```
***********************************************************
    2490 Tue Feb 10 16:43:45 2015
new/usr/src/lib/print/Makefile
5605 Disable IPP printing like SMB printing can be disabled
***********************************************************
```

```
#
# 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 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

SUBDIRS = \
        libprint \
        libpapi-common \
        libpapi-dynamic \
        libpapi-lpd \
        libipp-core \
        libhttp-core \
        libpapi-ipp \
        libipp-listener
        libipp-listener \
        mod_ipp

$(ENABLE_IPP_PRINTING) SUBDIRS +=       mod_ipp

all :=                  TARGET = all
clean :=                TARGET = clean
clobber :=              TARGET = clobber
install :=              TARGET = install
install_h :=              TARGET = install_h
lint :=                 TARGET = lint

include $(SRC)/Makefile.master

TEXT_DOMAIN=    SUNW_OST_OSLIB
POFILE= print-lib.po

.KEEP_STATE:

all:    $(TXTS) $(SUBDIRS)

#
# Each message catalog file is generated in each sub
# directory and copied to the usr/src/cmd/lp/ directory.
# Those message catalog files are consolidated into one
```



```
# message catalog file.  The consolidated one will be copied
# into the $(ROOT)/catalog/SUNW_OST_OSCMD/ directory.
#

_msg:   $(MSGDOMAIN)
        @$(RM) $(POFILE)
        $(XGETTEXT) -s '/bin/find . -type d -name SCCS -prune -o -type f -name '
        @/bin/cat messages.po | sed '/domain/d' > $(POFILE)
        @$(RM) messages.po
        $(RM)  $(MSGDOMAIN)/$(POFILE)
        /bin/cp $(POFILE) $(MSGDOMAIN)

install: $(ROOTDIRS) $(ROOTSYMLINKDIRS) $(SUBDIRS)

install_h clean strip lint: $(SUBDIRS)

clobber: $(SUBDIRS) local_clobber

local_clobber:
        $(RM) $(CLOBBERFILES) $(POFILE)

$(SUBDIRS):     FRC
        @cd $@; pwd; $(MAKE) $(TARGET)

FRC:

include $(SRC)/Makefile.msg.targ

# Dependencies
libpapi-dynamic:        libpapi-common
libpapi-lpd:            libpapi-dynamic
libipp-core:            libpapi-common
libpapi-ipp:            libpapi-common libipp-core libhttp-core
libipp-listener:        libpapi-dynamic libipp-core
mod_ipp:                libipp-listener
```

```
***********************************************************
   24593 Tue Feb 10 16:43:45 2015
new/usr/src/pkg/Makefile
5605 Disable IPP printing like SMB printing can be disabled
***********************************************************
```

```
#
# 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.
# Copyright 2015, OmniTI Computer Consulting, Inc. All rights reserved.
#

include $(SRC)/Makefile.master
include $(SRC)/Makefile.buildnum

#
# Make sure we're getting a consistent execution environment for the
# embedded scripts.
#
SHELL= /usr/bin/ksh93

#
# To suppress package dependency generation on any system, regardless
# of how it was installed, set SUPPRESSPKGDEP=true in the build
# environment.
#
SUPPRESSPKGDEP= false

#
# Comment this line out or set "PKGDEBUG=" in your build environment
# to get more verbose output from the make processes in usr/src/pkg
#
PKGDEBUG= @

#
# Cross platform packaging notes
#
# By default, we package the proto area from the same architecture as
# the packaging build.  In other words, if you're running nightly or
# bldenv on an x86 platform, it will take objects from the x86 proto
# area and use them to create x86 repositories.
#
# If you want to create repositories for an architecture that's
# different from $(uname -p), you do so by setting PKGMACH in your
# build environment.
#
# For this to work correctly, the following must all happen:
```



```
#
#   1. You need the desired proto area, which you can get either by
#      doing a gatekeeper-style build with the -U option to
#      nightly(1), or by using rsync.  If you don't do this, you will
#      get packaging failures building all packages, because pkgsend
#      is unable to find the required binaries.
#   2. You need the desired tools proto area, which you can get in the
#      same ways as the normal proto area.  If you don't do this, you
#      will get packaging failures building onbld, because pkgsend is
#      unable to find the tools binaries.
#   3. The remainder of this Makefile should never refer directly to
#      $(MACH).  Instead, $(PKGMACH) should be used whenever an
#      architecture-specific path or token is needed.  If this is done
#      incorrectly, then packaging will fail, and you will see the
#      value of $(uname -p) instead of the value of $(PKGMACH) in the
#      commands that fail.
#   4. Each time a rule in this Makefile invokes $(MAKE), it should
#      pass PKGMACH=$(PKGMACH) explicitly on the command line.  If
#      this is done incorrectly, then packaging will fail, and you
#      will see the value of $(uname -p) instead of the value of
#      $(PKGMACH) in the commands that fail.
#
# Refer also to the convenience targets defined later in this
# Makefile.
#
PKGMACH=        $(MACH)

#
# ROOT, TOOLS_PROTO, and PKGARCHIVE should be set by nightly or
# bldenv.  These macros translate them into terms of $PKGMACH, instead
# of $ARCH.
#
PKGROOT.cmd=    print $(ROOT) | sed -e s:/root_$(MACH):/root_$(PKGMACH):
PKGROOT=        $(PKGROOT.cmd:sh)
TOOLSROOT.cmd=  print $(TOOLS_PROTO) | sed -e s:/root_$(MACH):/root_$(PKGMACH):
TOOLSROOT=      $(TOOLSROOT.cmd:sh)
PKGDEST.cmd=    print $(PKGARCHIVE) | sed -e s:/$(MACH)/:/$(PKGMACH)/:
PKGDEST=        $(PKGDEST.cmd:sh)

EXCEPTIONS= packaging

PKGMOGRIFY= pkgmogrify

#
# Always build the redistributable repository, but only build the
# nonredistributable bits if we have access to closed source.
#
# Some objects that result from the closed build are still
# redistributable, and should be packaged as part of an open-only
# build.  Access to those objects is provided via the closed-bins
# tarball.  See usr/src/tools/scripts/bindrop.sh for details.
#
REPOS= redist

#
# The packages directory will contain the processed manifests as
# direct build targets and subdirectories for package metadata extracted
# incidentally during manifest processing.
#
# Nothing underneath $(PDIR) should ever be managed by SCM.
#
PDIR= packages.$(PKGMACH)

#
# The tools proto must be specified for dependency generation.
# Publication from the tools proto area is managed in the
```

```
128 # publication rule.
129 #
130 $(PDIR)/developer-build-onbld.dep:= PKGROOT= $(TOOLSROOT)

132 PKGPUBLISHER= $(PKGPUBLISHER_REDIST)

134 #
135 # To get these defaults, manifests should simply refer to $(PKGVERS).
136 #
137 PKGVERS_COMPONENT= 0.$(RELEASE)
138 PKGVERS_BUILTON= $(RELEASE)
139 PKGVERS_BRANCH= 0.$(ONNV_BUILDNUM)
140 PKGVERS= $(PKGVERS_COMPONENT),$(PKGVERS_BUILTON)-$(PKGVERS_BRANCH)

142 #
143 # The ARCH32 and ARCH64 macros are used in the manifests to express
144 # architecture-specific subdirectories in the installation paths
145 # for isaexec'd commands.
146 #
147 # We can't simply use $(MACH32) and $(MACH64) here, because they're
148 # only defined for the build architecture.  To do cross-platform
149 # packaging, we need both values.
150 #
151 i386_ARCH32= i86
152 sparc_ARCH32= sparcv7
153 i386_ARCH64= amd64
154 sparc_ARCH64= sparcv9

156 #
157 # macros and transforms needed by pkgmogrify
158 #
159 # If you append to this list using target-specific assignments (:=),
160 # be very careful that the targets are of the form $(PDIR)/pkgname.  If
161 # you use a higher level target, or a package list, you'll trigger a
162 # complete reprocessing of all manifests because they'll fail command
163 # dependency checking.
164 #
165 PM_TRANSFORMS= common_actions publish restart_fmri facets defaults \
166         extract_metadata
167 PM_INC= transforms manifests

169 PKGMOG_DEFINES= \
170         i386_ONLY=$(POUND_SIGN) \
171         sparc_ONLY=$(POUND_SIGN) \
172         $(PKGMACH)_ONLY= \
173         ARCH=$(PKGMACH) \
174         ARCH32=$($(PKGMACH)_ARCH32) \
175         ARCH64=$($(PKGMACH)_ARCH64) \
176         PKGVERS_COMPONENT=$(PKGVERS_COMPONENT) \
177         PKGVERS_BUILTON=$(PKGVERS_BUILTON) \
178         PKGVERS_BRANCH=$(PKGVERS_BRANCH) \
179         PKGVERS=$(PKGVERS) \
180         PERL_ARCH=$(PERL_ARCH) \
181         PERL_VERSION=$(PERL_VERSION) \
182         PERL_PKGVERS=$(PERL_PKGVERS)

184 PKGDEP_TOKENS_i386= \
185         'PLATFORM=i86hvm' \
186         'PLATFORM=i86pc' \
187         'PLATFORM=i86xpv' \
188         'ISALIST=amd64' \
189         'ISALIST=i386'
190 PKGDEP_TOKENS_sparc= \
191         'PLATFORM=sun4u' \
192         'PLATFORM=sun4v' \
193         'ISALIST=sparcv9' \
```



```
194         'ISALIST=sparc'
195 PKGDEP_TOKENS= $(PKGDEP_TOKENS_$(PKGMACH))

197 #
198 # The package lists are generated with $(PKGDEP_TYPE) as their
199 # dependency types, so that they can be included by either an
200 # incorporation or a group package.
201 #
202 $(PDIR)/osnet-redist.mog := PKGDEP_TYPE= require
203 $(PDIR)/osnet-incorporation.mog:= PKGDEP_TYPE= incorporate

205 PKGDEP_INCORP= \
206         depend fmri=consolidation/osnet/osnet-incorporation type=require

208 #
209 # All packaging build products should go into $(PDIR), so they don't
210 # need to be included separately in CLOBBERFILES.
211 #
212 CLOBBERFILES= $(PDIR) proto_list_$(PKGMACH) install-$(PKGMACH).out \
213         license-list

215 #
216 # By default, PKGS will list all manifests.  To build and/or publish a
217 # subset of packages, override this on the command line or in the
218 # build environment and then reference (implicitly or explicitly) the all
219 # or install targets.
220 #

222 # Cheesy two-stage set is to enable the use of ENABLE_IPP_PRINTING.
223 MANIFESTS :sh= (cd manifests; print *.mf | \
224         sed 's/print-lp-ipp-ipp-listener.mf//')
225 $(ENABLE_IPP_PRINTING)MANIFESTS :sh= (cd manifests; print *.mf)

221 MANIFESTS :sh= (cd manifests; print *.mf)
227 PKGS= $(MANIFESTS:%.mf=%)
228 DEP_PKGS= $(PKGS:%=$(PDIR)/%.dep)
229 PROC_PKGS= $(PKGS:%=$(PDIR)/%.mog)

231 #
232 # Track the synthetic manifests separately so we can properly express
233 # build rules and dependencies.  The synthetic and real packages use
234 # different sets of transforms and macros for pkgmogrify.
235 #
236 SYNTH_PKGS= osnet-incorporation osnet-redist
237 DEP_SYNTH_PKGS= $(SYNTH_PKGS:%=$(PDIR)/%.dep)
238 PROC_SYNTH_PKGS= $(SYNTH_PKGS:%=$(PDIR)/%.mog)

240 #
241 # Root of pkg image to use for dependency resolution
242 # Normally / on the machine used to build the binaries
243 #
244 PKGDEP_RESOLVE_IMAGE = /

246 #
247 # For each package, we determine the target repository based on
248 # manifest-embedded metadata.  Because we make that determination on
249 # the fly, the publication target cannot be expressed as a
250 # subdirectory inside the unknown-by-the-makefile target repository.
251 #
252 # In order to limit the target set to real files in known locations,
253 # we use a ".pub" file in $(PDIR) for each processed manifest, regardless
254 # of content or target repository.
255 #
256 PUB_PKGS= $(SYNTH_PKGS:%=$(PDIR)/%.pub) $(PKGS:%=$(PDIR)/%.pub)

258 #
```

```
259 # Any given repository- and status-specific package list may be empty,
260 # but we can only determine that dynamically, so we always generate all
261 # lists for each repository we're building.
262 #
263 # The meanings of each package status are as follows:
264 #
265 #       PKGSTAT         meaning
266 #       ----------      ------------------------------------------------------
267 #       noincorp        Do not include in incorporation or group package
268 #       obsolete        Include in incorporation, but not group package
269 #       renamed         Include in incorporation, but not group package
270 #       current         Include in incorporation and group package
271 #
272 # Since the semantics of the "noincorp" package status dictate that
273 # such packages are not included in the incorporation or group packages,
274 # there is no need to build noincorp package lists.
275 #
276 PKGLISTS= \
277         $(REPOS:%=$(PDIR)/packages.%.current) \
278         $(REPOS:%=$(PDIR)/packages.%.renamed) \
279         $(REPOS:%=$(PDIR)/packages.%.obsolete)

281 .KEEP_STATE:

283 .PARALLEL: $(PKGS) $(PROC_PKGS) $(DEP_PKGS) \
284         $(PROC_SYNTH_PKGS) $(DEP_SYNTH_PKGS) $(PUB_PKGS)

286 #
287 # For a single manifest, the dependency chain looks like this:
288 #
289 #       raw manifest (mypkg.mf)
290 #               |
291 #               | use pkgmogrify to process raw manifest
292 #               |
293 #       processed manifest (mypkg.mog)
294 #               |
295 #          *    | use pkgdepend generate to generate dependencies
296 #               |
297 #       manifest with TBD dependencies (mypkg.dep)
298 #               |
299 #          %    | use pkgdepend resolve to resolve dependencies
300 #               |
301 #       manifest with dependencies resolved (mypkg.res)
302 #               |
303 #               | use pkgsend to publish the package
304 #               |
305 #       placeholder to indicate successful publication (mypkg.pub)
306 #
307 # * This may be suppressed via SUPPRESSPKGDEP.  The resulting
308 #   packages will install correctly, but care must be taken to
309 #   install all dependencies, because pkg will not have the input
310 #   it needs to determine this automatically.
311 #
312 # % This is included in this diagram to make the picture complete, but
313 #   this is a point of synchronization in the build process.
314 #   Dependency resolution is actually done once on the entire set of
315 #   manifests, not on a per-package basis.
316 #
317 # The full dependency chain for generating everything that needs to be
318 # published, without actually publishing it, looks like this:
319 #
320 #       processed synthetic packages
321 #               |               |
322 #       package lists       synthetic package manifests
323 #               |
324 #       processed real packages
```



```
325 #           |           |
326 #       package dir     real package manifests
327 #
328 # Here, each item is a set of real or synthetic packages.  For this
329 # portion of the build, no reference is made to the proto area.  It is
330 # therefore suitable for the "all" target, as opposed to "install."
331 #
332 # Since each of these steps is expressed explicitly, "all" need only
333 # depend on the head of the chain.
334 #
335 # From the end of manifest processing, the publication dependency
336 # chain looks like this:
337 #
338 #               repository metadata (catalogs and search indices)
339 #                       |
340 #                       | pkgrepo refresh
341 #                       |
342 #               published packages
343 #                 |             |
344 #                 |             | pkgsend publish
345 #                 |             |
346 #         repositories      resolved dependencies
347 #                 |                     |
348 # pkgsend         |                     | pkgdepend resolve
349 # create-repository                     |
350 #                 |             generated dependencies
351 #        repo directories               |
352 #                                       | pkgdepend
353 #                                       |
354 #                                 processed manifests
355 #

357 ALL_TARGETS= $(PROC_SYNTH_PKGS) proto_list_$(PKGMACH)

359 all: $(ALL_TARGETS)

361 #
362 # This will build the directory to contain the processed manifests
363 # and the metadata symlinks.
364 #
365 $(PDIR):
366         @print "Creating $(@)"
367         $(PKGDEBUG)$(INS.dir)

369 #
370 # This rule resolves dependencies across all published manifests.
371 #
372 # We shouldn't have to ignore the error from pkgdepend, but until
373 # 16012 and its dependencies are resolved, pkgdepend will always exit
374 # with an error.
375 #
376 $(PDIR)/gendeps: $(DEP_SYNTH_PKGS) $(DEP_PKGS)
377         -$(PKGDEBUG)if [ "$(SUPPRESSPKGDEP)" = "true" ]; then \
378                 print "Suppressing dependency resolution"; \
379                 for p in $(DEP_PKGS:%.dep=%); do \
380                         $(CP) $$p.dep $$p.res; \
381                 done; \
382         else \
383                 print "Resolving dependencies"; \
384                 pkgdepend -R $(PKGDEP_RESOLVE_IMAGE) resolve \
385                     -m $(DEP_SYNTH_PKGS) $(DEP_PKGS); \
386                 for p in $(DEP_SYNTH_PKGS:%.dep=%) $(DEP_PKGS:%.dep=%); do \
387                         if [ "$$(print $$p.metadata.*)" = \
388                              "$$(print $$p.metadata.noincorp.*)" ]; \
389                         then \
390                                 print "Removing dependency versions from $$p"; \
```

```
391                                 $(PKGMOGRIFY) $(PKGMOG_VERBOSE) \
392                                     -O $$p.res -I transforms \
393                                     strip_versions $$p.dep.res; \
394                                 $(RM) $$p.dep.res; \
395                         else \
396                                 $(MV) $$p.dep.res $$p.res; \
397                         fi; \
398                 done; \
399         fi
400         $(PKGDEBUG)$(TOUCH) $(@)

402 install: $(ALL_TARGETS) repository-metadata

404 repository-metadata: publish_pkgs
405         $(PKGDEBUG)for r in $(REPOS); do \
406                 pkgrepo refresh -s $(PKGDEST)/repo.$$r; \
407         done

409 #
410 # Since we create zero-length processed manifests for a graceful abort
411 # from pkgmogrify, we need to detect that here and make no effort to
412 # publish the package.
413 #
414 # For all other packages, we publish them regardless of status.  We
415 # derive the target repository as a component of the metadata-derived
416 # symlink for each package.
417 #
418 publish_pkgs: $(REPOS:%=$(PKGDEST)/repo.%) $(PDIR)/gendeps .WAIT $(PUB_PKGS)

420 #
421 # Before publishing, we want to pull the license files from $CODEMGR_WS
422 # into the proto area.  This allows us to NOT pass $SRC (or
423 # $CODEMGR_WS) as a basedir for publication.
424 #
425 $(PUB_PKGS): stage-licenses

427 #
428 # Initialize the empty on-disk repositories
429 #
430 $(REPOS:%=$(PKGDEST)/repo.%):
431         @print "Initializing $(@F)"
432         $(PKGDEBUG)$(INS.dir)
433         $(PKGDEBUG)pkgsend -s file://$(@) create-repository \
434                 --set-property publisher.prefix=$(PKGPUBLISHER)

436 #
437 # rule to process real manifests
438 #
439 # To allow redistributability and package status to change, we must
440 # remove not only the actual build target (the processed manifest), but
441 # also the incidental ones (the metadata-derived symlinks).
442 #
443 # If pkgmogrify exits cleanly but fails to create the specified output
444 # file, it means that it encountered an abort directive.  That means
445 # that this package should not be published for this particular build
446 # environment.  Since we can't prune such packages from $(PKGS)
447 # retroactively, we need to create an empty target file to keep make
448 # from trying to rebuild it every time.  For these empty targets, we
449 # do not create metadata symlinks.
450 #
451 # Automatic dependency resolution to files is also done at this phase of
452 # processing.  The skipped packages are skipped due to existing bugs
453 # in pkgdepend.
454 #
455 # The incorporation dependency is tricky: it needs to go into all
456 # current and renamed manifests (ie all incorporated packages), but we
```



```
457 # don't know which those are until after we run pkgmogrify.  So
458 # instead of expressing it as a transform, we tack it on ex post facto.
459 #
460 # Implementation notes:
461 #
462 # - The first $(RM) must not match other manifests, or we'll run into
463 #   race conditions with parallel manifest processing.
464 #
465 # - The make macros [ie $(MACRO)] are evaluated when the makefile is
466 #   read in, and will result in a fixed, macro-expanded rule for each
467 #   target enumerated in $(PROC_PKGS).
468 #
469 # - The shell variables (ie $$VAR) are assigned on the fly, as the rule
470 #   is executed.  The results may only be referenced in the shell in
471 #   which they are assigned, so from the perspective of make, all code
472 #   that needs these variables needs to be part of the same line of
473 #   code.  Hence the use of command separators and line continuation
474 #   characters.
475 #
476 # - The extract_metadata transforms are designed to spit out shell
477 #   variable assignments to stdout.  Those are published to the
478 #   .vars temporary files, and then used as input to the eval
479 #   statement.  This is done in stages specifically so that pkgmogrify
480 #   can signal failure if the manifest has a syntactic or other error.
481 #   The eval statement should begin with the default values, and the
482 #   output from pkgmogrify (if any) should be in the form of a
483 #   variable assignment to override those defaults.
484 #
485 # - When this rule completes execution, it must leave an updated
486 #   target file ($@) in place, or make will reprocess the package
487 #   every time it encounters it as a dependency.  Hence the "touch"
488 #   statement to ensure that the target is created, even when
489 #   pkgmogrify encounters an abort in the publish transforms.
490 #

492 .SUFFIXES: .mf .mog .dep .res .pub

494 $(PDIR)/%.mog: manifests/%.mf
495         @print "Processing manifest $(<F)"
496         @env PKGFMT_OUTPUT=v1 pkgfmt -c $<
497         $(PKGDEBUG)$(RM) $(@) $(@:%.mog=%) $(@:%.mog=%.nodepend) \
498                 $(@:%.mog=%.lics) $(PDIR)/$(@F:%.mog=%).metadata.* $(@).vars
499         $(PKGDEBUG)$(PKGMOGRIFY) $(PKGMOG_VERBOSE) $(PM_INC:%= -I %) \
500                 $(PKGMOG_DEFINES:%=-D %) -P $(@).vars -O $(@) \
501                 $(<) $(PM_TRANSFORMS)
502         $(PKGDEBUG)eval REPO=redist PKGSTAT=current NODEPEND=$(SUPPRESSPKGDEP) \
503                 '$(CAT) -s $(@).vars'; \
504         if [ -f $(@) ]; then \
505                 if [ "$$NODEPEND" != "false" ]; then \
506                         $(TOUCH) $(@:%.mog=%.nodepend); \
507                 fi; \
508                 $(LN) -s $(@F) \
509                         $(PDIR)/$(@F:%.mog=%).metadata.$$PKGSTAT.$$REPO; \
510                 if [ \( "$$PKGSTAT" = "current" \) -o \
511                      \( "$$PKGSTAT" = "renamed" \) ]; \
512                         then print $(PKGDEP_INCORP) >> $(@); \
513                 fi; \
514                 print $$LICS > $(@:%.mog=%.lics); \
515         else \
516                 $(TOUCH) $(@) $(@:%.mog=%.lics); \
517         fi
518         $(PKGDEBUG)$(RM) $(@).vars

520 $(PDIR)/%.dep: $(PDIR)/%.mog
521         @print "Generating dependencies for $(<F)"
522         $(PKGDEBUG)$(RM) $(@)
```

```
523         $(PKGDEBUG)if [ ! -f $(@:%.dep=%.nodepend) ]; then \
524                 pkgdepend generate -m $(PKGDEP_TOKENS:%=-D %) $(<) \
525                         $(PKGROOT) > $(@); \
526         else \
527                 $(CP) $(<) $(@); \
528         fi

530 #
531 # The full chain implies that there should be a .dep.res suffix rule,
532 # but dependency generation is done on a set of manifests, rather than
533 # on a per-manifest basis.  Instead, see the gendeps rule above.
534 #

536 $(PDIR)/%.pub: $(PDIR)/%.res
537         $(PKGDEBUG)m=$$(basename $(@:%.pub=%).metadata.*); \
538         r=$${m#$(@F:%.pub=%.metadata.)+(?).}; \
539         if [ -s $(<) ]; then \
540                 print "Publishing $(@F:%.pub=%) to $$r repository"; \
541                 pkgsend -s file://$(PKGDEST)/repo.$$r publish \
542                     -d $(PKGROOT) -d $(TOOLSROOT) \
543                     -d license_files -d $(PKGROOT)/licenses \
544                     --fmri-in-manifest --no-index --no-catalog $(<) \
545                     > /dev/null; \
546         fi; \
547         $(TOUCH) $(@);

549 #
550 # rule to build the synthetic manifests
551 #
552 # This rule necessarily has PKGDEP_TYPE that changes according to
553 # the specific synthetic manifest.  Rather than escape command
554 # dependency checking for the real manifest processing, or failing to
555 # express the (indirect) dependency of synthetic manifests on real
556 # manifests, we simply split this rule out from the one above.
557 #
558 # The implementation notes from the previous rule are applicable
559 # here, too.
560 #
561 $(PROC_SYNTH_PKGS): $(PKGLISTS) $$(@F:%.mog=%.mf)
562         @print "Processing synthetic manifest $(@F:%.mog=%.mf)"
563         $(PKGDEBUG)$(RM) $(@) $(PDIR)/$(@F:%.mog=%).metadata.* $(@).vars
564         $(PKGDEBUG)$(PKGMOGRIFY) $(PKGMOG_VERBOSE) -I transforms -I $(PDIR) \
565                 $(PKGMOG_DEFINES:%=-D %) -D PKGDEP_TYPE=$(PKGDEP_TYPE) \
566                 -P $(@).vars -O $(@) $(@F:%.mog=%.mf) \
567                 $(PM_TRANSFORMS) synthetic
568         $(PKGDEBUG)eval REPO=redist PKGSTAT=current `$(CAT) -s $(@).vars`; \
569         if [ -f $(@) ]; then \
570                 $(LN) -s $(@F) \
571                         $(PDIR)/$(@F:%.mog=%).metadata.$$PKGSTAT.$$REPO; \
572         else \
573                 $(TOUCH) $(@); \
574         fi
575         $(PKGDEBUG)$(RM) $(@).vars

577 $(DEP_SYNTH_PKGS): $$(@:%.dep=%.mog)
578         @print "Skipping dependency generation for $(@F:%.dep=%)"
579         $(PKGDEBUG)$(CP) $(@:%.dep=%.mog) $(@)

581 clean:

583 clobber: clean
584         $(RM) -r $(CLOBBERFILES)

586 #
587 # This rule assumes that all links in the $PKGSTAT directories
588 # point to valid manifests, and will fail the make run if one
```



```
589 # does not contain an fmri.
590 #
591 # We do this in the BEGIN action instead of using pattern matching
592 # because we expect the fmri to be at or near the first line of each input
593 # file, and this way lets us avoid reading the rest of the file after we
594 # find what we need.
595 #
596 # We keep track of a failure to locate an fmri, so we can fail the
597 # make run, but we still attempt to process each package in the
598 # repo/pkgstat-specific subdir, in hopes of maybe giving some
599 # additional useful info.
600 #
601 # The protolist is used for bfu archive creation, which may be invoked
602 # interactively by the user.  Both protolist and PKGLISTS targets
603 # depend on $(PROC_PKGS), but protolist builds them recursively.
604 # To avoid collisions, we insert protolist into the dependency chain
605 # here.  This has two somewhat subtle benefits: it allows bfu archive
606 # creation to work correctly, even when -a was not part of NIGHTLY_OPTIONS,
607 # and it ensures that a protolist file here will always correspond to the
608 # contents of the processed manifests, which can vary depending on build
609 # environment.
610 #
611 $(PKGLISTS): $(PROC_PKGS)
612         $(PKGDEBUG)sdotr=$(@F:packages.%=%); \
613         r=$${sdotr%.+(?)}; s=$${sdotr#+(?).}; \
614         print "Generating $$r $$s package list"; \
615         $(RM) $(@); $(TOUCH) $(@); \
616         $(NAWK) 'BEGIN { \
617                 if (ARGC < 2) { \
618                         exit; \
619                 } \
620                 retcode = 0; \
621                 for (i = 1; i < ARGC; i++) { \
622                         do { \
623                                 e = getline f < ARGV[i]; \
624                         } while ((e == 1) && (f !~ /name=pkg.fmri/)); \
625                         close(ARGV[i]); \
626                         if (e == 1) { \
627                                 l = split(f, a, "="); \
628                                 print "depend fmri=" a[l], \
629                                         "type=$$(PKGDEP_TYPE)"; \
630                         } else { \
631                                 print "no fmri in " ARGV[i] >> "/dev/stderr"; \
632                                 retcode = 2; \
633                         } \
634                 } \
635                 exit retcode; \
636         }' `find $(PDIR) -type l -a \( $(PKGS:%=-name %.metadata.$$s.$$r -o) \
637                 -name NOSUCHFILE \)` >> $(@)

639 #
640 # rules to validate proto area against manifests, check for safe
641 # file permission modes, and generate a faux proto list
642 #
643 # For the check targets, the dependencies on $(PROC_PKGS) is specified
644 # as a subordinate make process in order to suppress output.
645 #
646 makesilent:
647         @$(MAKE) -e $(PROC_PKGS) PKGMACH=$(PKGMACH) \
648                 SUPPRESSPKGDEP=$(SUPPRESSPKGDEP) > /dev/null

650 #
651 # The .lics files were created during pkgmogrification, and list the
652 # set of licenses to pull from $SRC for each package.  Because
653 # licenses may be duplicated between packages, we uniquify them as
654 # well as aggregating them here.
```

```
655 #
656 license-list: makesilent
657         $(PKGDEBUG)( for l in `cat $(PROC_PKGS:%.mog=%.lics)`; \
658                 do print $$l; done ) | sort -u > $@

660 #
661 # Staging the license and description files in the proto area allows
662 # us to do proper unreferenced file checking of both license and
663 # description files without blanket exceptions, and to pull license
664 # content without reference to $CODEMGR_WS during publication.
665 #
666 stage-licenses: license-list FRC
667         $(PKGDEBUG)$(MAKE) -e -f Makefile.lic \
668                 PKGDEBUG=$(PKGDEBUG) LICROOT=$(PKGROOT)/licenses \
669                 `$(NAWK) '{ \
670                         print "$(PKGROOT)/licenses/" $$0; \
671                         print "$(PKGROOT)/licenses/" $$0 ".descrip"; \
672                 }' license-list` > /dev/null;

674 protocmp: makesilent
675         @validate_pkg -a $(PKGMACH) -v \
676                 $(EXCEPTIONS:%=-e $(CODEMGR_WS)/exception_lists/%) \
677                 -m $(PDIR) -p $(PKGROOT) -p $(TOOLSROOT)

679 pmodes: makesilent
680         @validate_pkg -a $(PKGMACH) -M -m $(PDIR) \
681                 -e $(CODEMGR_WS)/exception_lists/pmodes

683 check: protocmp pmodes

685 protolist: proto_list_$(PKGMACH)

687 proto_list_$(PKGMACH): $(PROC_PKGS)
688         @validate_pkg -a $(PKGMACH) -L -m $(PDIR) > $(@)

690 $(PROC_PKGS): $(PDIR)

692 #
693 # This is a convenience target to allow package names to function as
694 # build targets.  Generally, using it is only useful when iterating on
695 # development of a manifest.
696 #
697 # When processing a manifest, use the basename (without extension) of
698 # the package.  When publishing, use the basename with a ".pub"
699 # extension.
700 #
701 # Other than during manifest development, the preferred usage is to
702 # avoid these targets and override PKGS on the make command line and
703 # use the provided all and install targets.
704 #
705 $(PKGS) $(SYNTH_PKGS): $(PDIR)/$$(@:%=%.mog)

707 $(PKGS:%=%.pub) $(SYNTH_PKGS:%=%.pub): $(PDIR)/$$(@)

709 #
710 # This is a convenience target to resolve dependencies without publishing
711 # packages.
712 #
713 gendeps: $(PDIR)/gendeps

715 #
716 # These are convenience targets for cross-platform packaging.  If you
717 # want to build any of "the normal" targets for a different
718 # architecture, simply use "arch/target" as your build target.
719 #
720 # Since the most common use case for this is "install," the architecture
```



```
721 # specific install targets have been further abbreviated to elide "/install."
722 #
723 i386/% sparc/%:
724         $(MAKE) -e $(@F) PKGMACH=$(@D) SUPPRESSPKGDEP=$(SUPPRESSPKGDEP)

726 i386 sparc: $$(@)/install

728 FRC:
```