD_2.3[ab]*$/));

 117         # The link-editor creates "base" versions using the SONAME of the
 118         # object to contain  linker generated symbols (_etext, _edata, etc.).
```

```
 119         return ('SONAME')
 120             if ($Ver eq $Soname) && ($Soname ne '');

 122         # The convention is to use SUNWprivate and ILLUMOSprivate to indicate
 123         # private versions. They may have a numeric suffix, but the
 124         # number is not significant for ELF versioning other than being part
 125         # of a unique name.
 122         # The Solaris convention is to use SUNWprivate to indicate
 123         # private versions. SUNWprivate can have a numeric suffix, but
 124         # the number is not significant for ELF versioning other than
 125         # being part of a unique name.
 126         return ('PRIVATE')
 127             if ($Ver =~ /^(SUNW|ILLUMOS)private(_[0-9.]+)?$/);
 127             if ($Ver =~ /^SUNWprivate(_[0-9.]+)?$/);

 129         # Anything else is a version we don't recognize.
 130         return ('UNKNOWN');
 131 }
```

_____***unchanged_portion_omitted_***