****
3908 Sun Jan 26 22:03:16 2014
new/exception_lists/interface_check
4519 ABI checking needs to adapt to modern times, run by default
****

```
#
# 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 Sun Jan 26 22:03:16 2014
new/usr/src/tools/env/illumos.sh
4519 ABI checking needs to adapt to modern times, run by default
**********************************************************
#
# 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 Sun Jan 26 22:03:17 2014
new/usr/src/tools/scripts/nightly.sh
4522 the build doesn't fail nearly often enough
**********************************************************
______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:' \
                        | egrep -v '^Solo size statistics:' \
                        | egrep -v '^Using ROM API Version' \
                        | egrep -v '^Zero Signature length:' \
			| egrep -v '^Note \(probably harmless\):' \
			| egrep -v '::' \
			| egrep -v -- '-xcache' \
			| egrep -v '^\+' \
			| egrep -v '^cc1: note: -fwritable-strings' \
			| egrep -v 'svccfg-native -s svc:/' \
			| sort | uniq >$SRC/${NOISE}.out
		if [ ! -f $SRC/${NOISE}.ref ]; then
			cp $SRC/${NOISE}.out $SRC/${NOISE}.ref
		fi
		echo "\n==== Build noise differences ($LABEL) ====\n" \
			>>$mail_msg_file
		diff $SRC/${NOISE}.ref $SRC/${NOISE}.out >>$mail_msg_file
	fi

	#
	#	Re-sign selected binaries using signing server
	#	(gatekeeper builds only)
	#
	if [ -n "$CODESIGN_USER" -a "$this_build_ok" = "y" ]; then
		echo "\n==== Signing proto area at `date` ====\n" >> $LOGFILE
		signing_file="${TMPDIR}/signing"
		rm -f ${signing_file}
		export CODESIGN_USER
		signproto $SRC/tools/codesign/creds 2>&1 | \
			tee -a ${signing_file} >> $LOGFILE
		echo "\n==== Finished signing proto area at `date` ====\n" \
		    >> $LOGFILE
		echo "\n==== Crypto module signing errors ($LABEL) ====\n" \
		    >> $mail_msg_file
		egrep 'WARNING|ERROR' ${signing_file} >> $mail_msg_file
		if (( $? == 0 )) ; then
			build_ok=n
			this_build_ok=n
		fi
	fi

	#
	#	Building Packages
	#
	if [ "$p_FLAG" = "y" -a "$this_build_ok" = "y" ]; then
		if [ -d $SRC/pkg ]; then
			echo "\n==== Creating $LABEL packages at `date` ====\n" \
				>> $LOGFILE
			echo "Clearing out $PKGARCHIVE ..." >> $LOGFILE
			rm -rf $PKGARCHIVE >> "$LOGFILE" 2>&1
			mkdir -p $PKGARCHIVE >> "$LOGFILE" 2>&1

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

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

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

```
				    >> $mail_msg_file
		else
			#
			# Handle it gracefully if -p was set but there are
			# neither pkg directories.
			#
			echo "\n==== No $LABEL packages to build ====\n" \
				>> $LOGFILE
		fi
	else
		echo "\n==== Not creating $LABEL packages ====\n" >> $LOGFILE
	fi

	ROOT=$ORIGROOT
}

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

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

	lintdir=$1
	dodiff=$2
	base=`basename $lintdir`
	LINTOUT=$lintdir/lint-${MACH}.out
	LINTNOISE=$lintdir/lint-noise-${MACH}
	export ENVLDLIBS1=`myldlibs $ROOT`
	export ENVCPPFLAGS1=`myheaders $ROOT`

	set_debug_build_flags

	#
	#	'$MAKE lint' in $lintdir
	#
	echo "\n==== Begin '$MAKE lint' of $base at `date` ====\n" >> $LOGFILE

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

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

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

#endif /* ! codereview */
	echo "\n==== '$MAKE lint' of $base ERRORS ====\n" >> $mail_msg_file

#endif /* ! codereview */
```

```
        grep "$MAKE:" $LINTOUT |
                egrep -v "Ignoring unknown host" | \
                tee $TMPDIR/lint_errs >> $mail_msg_file
        if [[ -s $TMPDIR/lint_errs ]]; then
                build_extras_ok=n
        fi
                egrep -v "Ignoring unknown host" \
                >> $mail_msg_file

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

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

        rm -f ${LINTNOISE}.ref
        if [ -f ${LINTNOISE}.out ]; then
                mv ${LINTNOISE}.out ${LINTNOISE}.ref
        fi
        grep : $LINTOUT | \
                egrep -v '^(real|user|sys)' |
                egrep -v '(library construction)' | \
                egrep -v ': global crosschecks' | \
                egrep -v 'Ignoring unknown host' | \
                egrep -v '\.c:$' | \
                sort | uniq > ${LINTNOISE}.out
        if [ ! -f ${LINTNOISE}.ref ]; then
                cp ${LINTNOISE}.out ${LINTNOISE}.ref
        fi

#endif /* ! codereview */
        if [ "$dodiff" != "n" ]; then
                echo "\n==== lint warnings $base ====\n" \
                        >>$mail_msg_file
                # should be none, though there are a few that were filtered out
                # above
                egrep -i '(warning|lint):' ${LINTNOISE}.out \
                        | sort | uniq | tee $TMPDIR/lint_warns >> $mail_msg_file
                if [[ -s $TMPDIR/lint_warns ]]; then
                        build_extras_ok=n
                fi
                        | sort | uniq >> $mail_msg_file
                echo "\n==== lint noise differences $base ====\n" \
                        >> $mail_msg_file
                diff ${LINTNOISE}.ref ${LINTNOISE}.out \
                        >> $mail_msg_file
        fi
}

#
# Build and install the onbld tools.
#
# usage: build_tools DESTROOT
#
# returns non-zero status if the build was successful.
#
function build_tools {
        DESTROOT=$1

        INSTALLOG=install-${MACH}

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

        rm -f ${TOOLS}/${INSTALLOG}.out
        cd ${TOOLS}
```

```
        /bin/time $MAKE TOOLS_PROTO=${DESTROOT} -e install 2>&1 | \
            tee -a ${TOOLS}/${INSTALLOG}.out >> $LOGFILE

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

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

        if [[ -s $TMPDIR/tools_errors ]]; then
                return 1
        fi
        return 0
                egrep -v warning >> $mail_msg_file
        return $?
}
______unchanged_portion_omitted_

MACH=`uname -p`

if [ "$OPTHOME" = "" ]; then
        OPTHOME=/opt
        export OPTHOME
fi

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

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

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

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

        -A      check for ABI differences in .so files
        -C      check for cstyle/hdrchk errors
        -D      do a build with DEBUG on
        -F      do _not_ do a non-DEBUG build
        -G      gate keeper default group of options (-au)
        -I      integration engineer default group of options (-ampu)
        -M      do not run pmodes (safe file permission checker)
        -N      do not run protocmp
        -R      default group of options for building a release (-mp)
        -U      update proto area in the parent
        -V VERS set the build version string to VERS
        -f      find unreferenced files
        -i      do an incremental build (no "make clobber")
        -l      do "make lint" in $LINTDIRS (default: $SRC y)
        -m      send mail to $MAILTO at end of build
        -n      do not do a bringover
        -p      create packages
        -r      check ELF runtime attributes in the proto area
        -t      build and use the tools in $SRC/tools (default setting)
        +t      Use the build tools in $ONBLD_TOOLS/bin
        -u      update proto_list_$MACH and friends in the parent workspace;
                when used with -f, also build an unrefmaster.out in the parent
        -w      report on differences between previous and current proto areas
```

```
'
#
#       A log file will be generated under the name $LOGFILE
#       for partially completed build and log.'date '+%F''
#       in the same directory for fully completed builds.
#

# default values for low-level FLAGS; G I R are group FLAGS
A_FLAG=n
C_FLAG=n
D_FLAG=n
F_FLAG=n
f_FLAG=n
i_FLAG=n; i_CMD_LINE_FLAG=n
l_FLAG=n
M_FLAG=n
m_FLAG=n
N_FLAG=n
n_FLAG=n
p_FLAG=n
r_FLAG=n
t_FLAG=y
U_FLAG=n
u_FLAG=n
V_FLAG=n
w_FLAG=n
#
build_ok=y
build_extras_ok=y
#endif /* ! codereview */

#
# examine arguments
#

OPTIND=1
while getopts +intV: FLAG
do
        case $FLAG in
          i )   i_FLAG=y; i_CMD_LINE_FLAG=y
                ;;
          n )   n_FLAG=y
                ;;
         +t )   t_FLAG=n
                ;;
          V )   V_FLAG=y
                V_ARG="$OPTARG"
                ;;
         \? )   echo "$USAGE"
                exit 1
                ;;
        esac
done

# correct argument count after options
shift 'expr $OPTIND - 1'

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

# check if user is running nightly as root
# ISUSER is set non-zero if an ordinary user runs nightly, or is zero
# when root invokes nightly.
```

```
/usr/bin/id | grep '^uid=0(' >/dev/null 2>&1
ISUSER=$?;      export ISUSER

#
# force locale to C
LC_COLLATE=C;   export LC_COLLATE
LC_CTYPE=C;     export LC_CTYPE
LC_MESSAGES=C;  export LC_MESSAGES
LC_MONETARY=C;  export LC_MONETARY
LC_NUMERIC=C;   export LC_NUMERIC
LC_TIME=C;      export LC_TIME

# clear environment variables we know to be bad for the build
unset LD_OPTIONS
unset LD_AUDIT          LD_AUDIT_32             LD_AUDIT_64
unset LD_BIND_NOW       LD_BIND_NOW_32          LD_BIND_NOW_64
unset LD_BREADTH        LD_BREADTH_32           LD_BREADTH_64
unset LD_CONFIG         LD_CONFIG_32            LD_CONFIG_64
unset LD_DEBUG          LD_DEBUG_32             LD_DEBUG_64
unset LD_DEMANGLE       LD_DEMANGLE_32          LD_DEMANGLE_64
unset LD_FLAGS          LD_FLAGS_32             LD_FLAGS_64
unset LD_LIBRARY_PATH   LD_LIBRARY_PATH_32      LD_LIBRARY_PATH_64
unset LD_LOADFLTR       LD_LOADFLTR_32          LD_LOADFLTR_64
unset LD_NOAUDIT        LD_NOAUDIT_32           LD_NOAUDIT_64
unset LD_NOAUXFLTR      LD_NOAUXFLTR_32         LD_NOAUXFLTR_64
unset LD_NOCONFIG       LD_NOCONFIG_32          LD_NOCONFIG_64
unset LD_NODIRCONFIG    LD_NODIRCONFIG_32       LD_NODIRCONFIG_64
unset LD_NODIRECT       LD_NODIRECT_32          LD_NODIRECT_64
unset LD_NOLAZYLOAD     LD_NOLAZYLOAD_32        LD_NOLAZYLOAD_64
unset LD_NOOBJALTER     LD_NOOBJALTER_32        LD_NOOBJALTER_64
unset LD_NOVERSION      LD_NOVERSION_32         LD_NOVERSION_64
unset LD_ORIGIN         LD_ORIGIN_32            LD_ORIGIN_64
unset LD_PRELOAD        LD_PRELOAD_32           LD_PRELOAD_64
unset LD_PROFILE        LD_PROFILE_32           LD_PROFILE_64

unset CONFIG
unset GROUP
unset OWNER
unset REMOTE
unset ENV
unset ARCH
unset CLASSPATH
unset NAME

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

#
#       Setup environmental variables
#
if [ -f /etc/nightly.conf ]; then
        . /etc/nightly.conf
fi

if [ -f $1 ]; then
        if [[ $1 = */* ]]; then
                . $1
        else
                . ./$1
        fi
else
        if [ -f $OPTHOME/onbld/env/$1 ]; then
```

```
                . $OPTHOME/onbld/env/$1
        else
                echo "Cannot find env file as either $1 or $OPTHOME/onbld/env/$1
                exit 1
        fi
fi

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

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

#
# place ourselves in a new task, respecting BUILD_PROJECT if set.
#
if [ -z "$BUILD_PROJECT" ]; then
        /usr/bin/newtask -c $$
else
        /usr/bin/newtask -c $$ -p $BUILD_PROJECT
fi

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

#
# See if NIGHTLY_OPTIONS is set
#
if [ "$NIGHTLY_OPTIONS" = "" ]; then
        NIGHTLY_OPTIONS="-aBm"
fi

#
# If BRINGOVER_WS was not specified, let it default to CLONE_WS
#
if [ "$BRINGOVER_WS" = "" ]; then
        BRINGOVER_WS=$CLONE_WS
fi

#
# If BRINGOVER_FILES was not specified, default to usr
#
if [ "$BRINGOVER_FILES" = "" ]; then
        BRINGOVER_FILES="usr"
fi

check_closed_bins

#
# Note: changes to the option letters here should also be applied to the
#       bldenv script.  'd' is listed for backward compatibility.
#
NIGHTLY_OPTIONS=-${NIGHTLY_OPTIONS#-}
OPTIND=1
while getopts +ABCDdFfGIilMmNnpRrtUuw FLAG $NIGHTLY_OPTIONS
do
        case $FLAG in
          A )   A_FLAG=y
```

```
                ;;
          B )   D_FLAG=y
                ;; # old version of D
          C )   C_FLAG=y
                ;;
          D )   D_FLAG=y
                ;;
          F )   F_FLAG=y
                ;;
          f )   f_FLAG=y
                ;;
          G )   u_FLAG=y
                ;;
          I )   m_FLAG=y
                p_FLAG=y
                u_FLAG=y
                ;;
          i )   i_FLAG=y
                ;;
          l )   l_FLAG=y
                ;;
          M )   M_FLAG=y
                ;;
          m )   m_FLAG=y
                ;;
          N )   N_FLAG=y
                ;;
          n )   n_FLAG=y
                ;;
          p )   p_FLAG=y
                ;;
          R )   m_FLAG=y
                p_FLAG=y
                ;;
          r )   r_FLAG=y
                ;;
         +t )   t_FLAG=n
                ;;
          U )   if [ -z "${PARENT_ROOT}" ]; then
                        echo "PARENT_ROOT must be set if the U flag is" \
                            "present in NIGHTLY_OPTIONS."
                        exit 1
                fi
                NIGHTLY_PARENT_ROOT=$PARENT_ROOT
                if [ -n "${PARENT_TOOLS_ROOT}" ]; then
                        NIGHTLY_PARENT_TOOLS_ROOT=$PARENT_TOOLS_ROOT
                fi
                U_FLAG=y
                ;;
          u )   u_FLAG=y
                ;;
          w )   w_FLAG=y
                ;;
         \? )   echo "$USAGE"
                exit 1
                ;;
        esac
done

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

```
if [ -z "$MAILTO" -o "$MAILTO" = "nobody" ]; then
        MAILTO=$STAFFER
        export MAILTO
fi

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

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

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

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

maketype="distributed"
if [[ -z "$MAKE" ]]; then
        MAKE=dmake
elif [[ ! -x "$MAKE" ]]; then
        echo "\$MAKE is set to garbage in the environment"
        exit 1
fi
# get the dmake version string alone
DMAKE_VERSION=$( $MAKE -v )
DMAKE_VERSION=${DMAKE_VERSION#*: }
# focus in on just the dotted version number alone
DMAKE_MAJOR=$( echo $DMAKE_VERSION | \
        sed -e ’s/.*\<\([^.]*\.[^   ]*\).*$/\1/’ )
# extract the second (or final) integer
DMAKE_MINOR=${DMAKE_MAJOR#*.}
DMAKE_MINOR=${DMAKE_MINOR%%.*}
# extract the first integer
DMAKE_MAJOR=${DMAKE_MAJOR%%.*}
CHECK_DMAKE=${CHECK_DMAKE:-y}
# x86 was built on the 12th, sparc on the 13th.
if [ "$CHECK_DMAKE" = "y" -a \
     "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/12" -a \
     "$DMAKE_VERSION" != "Sun Distributed Make 7.3 2003/03/13" -a \( \
     "$DMAKE_MAJOR" -lt 7 -o \
     "$DMAKE_MAJOR" -eq 7 -a "$DMAKE_MINOR" -lt 4 \) ]; then
        if [ -z "$DMAKE_VERSION" ]; then
                echo "$MAKE is missing."
                exit 1
        fi
        echo ‘whence $MAKE‘" version is:"
        echo "  ${DMAKE_VERSION}"
        cat <<EOF

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

```
  CHECK_DMAKE=n
EOF
        exit 1
fi
export PATH
export MAKE

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

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

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

        export DMAKE_MAX_JOBS=$maxjobs
fi

DMAKE_MODE=parallel;
export DMAKE_MODE

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

#
# if -V flag was given, reset VERSION to V_ARG
#
if [ "$V_FLAG" = "y" ]; then
        VERSION=$V_ARG
fi

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

#
# Keep elfsign’s use of pkcs11_softtoken from looking in the user home
# directory, which doesn’t always work.   Needed until all build machines
# have the fix for 6271754
#
SOFTTOKEN_DIR=$TMPDIR
export SOFTTOKEN_DIR

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

unset   CFLAGS LD_LIBRARY_PATH LDFLAGS

# create directories that are automatically removed if the nightly script
# fails to start correctly
function newdir {
        dir=$1
        toadd=
        while [ ! -d $dir ]; do
                toadd="$dir $toadd"
                dir=`dirname $dir`
        done
        torm=
        newlist=
        for dir in $toadd; do
                if staffer mkdir $dir; then
                        newlist="$ISUSER $dir $newlist"
                        torm="$dir $torm"
                else
                        [ -z "$torm" ] || staffer rmdir $torm
                        return 1
                fi
        done
        newdirlist="$newlist $newdirlist"
        return 0
}
newdirlist=

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

# since this script assumes the build is from full source, it nullifies
# variables likely to have been set by a "ws" script; nullification
# confines the search space for headers and libraries to the proto area
# built from this immediate source.
ENVLDLIBS1=
ENVLDLIBS2=
ENVLDLIBS3=
ENVCPPFLAGS1=
ENVCPPFLAGS2=
ENVCPPFLAGS3=
ENVCPPFLAGS4=
PARENT_ROOT=

export ENVLDLIBS3 ENVCPPFLAGS1 ENVCPPFLAGS2 ENVCPPFLAGS3 ENVCPPFLAGS4 \
        ENVLDLIBS1 ENVLDLIBS2 PARENT_ROOT

PKGARCHIVE_ORIG=$PKGARCHIVE

#
# Juggle the logs and optionally send mail on completion.
#

function logshuffle {
        LLOG="$ATLOG/log.`date '+%F.%H:%M'`"
        if [ -f $LLOG -o -d $LLOG ]; then
                LLOG=$LLOG.$$
        fi
```

```
        mkdir $LLOG
        export LLOG

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

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

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

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

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

        exec >>$LOGFILE 2>&1
        if [ -s $build_noise_file ]; then
                echo "\n==== Nightly build noise ====\n" |
                    tee -a $LOGFILE >>$mail_msg_file
                cat $build_noise_file >>$LOGFILE
                cat $build_noise_file >>$mail_msg_file
                echo | tee -a $LOGFILE >>$mail_msg_file
        fi
        rm -f $build_noise_file

        case "$build_ok" in
                y)
                        state=Completed
                        ;;
                i)
                        state=Interrupted
                        ;;
                *)
                        state=Failed
                        ;;
        esac

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

#endif /* ! codereview */
        NIGHTLY_STATUS=$state
        export NIGHTLY_STATUS

        run_hook POST_NIGHTLY $state
        run_hook SYS_POST_NIGHTLY $state

        #
        # mailx(1) sets From: based on the -r flag
        # if it is given.
        #
        mailx_r=
        if [[ -n "${MAILFROM}" ]]; then
```

```
                mailx_r="-r ${MAILFROM}"
        fi

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

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

        mv $LOGFILE $LLOG
}

#
#       Remove the locks and temporary files on any exit
#
function cleanup {
        logshuffle

        [ -z "$lockfile" ] || staffer rm -f $lockfile
        [ -z "$atloglockfile" ] || rm -f $atloglockfile
        [ -z "$ulockfile" ] || staffer rm -f $ulockfile
        [ -z "$Ulockfile" ] || rm -f $Ulockfile

        set -- $newdirlist
        while [ $# -gt 0 ]; do
                ISUSER=$1 staffer rmdir $2
                shift; shift
        done
        rm -rf $TMPDIR
}

function cleanup_signal {
        build_ok=i
        # this will trigger cleanup(), above.
        exit 1
}

trap cleanup 0
trap cleanup_signal 1 2 3 15

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

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

        while ! staffer ln -s $hostname.$STAFFER.$$ $lockf 2> /dev/null; do
                basews=`basename $CODEMGR_WS`
                ls -l $lockf | nawk '{print $NF}' | IFS=. read host user pid
                if [ "$host" != "$hostname" ]; then
```

```
                        echo "$MACH build of $basews apparently" \
                            "already started by $user on $host as $pid."
                        exit 1
                elif kill -s 0 $pid 2>/dev/null; then
                        echo "$MACH build of $basews already started" \
                            "by $user as $pid."
                        exit 1
                else
                        # stale lock; clear it out and try again
                        rm -f $lockf
                fi
        done
}

#
# Return the list of interesting proto areas, depending on the current
# options.
#
function allprotos {
        typeset roots="$ROOT"

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

        echo $roots
}

# Ensure no other instance of this script is running on this host.
# LOCKNAME can be set in <env_file>, and is by default, but is not
# required due to the use of $ATLOG below.
if [ -n "$LOCKNAME" ]; then
        create_lock /tmp/$LOCKNAME "lockfile"
fi
#
# Create from one, two, or three other locks:
#       $ATLOG/nightly.lock
#               - protects against multiple builds in same workspace
#       $PARENT_WS/usr/src/nightly.$MACH.lock
#               - protects against multiple 'u' copy-backs
#       $NIGHTLY_PARENT_ROOT/nightly.lock
#               - protects against multiple 'U' copy-backs
#
# Overriding ISUSER to 1 causes the lock to be created as root if the
# script is run as root.  The default is to create it as $STAFFER.
ISUSER=1 create_lock $ATLOG/nightly.lock "atloglockfile"
if [ "$u_FLAG" = "y" ]; then
        create_lock $PARENT_WS/usr/src/nightly.$MACH.lock "ulockfile"
fi
if [ "$U_FLAG" = "y" ]; then
        # NIGHTLY_PARENT_ROOT is written as root if script invoked as root.
        ISUSER=1 create_lock $NIGHTLY_PARENT_ROOT/nightly.lock "Ulockfile"
fi

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

#
# Create mail_msg_file
#
mail_msg_file="${TMPDIR}/mail_msg"
touch $mail_msg_file
build_time_file="${TMPDIR}/build_time"
build_environ_file="${TMPDIR}/build_environ"
touch $build_environ_file
```

```
#
#	Move old LOGFILE aside
#	ATLOG directory already made by 'create_lock' above
#
if [ -f $LOGFILE ]; then
	mv -f $LOGFILE ${LOGFILE}-
fi
#
#	Build OsNet source
#
START_DATE=`date`
SECONDS=0
echo "\n==== Nightly $maketype build started:   $START_DATE ====" \
    | tee -a $LOGFILE > $build_time_file

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

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

run_hook SYS_PRE_NIGHTLY
run_hook PRE_NIGHTLY

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

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

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

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

if [ "$f_FLAG" = "y" ]; then
	if [ "$i_FLAG" = "y" ]; then
		echo "WARNING: the -f flag cannot be used during incremental" \
		    "builds; ignoring -f\n" | tee -a $mail_msg_file >> $LOGFILE
		f_FLAG=n
	fi
	if [ "${l_FLAG}${p_FLAG}" != "yy" ]; then
		echo "WARNING: the -f flag requires -l, and -p;" \
		    "ignoring -f\n" | tee -a $mail_msg_file >> $LOGFILE
```

```
		f_FLAG=n
	fi
fi

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

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

case $MULTI_PROTO in
yes|no) ;;
*)
	echo "WARNING: MULTI_PROTO is \"$MULTI_PROTO\"; " \
	    "should be \"yes\" or \"no\"." | tee -a $mail_msg_file >> $LOGFILE
	echo "Setting MULTI_PROTO to \"no\".\n" | \
	    tee -a $mail_msg_file >> $LOGFILE
	export MULTI_PROTO=no
	;;
esac

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

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

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

#
# Echo the SCM type of the parent workspace, this can't just be which_scm
# as that does not know how to identify various network repositories.
#
function parent_wstype {
        typeset scm_type junk

        CODEMGR_WS="$BRINGOVER_WS" "$WHICH_SCM" 2>/dev/null \
            | read scm_type junk
        if [[ -z "$scm_type" || "$scm_type" == unknown ]]; then
                # Probe BRINGOVER_WS to determine its type
                if [[ $BRINGOVER_WS == ssh://* ]]; then
                        scm_type="mercurial"
                elif [[ $BRINGOVER_WS == http://* ]] && \
                    wget -q -O- --save-headers "$BRINGOVER_WS/?cmd=heads" | \
                    egrep -s "application/mercurial" 2> /dev/null; then
                        scm_type="mercurial"
                else
                        scm_type="none"
                fi
        fi

        # fold both unsupported and unrecognized results into "none"
        case "$scm_type" in
        mercurial)
                ;;
        *)      scm_type=none
                ;;
        esac

        echo $scm_type
}

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

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

        case "$scm_type" in
        none|git|mercurial)
                ;;
        *)      scm_type=none
                ;;
        esac

        echo $scm_type
}

SCM_TYPE=$(child_wstype)

#
#       Decide whether to clobber
#
if [ "$i_FLAG" = "n" -a -d "$SRC" ]; then
        echo "\n==== Make clobber at `date` ====\n" >> $LOGFILE

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

        # Remove all .make.state* files, just in case we are restarting
        # the build after having interrupted a previous 'make clobber'.
        find . \( -name SCCS -o -name .hg -o -name .svn -o -name .git \
                -o -name 'interfaces.*' \) -prune \
                -o -name '.make.*' -print | xargs rm -f
```

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

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

        if [[ "$t_FLAG" = "y" ]]; then
                echo "\n==== Make tools clobber at `date` ====\n" >> $LOGFILE
                cd ${TOOLS}
                rm -f ${TOOLS}/clobber-${MACH}.out
                $MAKE TOOLS_PROTO=$TOOLS_PROTO -ek clobber 2>&1 | \
                        tee -a ${TOOLS}/clobber-${MACH}.out >> $LOGFILE
                echo "\n==== Make tools clobber ERRORS ====\n" \
                        >> $mail_msg_file
                grep "$MAKE:" ${TOOLS}/clobber-${MACH}.out \
                        >> $mail_msg_file
                if (( $? == 0 )); then
                        build_extras_ok=n
                fi
#endif /* ! codereview */
                rm -rf ${TOOLS_PROTO}
                mkdir -p ${TOOLS_PROTO}
        fi

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

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

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

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

        typeset -x HGMERGE="/bin/false"

        #
        # If the user has changes, regardless of whether those changes are
        # committed, and regardless of whether those changes conflict, then
        # we’ll attempt to merge them either implicitly (uncommitted) or
        # explicitly (committed).
        #
        # These are the messages we’ll use to help clarify mercurial output
        # in those cases.
        #
        typeset mergefailmsg="\
***\n\
*** nightly was unable to automatically merge your changes.  You should\n\
*** redo the full merge manually, following the steps outlined by mercurial\n\
*** above, then restart nightly.\n\
***\n"
        typeset mergepassmsg="\
***\n\
*** nightly successfully merged your changes.  This means that your working\n\
*** directory has been updated, but those changes are not yet committed.\n\
*** After nightly completes, you should validate the results of the merge,\n\
*** then use hg commit manually.\n\
***\n"

        #
        # For each repository in turn:
        #
        # 1. Do the pull.  If this fails, dump the output and bail out.
        #
        # 2. If the pull resulted in an extra head, do an explicit merge.
        #    If this fails, dump the output and bail out.
        #
        # Because we can’t rely on Mercurial to exit with a failure code
        # when a merge fails (Mercurial issue #186), we must grep the
        # output of pull/merge to check for attempted and/or failed merges.
        #
        # 3. If a merge failed, set the message and fail the bringover.
        #
        # 4. Otherwise, if a merge succeeded, set the message
        #
        # 5. Dump the output, and any message from step 3 or 4.
        #

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

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

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

```
                return
        fi

        if grep "not updating" $TMPDIR/pull_open.out > /dev/null 2>&1; then
                staffer hg --cwd $CODEMGR_WS merge \
                    >> $TMPDIR/pull_open.out 2>&1
                if (( $? != 0 )); then
                        printf "%s: merge failed as follows:\n\n" \
                            "$CODEMGR_WS"
                        cat $TMPDIR/pull_open.out
                        if grep "^merging.*failed" $TMPDIR/pull_open.out \
                            > /dev/null 2>&1; then
                                printf "$mergefailmsg"
                        fi
                        touch $TMPDIR/bringover_failed
                        return
                fi
        fi

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

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

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

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

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

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

        run_hook PRE_BRINGOVER

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

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

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

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

        run_hook POST_BRINGOVER

        check_closed_bins

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

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

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

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

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

#
# Report the compiler versions.
#

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

( cd $SRC
  for target in cc-version cc64-version java-version; do
        echo
        #
        # Put statefile somewhere we know we can write to rather than trip
        # over a read-only $srcroot.
        #
        rm -f $TMPDIR/make-state
        export SRC
        if $MAKE -K $TMPDIR/make-state -e $target 2>/dev/null; then
                continue
        fi
        touch $TMPDIR/nocompiler
  done
  echo
) | tee -a $build_environ_file >> $LOGFILE
```

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

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

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

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

        build_tools ${TOOLS_PROTO}
        if (( $? != 0 )); then
                build_ok=n
        else
        if [[ $? != 0  && "$t_FLAG" = y ]]; then
                use_tools $TOOLS_PROTO
        fi
fi

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

normal_build

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


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

if [ "$build_ok" = "y" ]; then
        echo "\n==== Creating protolist system file at ‘date‘ ====" \
                >> $LOGFILE
        protolist $checkroot > $ATLOG/proto_list_${MACH}
        echo "==== protolist system file created at ‘date‘ ====\n" \
                >> $LOGFILE
        if [ "$N_FLAG" != "y" ]; then

                E1=
                f1=
                for f in $f1; do
                        if [ -f "$f" ]; then
                                E1="$E1 -e $f"
                        fi
                done

                E2=
                f2=
                if [ -d "$SRC/pkg" ]; then
                        f2="$f2 exceptions/packaging"
                fi

                for f in $f2; do
                        if [ -f "$f" ]; then
                                E2="$E2 -e $f"
                        fi
                done
        fi

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

        fi

        if [ "$N_FLAG" != "y" -a -f "$REF_PROTO_LIST" ]; then
                echo "\n==== Impact on proto area ====\n" >> $mail_msg_file
                if [ -n "$E2" ]; then
                        ELIST=$E2
                else
                        ELIST=$E1
                fi
                $PROTOCMPTERSE \
                        "Files in yesterday’s proto area, but not today’s:" \
                        "Files in today’s proto area, but not yesterday’s:" \
                        "Files that changed between yesterday and today:" \
                        ${ELIST} \
                        -d $REF_PROTO_LIST \
                        $ATLOG/proto_list_${MACH} \
                        >> $mail_msg_file
        fi
fi

if [ "$u_FLAG" = "y"  -a "$build_ok" = "y" -a "$build_extras_ok" = "y" ]; then
if [ "$u_FLAG" = "y"  -a "$build_ok" = "y" ]; then
        staffer cp $ATLOG/proto_list_${MACH} \
                $PARENT_WS/usr/src/proto_list_${MACH}
fi

# Update parent proto area if necessary. This is done now
# so that the proto area has either DEBUG or non-DEBUG kernels.
# Note that this clears out the lock file, so we can dispense with
# the variable now.
if [ "$U_FLAG" = "y" -a "$build_ok" = "y" ]; then
```

```
        echo "\n==== Copying proto area to $NIGHTLY_PARENT_ROOT ====\n" | \
            tee -a $LOGFILE >> $mail_msg_file
        rm -rf $NIGHTLY_PARENT_ROOT/*
        unset Ulockfile
        mkdir -p $NIGHTLY_PARENT_ROOT
        if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
                ( cd $ROOT; tar cf - . |
                    ( cd $NIGHTLY_PARENT_ROOT;  umask 0; tar xpf - ) ) 2>&1 |
                    tee -a $mail_msg_file >> $LOGFILE
        fi
        if [[ "$MULTI_PROTO" = yes && "$F_FLAG" = n ]]; then
                rm -rf $NIGHTLY_PARENT_ROOT-nd/*
                mkdir -p $NIGHTLY_PARENT_ROOT-nd
                cd $ROOT-nd
                ( tar cf - . |
                    ( cd $NIGHTLY_PARENT_ROOT-nd; umask 0; tar xpf - ) ) 2>&1 |
                    tee -a $mail_msg_file >> $LOGFILE
        fi
        if [ -n "${NIGHTLY_PARENT_TOOLS_ROOT}" ]; then
                echo "\n==== Copying tools proto area to $NIGHTLY_PARENT_TOOLS_R
                    tee -a $LOGFILE >> $mail_msg_file
                rm -rf $NIGHTLY_PARENT_TOOLS_ROOT/*
                mkdir -p $NIGHTLY_PARENT_TOOLS_ROOT
                if [[ "$MULTI_PROTO" = no || "$D_FLAG" = y ]]; then
                        ( cd $TOOLS_PROTO; tar cf - . |
                            ( cd $NIGHTLY_PARENT_TOOLS_ROOT;
                            umask 0; tar xpf - ) ) 2>&1 |
                            tee -a $mail_msg_file >> $LOGFILE
                fi
        fi
fi

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

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

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

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

                # Produce interface description for the proto. Report errors.
                interface_check -o -w $elf_ddir -f object_list \
                        -i interface -E interface.err
                if [[ -s $elf_ddir/interface.err ]]; then
                        tee -a $LOGFILE < $elf_ddir/interface.err \
                            >> $mail_msg_file
                        build_extras_ok=n
#endif /* ! codereview */
```

```
                fi

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

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

                        if [[ -f $base_ifile ]]; then
                                interface_cmp -d -o $base_ifile \
                                    $elf_ddir/interface > $elf_ddir/interface.cm
                                if [[ -s $elf_ddir/interface.cmp ]]; then
                                        echo | tee -a $LOGFILE >> $mail_msg_file
                                        tee -a $LOGFILE < \
                                            $elf_ddir/interface.cmp \
                                            >> $mail_msg_file
                                        build_extras_ok=n
#endif /* ! codereview */
                                fi
                        else
                                echo "baseline not available. comparison" \
                                    "skipped" | \
                                    tee -a $LOGFILE >> $mail_msg_file
                        fi

                fi
        fi

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

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

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

                # Report errors
                if [[ -s $elf_ddir/runtime.err ]]; then
                        tee -a $LOGFILE < $elf_ddir/runtime.err \
                                >> $mail_msg_file
                        build_extras_ok=n
#endif /* ! codereview */
                fi

                # If there is an ELF-data directory from a previous build,
```

```
                # then diff the attr files. These files contain information
                # about dependencies, versioning, and runpaths. There is some
                # overlap with the ABI checking done above, but this also
                # flushes out non-ABI interface differences along with the
                # other information.
                echo "\n==== Diff ELF runtime attributes" \
                    "(since last build) ====\n" | \
                    tee -a $LOGFILE >> $mail_msg_file >> $mail_msg_file

                if [[ -f $elf_ddir.ref/runtime.attr ]]; then
                        diff $elf_ddir.ref/runtime.attr \
                                $elf_ddir/runtime.attr \
                                >> $mail_msg_file
                fi
        fi

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

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

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

# DEBUG lint of kernel begins

if [ "$i_CMD_LINE_FLAG" = "n" -a "$l_FLAG" = "y" ]; then
        if [ "$LINTDIRS" = "" ]; then
                # LINTDIRS="$SRC/uts y $SRC/stand y $SRC/psm y"
                LINTDIRS="$SRC y"
        fi
        set $LINTDIRS
        while [ $# -gt 0 ]; do
                dolint $1 $2; shift; shift
        done
else
        echo "\n==== No '$MAKE lint' ====\n" >> $LOGFILE
fi

# "make check" begins

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

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

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

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

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

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

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

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

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

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

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

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

        abspkg=
        for d in $abssrcdirs; do
                if [ -d "$d/pkg" ]; then
                        abspkg="$abspkg $d"
                fi
        done
```

```
        if [ -n "$abspkg" ]; then
                for d in "$abspkg"; do
                        ( cd $d/pkg ; $MAKE -e pmodes ) >> $mail_msg_file
                done
        fi
fi

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

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

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

typeset -i10 hours
typeset -Z2 minutes
typeset -Z2 seconds

elapsed_time=$SECONDS
((hours = elapsed_time / 3600 ))
((minutes = elapsed_time / 60  % 60))
((seconds = elapsed_time % 60))

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

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

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

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

        for unreffile; do
                comm -12 ${TMPDIR}/unref.merge "$unreffile" > ${TMPDIR}/unref.$$
                mv ${TMPDIR}/unref.$$ ${TMPDIR}/unref.merge
        done

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

#
```

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

****
    4995 Sun Jan 26 22:03:17 2014
new/usr/src/tools/scripts/onbld_elfmod_vertype.pm
4519 ABI checking needs to adapt to modern times, run by default
****

```
package onbld_elfmod_vertype;

#
# 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) 2010, Oracle and/or its affiliates. All rights reserved.
#

#
# This perl module implements the rules used to categorize ELF versions
# for the core Solaris OS and related code repositories. Although this
# code fits logically into the onbld_elfmod module, it is maintained as
# a separate module in order to allow maintainers of other code to provide
# an implementation appropriate to their local conventions.
#
# By isolating the codebase specific details of ELF version names in this
# module and reporting the results via a fixed interface, we allow
# interface_check and interface_cmp to be written in a way that isolates
# them from the specific names that apply to a given body of code.
# Those tools allow you to substitute your own module in place of this one
# to customize their behavior.
#
# The types of versions understood by interface_check and interface_cmp
# fall into the following categories:
#
#       NUMBERED:       A public version that follows the standard numbering
#                       convention of a known prefix (e.g. SUNW_), followed
#                       by 2 or 3 dot separated numeric values:
#
#                               <PREFIX>major.minor[.micro]
#
#       PLAIN:          A public version that may or may not contain
#                       numeric characters, but for which numeric characters
#                       are not treated as such.
#
#       SONAME:         Base version with the same name as the object SONAME
#
#       PRIVATE:        A private version that follows the same rules as PLAIN.
#
#       UNKNOWN:        A version string that does not fit any of the
#                       above categories
#
```

```
# The above categories are generic, in the sense that they apply to any
# code base. However, each code base will have different well known prefix
# and name strings that map to these categories. The purpose of this module
# is to map these special well known strings to the category they represent
# for the code base in question.
#

use strict;


## Category(Version, Soname)
#
# Return an array containing the category of ELF version represented
# by the given Version, and other category dependent information.
#
# entry:
#       Version - Version string to examine
#       Soname - Empty ('') string, or SONAME of object that contains the
#               given version if it is available. In some environments,
#               the valid versions depend on the particular object in
#               question. This argument can be used to customize the
#               results of this routine based on the object being analyzed.
#
# exit:
#       This routine returns an array to describe the type of version
#       encountered. Element [0] is always a string token that gives one
#       of the version categories described in the module header comment.
#       For types other than NUMBERED, this is the only element in the
#       return array.
#
#       NUMBERED versions receive a return array with additional values
#       describing the version:
#
#               ( 'NUMBERED', cnt, prefix, major, minor[, micro])
#
#       If the version has 3 numberic values, cnt is 3, and micro
#       is present. If there are 2 numeric values, cnt is 2, and micro
#       is omitted.
#
sub Category {
        my ($Ver, $Soname) = @_;

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

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

        # The link-editor creates "base" versions using the SONAME of the
        # object to contain  linker generated symbols (_etext, _edata, etc.).
        return ('SONAME')
            if ($Ver eq $Soname) && ($Soname ne '');

        # The Solaris convention is to use SUNWprivate to indicate
        # private versions. SUNWprivate can have a numeric suffix, but
        # the number is not significant for ELF versioning other than
        # being part of a unique name.
        return ('PRIVATE')
```

```
            if ($Ver =~ /^(SUNW|ILLUMOS)private(_[0-9.]+)?$/);
            if ($Ver =~ /^SUNWprivate(_[0-9.]+)?$/);

        # Anything else is a version we don’t recognize.
        return (’UNKNOWN’);
}
```

______***unchanged_portion_omitted_***