

new/exception_lists/check_rtime

1

```

*****
10036 Tue Feb 20 01:00:50 2018
new/exception_lists/check_rtime
XXXX check_rtime should be able to forbid libraries
XXXX check_rtime needs to learn libnsl is safe now
XXXX check_rtime could use MACH() more thoroughly
XXXX check_rtime exceptions could be tidied up.
*****
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 #
21 #
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 #
26 #
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
41 #
42 #
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
48 #
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
56 #
57 # picl file exclusions (4385799)
58 SKIP      ^usr/platform/.*/*libpsvcplugin_psr\.so\.1

```

new/exception_lists/check_rtime

2

```

59 SKIP      ^usr/platform/.*/*libpsvcpolicy_psr\.so\.1
60 SKIP      ^usr/platform/.*/*libpsvcpolicy\.so\.1
61 SKIP      ^usr/lib/syevent/modules/picl_slm.so$
62 #
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
67 EXEC_DATA ^lib/amd64/libumem\.so\.1$ # ptcumem
68 EXEC_DATA ^lib/libumem\.so\.1$ # ptcumem
69 EXEC_DATA ^opt/SUNWdtrt/tst/.*/*ustack/tst\.helper\.exe$
70 EXEC_DATA ^platform/.*/*MACH(kernel)/unix$
71 EXEC_DATA ^platform/.*/*multiboot$
72 #
73 # Objects that are allowed to have an executable stack
74 EXEC_STACK ^platform/.*/*MACH(kernel)/unix$
75 EXEC_STACK ^platform/.*/*multiboot$
76 EXEC_STACK ^opt/os-tests/tests/secflags/stacky$
77 #
78 # Objects for which we allow relocations to the text segment
79 TEXTREL ^platform/.*/*MACH(kernel)/unix$
80 #
81 # Directories and files that are allowed to have no direct bound symbols
82 NODIRECT ^platform/.*/*MACH(kernel)/unix$
83 NODIRECT ^usr/ucb
84 NODIRECT ^usr/4lib/sbcp$
85 #
86 # Identify any files that should be skipped when building a crle(1)
87 # configuration file. As the hwcap libraries can be loop-back mounted onto
88 # libc, these can confuse crle(1) because of their identical dev/inode.
89 NOCRLEALT ^usr/lib/libc/libc_hwcap[1-3].so.1$
90 #
91 # Files that should contain debugging information.
92 STAB ^platform/.*/*MACH(kernel)/unix$
93 #
94 # Files that are allowed undefined references
95 UNDEF_REF ^usr/lib/libnisdb\.so\.2$
96 #
97 # Objects allowed to have unused dependencies
98 UNUSED_DEPS ^usr/lib/picl/plugins/ # require devtree dependencies
99 # libm.so.2 dependency
100 UNUSED_OBJ unused object=.*MACH(lib)/libm_hwcap1\.so\.2
101 #
102 # libnetsnmphelpers.so is empty in some net-snmp versions
103 UNUSED_OBJ unused object=.*/*libnetsnmphelpers\.so\..*
104 UNREF_OBJ unreferenced object=.*/*libnetsnmphelpers\.so\..*
105 #
106 # Unused runpaths due to dlopen() use
107 UNUSED_RPATH /usr/lib/fs/autofs.*\ from\ .automountd
108 UNUSED_RPATH /etc/ppp/plugins.*\ from\ .pppd
109 UNUSED_RPATH /usr/lib/inet/ppp.*\ from\ .pppd
110 UNUSED_RPATH /usr/sfw/lib.*\ from\ .*libipsecutil\.so\.1
111 UNUSED_RPATH /usr/platform/.*rsmlib.*\ from\ .*librsm\.so\.2
112 UNUSED_RPATH $ORIGIN.*\ from\ .*fcode.so
113 UNUSED_RPATH /opt/VRTSvxvm/lib.*\ from\ .*libdiskmgt\.so\.1
114 #
115 # Unused runpaths in picl code
116 UNUSED_RPATH /usr/platform/.*\ from\ .*/*usr/platform
117 UNUSED_RPATH /usr/lib/picl.*\ from\ .*/*usr/platform
118 UNUSED_RPATH /usr/platform/.*\ from\ .*/*usr/lib/picl
119 #
120 # Unused runpaths in non-OSNET objects we can't change
121 UNUSED_RPATH /usr/lib/mps.*\ from\ .*libnss3\.so
122 UNUSED_RPATH /usr/lib/mps.*\ from\ .*libnssutil3\.so
123 UNUSED_RPATH /usr/lib/mps.*\ from\ .*libmime3\.so

```

new/exception_lists/check_rtime

3

```

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
141 UNUSED_RPATH /usr/lib.*\ from\ .*usr/lib/mps
141 UNUSED_RPATH /usr/ccs/lib.*\ from\ .*usr/lib/mps
140 UNUSED_RPATH /usr/gnu/lib.*\ from\ .*usr/lib/libpython2\..
141 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/smb.*\ 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\ .*libspr4\.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\ .*libspmicommon\.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 /lib.*\ of\ .*libstdc\+\+\.so\6 # gcc build
160 UNREF_OBJ /lib.*\ of\ .*libpicl/plugins/ # picl
161 UNREF_OBJ /lib.*\ of\ .*kcfid # 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

```

new/exception_lists/check_rtime

4

```

164 # statics which have the dependence.
165 UNREF_OBJ /libc\.so\1.*\ of\ .*kldap\.so\1

```

```

168 # Objects that used to contain system functionality that has since
169 # migrated to libc. We preserve these libraries as pure filters for
170 # backward compatibility 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 assembler 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
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/libns1\.so\1 # C++
220 NOSYMSORT lib/sparcv9/libns1\.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 accepted.

```

```

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(smb)/libfksmb.so.1
219 FORBIDDEN_DEP usr/lib/smb/fksmb

```

```
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/uch/
228 FORBIDDEN_DEP usr/ucplib/

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-CP2300/1
```

```

*****
20847 Tue Feb 20 01:00:51 2018
new/usr/src/tools/scripts/check_rtime.1onbld
XXXX check_rtime should be able to forbid libraries
XXXX check_rtime needs to learn libnsl is safe now
XXXX check_rtime could use MACH() more thoroughly
XXXX check_rtime exceptions could be tidied up.
*****
1  \. Copyright (c) 2001, 2010, Oracle and/or its affiliates. All rights reserved.
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 \.
22 .Dd February 19, 2018
23 .Dt CHECK_RUNTIME 1ONBLD
24 .Os
25 .Sh NAME
26 .Nm check_rtime
27 .Nd check ELF runtime attributes
28 .Sh SYNOPSIS
29 .Nm check_rtime
30 .Op Fl imosv
31 .Op Fl D Ar depfile | Fl d depdir
32 .Op Fl E Ar errfile
33 .Op Fl e Ar exfile
34 .Op Fl f Ar listfile
35 .Op Fl I Ar infofile
36 .Op Fl w Ar outdir
37 .Ar file | dir ...
38 .Sh DESCRIPTION
39 .Nm check_rtime
40 .TH CHECK_RUNTIME 1ONBLD "Mar 09, 2010"
41 .SH NAME
42 .I check_rtime
43 \- check ELF runtime attributes
44 .SH SYNOPSIS
45 \fBcheck_rtime [-imosv] [-D depfile | -d depdir] [-E errfile] [-e exfile] [-f li
46 .SH DESCRIPTION
47 .LP
48 .I check_rtime
49 attempts to check a number of ELF runtime attributes
50 for consistency with common build rules.
51 These checks involve running
52 .Xr ldd 1
53 and
54 .Xr elfdump 1
55 against a family of dynamic objects.
56 A dynamic object can be defined explicitly as a
57 .Ar file
58 or multiple dynamic objects can be located under the directory

```

```

50 .Ar dir .
51 .Pp
52 .Nm check_rtime
53 is typically called from
54 .Xr nightly 1ONBLD
55 when the
56 .Fl r
57 option is in effect.
58 In this case the dynamic objects under
59 the associated
60 .Em proto area
61 .Pq Ev $ROOT
62 are checked.
63 .Nm check_rtime
64 These checks involve running \fBldd(1)\fP and
65 \fBelfdump(1)\fP against a family of dynamic objects.
66 A dynamic object can be defined explicitly as a \fIfile\fP
67 or multiple dynamic objects can be located under the directory \fIIdir\fP.
68 .LP
69 .I check_rtime
70 is typically called from \fBnightly(1ONBLD)\fP when the \fB-r\fP
71 option is in effect. In this case the dynamic objects under
72 the associated \fIproto\fP area (\fB$ROOT\fP) are checked.
73 .I check_rtime
74 can also be run standalone against any set of dynamic objects.
75 .Pp
76 .Nm check_rtime
77 uses
78 .Xr ldd 1
79 to verify dependencies.
80 This implies that by default any object inspected will bind to its dependencies
81 as they are found in the
82 .Em underlying system .
83 Use of the
84 .Fl D ,
85 .Fl d ,
86 .LP
87 .I check_rtime
88 uses \fBldd(1)\fP to verify dependencies. This implies that
89 by default any object inspected will bind to its dependencies
90 as they are found in the \fBunderlying\fP system. Use of the \fB-D\fP, \fB-d\fP
91 option, or the existence of the environment variables
92 .Ev $CODEMGR_WS
93 or
94 .Ev $ROOT
95 instruct
96 .Nm check_rtime
97 \fB$CODEMGR_WS/$ROOT\fP instruct
98 .I check_rtime
99 to establish an alternative dependency mapping using
100 runtime configuration files generated with
101 .Xr crle 1 .
102 .Pp
103 .Nm check_rtime
104 uses
105 .Xr ldd 1
106 to completely relocate any dynamic object and thus detect missing
107 dependencies, unsatisfied symbol relocations, unused and unreferenced
108 dependencies.
109 These checks are carried out for the following reasons:
110 .Bl -bullet
111 .It
112 runtime configuration files generated with \fBcrle(1)\fP.
113 .LP
114 .I check_rtime
115 uses \fBldd(1)\fP to completely relocate any dynamic

```

```

57 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 $(LDLIBS) )
150 that have been copy and pasted
151 between
152 .Pa Makefiles
153 without verifying their need, are a typical reason why unused dependencies
154 exist.
155 Unused dependencies are displayed as:
156 .Pp
157 .Dl foo: unused object=bar.so.1 <remove lib or -zignore?>
158 .It
106 load and relocate, but will not be used by the calling object. They
107 also result in unnecessary processing at link-edit time.
108 .sp
109 Dependency lists (typically defined via \fB$(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 combrelloc
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 nocombrelloc
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 -zcombrelloc?>
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
260 .TP 4
261 \(\bu
262 A concatenated relocation section (\fI.SUNW_reloc\fP)
263 provides optimal symbol table
264 access at runtime, and thus reduces the overhead of relocating
265 the shared object. In past releases, the link-edit of a dynamic object with
266 the \fB-z combrelloc\fP option was required to generate a combined
267 relocation section. However, with the integration of 6642769, this section
268 combination is a default behavior of the link-editor.
269 .sp
270 In past releases, not inheriting \fB$(DYNFLAGS)\fP from
271 \fIlib/Makefile.lib\fP was the typical reason for not having a
272 concatenated relocation section. The misguided use of the
273 \fB-z nocombrelloc\fP option will also prevent the creation of a
274 concatenated relocation section. A missing concatenated relocation section
275 is displayed as:
276 .sp
277 .RS 6
278 foo: .SUNW_reloc section missing <no -zcombrelloc?>
279 .RE
280 .TP
281 \(\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$(LD_FLAGS)\fP from \fIcmd/Makefile.cmd\fP
207 (which asserts \fP-s\fP), or \fB$(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
213 foo: debugging sections should be deleted \(\bu
214 .br
215 <no strip -x?>
216 .RE
217 .TP
218 \(\bu
219 All objects should retain their full \fI.syntab\fP symbol table.
220 Although this consumes disk space, it provides for more extensive stack
221 tracing when debugging user applications.
222 .Pp
223 Hard coding a
224 .Fl s
225 flag with
226 \fB$(LD_FLAGS) or
227 \fB$(DYN_FLAGS)
228 is the typical reason for symbol tables being removed.
229 .sp
230 Hard coding a \fI-s\fP flag with \fB$(LD_FLAGS)\fP or
231 \fB$(DYN_FLAGS)\fP is the typical
232 reason for symbol tables being removed.
233 Objects that do not contain a symbol table are displayed as:
234 .Bd -literal
235 .sp
236 .RS 6
237 foo.so.1: symbol table should not be stripped \(\bu
238 .br
239 <remove -s?>
240 .RE

```

```

233 .TP
234 \(\bu
235 Applications should have a non-executable stack defined to make
236 them less vulnerable to buffer overflow attacks.
237 .Pp
238 Not inheriting the
239 \fB$(LD_FLAGS)
240 macro in
241 \fIcmd/Makefile.cmd
242 .sp
243 Not inheriting the \fB$(LD_FLAGS)\fP macro in \fIcmd/Makefile.cmd\fP
244 is the typical reason for not having a non-executable stack definition.
245 Applications without this definition are displayed as:
246 .Bd -literal
247 .sp
248 .RS 6
249 foo: application requires non-executable stack \(\bu
250 .br
251 .nf
252 <no -Mmapfile_noexstk?>
253 .RE
254 .TP
255 x86 applications should have a non-executable data segment defined to make
256 them less vulnerable to buffer overflow attacks.
257 .Pp
258 Not inheriting the
259 \fB$(LD_FLAGS)
260 macro in
261 \fIcmd/Makefile.cmd
262 .sp
263 Not inheriting the \fB$(LD_FLAGS)\fP macro in \fIcmd/Makefile.cmd\fP
264 is the typical reason for not having a non-executable data definition.
265 Applications without this definition are displayed as:
266 .Bd -literal
267 .sp
268 .RS 6
269 foo: application requires non-executable data \(\bu
270 .br
271 .nf
272 <no -Mmapfile_noexdata?>
273 .RE
274 .TP
275 Solaris ELF files contain symbol sort sections used by DTrace to
276 map addresses in memory to the related function or variable symbols.
277 There are two such sections,
278 .Em .SUNW_dynsymsort
279 for regular symbols, and
280 .Em .SUNW_dyntlssort
281 for thread-local symbols.
282 To ensure that the best names are shown for each such address, and that the
283 same name is given across Solaris releases,
284 .Nm check_rtime
285 map addresses in memory to the related function or variable symbols. There
286 are two such sections, \fI.SUNW_dynsymsort\fP for
287 regular symbols, and \fI.SUNW_dyntlssort\fP for thread-local
288 symbols. To ensure that the best names are shown for each

```

```

274 such address, and that the same name is given across Solaris releases,
275 .I check_rtime
318 enforces the rule that only one symbol can appear in the sort sections for
319 any given address.
320 There are two common ways in which multiple symbols
321 or a given address occur in the ON distribution.
322 The first is from code written in assembly language.
323 The second is as a result of using
324 .Ic #pragma weak
325 in C to create weak symbols.
326 The best solution to this situation is to modify the code to avoid symbol
327 aliasing.
328 Alternatively, the
329 .Va NODYNSORT
330 mapfile attribute can be used to eliminate the unwanted symbol.
331 .Pp
279 or a given address occur in the ON distribution. The first is from
280 code written in assembly language. The second is as a
281 result of using \fB#pragma weak\fP in C to create weak symbols. The
282 best solution to this
283 situation is to modify the code to avoid symbol aliasing. Alternatively,
284 the \fBNODYNSORT\fP mapfile attribute can be used to eliminate the unwanted
285 symbol.
286 .sp
332 Duplicate entries in a symbol sort section are
333 displayed in one of the following ways, depending on
334 whether the section is for regular or thread-local symbols:
335 .Bd -literal
290 .sp
291 .RS 6
336 foo: .SUNW_dynsymsort: duplicate ADDRESS: sym1, sym2
293 .br
337 foo: .SUNW_dyntlssort: duplicate OFFSET: sym1, sym2
338 .Ed
339 .It
340 illumos dynamic ELF objects are expected to employ direct bindings whenever
341 feasible.
342 This runtime binding technique helps to avoid accidental interposition
343 problems, and provides a more optimal runtime symbol search model.
344 .Pp
345 Not inheriting the correct
346 .Dv $(LD_FLAGS) from
347 .Pa cmd/Makefile.cmd ,
348 or the correct
349 .Dv $(DYN_FLAGS)
350 from
351 .Pa lib/Makefile.lib ,
352 are the typical reasons for not enabling direct bindings.
353 Dynamic objects that do not contain direct binding information are displayed
354 as:
355 .Bd -literal
295 .RE
296 .sp
297 .TP
298 \{(bu
299 \fBOSNet\fP dynamic ELF objects are expected to employ direct bindings whenever
300 feasible. This runtime binding technique helps to avoid accidental
301 interposition problems, and provides a more optimal
302 runtime symbol search model.
303 .sp
304 Not inheriting the correct \fB$(LD_FLAGS)\fP from \fIcmd/Makefile.cmd\fP,
305 or the correct \fB$(DYN_FLAGS)\fP from \fIlib/Makefile.lib\fP, are the
306 typical reasons for not enabling direct bindings. Dynamic objects that
307 do not contain direct binding information are displayed as:
308 .sp
309 .RS 6

```

```

356 foo: object has no direct bindings \\
311 .br
312 .nf
357 <no -B direct or -z direct?>
358 .Ed
359 .El
360 .Pp
361 .Nm check_rtime
362 also
363 uses
364 .Xr elfdump 1
365 to display useful dynamic entry information under the
366 .Fl -i
367 option.
314 .fi
315 .RE

317 .sp
318 .LP
319 .I check_rtime also
320 uses \fBelfdump(1)\fP
321 to display useful dynamic entry information under the \fB-i\fP option.
368 This doesn't necessarily indicate an error condition, but
369 provides information that is often useful for gatekeepers to track
370 changes in a release.
371 Presently the information listed is:
372 .Bl -bullet
373 .It
374 Runpaths are printed for any dynamic object.
375 This is a historic sanity check to insure compiler supplied runpaths
376 (typically from
377 .Nm CC )
378 are not recorded in any objects.
379 Runpaths are displayed as:
380 .Pp
381 .Dl foo: RPATH=/usr/bar/lib
382 .It
324 changes in a release. Presently the information listed is:
325 .TP
326 \{(bu
327 Runpaths are printed for any dynamic object. This is a historic
328 sanity check to insure compiler supplied runpaths (typically from \fBCC\fP)
329 are not recorded in any objects. Runpaths are displayed as:
330 .sp
331 .RS 6
332 foo: RPATH=/usr/bar/lib
333 .RE
334 .TP
335 \{(bu
383 Needed dependencies are printed for any dynamic object.
384 In the freeware world this often helps the introducer of a new
385 shared object discover that an existing binary has become its
386 consumer, and thus that binaries package dependencies may require updating.
387 Dependencies are printed as:
388 .Pp
389 .Dl foo: NEEDED=bar.so.1
390 .It
391 Dependencies may be marked as forbidden
392 .Pq see Sx EXCEPTION FILE FORMAT
393 this allows the build to warn should people use them accidentally.
394 Forbidden dependencies are printed as:
395 .Pp
396 .Dl foo: NEEDED=bar.so.1 <forbidden dependency, missing -nodefaultlibs?>
397 .El
398 .Pp
399 .Nm check_rtime

```

```

400 uses
401 .Xr mcs 1
402 to inspect an object's
403 .Em .comment
404 section.
405 .sp
406 .RS 6
407 foo: NEEDED=bar.so.1
408 .RE
409 .sp
410 .LP
411 .I check_rtime
412 uses \fBmcs(1)\fP to inspect an object's \fI.comment\fP section.
413 During development, this section contains numerous file identifiers
414 marked with the tag
415 .Qq @(#) .
416 For release builds these sections are deleted and rewritten under control of
417 the
418 .Dv $(POST_PROCESS)
419 macro to produce a common release identifier.
420 This identifier typically consists of three lines including a single comment
421 starting with the string
422 .Qq @(#) SunOS .
423 If this common identifier isn't found the following diagnostic is generated:
424 .Pp
425 .Dl foo: non-conforming mcs(1) comment <no $(POST_PROCESS)?>
426 .Pp
427 .Nm check_rtime
428 uses
429 .Xr pvs 1
430 to display version definitions under the
431 .Fl v
432 option.
433 marked with the tag "\fB@(#) \fP". For release builds these sections
434 are deleted and rewritten under control of the \fB$(POST_PROCESS)\fP
435 macro to produce a common release identifier. This identifier
436 typically consists of three lines including a single comment starting
437 with the string "\fB@(#) SunOS \fP". If this common identifier isn't
438 found the following diagnostic is generated:
439 .sp
440 .RS 6
441 foo: non-conforming mcs(1) comment <no $(POST_PROCESS)?>
442 .RE
443 .sp
444 .LP
445 .I check_rtime
446 uses \fBpvs(1)\fP to display version definitions under the \fB-v\fP option.
447 Each symbol defined by the object is shown along with the version it belongs to.
448 Changes to the symbols defined by an object, or the versions they belong to,
449 do not necessarily indicate an error condition, but
450 provides information that is often useful for gatekeepers to track
451 changes in a release.
452 .Sh OPTIONS
453 .sp
454 .SH OPTIONS
455 .LP
456 The following options are supported:
457 .Bl -tag -width indent
458 .It Fl D Ar depfile
459 Use
460 .Ar depfile
461 to generate an alternative dependency mapping.
462 .Ar depfile
463 must be created by
464 .Ic find_elf -r .
465 The

```

```

441 .Fl D
442 and
443 .Fl d
444 options are mutually exclusive.
445 .It Fl d Ar depfile
446 Use
447 .Ar depdir
448 to generate an alternative dependency mapping.
449 .Xr find_elf 1ONBLD
450 is used to locate the ELF sharable objects for which alternative mappings are
451 required.
452 The
453 .Fl D
454 and
455 .Fl d
456 options are mutually exclusive.
457 .It Fl E Ar errfile
458 Direct error messages for the analyzed objects to
459 .Ar errfile
460 instead of stdout.
461 .It Fl e Ar exfile
462 .TP 4
463 .B \-D depfile
464 Use \fIdepfile\fP to generate an alternative dependency mapping.
465 \fIdepfile\fP must be created by '\fBfind_elf -r\fP'.
466 The \fB-D\fP and \fB-d\fP options are mutually exclusive.
467 .TP
468 .B \-d depdir
469 Use \fIdepdir\fP to generate an alternative dependency mapping.
470 \fBfind_elf(1ONBLD)\fP is used to locate the ELF sharable objects for
471 which alternative mappings are required. The \fB-D\fP and \fB-d\fP options
472 are mutually exclusive.
473 .TP 4
474 .B \-E errfile
475 Direct error messages for the analyzed objects to \fIerrfile\fP instead
476 of stdout.
477 .TP 4
478 .B \-e exfile
479 An exception file is used to exclude objects from
480 the usual rules.
481 See
482 .Sx EXCEPTION FILE FORMAT .
483 .It Fl f Ar listfile
484 the usual rules. See EXCEPTION FILE FORMAT.
485 .TP
486 .B \-f listfile
487 Normally,
488 .Ic interface_check
489 .I interface_check
490 runs
491 .Ic find_elf
492 to locate the ELF objects to analyze.
493 The
494 .Fl f
495 option can be used to instead provide a file containing the list of objects to
496 analyze, in the format produced by
497 .Ic find_elf -r .
498 .It Fl I Ar infofile
499 Direct informational messages (
500 .Fl i ,
501 and
502 .Fl v
503 options) for the analyzed objects to
504 .Ar infofile
505 instead of stdout.
506 .It Fl i

```

```

486 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
493 .I find_elf
494 to locate the ELF objects to analyze. The \fB-f\fP option can be
495 used to instead provide a file containing the list of objects to
496 analyze, in the format produced by '\fbfind_elf -r\fP'.
497 .TP
498 .B -I infofile
499 Direct informational messages (\fB-i\fP, and \fB-v\fP options) for the
500 analyzed objects to \fIinfile\fP instead of stdout.
501 .TP
502 .B \-i
503 Provide dynamic entry information. Presently only dependencies and
504 runpaths are printed.
505 .TP
506 .B \-m
507 Enable \fBmcs(1)\fP checking.
508 .TP
509 .B \-o
510 Produce a one-line output for each condition discovered, prefixed
511 by the objects name.
512 This output style is more terse, but is more appropriate for sorting and
513 diffing with previous build results.
514 .It Fl s
515 Determine whether
516 .Em .stabs
517 sections exist.
518 .It Fl v
519 Provide version definition information.
520 Each symbol defined by the object is printed along with the version it is
521 assigned to.
522 .It Fl w Ar outdir
523 Interpret the paths of all input and output files relative to
524 .Ar outdir .
525 .El
526 .Sh EXCEPTION FILE FORMAT
527 by the objects name. This output style is more terse, but is
528 more appropriate for sorting and diffing with previous build results.
529 .TP
530 .B \-s
531 Determine whether \fI.stabs\fP sections exist.
532 .TP
533 .B \-v
534 Provide version definition information. Each symbol defined by the object
535 is printed along with the version it is assigned to.
536 .TP
537 .B -w outdir
538 Interpret the paths of all input and output files relative to \fIoutdir\fP.
539 .SH EXCEPTION FILE FORMAT
540 Exceptions to the rules enforced by
541 .Nm check_rtime
542 are specified using an exception file.
543 The
544 .Fl -e
545 option is used to specify an explicit exception file.
546 Otherwise, if used in an activated workspace, the default exception file is
547 .Pa $CODEMGR_WS/exception_list/check_rtime
548 if that file exists.
549 If not used in an activated workspace, or if
550 .Pa $CODEMGR_WS/exception_list/check_rtime
551 does not exist,

```

```

522 .Nm check_rtime
523 .I check_rtime
524 are specified using an exception file. The \fB-e\fP option is used to
525 specify an explicit exception file. Otherwise, if used in an activated
526 workspace, the default exception file is
527 $CODEMGR_WS/exception_list/check_rtime
528 if that file exists. If not used in an activated workspace, or if
529 $CODEMGR_WS/exception_list/check_rtime does not exist,
530 .I check_rtime
531 will use
532 .Pa /opt/onbld/etc/exception_list/check_rtime
533 .I /opt/onbld/etc/exception_list/check_rtime
534 as a fallback default exception file.
535 .Pp
536 .P
537 To run
538 .Nm check_rtime
539 without applying exceptions, specify
540 .Fl e
541 with a value of
542 .Pa /dev/null .
543 .Pp
544 A
545 .Ql #
546 character at the beginning of a line, or at any point in
547 a line when preceded by whitespace, introduces a comment.
548 Empty lines, and lines containing only comments, are ignored by
549 .Nm check_rtime .
550 Exceptions are specified as space separated keyword, and
551 .Xr perl 1
552 .I check_rtime
553 without applying exceptions, specify \fB-e\fP with a value of /dev/null.
554 .P
555 A '#' character at the beginning of a line, or at any point in
556 a line when preceded by whitespace, introduces a comment. Empty lines,
557 and lines containing only comments, are ignored by
558 .I check_rtime.
559 Exceptions are specified as space separated keyword, and \fBperl(1)\fP
560 regular expression:
561 .Pp
562 .Dl keyword perl-regex
563 .Pp
564 .sp
565 .in +4
566 .nf
567 keyword perl-regex
568 .fi
569 .in -4
570 .sp
571 Since whitespace is used as a separator, the regular
572 expression cannot itself contain whitespace.
573 Use of the
574 .Ql \s
575 character class to represent whitespace within the regular expression is
576 recommended.
577 .Pp
578 expression cannot itself contain whitespace. Use of the \s character
579 class to represent whitespace within the regular expression is recommended.
580 Before the perl regular expression is used, constructs of the form
581 .Em MACH(dir)
582 are expanded into a regular expression that matches the directory given, as
583 well as any 64-bit architecture subdirectory that might be present
584 (i.e. amd64, sparcv9). For instance,
585 .Em MACH(lib)
586 will match any of the following:
587 .Bl -tag -width indent

```

```

561 .It Pa lib
562 .It Pa lib/amd64
563 .It Pa lib/sparcv9
564 .El
565 .Pp
566 MACH(dir) are expanded into a regular expression that matches the directory
567 given, as well as any 64-bit architecture subdirectory that
568 might be present (i.e. amd64, sparcv9). For instance, MACH(lib) will
569 match any of the following:
570 .sp
571 .in +4
572 .nf
573 lib
574 lib/amd64
575 lib/sparcv9
576 .fi
577 .in -4
578 .sp
579 The exceptions understood by
580 .Nm check_rtime
581 .I check_rtime
582 are:
583 .Bl -tag -width indent
584 .It EXEC_DATA
585 .sp
586 .ne 2
587 .na
588 \fBEXEC_DATA\fR
589 .ad
590 .RS 17n
591 .sp
592 Executables that are not required to have non-executable writable
593 data segments
594 .It EXEC_STACK
595 .RE
596
597 .sp
598 .ne 2
599 .na
600 \fBEXEC_STACK\fR
601 .ad
602 .RS 17n
603 .sp
604 Executables that are not required to have a non-executable stack
605 .It NOCRLEALT
606 .RE
607
608 .sp
609 .ne 2
610 .na
611 \fBNOCRLEALT\fR
612 .ad
613 .RS 17n
614 .sp
615 Objects that should be skipped when building the alternative dependency
616 mapping via the
617 .Fl d
618 option.
619 .It NODIRECT
620 mapping via the \fB-d\fP option.
621 .RE
622
623 .sp
624 .ne 2
625 .na
626 \fBNODIRECT\fR

```

```

514 .ad
515 .RS 17n
516 .sp
517 Directories and files that are allowed to have no direct bound symbols.
518 .It NOSYMSORT
519 .RE
520
521 .sp
522 .ne 2
523 .na
524 \fBNOSYMSORT\fR
525 .ad
526 .RS 17n
527 .sp
528 Files for which we skip checking of duplicate addresses in the
529 symbol sort sections.
530 .It OLDDEP
531 .RE
532
533 .sp
534 .ne 2
535 .na
536 \fBOLDDEP\fR
537 .ad
538 .RS 17n
539 .sp
540 Objects that used to contain system functionality that has since
541 migrated to libc.
542 We preserve these libraries as pure filters for backward compatibility but
543 nothing needs to link to them.
544 .It SKIP
545 Directories and/or individual objects to skip.
546 Note that SKIP should be a last resort, used only when one of the other
547 exceptions will not suffice.
548 .It STAB
549 migrated to libc. We preserve these libraries as pure filters for
550 backward compatibility but nothing needs to link to them.
551 .RE
552
553 .sp
554 .ne 2
555 .na
556 \fBSKIP\fR
557 .ad
558 .RS 17n
559 .sp
560 Directories and/or individual objects to skip. Note that SKIP should be
561 a last resort, used only when one of the other exceptions will not suffice.
562 .RE
563
564 .sp
565 .ne 2
566 .na
567 \fBSTAB\fR
568 .ad

```

```

569 .RS 17n
570 .sp
597 Objects for which we allow relocations to the text segment.
598 .It BUNDEF_OBJ
572 .RE

574 .sp
575 .ne 2
576 .na
577 \fBUNDEF_OBJ\fR
578 .ad
579 .RS 17n
580 .sp
599 Objects that are allowed to be unreferenced.
600 .It UNDEF_REF
582 .RE

584 .sp
585 .ne 2
586 .na
587 \fBUNDEF_REF\fR
588 .ad
589 .RS 17n
590 .sp
601 Objects that are allowed undefined references.
602 .It UNUSED_DEPS
592 .RE

594 .sp
595 .ne 2
596 .na
597 \fBUNUSED_DEPS\fR
598 .ad
599 .RS 17n
600 .sp
603 Objects that are allowed to have unused dependencies.
604 .It BUNUSED_OBJ
602 .RE

604 .sp
605 .ne 2
606 .na
607 \fBUNUSED_OBJ\fR
608 .ad
609 .RS 17n
610 .sp
605 Objects that are always allowed to be unused dependencies.
606 .It UNUSED_RPATH
612 .RE

614 .sp
615 .ne 2
616 .na
617 \fBUNUSED_RPATH\fR
618 .ad
619 .RS 17n
620 .sp
607 Objects that are allowed to have unused runpath directories.
608 .It FORBIDDEN
609 Specifies that dependencies on a given object are forbidden.
610 .It FORBIDDEN_DEP
611 Specifies that a given object is permitted a forbidden dependency.
612 .El
613 .Sh ALTERNATIVE DEPENDENCY MAPPING
614 .Nm check_rtime
615 was primarily designed to process a nightly builds

```

```

616 .Ev $ROOT
617 hierarchy.
618 It is often the case that objects within this hierarchy must bind to
619 dependencies within the same hierarchy to satisfy their requirements.
620 .Pp
622 .RE

624 .SH ALTERNATIVE DEPENDENCY MAPPING
625 .I check_rtime
626 was primarily designed to process a nightly builds \fB$ROOT\fP
627 hierarchy. It is often the case that objects within this hierarchy
628 must bind to dependencies within the same hierarchy to satisfy
629 their requirements.
630 .LP
621 To achieve this,
622 .Nm check_rtime
623 uses the shared objects specified with the
624 .Fl D
625 or
626 .Fl d
627 options.
628 If neither option is specified, and the
629 .Ev $CODEMGR_WS
630 and
631 .Ev $ROOT
632 environment variables are defined, the proto area for the workspace is
633 used.
634 The objects found are used to create runtime configuration files via
635 .Xr crle 1 ,
636 that establish the new shared objects as alternatives to their underlying
637 system location.
638 .Nm check_rtime
639 passes these configuration files as
640 .Ev LD_CONFIG
641 environment variable settings to
642 .Xr ldd 1
643 using its
644 .Fl -e
645 option.
646 .Pp
632 .I check_rtime
633 uses the shared objects specified with the \fB-D\fP or \fB-d\fP options.
634 If neither option is specified, and the \fB$CODEMGR_WS\fP and \fB$ROOT\fP
635 environment variables are defined, the proto area for the workspace
636 is used. The objects found are used
637 to create runtime configuration files via \fBcrle(1)\fP, that establish
638 the new shared objects as alternatives to their underlying system location.
639 .I check_rtime
640 passes these configuration files as \fBBLD_CONFIG\fP environment
641 variable settings to \fBldd(1)\fP using its \fB-e\fP option.
642 .LP
647 The effect of these configuration files is that the execution of an
648 object under
649 .Xr ldd 1
650 will bind to the dependencies defined as alternatives.
651 Simply put, an object inspected in the
652 .Pa proto
653 area will bind to its dependencies found in the
654 .Pa proto
655 area.
656 Dependencies that have no alternative mapping will continue to bind to the
657 underlying system.
658 .Sh ENVIRONMENT VARIABLES
659 When the
660 .Fl D
661 or

```

```

662 .Fl d
663 option isn't in use,
664 .Nm check_rtime
664 object under \fBldd(1)\fP will bind to the dependencies defined as
665 alternatives. Simply put, an object inspected in the \fIproto\fP
666 area will bind to its dependencies found in the \fIproto\fP area.
667 Dependencies that have no alternative mapping will continue to
668 bind to the underlying system.
669 .SH ENVIRONMENT VARIABLES
670 .LP
671 When the \fB-D\fP or \fB-d\fP option isn't in use,
672 .I check_rtime
673 uses the following environment variables to
674 establish an alternative dependency mapping:
675 .Bl -tag -width indent
676 .It Ev CODEMGR_WS
677 .LP
678 .B CODEMGR_WS
679 .RS 4
680 The root of your workspace, which is the directory
681 containing
682 .Pa .git .
683 Existence of this environment variable indicates that
684 .Ev $ROOT
685 should be investigated.
686 .It Ev ROOT
687 Root of the
688 .Pa proto
689 area of your workspace.
690 Any shared objects under this directory will be used to establish an
691 alternative dependency mapping.
692 .El
693 If
694 .Xr ldd 1
695 supports the
696 .Fl U
697 option, it will be used to determine any unreferenced dependencies.
698 Otherwise
699 .Xr ldd 1
700 uses the older
701 .Fl u
702 option which only detects unused references.
703 If the following environment variable exists, and indicates an earlier release
704 than \fB5.10\fP then
705 .Xr ldd 1
706 also falls back to using the
707 .Fl u
708 option.
709 .Bl -tag -width indent
710 .It Ev RELEASE
711 containing \fICodemgr_wsdata\fP. Existence of this environment variable
712 indicates that \fB$ROOT\fP should be investigated.
713 .RE
714 .LP
715 .B ROOT
716 .RS 4
717 Root of the \fIproto\fP area of your workspace. Any shared objects
718 under this directory will be used to establish an alternative dependency
719 mapping.
720 .RE
721 .sp
722 If \fBldd(1)\fP supports the \fB-U\fP option, it will be used to determine
723 any unreferenced dependencies. Otherwise \fBldd(1)\fP uses the older
724 \fB-u\fP option which only detects unused references. If the following
725 environment variable exists, and indicates an earlier release than \fB5.10\fP
726 then \fBldd(1)\fP also falls back to using the \fB-u\fP option.

```

```

675 .LP
676 .B RELEASE
677 .RS 4
678 The release version number of the environment being built.
679 .El
680 .Sh ERROR CONDITIONS
681 Inspection of an object with
682 .Xr ldd 1
683 assumes it is compatible with the machine on which
684 .Nm check_rtime
685 is being run.
686 Incompatible objects such as a 64-bit object encountered on a 32-bit system,
687 or an i386 object encountered on a sparc system, can not be fully inspected.
688 These objects are displayed as:
689 .Pp
690 .Dl foo: has wrong class or data encoding
691 .Sh FILES
692 .Bl -tag -width indent
693 .It Pa $CODEMGR_WS/exception_list/check_rtime
694 .It Pa /opt/onblld/etc/exception_list/check_rtime
695 .El
696 .Sh SEE ALSO
697 .Xr crle 1 ,
698 .Xr elfdump 1 ,
699 .Xr ld.so.1 1 ,
700 .Xr ldd 1 ,
701 .Xr mcs 1 ,
702 .Xr find_elf 1ONBLD
703 .RE
704 .Sh ERROR CONDITIONS
705 .LP
706 Inspection of an object with \fBldd(1)\fP assumes it is compatible
707 with the machine on which
708 .I check_rtime
709 is being run. Incompatible objects such as a 64-bit object encountered on
710 a 32-bit system, or an i386 object encountered on a sparc system,
711 can not be fully inspected. These objects are displayed as:
712 .sp
713 .RS 4
714 foo: has wrong class or data encoding
715 .RE
716 .Sh FILES
717 .LP
718 .RS 5
719 $CODEMGR_WS/exception_list/check_rtime
720 /opt/onblld/etc/exception_list/check_rtime
721 .Sh SEE ALSO
722 .B crle(1),
723 .B elfdump(1),
724 .B find_elf(1ONBLD),
725 .B ldd(1),
726 .B ld.so.1(1),
727 .B mcs(1).

```

new/usr/src/tools/scripts/check_rtime.pl

1

```
*****
35527 Tue Feb 20 01:00:53 2018
new/usr/src/tools/scripts/check_rtime.pl
XXXX check_rtime should be able to forbid libraries
XXXX check_rtime needs to learn libnsl is safe now
XXXX check_rtime could use MACH() more thoroughly
XXXX check_rtime exceptions could be tidied up.
*****
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 #
22 #
23 #
24 # Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
25 #
26 #
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.
54 #
55 #
56 # Define all global variables (required for strict)
57 use vars qw($Prog $Env $Ena64 $Tmpdir);
58 use vars qw($LddNoU $Conf32 $Conf64);
```

new/usr/src/tools/scripts/check_rtime.pl

2

```
59 use vars qw(%opt);
60 use vars qw($ErrFH $ErrTtl $InfoFH $InfoTtl $OutCnt1 $OutCnt2);
61 #
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 functionality
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     }
212     return;

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, $Line);
389     last;
390 }
391
392 # It's possible this binary can't be executed, ie. we've
393 # found a sparc binary while running on an intel system,
394 # or a sparcv9 binary on a sparcv7/8 system.
395 if ($Line =~ /wrong class/) {
396     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
397         "has wrong class or data encoding");
398     next;
399 }
400
401 # Historically, ldd(1) likes executable objects to have
402 # their execute bit set.
403 if ($Line =~ /not executable/) {
404     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
405         "is not executable");
406     next;
407 }
408 }
409
410 # Look for "file" or "versions" that aren't found. Note that
411 # these lines will occur before we find any symbol referencing
412 # errors.
413 if (($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 =~ $EXRE_unused_rpath);
445
446     if ($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]*(.*)\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 =~ $EXRE_unused_deps);
476
477     # Skip if dependency is always allowed to be unused
478     next if defined($EXRE_unused_obj) &&
479         ($Line =~ $EXRE_unused_obj);
480
481     $Line =~ s!$Tmpdir!! if $Secure;
482     $Line =~ s/^[ \t]*(.*)\t$1\t<remove lib or -zignore?>/;
483     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath, $Line);
484     next;
485 }
486 }
487
488 # Reuse the elfdump(1) data to investigate additional dynamic linking
489 # information.
490
491 $Sun = $Relsz = $Pltsz = $Dyn = $Stab = $SymSort = 0;
492 $Tex = $Strip = 1;
493 $HasDirectBinding = 0;
494
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         $Header = 'Shdr';
501
502         if (($Sun == 0) && ($Line =~ /\.SUNW_reloc/)) {
503             # This object has a combined relocation section.
504             $Sun = 1;
505
506         } elsif (($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             $SymSort = 1;
513         }
514
515         if (($Strip == 1) && ($Line =~ /\.symtab/)) {
516             # This object contains a complete symbol table.
517             $Strip = 0;
518         }
519     }
520     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') && ($Header ne 'Syminfo')) {
528         next;
529     }
531
532 # Look into the Syminfo section.
533 # Does this object have at least one Directly Bound symbol?
534 if (($Header eq 'Syminfo')) {
535     my(@Symword);
536
537     if ($HasDirectBinding == 1) {
538         next;
539     }
541
542     @Symword = split(' ', $Line);
543
544     if (!defined($Symword[1])) {
545         next;
546     }
547     if ($Symword[1] =~ /B/) {
548         $HasDirectBinding = 1;
549     }
551
552 # Does this object contain text relocations.
553 if ($Tex && ($Line =~ /TEXTREL/)) {
554     # Determine if this file is allowed text relocations.
555     if (defined($EXRE_textrel) &&
556         ($RelPath =~ $EXRE_textrel)) {
557         $Tex = 0;
558         next ELF;
559     }
560     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
561         "TEXTREL .dynamic tag\t\t\t\t<no -Kpic?>");
562     $Tex = 0;
563     next;
564 }
565
566 # Does this file have any relocation sections (there are a few
567 # psr libraries with no relocations at all, thus a .SUNW_reloc
568 # section won't exist either).
569 if (($RelSZ == 0) && ($Line =~ /RELA?SZ/)) {
570     $RelSZ = hex((split(' ', $Line))[2]);
571     next;
572 }
573
574 # Does this file have any plt relocations. If the plt size is
575 # equivalent to the total relocation size then we don't have
576 # any relocations suitable for combining into a .SUNW_reloc
577 # section.
578 if (($PltSZ == 0) && ($Line =~ /PLTRELSZ/)) {
579     $PltSZ = hex((split(' ', $Line))[2]);
580     next;
581 }
582
583 # Does this object have any dependencies.
584 if ($Line =~ /NEEDED/) {
585     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 =~ $EXRE_forbidden_dep);
595
596             onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
597                 "NEEDED=$Need\t<forbidden dependency, " .
598                 "missing -nodefaultlibs?>");
599             next;
600             "NEEDED=$Need\t<dependency no longer necessary>");
601         } elsif ($opt{i}) {
602             # Under the -i (information) option print out
603             # any useful dynamic entries.
604             onbld_elfmod::OutMsg($InfoFH, $InfoTtl, $RelPath,
605                 "NEEDED=$Need");
606         }
607         next;
609
610 # Is this object built with -B direct flag on?
611 if ($Line =~ / DIRECT /) {
612     $HasDirectBinding = 1;
613 }
614
615 # Does this object specify a runpath.
616 if ($opt{i} && ($Line =~ /RPATH/)) {
617     my($Rpath) = (split(' ', $Line))[3];
618     onbld_elfmod::OutMsg($InfoFH, $InfoTtl,
619         $RelPath, "RPATH=$Rpath");
620     next;
621 }
622
623 # A shared object, that contains non-plt relocations, should have a
624 # combined relocation section indicating it was built with -z combrelloc.
625 if (($Type eq 'DYN') && $RelSZ && ($RelSZ != $PltSZ) && ($Sun == 0)) {
626     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
627         ".SUNW_reloc section missing\t\t\t\t<no -zcombrelloc?>");
628 }
629
630 # No objects released to a customer should have any .stabs sections
631 # remaining, they should be stripped.
632 if ($opt{s} && $Stab) {
633     goto DONEYTAB if defined($EXRE_stab) && ($RelPath =~ $EXRE_stab)
634
635     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
636         "debugging sections should be deleted\t\t\t\t<no strip -x?>");
637 }
638
639 # Identify an object that is not built with either -B direct or
640 # -z direct.
641 goto DONEYTAB
642     if (defined($EXRE_nodirect) && ($RelPath =~ $EXRE_nodirect));
643
644 if ($RelSZ && ($HasDirectBinding == 0)) {
645     onbld_elfmod::OutMsg($ErrFH, $ErrTtl, $RelPath,
646         "object has no direct bindings\t\t\t\t<no -B direct or -z direct?>");
647 }
648
649 DONEYTAB:

```

```
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 $Strip;
655
656     # If there are symbol sort sections in this object, report on
657     # any that have duplicate addresses.
658     ProcSymSort($FullPath, $RelPath) if $SymSort;
659
660     # If -v was specified, and the object has a version definition
661     # section, generate output showing each public symbol and the
662     # version it belongs to.
663     ProcVerdef($FullPath, $RelPath)
664       if ($Verdef eq 'VERDEF') && $opt{v};
665 }
```

unchanged portion omitted