```
*********************************************************
   10036 Tue Feb 20 04:35:58 2018
new/exception_lists/check_rtime
9139 check_rtime should be able to forbid libraries
9140 check_rtime should learn libnsl is safe now
9141 check_rtime exceptions could be cleaner
*********************************************************
```

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

  22 #
  23 # Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
  24 # Copyright 2015, OmniTI Computer Consulting, Inc. All rights reserved.
  25 #

  27 # This file provides exceptions to the usual rules applied to ELF objects by
  28 # check_rtime. All strings are Perl regular expressions that are compared to
  29 # file paths. In addition to the standard Perl syntax, there is one extension:
  30 #
  31 #       MACH(dir)
  32 #
  33 # is expanded into a regular expression that matches the given
  34 # directory, or a 64-bit subdirectory of the directory with the
  35 # name of a 64-bit architecture. For example, MACH(lib) will match
  36 # any of the following:
  37 #
  38 #       lib
  39 #       lib/amd64
  40 #       lib/sparcv9


  43 # Directory hierarchies to skip completely
  44 SKIP            ^usr/lib/libc/                  # optimized libc
  45 SKIP            ^usr/lib/rcm/                   # 4426119
  46 SKIP            ^usr/perl5/                     # alan's taking care of these :-
  47 SKIP            ^usr/src/                       # no objects in source code

  49 # Individual files that we don't examine
  50 SKIP            ^boot/grub/bin/grub$
  51 # USIII specific extns. cause ldd noise on USII bld. m/c
  52 SKIP            ^usr/lib/fps/sun4u/UltraSPARC.*/fptest$
  53 SKIP            ^usr/MACH(lib)/lddstub$         # lddstub has no dependencies
  54 SKIP            ^usr/MACH(lib)/libssagent\.so\.1$       # 4328854
  55 SKIP            ^usr/lib/MACH(iconv)/geniconvtbl.so$    # 4384329

  57 # picl file exclusions (4385799)
  58 SKIP            ^usr/platform/.*/libpsvcplugin_psr\.so\.1
  59 SKIP            ^usr/platform/.*/libpsvcpolicy_psr\.so\.1
```



```
  60 SKIP            ^usr/platform/.*/libpsvcpolicy\.so\.1
  61 SKIP            ^usr/lib/sysevent/modules/picl_slm.so$

  63 # Objects that are allowed to have executable data segments
  64 EXEC_DATA       ^MACH(lib)/ld\.so\.1$
  65 EXEC_DATA       ^lib/libc\.so\.1$       # 6524709, 32-bit, needed for x86 only
  66 EXEC_DATA       ^MACH(lib)/libumem\.so\.1$ # ptcumem
  66 EXEC_DATA       ^lib/amd64/libumem\.so\.1$ # ptcumem
  67 EXEC_DATA       ^lib/libumem\.so\.1$    # ptcumem
  67 EXEC_DATA       ^opt/SUNWdtrt/tst/.*/ustack/tst\.helper\.exe$
  68 EXEC_DATA       ^platform/.*/MACH(kernel)/unix$
  70 EXEC_DATA       ^platform/.*/multiboot$

  70 # Objects that are allowed to have an executable stack
  71 EXEC_STACK      ^platform/.*/MACH(kernel)/unix$
  72 EXEC_STACK      ^platform/.*/multiboot$
  73 EXEC_STACK      ^opt/os-tests/tests/secflags/stacky$

  75 # Objects for which we allow relocations to the text segment
  76 TEXTREL         ^platform/.*/MACH(kernel)/unix$

  78 # Directories and files that are allowed to have no direct bound symbols
  79 NODIRECT        ^platform/.*/MACH(kernel)/unix$
  80 NODIRECT        ^usr/ucb
  81 NODIRECT        ^usr/4lib/sbcp$

  83 # Identify any files that should be skipped when building a crle(1)
  84 # configuration file.  As the hwcap libraries can be loop-back mounted onto
  85 # libc, these can confuse crle(1) because of their identical dev/inode.
  86 NOCRLEALT       ^usr/lib/libc/libc_hwcap[1-3].so.1$

  88 # Files that should contain debugging information.
  89 STAB    ^platform/.*/MACH(kernel)/unix$

  91 # Files that are allowed undefined references
  92 UNDEF_REF       ^usr/lib/libnisdb\.so\.2$

  94 # Objects allowed to have unused dependencies
  95 UNUSED_DEPS     ^usr/lib/picl/plugins/          # require devtree dependencies

  99 # libm.so.2 dependency
 100 UNUSED_OBJ      unused object=.*MACH(libm)/libm_hwcap1\.so\.2

  97 # libnetsnmphelpers.so is empty in some net-snmp versions
  98 UNUSED_OBJ      unused object=.*/libnetsnmphelpers\.so\..*
  99 UNREF_OBJ       unreferenced object=.*/libnetsnmphelpers\.so\..*

 101 # Unused runpaths due to dlopen() use
 102 UNUSED_RPATH    /usr/lib/fs/autofs.*\ from\ .automountd
 103 UNUSED_RPATH    /etc/ppp/plugins.*\ from\ .*pppd
 104 UNUSED_RPATH    /usr/lib/inet/ppp.*\ from\ .*pppd
 110 UNUSED_RPATH    /usr/sfw/lib.*\ from\ .*libipsecutil\.so\.1
 105 UNUSED_RPATH    /usr/platform/.*rsmlib.*\ from\ .*librsm\.so\.2
 106 UNUSED_RPATH    \$ORIGIN.*\ from\ .*fcode.so
 107 UNUSED_RPATH    /opt/VRTSvxvm/lib.*\ from\ .*libdiskmgt\.so\.1

 109 # Unused runpaths in picl code
 110 UNUSED_RPATH    /usr/platform/.*\ from\ .*/usr/platform
 111 UNUSED_RPATH    /usr/lib/picl/.*\ from\ .*/usr/platform
 112 UNUSED_RPATH    /usr/platform/.*\ from\ .*/usr/lib/picl

 114 # Unused runpaths in non-OSNET objects we can't change
 115 UNUSED_RPATH    /usr/lib/mps.*\ from\ .*libnss3\.so
 116 UNUSED_RPATH    /usr/lib/mps.*\ from\ .*libnssutil3\.so
 117 UNUSED_RPATH    /usr/lib/mps.*\ from\ .*libsmime3\.so
 118 UNUSED_RPATH    /usr/lib/mps.*\ from\ .*libssl3\.so
```

```
125 UNUSED_RPATH    /usr/sfw/lib.*\ from\ .*libdbus-1\.so\.3
126 UNUSED_RPATH    /usr/sfw/lib.*\ from\ .*libdbus-glib-1\.so\.2
127 UNUSED_RPATH    /usr/sfw/lib.*\ from\ .*libglib-2\.0\.so\.0
128 UNUSED_RPATH    /usr/X11/lib.*\ from\ .*libglib-2\.0\.so\.0
129 UNUSED_RPATH    /usr/sfw/lib.*\ from\ .*libgobject-2\.0\.so\.0
130 UNUSED_RPATH    /usr/X11/lib.*\ from\ .*libgobject-2\.0\.so\.0
131 UNUSED_RPATH    /usr/sfw/lib.*\ from\ .*libgthread-2\.0\.so\.0
132 UNUSED_RPATH    /usr/X11/lib.*\ from\ .*libgthread-2\.0\.so\.0
133 UNUSED_RPATH    /usr/sfw/lib.*\ from\ .*libcrypto\.so\.0\.9\.8
134 UNUSED_RPATH    /usr/sfw/lib.*\ from\ .*libnetsnmp\.so\..*
135 UNUSED_RPATH    /usr/sfw/lib.*\ from\ .*libgcc_s\.so\.1
136 UNUSED_RPATH    /usr/ccs/lib.*\ from\ .*libgcc_s\.so\.1
137 UNUSED_RPATH    /usr/lib.*\ from\ .*libgcc_s\.so\.1
138 UNUSED_RPATH    /usr/postgres/8.3/lib.*\ from\ .*libpq\.so\.5
139 UNUSED_RPATH    /usr/sfw/lib.*\ from\ .*libpq\.so\.5
119 UNUSED_RPATH    /usr/lib.*\ from\ .*/usr/lib/mps
141 UNUSED_RPATH    /usr/ccs/lib.*\ from\ .*/usr/lib/mps
120 UNUSED_RPATH    /usr/gnu/lib.*\ from\ .*/usr/lib/libpython2\..
121 UNUSED_RPATH    /usr/gnu/lib.*\ from\ .*/usr/lib/64/libpython2\..
144 UNUSED_RPATH    /usr/snadm/lib.*\ from\ .*/usr/snadm/lib/libspmicommon\.so\.1


123 # Unused runpaths for reasons not captured above
124 UNUSED_RPATH    /usr/lib/smbsrv.*\ from\ .*libsmb\.so\.1        # future needs
149 UNUSED_RPATH    /usr.*\ from\ .*tst\.gcc\.exe                   # gcc built


126 # Unreferenced objects of non-OSnet objects we can't change
127 UNREF_OBJ       /lib.*\ of\ .*libcimapi\.so
128 UNREF_OBJ       /lib.*\ of\ .*libdbus-1\.so\.3
129 UNREF_OBJ       /lib.*\ of\ .*libdbus-glib-1\.so\.2
130 UNREF_OBJ       /lib.*\ of\ .*libgio-2.0\.so\.0
131 UNREF_OBJ       /lib.*\ of\ .*libglib-2.0\.so\.0
132 UNREF_OBJ       /lib.*\ of\ .*libgobject-2.0\.so\.0
133 UNREF_OBJ       /lib.*\ of\ .*libgthread-2\.0\.so\.0
134 UNREF_OBJ       /lib.*\ of\ .*libjvm\.so
135 UNREF_OBJ       /lib.*\ of\ .*libnetsnmp\.so\..*
136 UNREF_OBJ       /lib.*\ of\ .*libnetsnmpagent\.so\..*
137 UNREF_OBJ       /lib.*\ of\ .*libnetsnmpmibs\.so\..*
138 UNREF_OBJ       /lib.*\ of\ .*libnetsnmphelpers\.so\..*
139 UNREF_OBJ       /lib.*\ of\ .*libnspr4\.so
140 UNREF_OBJ       /lib.*\ of\ .*libpq\.so\.5
141 UNREF_OBJ       /lib.*\ of\ .*libsoftokn3\.so
142 UNREF_OBJ       /lib.*\ of\ .*libspmicommon\.so\.1
143 UNREF_OBJ       /lib.*\ of\ .*libspmocommon\.so\.1
144 UNREF_OBJ       /lib.*\ of\ .*libssl3\.so
145 UNREF_OBJ       /lib.*\ of\ .*libtspi\.so\.1
146 UNREF_OBJ       /lib.*\ of\ .*libxml2\.so\.2
147 UNREF_OBJ       /lib.*\ of\ .*libxslt\.so\.1
148 UNREF_OBJ       /lib.*\ of\ .*libpq\.so\.4
149 UNREF_OBJ       /lib.*\ of\ .*libpython2\.4\.so\.1\.0
150 UNREF_OBJ       /lib.*\ of\ .*libpython2\.6\.so\.1\.0
151 UNREF_OBJ       /lib.*\ of\ .*libpython2\.7\.so\.1\.0
152 UNREF_OBJ       /libgcc_s.*\ of\ .*libstdc\+\+\.so\.6
153 UNREF_OBJ       /libgcc_s.*\ of\ .*libgmodule-2\.0\.so\.0

155 # Unreferenced object of objects we can't change for other reasons
156 UNREF_OBJ       /libmapmalloc\.so\.1;\ unused\ dependency\ of   # interposer
157 UNREF_OBJ       /libstdc\+\+\.so\.6;\ unused\ dependency\ of    # gcc build
158 UNREF_OBJ       /libgcc_s\.so\.1.*\ of\ .*libstdc\+\+\.so\.6    # omnios gcc mix
159 UNREF_OBJ       /libm\.so\.2.*\ of\ .*libstdc\+\+\.so\.6        # gcc build
160 UNREF_OBJ       /lib.*\ of\ .*/lib/picl/plugins/                # picl
161 UNREF_OBJ       /lib.*\ of\ .*kcfd                              # interposer
162 UNREF_OBJ       /libpkcs11\.so\.1; .*\ of\ .*libkmf\.so\.1      # interposed
163 # Referenced by the Studio build, not the GCC build.  GCC eliminates the unused
164 # statics which have the dependence.
```



```
165 UNREF_OBJ       /libc\.so\.1.*\ of\ .*kldap\.so\.1


168 # Objects that used to contain system functionalty that has since
169 # migrated to libc. We preserve these libraries as pure filters for
170 # backward compatability but nothing needs to link to them.
171 OLDDEP          libaio\.so\.1                   # onnv build 44
172 OLDDEP          libdl\.so\.1                    # on10 build 49
173 OLDDEP          libdoor\.so\.1                  # onnv build 12
174 OLDDEP          libintl\.so\.1                  # on297 build 7
175 OLDDEP          libpthread\.so\.1               # on10 build 53
176 OLDDEP          librt\.so\.1                    # onnv build 44
177 OLDDEP          libsched\.so\.1                 # on10 build 36
178 OLDDEP          libthread\.so\.1                # on10 build 53
179 OLDDEP          libw\.so\.1                     # on297 build 7

181 # Files for which we skip checking of duplicate addresses in the
182 # symbol sort sections. Such exceptions should be rare --- most code will
183 # not have duplicate addresses, since it takes assember or a "#pragma weak"
184 # to do such aliasing in C. C++ is different: The compiler generates aliases
185 # for implementation reasons, and the mangled names used to encode argument
186 # and return value types are difficult to handle well in mapfiles.
187 # Furthermore, the Sun compiler and gcc use different and incompatible
188 # name mangling conventions. Since illumos must be buildable by either, we
214 # name mangling conventions. Since ON must be buildable by either, we
189 # would have to maintain two sets of mapfiles for each such object.
190 # C++ use is rare in illumos, so this is not worth pursuing.
216 # C++ use is rare in ON, so this is not worth pursuing.
191 #
192 NOSYMSORT       opt/SUNWdtrt/tst/common/pid/tst.weak2.exe       # DTrace test
219 NOSYMSORT       lib/amd64/libnsl\.so\.1                         # C++
220 NOSYMSORT       lib/sparcv9/libnsl\.so\.1                       # C++
221 NOSYMSORT       lib/sparcv9/libfru\.so\.1                       # C++
222 NOSYMSORT       usr/lib/lms                                     # C++
193 NOSYMSORT       ld\.so\.1                                       # libc_pic.a use
194 NOSYMSORT       usr/MACH(lib)/libsun_fc\.so\.1                  # C++
195 NOSYMSORT       usr/MACH(lib)/libfru\.so\.1                     # C++
224 NOSYMSORT       lib/libsun_fc\.so\.1                            # C++
225 NOSYMSORT       lib/amd64/libsun_fc\.so\.1                      # C++
226 NOSYMSORT       lib/sparcv9/libsun_fc\.so\.1                    # C++
227 NOSYMSORT       usr/lib/amd64/libfru\.so\.1                     # C++

197 # The majority of illumos deliverables should not depend on the GCC runtime
198 # (any necessary runtime symbol should be provided by libc.so, instead).
199 # However, the GNU C++ runtime requires the GCC runtime, so certain objects
200 # must be excepted.
201 FORBIDDEN       libgcc_s\.so
202 FORBIDDEN_DEP   usr/bin/audioconvert            # C++
203 FORBIDDEN_DEP   usr/bin/make                    # C++
204 FORBIDDEN_DEP   usr/MACH(lib)/libfru.so.1       # C++
205 FORBIDDEN_DEP   usr/MACH(lib)/libsun_fc.so.1    # C++
206 FORBIDDEN_DEP   usr/lib/netsvc/yp/rpc.yppasswdd # C++
207 FORBIDDEN_DEP   usr/lib/netsvc/yp/ypserv        # C++
208 FORBIDDEN_DEP   usr/lib/netsvc/yp/ypxfr        # C++
209 FORBIDDEN_DEP   usr/lib/netsvc/yp/ypxfrd       # C++

211 # libfakekernel is a test environment, not intended for general use
212 FORBIDDEN libfakekernel\.so
213 FORBIDDEN_DEP usr/MACH(lib)/libzpool.so.1
214 FORBIDDEN_DEP usr/bin/amd64/ztest
215 FORBIDDEN_DEP usr/bin/i86/ztest
216 FORBIDDEN_DEP usr/bin/sparcv7/ztest
217 FORBIDDEN_DEP usr/bin/sparcv9/ztest
218 FORBIDDEN_DEP usr/lib/MACH(smbsrv)/libfksmbsrv.so.1
219 FORBIDDEN_DEP usr/lib/smbsrv/fksmbd
220 FORBIDDEN_DEP usr/sbin/amd64/zdb
```

```
221 FORBIDDEN_DEP usr/sbin/i86/zdb
222 FORBIDDEN_DEP usr/sbin/sparcv7/zdb
223 FORBIDDEN_DEP usr/sbin/sparcv9/zdb

225 # libucb is intended for legacy compatibility, not general use
226 FORBIDDEN libucb\.so
227 FORBIDDEN_DEP usr/ucb/
228 FORBIDDEN_DEP usr/ucblib/

230 # Older versions of libraries only provided for binary compatibility
231 FORBIDDEN libm\.so\.1
232 FORBIDDEN libresolv\.so\.1
233 FORBIDDEN libxcurses\.so\.1
234 #endif /* ! codereview */

236 # The libprtdiag_psr.so.1 objects built under usr/src/lib/libprtdiag_psr
237 # are a family, all built using the same makefile, targeted at different
238 # sparc hardware variants. There are a small number of cases where this
239 # one size fits all approach causes an object to be linked against an
240 # unneeded library.
241 UNREF_OBJ       lib/(libdevinfo|libcfgadm)\.so\.1; .*\ of\ .*SUNW,Netra-CP2300/l
```

```
**********************************************************
   20847 Tue Feb 20 04:35:58 2018
new/usr/src/tools/scripts/check_rtime.1onbld
9139 check_rtime should be able to forbid libraries
9140 check_rtime should learn libnsl is safe now
9141 check_rtime exceptions could be cleaner
**********************************************************
```

```
.\" Copyright (c) 2001, 2010, Oracle and/or its affiliates. All rights reserved.
.\"
.\" 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
.\"
.Dd February 19, 2018
.Dt CHECK_RTIME 1ONBLD
.Os
.Sh NAME
.Nm check_rtime
.Nd check ELF runtime attributes
.Sh SYNOPSIS
.Nm check_rtime
.Op Fl imosv
.Op Fl D Ar depfile | Fl d depdir
.Op Fl E Ar errfile
.Op Fl e Ar exfile
.Op Fl f Ar listfile
.Op Fl I Ar infofile
.Op Fl w Ar outdir
.Ar file | dir ...
.Sh DESCRIPTION
.Nm check_rtime
.TH CHECK_RTIME 1ONBLD "Mar 09, 2010"
.SH NAME
.I check_rtime
\- check ELF runtime attributes
.SH SYNOPSIS
\fBcheck_rtime [-imosv] [-D depfile | -d depdir] [-E errfile] [-e exfile] [-f li
.SH DESCRIPTION
.LP
.I check_rtime
attempts to check a number of ELF runtime attributes
for consistency with common build rules.
These checks involve running
.Xr ldd 1
and
.Xr elfdump 1
against a family of dynamic objects.
A dynamic object can be defined explicitly as a
.Ar file
or multiple dynamic objects can be located under the directory
.Ar dir .
```



```
.Pp
.Nm check_rtime
is typically called from
.Xr nightly 1ONBLD
when the
.Fl r
option is in effect.
In this case the dynamic objects under
the associated
.Em proto area
.Pq Ev $ROOT
are checked.
.Nm check_rtime
.I check_rtime
is typically called from \fBnightly(1ONBLD)\fP when the \fB-r\fP
option is in effect. In this case the dynamic objects under
the associated \fIproto\fP area (\fB$ROOT\fP) are checked.
.I check_rtime
can also be run standalone against any set of dynamic objects.
.Pp
.Nm check_rtime
uses
.Xr  ldd 1
to verify dependencies.
This implies that by default any object inspected will bind to its dependencies
as they are found in the
.Em underlying system .
Use of the
.Fl D ,
.Fl d
.LP
.I check_rtime
uses \fBldd(1)\fP to verify dependencies. This implies that
by default any object inspected will bind to its dependencies
as they are found in the \fBunderlying\fP system.  Use of the \fB-D\fP, \fB-d\fP
option, or the existence of the environment variables
.Ev $CODEMGR_WS
or
.Ev $ROOT
instruct
.Nm check_rtime
\fB$CODEMGR_WS/$ROOT\fP instruct
.I check_rtime
to establish an alternative dependency mapping using
runtime configuration files generated with
.Xr crle 1 .
.Pp
.Nm check_rtime
uses
.Xr ldd 1
to completely relocate any dynamic object and thus detect missing
dependencies, unsatisfied symbol relocations, unused and unreferenced
dependencies.
These checks are carried out for the following reasons:
.Bl -bullet
.It
runtime configuration files generated with \fBcrle(1)\fP.
.LP
.I check_rtime
uses \fBldd(1)\fP to completely relocate any dynamic
object and thus detect missing dependencies, unsatisfied
```

```
 58 symbol relocations, unused and unreferenced dependencies. These checks
 59 are carried out for the following reasons:
 60 .TP 4
 61 \(bu
 95 An object that cannot find its dependencies may fail to load
 96 at runtime.
 97 This error condition often goes unnoticed because the existing use of the
 98 object is as a dependency itself, and the objects' dependencies are already
 99 satisfied by the caller.
100 However, if the object itself is unable to satisfy its dependencies, its use
101 in new environments may be compromised.
102 .Pp
103 A missing or erroneous
104 .Em runpath
105 is the typical reason why an object can not locate its dependencies.
106 Use of the link-editors
107 .Fl zdefs
108 option when building a shared object ensures required dependencies are
109 established.
110 This flag is inherited from
111 .Dv $(DYNFLAGS)
112 in
113 .Pa lib/Makefile.lib .
114 Missing dependencies are displayed as:
115 .Pp
116 .Dl foo: bar.so.1 => (file not found)  <no -zdefs?>
117 .It
 63 at runtime.  This error condition often goes unnoticed
 64 because the existing use of the object is as a dependency itself,
 65 and the objects' dependencies are already satisfied by the
 66 caller.  However, if the object itself is unable to satisfy its
 67 dependencies, its use in new environments may be compromised.
 68 .sp
 69 A missing or erroneous \fBrunpath\fP is the typical reason why
 70 an object can not locate its dependencies.  Use of the link-editors
 71 \fB-zdefs\fP option when building a shared object ensures required
 72 dependencies are established.  This flag is inherited from
 73 \fB$(DYNFLAGS)\fP in \fIlib/Makefile.lib\fP. Missing dependencies
 74 are displayed as:
 75 .sp
 76 .RS 6
 77 foo: bar.so.1 => (file not found)  <no -zdefs?>
 78 .RE
 79 .TP
 80 \(bu
118 Unsatisfied symbol relocations indicate that some thread of
119 execution through the object will fail when it is unable to
120 locate a referenced symbol.
121 .Pp
 84 .sp
122 A missing, or mismatched version of a dependency is the typical
123 reason for unsatisfied symbol relocations (see missing dependency
124 discussion above). Unsatisfied symbol relocations are displayed as:
125 .Pp
126 .Dl foo: symbol not found: bar  <no -zdefs?>
127 .Pp
 88 .sp
 89 .RS 6
 90 foo: symbol not found: bar  <no -zdefs?>
 91 .RE
 92 .RS 4
 93 .sp
128 Note: Shared objects can make reference to symbol definitions
129 that are expected to be defined by the caller.
130 To indicate that such symbols are not undefined in the usual sense, you must
131 specify these symbols in a
```



```
132 .Em mapfile ,
133 using the
134 .Va EXTERN
135 or
136 .Va PARENT
137 symbol attributes.
138 Without these symbol attributes,
139 .Xr ldd 1
140 is unable to determine the symbols special nature, and
141 .Nm check_rtime
 95 that are expected to be defined by the caller. To indicate that
 96 such symbols are not undefined in the usual sense, you must
 97 specify these symbols in a \fImapfile\fP, using the \fBEXTERN\fP
 98 or \fBPARENT\fP symbol attribute. Without these symbol attributes,
 99 \fBldd(1)\fP is unable to determine the symbols special nature, and
100 .I check_rtime
142 will report these symbols as undefined.
143 .It
102 .RE
103 .TP
104 \(bu
144 Unused dependencies are wasteful at runtime, as they take time to
145 load and relocate, but will not be used by the calling object.
146 They also result in unnecessary processing at link-edit time.
147 .Pp
148 Dependency lists (typically defined via
149 .Dv $(LDLIBS) )
150 that have been copy and pasted
151 between
152 .Pa Makefiles
153 without verifying their need, are a typicalreason why unused dependencies
154 exist.
155 Unused dependencies are displayed as:
156 .Pp
157 .Dl foo: unused object=bar.so.1  <remove lib or -zignore?>
158 .It
106 load and relocate, but will not be used by the calling object.  They
107 also result in unnecessary processing at link-edit time.
108 .sp
109 Dependency lists (typically defined via \fB$(LDLIBS)\fP)
110 that have been yanked-and-put
111 between \fIMakefiles\fP without verifying their need, are a typical
112 reason why unused dependencies exist.  Unused dependencies are
113 displayed as:
114 .sp
115 .RS 6
116 foo: unused object=bar.so.1  <remove lib or -zignore?>
117 .RE
118 .TP
119 \(bu
159 Unreferenced dependencies are also wasteful at runtime, although not
160 to the extent of unused dependencies.
161 They also result in unnecessary processing at link-edit time.
162 .Pp
121 to the extent of unused dependencies.  They also result in unnecessary
122 processing at link-edit time.
123 .sp
163 Unreferenced dependency removal guards against a dependency becoming
164 unused when combined with
165 different objects, or as the other object dependencies evolve.
166 Unreferenced dependencies are displayed as:
167 .Bd -literal
128 .sp
129 .RS 6
168 foo: unreferenced object=bar.so.1;  \\
131 .br
```

```
169     unused dependency of libfoo.so.1  \\
133 .br
170     <remove lib or -zignore?>
171 .Ed
172 .Pp
173 See also the section
174 .Sx ENVIRONMENT VARIABLES .
175 .It
135 .RE
136 .RS 4
137 .sp
138 See also the section ENVIRONMENT VARIABLES.
139 .RE
140 .TP
141 \(bu
176 Unused search paths are wasteful at runtime.
177 Unused search paths are displayed as:
178 .Bd -literal
144 .sp
145 .RS 6
179 foo: unused search path=/usr/foo/lib  \\
147 .br
180     (RUNPATH/RPATH from file libfoo.so.1)  \\
149 .br
181     <remove search path?>
182 .Ed
183 .El
184 .Pp
185 .Nm check_rtime
186 uses
187 .Xr elfdump 1
188 to look for a concatenated relocation section in shared objects, the existence
189 of text relocations, whether debugging or symbol table information exists,
190 whether applications have a non-executable stack defined, duplicate entries in
191 the symbol sorting sections, and for direct bindings.
151 .RE
152 .LP
153 .I check_rtime
154 uses \fBelfdump(1)\fP to look for a concatenated relocation
155 section in shared objects, the existence of text relocations,
156 whether debugging or symbol table information exists, whether
157 applications have a non-executable stack defined, duplicate
158 entries in the symbol sorting sections, and for direct bindings.
192 These checks are carried out for the following reasons:
193 .Bl -bullet
194 .It
195 A concatenated relocation section
196 .Pq Em .SUNW_reloc
197 provides optimal symbol table access at runtime, and thus reduces the overhead
198 of relocating the shared object.
199 In past releases, the link-edit of a dynamic object with the
200 .Fl z Ar combreloc
201 option was required to generate a combined relocation section.
202 However, with the integration of 6642769, this section combination is a default
203 the link-editor.
204 .Pp
205 In past releases, not inheriting
206 .Dv $(DYNFLAGS)
207 from
208 .Pa lib/Makefile.lib
209 was the typical reason for not having a concatenated relocation section.
210 The misguided use of the
211 .Fl z Ar nocombreloc
212 option will also prevent the creation of a concatenated relocation section.
213 A missing concatenated relocation section is displayed as:
214 .Pp
```



```
215 .Dl foo: .SUNW_reloc section missing  <no -zcombreloc?>
216 .It
217 Text relocations result in impure text segments.
218 As text segments are typically read-only, they can be shared between numerous
219 processes.
220 If they must be updated as part of the relocation then the updated pages
221 become unsharable and swap space must be allocated to back these pages.
222 These events consume unnecessary system resources and reduce overall system
223 performance.
224 .Pp
225 Not inheriting the
226 .Dv $(PICS)
227 rules from
228 .Pa lib/Makefile.lib
229 is the typical reason for having non-pic code in shared objects.
230 Text relocations are displayed as:
231 .Pp
232 .Dl foo: TEXTREL .dynamic tag  <no -Kpic?>
233 .It
234 Debugging information is unnecessary in released objects.
235 Although extensive when compiled
236 .Fl g ,
237 small quantities of debugging information are stored in
238 .Em .stabs
239 sections under normal compilations.
240 This debugging information is geared towards aiding debuggers locate
241 relocatable objects associated with the dynamic objects being debugged.
242 As relocatable objects aren't made available as part of a software release
243 this information has no use.
244 .Pp
245 Not inheriting the correct
246 .Dv $(LDFLAGS)
247 from
248 .Pa cmd/Makefile.cmd
249 .Pq which asserts Fl s
250 or
251 .Dv $(POST_PROCESS_SO)
252 .Pq which asserts Ic strip -x
253 are typical reasons for not removing debugging information.
254 Note, removal of debugging information is only enabled
255 for release builds.
256 The existence of debugging information is displayed as:
257 .Bd -literal
160 .TP 4
161 \(bu
162 A concatenated relocation section (\fI.SUNW_reloc\fP)
163 provides optimal symbol table
164 access at runtime, and thus reduces the overhead of relocating
165 the shared object.  In past releases, the link-edit of a dynamic object with
166 the \fB-z combreloc\fP option was required to generate a combined
167 relocation section.  However, with the integration of 6642769, this section
168 combination is a default behavior of the link-editor.
169 .sp
170 In past releases, not inheriting \fB$(DYNFLAGS)\fP from
171 \fIlib/Makefile.lib\fP was the typical reason for not having a
172 concatenated relocation section. The misguided use of the
173 \fB-z nocombreloc\fP option will also prevent the creation of a
174 concatenated relocation section. A missing concatenated relocation section
175 is displayed as:
176 .sp
177 .RS 6
178 foo: .SUNW_reloc section missing  <no -zcombreloc?>
179 .RE
180 .TP
181 \(bu
182 Text relocations result in impure text segments.  As text segments
```

```
183 are typically read-only, they can be shared between numerous processes.
184 If they must be updated as part of the relocation then the updated
185 pages become unsharable and swap space must be allocated to back
186 these pages.  These events consume unnecessary system resources and
187 reduce overall system performance.
188 .sp
189 Not inheriting the \fB$(PICS)\fP
190 rules from \fIlib/Makefile.lib\fP is the typical reason for having
191 non-pic code in shared objects.  Text relocations are displayed as:
192 .sp
193 .RS 6
194 foo: TEXTREL .dynamic tag  <no -Kpic?>
195 .RE
196 .TP
197 \(bu
198 Debugging information is unnecessary in released objects.  Although
199 extensive when compiled \fB-g\fP, small quantities of debugging
200 information are stored in \fI.stabs\fP sections under normal
201 compilations.  This debugging information is geared towards aiding
202 debuggers locate relocatable objects associated with the dynamic
203 objects being debugged.  As relocatable objects aren't made available
204 as part of a software release this information has no use.
205 .sp
206 Not inheriting the correct \fB$(LDFLAGS)\fP from \fIcmd/Makefile.cmd\fP
207 (which asserts \fP-s\fP), or \fB$(POST_PROCESS_SO)\fP (which asserts
208 \fIstrip -x\fP) are typical reasons for not removing debugging
209 information.  Note, removal of debugging information is only enabled
210 for release builds. The existence of debugging information is displayed as:
211 .sp
212 .RS 6
258 foo: debugging sections should be deleted  \\
214 .br
259     <no strip -x?>
260 .Ed
261 .It
262 All objects should retain their full
263 .Em .symtab
264 symbol table.
216 .RE
217 .TP
218 \(bu
219 All objects should retain their full \fI.symtab\fP symbol table.
265 Although this consumes disk space, it provides for more extensive stack
266 tracing when debugging user applications.
267 .Pp
268 Hard coding a
269 .Fl s
270 flag with
271 .Dv $(LDFLAGS) or
272 .Dv $(DYNFLAGS)
273 is the typical reason for symbol tables being removed.
222 .sp
223 Hard coding a \fI-s\fP flag with \fB$(LDFLAGS)\fP or
224 \fB$(DYNFLAGS)\fP is the typical
225 reason for symbol tables being removed.
274 Objects that do not contain a symbol table are displayed as:
275 .Bd -literal
227 .sp
228 .RS 6
276 foo.so.1: symbol table should not be stripped  \\
230 .br
277     <remove -s?>
278 .Ed
279 .It
232 .RE
233 .TP
```



```
234 \(bu
280 Applications should have a non-executable stack defined to make
281 them less vulnerable to buffer overflow attacks.
282 .Pp
283 Not inheriting the
284 .Dv $(LDFLAGS)
285 macro in
286 .Pa cmd/Makefile.cmd
237 .sp
238 Not inheriting the \fB$(LDFLAGS)\fP macro in \fIcmd/Makefile.cmd\fP
287 is the typical reason for not having a non-executable stack definition.
288 Applications without this definition are displayed as:
289 .Bd -literal
241 .sp
242 .RS 6
290 foo: application requires non-executable stack \\
244 .br
245 .nf
291         <no -Mmapfile_noexstk?>
292 .Ed
293 .It
247 .fi
248 .RE
249 .sp
250 .TP
251 \(bu
294 x86 applications should have a non-executable data segment defined to make
295 them less vulnerable to buffer overflow attacks.
296 .Pp
297 Not inheriting the
298 .Dv $(LDFLAGS)
299 macro in
300 .Pa cmd/Makefile.cmd
254 .sp
255 Not inheriting the \fB$(LDFLAGS)\fP macro in \fIcmd/Makefile.cmd\fP
301 is the typical reason for not having a non-executable data definition.
302 Applications without this definition are displayed as:
303 .Bd -literal
258 .sp
259 .RS 6
304 foo: application requires non-executable data \\
261 .br
262 .nf
305         <no -Mmapfile_noexdata?>
306 .Ed
307 .It
264 .fi
265 .RE
266 .sp
267 .TP
268 \(bu
308 Solaris ELF files contain symbol sort sections used by DTrace to
309 map addresses in memory to the related function or variable symbols.
310 There are two such sections,
311 .Em .SUNW_dynsymsort
312 for regular symbols, and
313 .Em .SUNW_dyntlssort
314 for thread-local symbols.
315 To ensure that the best names are shown for each such address, and that the
316 same name is given across Solaris releases,
317 .Nm check_rtime
270 map addresses in memory to the related function or variable symbols. There
271 are two such sections, \fI.SUNW_dynsymsort\fP for
272 regular symbols, and \fI.SUNW_dyntlssort\fP for thread-local
273 symbols. To ensure that the best names are shown for each
274 such address, and that the same name is given across Solaris releases,
```

```
.I check_rtime
```
enforces the rule that only one symbol can appear in the sort sections for
any given address.
There are two common ways in which multiple symbols
```
or a given address occur in the ON distribution.
The first is from code written in assembly language.
The second is as a result of using
.Ic #pragma weak
in C to create weak symbols.
The best solution to this situation is to modify the code to avoid symbol
aliasing.
Alternatively, the
.Va NODYNSORT
mapfile attribute can be used to eliminate the unwanted symbol.
.Pp
```
```
or a given address occur in the ON distribution. The first is from
code written in assembly language. The second is as a
result of using \fB#pragma weak\fP in C to create weak symbols. The
best solution to this
situation is to modify the code to avoid symbol aliasing. Alternatively,
the \fBNODYNSORT\fP mapfile attribute can be used to eliminate the unwanted
symbol.
.sp
```
Duplicate entries in a symbol sort section are
displayed in one of the following ways, depending on
whether the section is for regular or thread-local symbols:
```
.Bd -literal
```
```
.sp
.RS 6
```
foo: .SUNW_dynsymsort: duplicate ADDRESS: sym1, sym2
```
.br
```
foo: .SUNW_dyntlssort: duplicate OFFSET: sym1, sym2
```
.Ed
.It
illumos dynamic ELF objects are expected to employ direct bindings whenever
feasible.
This runtime binding technique helps to avoid accidental interposition
problems, and provides a more optimal runtime symbol search model.
.Pp
Not inheriting the correct
.Dv $(LDFLAGS) from
.Pa cmd/Makefile.cmd ,
or the correct
.Dv $(DYNFLAGS)
from
.Pa lib/Makefile.lib ,
are the typical reasons for not enabling direct bindings.
Dynamic objects that do not contain direct binding information are displayed
as:
.Bd -literal
```
```
.RE
.sp
.TP
\(bu
\fBOSNet\fP dynamic ELF objects are expected to employ direct bindings whenever
feasible.  This runtime binding technique helps to avoid accidental
interposition problems, and provides a more optimal
runtime symbol search model.
.sp
Not inheriting the correct \fB$(LDFLAGS)\fP from \fIcmd/Makefile.cmd\fP,
or the correct \fB$(DYNFLAGS)\fP from \fIlib/Makefile.lib\fP, are the
typical reasons for not enabling direct bindings. Dynamic objects that
do not contain direct binding information are displayed as:
.sp
.RS 6
```
foo: object has no direct bindings \\
```



```
.br
.nf
```
        <no -B direct or -z direct?>
```
.Ed
.El
.Pp
.Nm check_rtime
also
uses
.Xr elfdump 1
to display useful dynamic entry information under the
.Fl -i
option.
```
.fi
.RE

.sp
.LP
.I check_rtime also
uses \fBelfdump(1)\fP
to display useful dynamic entry information under the \fB-i\fP option.
```
This doesn’t necessarily indicate an error condition, but
provides information that is often useful for gatekeepers to track
```
changes in a release.
Presently the information listed is:
.Bl -bullet
.It
Runpaths are printed for any dynamic object.
This is a historic sanity check to insure compiler supplied runpaths
(typically from
.Nm CC )
are not recorded in any objects.
Runpaths are displayed as:
.Pp
.Dl foo: RPATH=/usr/bar/lib
.It
```
changes in a release.  Presently the information listed is:
.TP
\(bu
Runpaths are printed for any dynamic object.  This is a historic
sanity check to insure compiler supplied runpaths (typically from \fBCC\fP)
are not recorded in any objects.  Runpaths are displayed as:
.sp
.RS 6
foo: RPATH=/usr/bar/lib
.RE
.TP
\(bu
```
Needed dependencies are printed for any dynamic object.
In the freeware world this often helps the introducer of a new
shared object discover that an existing binary has become its
consumer, and thus that binaries package dependencies may require updating.
Dependencies are printed as:
```
.Pp
.Dl foo: NEEDED=bar.so.1
.It
Dependencies may be marked as forbidden
.Pq see Sx EXCEPTION FILE FORMAT
this allows the build to warn should people use them accidentally.
Forbidden dependencies are printed as:
.Pp
.Dl foo: NEEDED=bar.so.1        <forbidden dependency, missing -nodefaultlibs?>
.El
.Pp
.Nm check_rtime
uses
```
```

```
.Xr mcs 1
to inspect an object's
.Em .comment
section.
.sp
.RS 6
foo: NEEDED=bar.so.1
.RE
.sp
.LP
.I check_rtime
uses \fBmcs(1)\fP to inspect an object's \fI.comment\fP section.
```
During development, this section contains numerous file identifiers
```
marked with the tag
.Qq @(#) .
For release builds these sections are deleted and rewritten under control of
the
.Dv $(POST_PROCESS)
macro to produce a common release identifier.
This identifier typically consists of three lines including a single comment
starting with the string
.Qq @(#) SunOS .
If this common identifier isn't found the following diagnostic is generated:
.Pp
.Dl foo: non-conforming mcs(1) comment  <no $(POST_PROCESS)?>
.Pp
.Nm check_rtime
uses
.Xr pvs 1
to display version definitions under the
.Fl v
option.
marked with the tag "\fB@(#)\fP".  For release builds these sections
are deleted and rewritten under control of the \fB$(POST_PROCESS)\fP
macro to produce a common release identifier.  This identifier
typically consists of three lines including a single comment starting
with the string "\fB@(#) SunOS\fP".  If this common identifier isn't
found the following diagnostic is generated:
.sp
.RS 6
foo: non-conforming mcs(1) comment  <no $(POST_PROCESS)?>
.RE
.sp
.LP
.I check_rtime
uses \fBpvs(1)\fP to display version definitions under the \fB-v\fP option.
```
Each symbol defined by the object is shown along with the version it belongs to.
Changes to the symbols defined by an object, or the versions they belong to,
do not necessarily indicate an error condition, but
provides information that is often useful for gatekeepers to track
changes in a release.
```
.Sh OPTIONS
.sp
.SH OPTIONS
.LP
```
The following options are supported:
```
.Bl -tag -width indent
.It Fl D Ar depfile
Use
.Ar depfile
to generate an alternative dependency mapping.
.Ar depfile
must be created by
.Ic find_elf -r .
The
.Fl D
```



```
and
.Fl d
options are mutually exclusive.
.It Fl d Ar depfile
Use
.Ar depdir
to generate an alternative dependency mapping.
.Xr find_elf 1ONBLD
is used to locate the ELF sharable objects for which alternative mappings are
required.
The
.Fl D
and
.Fl d
options are mutually exclusive.
.It Fl E Ar errfile
Direct error messages for the analyzed objects to
.Ar errfile
instead of stdout.
.It Fl e Ar exfile
.TP 4
.B \-D depfile
Use \fIdepfile\fP to generate an alternative dependency mapping.
\fIdepfile\fP must be created by '\fBfind_elf -r\fP'.
The \fB-D\fP and \fB-d\fP options are mutually exclusive.
.TP
.B \-d depdir
Use \fIdepdir\fP to generate an alternative dependency mapping.
\fBfind_elf(1ONBLD)\fP is used to locate the ELF sharable objects for
which alternative mappings are required. The \fB-D\fP and \fB-d\fP options
are mutually exclusive.
.TP 4
.B \-E errfile
Direct error messages for the analyzed objects to \fIerrfile\fP instead
of stdout.
.TP 4
.B \-e exfile
```
An exception file is used to exclude objects from
```
the usual rules.
See
.Sx EXCEPTION FILE FORMAT .
.It Fl f Ar listfile
the usual rules. See EXCEPTION FILE FORMAT.
.TP
.B \-f listfile
```
Normally,
```
.Ic interface_check
.I interface_check
```
runs
```
.Ic find_elf
to locate the ELF objects to analyze.
The
.Fl f
option can be used to instead provide a file containing the list of objects to
analyze, in the format produced by
.Ic find_elf -r .
.It Fl I Ar infofile
Direct informational messages (
.Fl i ,
and
.Fl v
options) for the analyzed objects to
.Ar infofile
instead of stdout.
.It Fl i
Provide dynamic entry information.
```

```
487 Presently only dependencies and runpaths are printed.
488 .It Fl m
489 Enable
490 .Xr mcs 1
491 checking.
492 .It Fl o
397 .I find_elf
398 to locate the ELF objects to analyze. The \fB-f\fP option can be
399 used to instead provide a file containing the list of objects to
400 analyze, in the format produced by '\fBfind_elf -r\fP'.
401 .TP
402 .B -I infofile
403 Direct informational messages (\fB-i\fP, and \fB-v\fP options) for the
404 analyzed objects to \fIinfofile\fP instead of stdout.
405 .TP
406 .B \-i
407 Provide dynamic entry information.  Presently only dependencies and
408 runpaths are printed.
409 .TP
410 .B \-m
411 Enable \fBmcs(1)\fP checking.
412 .TP
413 .B \-o
493 Produce a one-line output for each condition discovered, prefixed
494 by the objects name.
495 This output style is more terse, but is more appropriate for sorting and
496 diffing with previous build results.
497 .It Fl s
498 Determine whether
499 .Em .stabs
500 sections exist.
501 .It Fl v
502 Provide version definition information.
503 Each symbol defined by the object is printed along with the version it is
504 assigned to.
505 .It Fl w Ar outdir
506 Interpret the paths of all input and output files relative to
507 .Ar outdir .
508 .El
509 .Sh EXCEPTION FILE FORMAT
415 by the objects name.  This output style is more terse, but is
416 more appropriate for sorting and diffing with previous build results.
417 .TP
418 .B \-s
419 Determine whether \fI.stabs\fP sections exist.
420 .TP
421 .B \-v
422 Provide version definition information. Each symbol defined by the object
423 is printed along with the version it is assigned to.
424 .TP
425 .B -w outdir
426 Interpret the paths of all input and output files relative to \fIoutdir\fP.
427 .SH EXCEPTION FILE FORMAT
510 Exceptions to the rules enforced by
511 .Nm check_rtime
512 are specified using an exception file.
513 The
514 .Fl -e
515 option is used to specify an explicit exception file.
516 Otherwise, if used in an activated workspace, the default exception file is
517 .Pa $CODEMGR_WS/exception_list/check_rtime
518 if that file exists.
519 If not used in an activated workspace, or if
520 .Pa $CODEMGR_WS/exception_list/check_rtime
521 does not exist,
522 .Nm check_rtime
```



```
429 .I check_rtime
430 are specified using an exception file. The \fB-e\fP option is used to
431 specify an explicit exception file. Otherwise, if used in an activated
432 workspace, the default exception file is
433 $CODEMGR_WS/exception_list/check_rtime
434 if that file exists. If not used in an activated workspace, or if
435 $CODEMGR_WS/exception_list/check_rtime does not exist,
436 .I check_rtime
523 will use
524 .Pa /opt/onbld/etc/exception_list/check_rtime
438 .I /opt/onbld/etc/exception_list/check_rtime
525 as a fallback default exception file.
526 .Pp
440 .P
527 To run
528 .Nm check_rtime
529 without applying exceptions, specify
530 .Fl e
531 with a value of
532 .Pa /dev/null .
533 .Pp
534 A
535 .Ql #
536 character at the beginning of a line, or at any point in
537 a line when preceded by whitespace, introduces a comment.
538 Empty lines, and lines containing only comments, are ignored by
539 .Nm check_rtime .
540 Exceptions are specified as space separated keyword, and
541 .Xr perl 1
442 .I check_rtime
443 without applying exceptions, specify \fB-e\fP with a value of /dev/null.
444 .P
445 A '#' character at the beginning of a line, or at any point in
446 a line when preceded by whitespace, introduces a comment. Empty lines,
447 and lines containing only comments, are ignored by
448 .I check_rtime.
449 Exceptions are specified as space separated keyword, and \fBperl(1)\fP
542 regular expression:
543 .Pp
544 .Dl keyword  perl-regex
545 .Pp
451 .sp
452 .in +4
453 .nf
454 keyword  perl-regex
455 .fi
456 .in -4
457 .sp
546 Since whitespace is used as a separator, the regular
547 expression cannot itself contain whitespace.
548 Use of the
549 .Ql \\s
550 character class to represent whitespace within the regular expression is
551 recommended.
552 .Pp
459 expression cannot itself contain whitespace. Use of the \\s character
460 class to represent whitespace within the regular expression is recommended.
553 Before the perl regular expression is used, constructs of the form
554 .Em MACH(dir)
555 are expanded into a regular expression that matches the directory given, as
556 well as any 64-bit architecture subdirectory that might be present
557 (i.e. amd64, sparcv9). For instance,
558 .Em MACH(lib)
559 will match any of the following:
560 .Bl -tag -width indent
561 .It Pa lib
```

```
.It Pa lib/amd64
.It Pa lib/sparcv9
.El
.Pp
MACH(dir) are expanded into a regular expression that matches the directory
given, as well as any 64-bit architecture subdirectory that
might be present (i.e. amd64, sparcv9). For instance, MACH(lib) will
match any of the following:
.sp
.in +4
.nf
lib
lib/amd64
lib/sparcv9
.fi
.in -4
.sp
```

The exceptions understood by
**.Nm check_rtime**
*.I check_rtime*
are:

```
.Bl -tag -width indent
.It EXEC_DATA
.sp
.ne 2
.na
\fBEXEC_DATA\fR
.ad
.RS 17n
.sp
```

Executables that are not required to have non-executable writable
data segments

```
.It EXEC_STACK
.RE

.sp
.ne 2
.na
\fBEXEC_STACK\fR
.ad
.RS 17n
.sp
```

Executables that are not required to have a non-executable stack

```
.It NOCRLEALT
.RE

.sp
.ne 2
.na
\fBNOCRLEALT\fR
.ad
.RS 17n
.sp
```

Objects that should be skipped when building the alternative dependency
**mapping via the**
**.Fl d**
**option.**

```
.It NODIRECT
mapping via the \fB-d\fP option.
.RE

.sp
.ne 2
.na
\fBNODIRECT\fR
.ad
```



```
.RS 17n
.sp
```

Directories and files that are allowed to have no direct bound symbols.

```
.It NOSYMSORT
.RE

.sp
.ne 2
.na
\fBNOSYMSORT\fR
.ad
.RS 17n
.sp
```

Files for which we skip checking of duplicate addresses in the
symbol sort sections.

```
.It OLDDEP
.RE

.sp
.ne 2
.na
\fBOLDDEP\fR
.ad
.RS 17n
.sp
```

Objects that used to contain system functionality that has since
**migrated to libc.**
**We preserve these libraries as pure filters for backward compatibility but**
**nothing needs to link to them.**

```
.It SKIP
```

**Directories and/or individual objects to skip.**
**Note that SKIP should be a last resort, used only when one of the other**
**exceptions will not suffice.**

```
.It STAB
migrated to libc. We preserve these libraries as pure filters for
backward compatibility but nothing needs to link to them.
.RE

.sp
.ne 2
.na
\fBSKIP\fR
.ad
.RS 17n
.sp
Directories and/or individual objects to skip. Note that SKIP should be
a last resort, used only when one of the other exceptions will not suffice.
.RE

.sp
.ne 2
.na
\fBSTAB\fR
.ad
.RS 17n
.sp
```

Objects that are allowed to contain debugging information (stabs).

```
.It TEXTREL
.RE

.sp
.ne 2
.na
\fBTEXTREL\fR
.ad
.RS 17n
```

```
.sp
Objects for which we allow relocations to the text segment.
.It BUNDEF_OBJ
.RE

.sp
.ne 2
.na
\fBUNDEF_OBJ\fR
.ad
.RS 17n
.sp
Objects that are allowed to be unreferenced.
.It UNDEF_REF
.RE

.sp
.ne 2
.na
\fBUNDEF_REF\fR
.ad
.RS 17n
.sp
Objects that are allowed undefined references.
.It UNUSED_DEPS
.RE

.sp
.ne 2
.na
\fBUNUSED_DEPS\fR
.ad
.RS 17n
.sp
Objects that are allowed to have unused dependencies.
.It BUNUSED_OBJ
.RE

.sp
.ne 2
.na
\fBUNUSED_OBJ\fR
.ad
.RS 17n
.sp
Objects that are always allowed to be unused dependencies.
.It UNUSED_RPATH
.RE

.sp
.ne 2
.na
\fBUNUSED_RPATH\fR
.ad
.RS 17n
.sp
Objects that are allowed to have unused runpath directories.
.It FORBIDDEN
Specifies that dependencies on a given object are forbidden.
.It FORBIDDEN_DEP
Specifies that a given object is permitted a forbidden dependency.
.El
.Sh ALTERNATIVE DEPENDENCY MAPPING
.Nm check_rtime
was primarily designed to process a nightly builds
.Ev $ROOT
```



```
hierarchy.
It is often the case that objects within this hierarchy must bind to
dependencies within the same hierarchy to satisfy their requirements.
.Pp
.RE

.SH ALTERNATIVE DEPENDENCY MAPPING
.I check_rtime
was primarily designed to process a nightly builds \fB$ROOT\fP
hierarchy. It is often the case that objects within this hierarchy
must bind to dependencies within the same hierarchy to satisfy
their requirements.
.LP
To achieve this,
.Nm check_rtime
uses the shared objects specified with the
.Fl D
or
.Fl d
options.
If neither option is specified, and the
.Ev $CODEMGR_WS
and
.Ev $ROOT
environment variables are defined, the proto area for the workspace is
used.
The objects found are used to create runtime configuration files via
.Xr crle 1 ,
that establish the new shared objects as alternatives to their underlying
system location.
.Nm check_rtime
passes these configuration files as
.Ev LD_CONFIG
environment variable settings to
.Xr ldd 1
using its
.Fl -e
option.
.Pp
.I check_rtime
uses the shared objects specified with the \fB-D\fP or \fB-d\fP options.
If neither option is specified, and the \fB$CODEMGR_WS\fP and \fB$ROOT\fP
environment variables are defined, the proto area for the workspace
is used. The objects found are used
to create runtime configuration files via \fBcrle(1)\fP, that establish
the new shared objects as alternatives to their underlying system location.
.I check_rtime
passes these configuration files as \fBLD_CONFIG\fP environment
variable settings to \fBldd(1)\fP using its \fB-e\fP option.
.LP
The effect of these configuration files is that the execution of an
object under
.Xr ldd 1
will bind to the dependencies defined as alternatives.
Simply put, an object inspected in the
.Pa proto
area will bind to its dependencies found in the
.Pa proto
area.
Dependencies that have no alternative mapping will continue to bind to the
underlying system.
.Sh ENVIRONMENT VARIABLES
When the
.Fl D
or
.Fl d
```

```
option isn't in use,
.Nm check_rtime
object under \fBldd(1)\fP will bind to the dependencies defined as
alternatives.  Simply put, an object inspected in the \fIproto\fP
area will bind to its dependencies found in the \fIproto\fP area.
Dependencies that have no alternative mapping will continue to
bind to the underlying system.
.SH ENVIRONMENT VARIABLES
.LP
When the \fB-D\fP or \fB-d\fP option isn't in use,
.I check_rtime
```
uses the following environment variables to
establish an alternative dependency mapping:
```
.Bl -tag -width indent
.It Ev CODEMGR_WS
.LP
.B CODEMGR_WS
.RS 4
```
The root of your workspace, which is the directory
```
containing
.Pa .git .
Existence of this environment variable indicates that
.Ev $ROOT
should be investigated.
.It Ev ROOT
Root of the
.Pa proto
area of your workspace.
Any shared objects under this directory will be used to establish an
alternative dependency mapping.
.El
If
.Xr ldd 1
supports the
.Fl U
option, it will be used to determine any unreferenced dependencies.
Otherwise
.Xr ldd 1
uses the older
.Fl u
option which only detects unused references.
If the following environment variable exists, and indicates an earlier release
than \fB5.10\fP then
.Xr ldd 1
also falls back to using the
.Fl u
option.
.Bl -tag -width indent
.It Ev RELEASE
containing \fICodemgr_wsdata\fP. Existence of this environment variable
indicates that \fB$ROOT\fP should be investigated.
.RE
.LP
.B ROOT
.RS 4
Root of the \fIproto\fP area of your workspace. Any shared objects
under this directory will be used to establish an alternative dependency
mapping.
.RE
.sp
If \fBldd(1)\fP supports the \fB-U\fP option, it will be used to determine
any unreferenced dependencies.  Otherwise \fBldd(1)\fP uses the older
\fB-u\fP option which only detects unused references.  If the following
environment variable exists, and indicates an earlier release than \fB5.10\fP
then \fBldd(1)\fP also falls back to using the \fB-u\fP option.
.LP
```



```
.B RELEASE
.RS 4
```
The release version number of the environment being built.
```
.El
.Sh ERROR CONDITIONS
Inspection of an object with
.Xr ldd 1
assumes it is compatible with the machine on which
.Nm check_rtime
is being run.
Incompatible objects such as a 64-bit object encountered on a 32-bit system,
or an i386 object encountered on a sparc system, can not be fully inspected.
These objects are displayed as:
.Pp
.Dl foo: has wrong class or data encoding
.Sh FILES
.Bl -tag -width indent
.It Pa $CODEMGR_WS/exception_list/check_rtime
.It Pa /opt/onbld/etc/exception_list/check_rtime
.El
.Sh SEE ALSO
.Xr crle 1 ,
.Xr elfdump 1 ,
.Xr ld.so.1 1 ,
.Xr ldd 1 ,
.Xr mcs 1 ,
.Xr find_elf 1ONBLD
.RE
.SH ERROR CONDITIONS
.LP
Inspection of an object with \fBldd(1)\fP assumes it is compatible
with the machine on which
.I check_rtime
is being run.  Incompatible objects such as a 64-bit object encountered on
a 32-bit system, or an i386 object encountered on a sparc system,
can not be fully inspected.  These objects are displayed as:
.sp
.RS 4
foo: has wrong class or data encoding
.RE
.SH FILES
.LP
.RS 5
$CODEMGR_WS/exception_list/check_rtime
/opt/onbld/etc/exception_list/check_rtime
.SH SEE ALSO
.B crle(1),
.B elfdump(1),
.B find_elf(1ONBLD),
.B ldd(1),
.B ld.so.1(1),
.B mcs(1).
```

```
**********************************************************
   35527 Tue Feb 20 04:36:00 2018
new/usr/src/tools/scripts/check_rtime.pl
9139 check_rtime should be able to forbid libraries
9140 check_rtime should learn libnsl is safe now
9141 check_rtime exceptions could be cleaner
**********************************************************
   1 #!/usr/perl5/bin/perl -w
   2 #
   3 # CDDL HEADER START
   4 #
   5 # The contents of this file are subject to the terms of the
   6 # Common Development and Distribution License (the "License").
   7 # You may not use this file except in compliance with the License.
   8 #
   9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  10 # or http://www.opensolaris.org/os/licensing.
  11 # See the License for the specific language governing permissions
  12 # and limitations under the License.
  13 #
  14 # When distributing Covered Code, include this CDDL HEADER in each
  15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  16 # If applicable, add the following below this CDDL HEADER, with the
  17 # fields enclosed by brackets "[]" replaced with your own identifying
  18 # information: Portions Copyright [yyyy] [name of copyright owner]
  19 #
  20 # CDDL HEADER END
  21 #

  23 #
  24 # Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
  25 #

  27 #
  28 # Check ELF information.
  29 #
  30 # This script descends a directory hierarchy inspecting ELF dynamic executables
  31 # and shared objects.  The general theme is to verify that common Makefile rules
  32 # have been used to build these objects.  Typical failures occur when Makefile
  33 # rules are re-invented rather than being inherited from "cmd/lib" Makefiles.
  34 #
  35 # As always, a number of components don't follow the rules, and these are
  36 # excluded to reduce this scripts output.
  37 #
  38 # By default any file that has conditions that should be reported is first
  39 # listed and then each condition follows.  The -o (one-line) option produces a
  40 # more terse output which is better for sorting/diffing with "nightly".
  41 #
  42 # NOTE: missing dependencies, symbols or versions are reported by running the
  43 # file through ldd(1).  As objects within a proto area are built to exist in a
  44 # base system, standard use of ldd(1) will bind any objects to dependencies
  45 # that exist in the base system.  It is frequently the case that newer objects
  46 # exist in the proto area that are required to satisfy other objects
  47 # dependencies, and without using these newer objects an ldd(1) will produce
  48 # misleading error messages.  To compensate for this, the -D/-d options, or the
  49 # existence of the CODEMSG_WS/ROOT environment variables, cause the creation of
  50 # alternative dependency mappings via crle(1) configuration files that establish
  51 # any proto shared objects as alternatives to their base system location.  Thus
  52 # ldd(1) can be executed against these configuration files so that objects in a
  53 # proto area bind to their dependencies in the same proto area.


  56 # Define all global variables (required for strict)
  57 use vars  qw($Prog $Env $Ena64 $Tmpdir);
  58 use vars  qw($LddNoU $Conf32 $Conf64);
  59 use vars  qw(%opt);
```



```
  60 use vars  qw($ErrFH $ErrTtl $InfoFH $InfoTtl $OutCnt1 $OutCnt2);

  62 # An exception file is used to specify regular expressions to match
  63 # objects. These directives specify special attributes of the object.
  64 # The regular expressions are read from the file and compiled into the
  65 # regular expression variables.
  66 #
  67 # The name of each regular expression variable is of the form
  68 #
  69 #       $EXRE_xxx
  70 #
  71 # where xxx is the name of the exception in lower case. For example,
  72 # the regular expression variable for EXEC_STACK is $EXRE_exec_stack.
  73 #
  74 # onbld_elfmod::LoadExceptionsToEXRE() depends on this naming convention
  75 # to initialize the regular expression variables, and to detect invalid
  76 # exception names.
  77 #
  78 # If a given exception is not used in the exception file, its regular
  79 # expression variable will be undefined. Users of these variables must
  80 # test the variable with defined() prior to use:
  81 #
  82 #       defined($EXRE_exec_stack) && ($foo =~ $EXRE_exec_stack)
  83 #
  84 # or if the test is to make sure the item is not specified:
  85 #
  86 #       !defined($EXRE_exec_stack) || ($foo !~ $EXRE_exec_stack)
  87 #
  88 # ----
  89 #
  90 # The exceptions are:
  91 #
  92 #   EXEC_DATA
  93 #       Objects that are not required to have non-executable writable
  94 #       data segments.
  95 #
  96 #   EXEC_STACK
  97 #       Objects that are not required to have a non-executable stack
  98 #
  99 #   FORBIDDEN_DEP
 100 #       Objects allowed to link to 'forbidden' objects
 101 #
 102 #   FORBIDDEN
 103 #       Objects to which nobody not excepted with FORBIDDEN_DEP may link
 104 #
 105 #endif /* ! codereview */
 106 #   NOCRLEALT
 107 #       Objects that should be skipped by AltObjectConfig() when building
 108 #       the crle script that maps objects to the proto area.
 109 #
 110 #    NODIRECT
 111 #       Objects that are not required to use direct bindings
 112 #
 113 #    NOSYMSORT
 114 #       Objects we should not check for duplicate addresses in
 115 #       the symbol sort sections.
 116 #
 117 #    OLDDEP
 118 #       Objects that are no longer needed because their functionalty
 119 #       has migrated elsewhere. These are usually pure filters that
 120 #       point at libc.
 121 #
 122 #    SKIP
 123 #       Files and directories that should be excluded from analysis.
 124 #
 125 #    STAB
```

```
126 #       Objects that are allowed to contain stab debugging sections
127 #
128 #    TEXTREL
129 #       Object for which relocations are allowed to the text segment
130 #
131 #    UNDEF_REF
132 #       Objects that are allowed undefined references
133 #
134 #    UNREF_OBJ
135 #       "unreferenced object=" ldd(1) diagnostics.
136 #
137 #    UNUSED_DEPS
138 #       Objects that are allowed to have unused dependencies
139 #
140 #    UNUSED_OBJ
141 #       Objects that are allowed to be unused dependencies
142 #
143 #    UNUSED_RPATH
144 #       Objects with unused runpaths
145 #

147 use vars  qw($EXRE_exec_data $EXRE_exec_stack $EXRE_nocrlealt);
148 use vars  qw($EXRE_nodirect $EXRE_nosymsort $EXRE_forbidden_dep $EXRE_forbidden)
 99 use vars  qw($EXRE_nodirect $EXRE_nosymsort);
149 use vars  qw($EXRE_olddep $EXRE_skip $EXRE_stab $EXRE_textrel $EXRE_undef_ref);
150 use vars  qw($EXRE_unref_obj $EXRE_unused_deps $EXRE_unused_obj);
151 use vars  qw($EXRE_unused_rpath);

153 use strict;
154 use Getopt::Std;
155 use File::Basename;


158 # Reliably compare two OS revisions.  Arguments are <ver1> <op> <ver2>.
159 # <op> is the string form of a normal numeric comparison operator.
160 sub cmp_os_ver {
161         my @ver1 = split(/\./, $_[0]);
162         my $op = $_[1];
163         my @ver2 = split(/\./, $_[2]);

165         push @ver2, ("0") x $#ver1 - $#ver2;
166         push @ver1, ("0") x $#ver2 - $#ver1;

168         my $diff = 0;
169         while (@ver1 || @ver2) {
170                 if (($diff = shift(@ver1) - shift(@ver2)) != 0) {
171                         last;
172                 }
173         }
174         return (eval "$diff $op 0" ? 1 : 0);
175 }

177 ## ProcFile(FullPath, RelPath, File, Class, Type, Verdef)
178 #
179 # Determine whether this a ELF dynamic object and if so investigate its runtime
180 # attributes.
181 #
182 sub ProcFile {
183         my($FullPath, $RelPath, $Class, $Type, $Verdef) = @_;
184         my(@Elf, @Ldd, $Dyn, $Sym, $Stack);
185         my($Sun, $Relsz, $Pltsz, $Tex, $Stab, $Strip, $Lddopt, $SymSort);
186         my($Val, $Header, $IsX86, $RWX, $UnDep);
187         my($HasDirectBinding);

189         # Only look at executables and sharable objects
190         return if ($Type ne 'EXEC') && ($Type ne 'DYN');
```



```
192         # Ignore symbolic links
193         return if -l $FullPath;

195         # Is this an object or directory hierarchy we don't care about?
196         return if (defined($EXRE_skip) && ($RelPath =~ $EXRE_skip));

198         # Bail if we can't stat the file. Otherwise, note if it is SUID/SGID.
199         return if !stat($FullPath);
200         my $Secure = (-u _ || -g _) ? 1 : 0;

202         # Reset output message counts for new input file
203         $$ErrTtl = $$InfoTtl = 0;

205         @Ldd = 0;

207         # Determine whether we have access to inspect the file.
208         if (!(-r $FullPath)) {
209                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
210                     "unable to inspect file: permission denied");
211                 return;
212         }

214         # Determine whether we have a executable (static or dynamic) or a
215         # shared object.
216         @Elf = split(/\n/, 'elfdump -epdcy $FullPath 2>&1');

218         $Dyn = $Stack = $IsX86 = $RWX = 0;
219         $Header = 'None';
220         foreach my $Line (@Elf) {
221                 # If we have an invalid file type (which we can tell from the
222                 # first line), or we're processing an archive, bail.
223                 if ($Header eq 'None') {
224                         if (($Line =~ /invalid file/) ||
225                             ($Line =~ /\Q$FullPath\E(.*):/)) {
226                                 return;
227                         }
228                 }

230                 if ($Line =~ /^ELF Header/) {
231                         $Header = 'Ehdr';
232                         next;
233                 }

235                 if ($Line =~ /^Program Header/) {
236                         $Header = 'Phdr';
237                         $RWX = 0;
238                         next;
239                 }

241                 if ($Line =~ /^Dynamic Section/) {
242                         # A dynamic section indicates we're a dynamic object
243                         # (this makes sure we don't check static executables).
244                         $Dyn = 1;
245                         next;
246                 }

248                 if (($Header eq 'Ehdr') && ($Line =~ /e_machine:/)) {
249                         # If it's a X86 object, we need to enforce RW- data.
250                         $IsX86 = 1 if $Line =~ /(EM_AMD64|EM_386)/;
251                         next;
252                 }

254                 if (($Header eq 'Phdr') &&
255                     ($Line =~ /\[ PF_X\s+PF_W\s+PF_R \]/)) {
256                         # RWX segment seen.
```

```
257                         $RWX = 1;
258                         next;
259                 }

261                 if (($Header eq 'Phdr') &&
262                     ($Line =~ /\[ PT_LOAD \]/ && $RWX && $IsX86)) {
263                         # Seen an RWX PT_LOAD segment.
264                         if (!defined($EXRE_exec_data) ||
265                             ($RelPath !~ $EXRE_exec_data)) {
266                             onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
267                                 "application requires non-executable " .
268                                 "data\t<no -Mmapfile_noexdata?>");
269                         }
270                         next;
271                 }

273                 if (($Header eq 'Phdr') && ($Line =~ /\[ PT_SUNWSTACK \]/)) {
274                         # This object defines a non-executable stack.
275                         $Stack = 1;
276                         next;
277                 }
278         }

280         # Determine whether this ELF executable or shared object has a
281         # conforming mcs(1) comment section.  If the correct $(POST_PROCESS)
282         # macros are used, only a 3 or 4 line .comment section should exist
283         # containing one or two "@(#)SunOS" identifying comments (one comment
284         # for a non-debug build, and two for a debug build). The results of
285         # the following split should be three or four lines, the last empty
286         # line being discarded by the split.
287         if ($opt{m}) {
288                 my(@Mcs, $Con, $Dev);

290                 @Mcs = split(/\n/, 'mcs -p $FullPath 2>&1');

292                 $Con = $Dev = $Val = 0;
293                 foreach my $Line (@Mcs) {
294                         $Val++;

296                         if (($Val == 3) && ($Line !~ /^@\(#\)SunOS/)) {
297                                 $Con = 1;
298                                 last;
299                         }
300                         if (($Val == 4) && ($Line =~ /^@\(#\)SunOS/)) {
301                                 $Dev = 1;
302                                 next;
303                         }
304                         if (($Dev == 0) && ($Val == 4)) {
305                                 $Con = 1;
306                                 last;
307                         }
308                         if (($Dev == 1) && ($Val == 5)) {
309                                 $Con = 1;
310                                 last;
311                         }
312                 }
313                 if ($opt{m} && ($Con == 1)) {
314                         onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
315                             "non-conforming mcs(1) comment\t<no \$(POST_PROCESS)?>");
316                 }
317         }

319         # Applications should contain a non-executable stack definition.
320         if (($Type eq 'EXEC') && ($Stack == 0) &&
321             (!defined($EXRE_exec_stack) || ($RelPath !~ $EXRE_exec_stack))) {
322                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
```



```
323                         "non-executable stack required\t<no -Mmapfile_noexstk?>");
324         }

326         # Having caught any static executables in the mcs(1) check and non-
327         # executable stack definition check, continue with dynamic objects
328         # from now on.
329         if ($Dyn eq 0) {
330                 return;
331         }

333         # Use ldd unless its a 64-bit object and we lack the hardware.
334         if (($Class == 32) || $Ena64) {
335                 my $LDDFullPath = $FullPath;

337                 if ($Secure) {
338                         # The execution of a secure application over an nfs file
339                         # system mounted nosuid will result in warning messages
340                         # being sent to /var/adm/messages.  As this type of
341                         # environment can occur with root builds, move the file
342                         # being investigated to a safe place first.  In addition
343                         # remove its secure permission so that it can be
344                         # influenced by any alternative dependency mappings.
345
346                         my $File = $RelPath;
347                         $File =~ s!^.*/!!;      # basename

349                         my($TmpPath) = "$Tmpdir/$File";

351                         system('cp', $LDDFullPath, $TmpPath);
352                         chmod 0777, $TmpPath;
353                         $LDDFullPath = $TmpPath;
354                 }

356                 # Use ldd(1) to determine the objects relocatability and use.
357                 # By default look for all unreferenced dependencies.  However,
358                 # some objects have legitimate dependencies that they do not
359                 # reference.
360                 if ($LddNoU) {
361                         $Lddopt = "-ru";
362                 } else {
363                         $Lddopt = "-rU";
364                 }
365                 @Ldd = split(/\n/, 'ldd $Lddopt $Env $LDDFullPath 2>&1');
366                 if ($Secure) {
367                         unlink $LDDFullPath;
368                 }
369         }

371         $Val = 0;
372         $Sym = 5;
373         $UnDep = 1;

375         foreach my $Line (@Ldd) {

377                 if ($Val == 0) {
378                         $Val = 1;
379                         # Make sure ldd(1) worked.  One possible failure is that
380                         # this is an old ldd(1) prior to -e addition (4390308).
381                         if ($Line =~ /usage:/) {
382                                 $Line =~ s/$/\t<old ldd(1)?>/;
383                                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl,
384                                     $RelPath, $Line);
385                                 last;
386                         } elsif ($Line =~ /execution failed/) {
387                                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl,
388                                     $RelPath, $Line);
```

```
389                             last;
390                     }
392                     # It's possible this binary can't be executed, ie. we've
393                     # found a sparc binary while running on an intel system,
394                     # or a sparcv9 binary on a sparcv7/8 system.
395                     if ($Line =~ /wrong class/) {
396                             onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
397                                 "has wrong class or data encoding");
398                             next;
399                     }
401                     # Historically, ldd(1) likes executable objects to have
402                     # their execute bit set.
403                     if ($Line =~ /not executable/) {
404                             onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
405                                 "is not executable");
406                             next;
407                     }
408             }
410             # Look for "file" or "versions" that aren't found.  Note that
411             # these lines will occur before we find any symbol referencing
412             # errors.
413             if (($Sym == 5) && ($Line =~ /not found\)/)) {
414                     if ($Line =~ /file not found\)/) {
415                             $Line =~ s/$/\t<no -zdefs?>/;
416                     }
417                     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath, $Line);
418                     next;
419             }
420             # Look for relocations whose symbols can't be found.  Note, we
421             # only print out the first 5 relocations for any file as this
422             # output can be excessive.
423             if ($Sym && ($Line =~ /symbol not found/)) {
424                     # Determine if this file is allowed undefined
425                     # references.
426                     if (($Sym == 5) && defined($EXRE_undef_ref) &&
427                         ($RelPath =~ $EXRE_undef_ref)) {
428                             $Sym = 0;
429                             next;
430                     }
431                     if ($Sym-- == 1) {
432                             onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
433                                 "continued ...") if !$opt{o};
434                             next;
435                     }
436                     # Just print the symbol name.
437                     $Line =~ s/$/\t<no -zdefs?>/;
438                     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath, $Line);
439                     next;
440             }
441             # Look for any unused search paths.
442             if ($Line =~ /unused search path=/) {
443                     next if defined($EXRE_unused_rpath) &&
444                         ($Line =~ $EXRE_unused_rpath);
446                     if ($Secure) {
447                             $Line =~ s!$Tmpdir/!!;
448                     }
449                     $Line =~ s/^[ \t]*(.*)/\t$1\t<remove search path?>/;
450                     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath, $Line);
451                     next;
452             }
453
454 #endif /* ! codereview */
```



```
455             # Look for unreferenced dependencies.  Note, if any unreferenced
456             # objects are ignored, then set $UnDep so as to suppress any
457             # associated unused-object messages.
458             if ($Line =~ /unreferenced object=/) {
459                     if (defined($EXRE_unref_obj) &&
460                         ($Line =~ $EXRE_unref_obj)) {
461                             $UnDep = 0;
462                             next;
463                     }
464                     if ($Secure) {
465                             $Line =~ s!$Tmpdir/!!;
466                     }
467                     $Line =~ s/^[ \t]*(.*)/$1\t<remove lib or -zignore?>/;
468                     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath, $Line);
469                     next;
470             }
471             # Look for any unused dependencies.
472             if ($UnDep && ($Line =~ /unused/)) {
473                     # Skip if object is allowed to have unused dependencies
474                     next if defined($EXRE_unused_deps) &&
475                         ($RelPath =~ $EXRE_unused_deps);
477                     # Skip if dependency is always allowed to be unused
478                     next if defined($EXRE_unused_obj) &&
479                         ($Line =~ $EXRE_unused_obj);
481                     $Line =~ s!$Tmpdir/!! if $Secure;
482                     $Line =~ s/^[ \t]*(.*)/$1\t<remove lib or -zignore?>/;
483                     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath, $Line);
484                     next;
485             }
486     }
488     # Reuse the elfdump(1) data to investigate additional dynamic linking
489     # information.
491     $Sun = $Relsz = $Pltsz = $Dyn = $Stab = $SymSort = 0;
492     $Tex = $Strip = 1;
493     $HasDirectBinding = 0;
495     $Header = 'None';
496 ELF:    foreach my $Line (@Elf) {
497             # We're only interested in the section headers and the dynamic
498             # section.
499             if ($Line =~ /^Section Header/) {
500                     $Header = 'Shdr';
502                     if (($Sun == 0) && ($Line =~ /\.SUNW_reloc/)) {
503                             # This object has a combined relocation section.
504                             $Sun = 1;
506                     } elsif (($Stab == 0) && ($Line =~ /\.stab/)) {
507                             # This object contain .stabs sections
508                             $Stab = 1;
509                     } elsif (($SymSort == 0) &&
510                              ($Line =~ /\.SUNW_dyn(sym)|(tls)sort/)) {
511                             # This object contains a symbol sort section
512                             $SymSort = 1;
513                     }
515                     if (($Strip == 1) && ($Line =~ /\.symtab/)) {
516                             # This object contains a complete symbol table.
517                             $Strip = 0;
518                     }
519                     next;
```

```
521                 } elsif ($Line =~ /^Dynamic Section/) {
522                         $Header = 'Dyn';
523                         next;
524                 } elsif ($Line =~ /^Syminfo Section/) {
525                         $Header = 'Syminfo';
526                         next;
527                 } elsif (($Header ne 'Dyn') && ($Header ne 'Syminfo')) {
528                         next;
529                 }

531                 # Look into the Syminfo section.
532                 # Does this object have at least one Directly Bound symbol?
533                 if (($Header eq 'Syminfo')) {
534                         my(@Symword);

536                         if ($HasDirectBinding == 1) {
537                                 next;
538                         }

540                         @Symword = split(' ', $Line);

542                         if (!defined($Symword[1])) {
543                                 next;
544                         }
545                         if ($Symword[1] =~ /B/) {
546                                 $HasDirectBinding = 1;
547                         }
548                         next;
549                 }

551                 # Does this object contain text relocations.
552                 if ($Tex && ($Line =~ /TEXTREL/)) {
553                         # Determine if this file is allowed text relocations.
554                         if (defined($EXRE_textrel) &&
555                             ($RelPath =~ $EXRE_textrel)) {
556                                 $Tex = 0;
557                                 next ELF;
558                         }
559                         onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
560                             "TEXTREL .dynamic tag\t\t\t<no -Kpic?>");
561                         $Tex = 0;
562                         next;
563                 }

565                 # Does this file have any relocation sections (there are a few
566                 # psr libraries with no relocations at all, thus a .SUNW_reloc
567                 # section won't exist either).
568                 if (($Relsz == 0) && ($Line =~ / RELA?SZ/)) {
569                         $Relsz = hex((split(' ', $Line))[2]);
570                         next;
571                 }

573                 # Does this file have any plt relocations.  If the plt size is
574                 # equivalent to the total relocation size then we don't have
575                 # any relocations suitable for combining into a .SUNW_reloc
576                 # section.
577                 if (($Pltsz == 0) && ($Line =~ / PLTRELSZ/)) {
578                         $Pltsz = hex((split(' ', $Line))[2]);
579                         next;
580                 }

582                 # Does this object have any dependencies.
583                 if ($Line =~ /NEEDED/) {
584                         my($Need) = (split(' ', $Line))[3];

586                         if (defined($EXRE_olddep) && ($Need =~ $EXRE_olddep)) {
```



```
587                                 # Catch any old (unnecessary) dependencies.
588                                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
589                                     "NEEDED=$Need\t<dependency no " .
590                                     "longer necessary>");
591                         } elsif (defined($EXRE_forbidden) &&
592                             ($Need =~ $EXRE_forbidden)) {
593                                 next if defined($EXRE_forbidden_dep) &&
594                                     ($FullPath =~ $EXRE_forbidden_dep);

596                                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
597                                     "NEEDED=$Need\t<forbidden dependency, " .
598                                     "missing -nodefaultlibs?>");
599                                 next;
404                         "NEEDED=$Need\t<dependency no longer necessary>");
600                         } elsif ($opt{i}) {
601                                 # Under the -i (information) option print out
602                                 # any useful dynamic entries.
603                                 onbld_elfmod::OutMsg($InfoFH, $InfoTtl, $RelPath
604                                     "NEEDED=$Need");
605                         }
606                         next;
607                 }

609                 # Is this object built with -B direct flag on?
610                 if ($Line =~ / DIRECT /) {
611                         $HasDirectBinding = 1;
612                 }

614                 # Does this object specify a runpath.
615                 if ($opt{i} && ($Line =~ /RPATH/)) {
616                         my($Rpath) = (split(' ', $Line))[3];
617                         onbld_elfmod::OutMsg($InfoFH, $InfoTtl,
618                             $RelPath, "RPATH=$Rpath");
619                         next;
620                 }
621         }

623         # A shared object, that contains non-plt relocations, should have a
624         # combined relocation section indicating it was built with -z combreloc.
625         if (($Type eq 'DYN') && $Relsz && ($Relsz != $Pltsz) && ($Sun == 0)) {
626                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
627                     ".SUNW_reloc section missing\t\t<no -zcombreloc?>");
628         }

630         # No objects released to a customer should have any .stabs sections
631         # remaining, they should be stripped.
632         if ($opt{s} && $Stab) {
633                 goto DONESTAB if defined($EXRE_stab) && ($RelPath =~ $EXRE_stab)

635                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
636                     "debugging sections should be deleted\t<no strip -x?>");
637         }

639         # Identify an object that is not built with either -B direct or
640         # -z direct.
641         goto DONESTAB
642             if (defined($EXRE_nodirect) && ($RelPath =~ $EXRE_nodirect));

644         if ($Relsz && ($HasDirectBinding == 0)) {
645                 onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
646                  "object has no direct bindings\t<no -B direct or -z direct?>");
647         }

649 DONESTAB:

651         # All objects should have a full symbol table to provide complete
```

```
 652         # debugging stack traces.
 653         onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
 654             "symbol table should not be stripped\t<remove -s?>") if $Strip;

 656         # If there are symbol sort sections in this object, report on
 657         # any that have duplicate addresses.
 658         ProcSymSort($FullPath, $RelPath) if $SymSort;

 660         # If -v was specified, and the object has a version definition
 661         # section, generate output showing each public symbol and the
 662         # version it belongs to.
 663         ProcVerdef($FullPath, $RelPath)
 664             if ($Verdef eq 'VERDEF') && $opt{v};
 665 }
```

***unchanged_portion_omitted_***