

```

*****
2509 Mon Jul 21 17:22:58 2014
new/usr/src/lib/libnvpair/Makefile.com
5005 libnvpair JSON output broken by lint fixes
5006 libnvpair JSON cannot print int16 arrays
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
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 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # Copyright (c) 2012 by Delphix. All rights reserved.
26 # Copyright (c) 2014, Joyent, Inc. All rights reserved.
26 # Copyright (c) 2013, Joyent, Inc. All rights reserved.
27 #

29 LIBRARY=      libnvpair.a
30 VERS=        .1

32 OBJECTS=     libnvpair.o \
33              nvpair_alloc_system.o \
34              nvpair_alloc_fixed.o \
35              nvpair.o \
36              fnvpair.o \
37              nvpair_json.o

39 include ../../Makefile.lib
40 include ../../Makefile.rootfs

42 SRCS=        ../libnvpair.c \
43              ../nvpair_alloc_system.c \
44              ../nvpair_json.c \
45              $(SRC)/common/nvpair/nvpair_alloc_fixed.c \
46              $(SRC)/common/nvpair/nvpair.c \
47              $(SRC)/common/nvpair/fnvpair.c

49 #
50 # Libraries added to the next line must be present in miniroot
51 #
52 LDLIBS +=    -lc -lnsl
53 LIBS =       $(DYNLIB) $(LINTLIB)

55 # turn off ptr-cast warnings
56 LINTFLAGS64 += -erroff=E_BAD_PTR_CAST_ALIGN

58 # turn off warning caused by lint bug: not understanding SCN18 "hhi"

```

```

59 LINTFLAGS += -erroff=E_BAD_FORMAT_STR2
60 LINTFLAGS += -erroff=E_INVALID_TOKEN_IN_DEFINE_MACRO
61 LINTFLAGS += -erroff=E_RET_INT_IMPLICITLY
62 LINTFLAGS += -erroff=E_FUNC_USED_VAR_ARG2
63 LINTFLAGS += -erroff=E_CONSTANT_CONDITION
64 LINTFLAGS64 += -erroff=E_BAD_FORMAT_STR2
65 LINTFLAGS64 += -erroff=E_INVALID_TOKEN_IN_DEFINE_MACRO
66 LINTFLAGS64 += -erroff=E_RET_INT_IMPLICITLY
67 LINTFLAGS64 += -erroff=E_FUNC_USED_VAR_ARG2
68 LINTFLAGS64 += -erroff=E_CONSTANT_CONDITION

70 CERRWARN += -_gcc=-Wno-type-limits
71 CERRWARN += -_gcc=-Wno-parentheses
72 CERRWARN += -_gcc=-Wno-uninitialized

74 CFLAGS += $(CVERBOSE)
75 CPPFLAGS += -D_REENTRANT

77 C99MODE= -xc99=%all
78 C99LMODE= -Xc99=%all

80 $(LINTLIB) := SRCS = $(SRCDIR)/$(LINTSRC)

82 .KEEP_STATE:

84 all: $(LIBS)

86 lint: lintcheck

88 include ../../Makefile.targ

90 pics/%.o: $(SRC)/common/nvpair/%.c
91           $(COMPILE.c) -o $@ $<
92           $(POST_PROCESS_O)

```

```

*****
8657 Mon Jul 21 17:22:58 2014
new/usr/src/lib/libnvpair/nvpair_json.c
5005 libnvpair JSON output broken by lint fixes
5006 libnvpair JSON cannot print int16 arrays
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */
11 */
12 * Copyright (c) 2014, Joyent, Inc. All rights reserved.
13 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
14 */

15 #include <stdio.h>
16 #include <stdlib.h>
17 #include <strings.h>
18 #include <wchar.h>
19 #include <sys/debug.h>

21 #include "libnvpair.h"

23 #define FPRINTF(fp, ...) \
24     do { \
25         if (fprintf(fp, __VA_ARGS__) < 0) \
26             return (-1); \
27     } while (0)
25     return (-1) \

29 /*
30  * When formatting a string for JSON output we must escape certain characters,
31  * as described in RFC4627. This applies to both member names and
32  * DATA_TYPE_STRING values.
33  *
34  * This function will only operate correctly if the following conditions are
35  * met:
36  *
37  *     1. The input String is encoded in the current locale.
38  *
39  *     2. The current locale includes the Basic Multilingual Plane (plane 0)
40  *        as defined in the Unicode standard.
41  *
42  * The output will be entirely 7-bit ASCII (as a subset of UTF-8) with all
43  * representable Unicode characters included in their escaped numeric form.
44  */
45 static int
46 nvlist_print_json_string(FILE *fp, const char *input)
47 {
48     mbstate_t mbr;
49     wchar_t c;
50     size_t sz;

52     bzero(&mbr, sizeof (mbr));

54     FPRINTF(fp, "\\");
55     while ((sz = mbrtowc(&c, input, MB_CUR_MAX, &mbr)) > 0) {
56         switch (c) {
57

```

```

58         FPRINTF(fp, "\\");
59         break;
60     case '\\n':
61         FPRINTF(fp, "\\n");
62         break;
63     case '\\r':
64         FPRINTF(fp, "\\r");
65         break;
66     case '\\':
67         FPRINTF(fp, "\\");
68         break;
69     case '\\f':
70         FPRINTF(fp, "\\f");
71         break;
72     case '\\t':
73         FPRINTF(fp, "\\t");
74         break;
75     case '\\b':
76         FPRINTF(fp, "\\b");
77         break;
78     default:
79         if ((c >= 0x00 && c <= 0x1f) ||
80             (c > 0x7f && c <= 0xffff)) {
81             /*
82              * Render both Control Characters and Unicode
83              * characters in the Basic Multilingual Plane
84              * as JSON-escaped multibyte characters.
85              */
86             FPRINTF(fp, "\\u%04x", (int)(0xffff & c));
87         } else if (c >= 0x20 && c <= 0x7f) {
88             /*
89              * Render other 7-bit ASCII characters directly
90              * and drop other, unrepresentable characters.
91              */
92             FPRINTF(fp, "%c", (int)(0xff & c));
93         }
94         break;
95     }
96     input += sz;
97 }

99     if (sz == (size_t)-1 || sz == (size_t)-2) {
100         /*
101          * We last read an invalid multibyte character sequence,
102          * so return an error.
103          */
104         return (-1);
105     }

107     FPRINTF(fp, "");
108     return (0);
109 }

111 /*
112  * Dump a JSON-formatted representation of an nvlist to the provided FILE *.
113  * This routine does not output any new-lines or additional whitespace other
114  * than that contained in strings, nor does it call fflush(3C).
115  */
116 int
117 nvlist_print_json(FILE *fp, nvlist_t *nvl)
118 {
119     nvpair_t *curr;
120     boolean_t first = B_TRUE;

122     FPRINTF(fp, "{");

```

```

124     for (curr = nvlist_next_nvpair(nvl, NULL); curr;
125         curr = nvlist_next_nvpair(nvl, curr)) {
126         data_type_t type = nvpair_type(curr);

128         if (!first)
129             FPRINTF(fp, ",");
130         else
131             first = B_FALSE;

133         if (nvlist_print_json_string(fp, nvpair_name(curr)) == -1)
134             return (-1);
135         FPRINTF(fp, ":");

137         switch (type) {
138         case DATA_TYPE_STRING: {
139             char *string = fnvpair_value_string(curr);
140             if (nvlist_print_json_string(fp, string) == -1)
141                 return (-1);
142             break;
143         }

145         case DATA_TYPE_BOOLEAN: {
146             FPRINTF(fp, "true");
147             break;
148         }

150         case DATA_TYPE_BOOLEAN_VALUE: {
151             FPRINTF(fp, "%s", fnvpair_value_boolean_value(curr) ==
152                 B_TRUE ? "true" : "false");
153             break;
154         }

156         case DATA_TYPE_BYTE: {
157             FPRINTF(fp, "%hhu", fnvpair_value_byte(curr));
158             break;
159         }

161         case DATA_TYPE_INT8: {
162             FPRINTF(fp, "%hhd", fnvpair_value_int8(curr));
163             break;
164         }

166         case DATA_TYPE_UINT8: {
167             FPRINTF(fp, "%hhu", fnvpair_value_uint8_t(curr));
168             break;
169         }

171         case DATA_TYPE_INT16: {
172             FPRINTF(fp, "%hd", fnvpair_value_int16(curr));
173             break;
174         }

176         case DATA_TYPE_UINT16: {
177             FPRINTF(fp, "%hu", fnvpair_value_uint16(curr));
178             break;
179         }

181         case DATA_TYPE_INT32: {
182             FPRINTF(fp, "%d", fnvpair_value_int32(curr));
183             break;
184         }

186         case DATA_TYPE_UINT32: {
187             FPRINTF(fp, "%u", fnvpair_value_uint32(curr));
188             break;
189         }

```

```

191     case DATA_TYPE_INT64: {
192         FPRINTF(fp, "%lld",
193             (long long)fnvpair_value_int64(curr));
194         break;
195     }

197     case DATA_TYPE_UINT64: {
198         FPRINTF(fp, "%llu",
199             (unsigned long long)fnvpair_value_uint64(curr));
200         break;
201     }

203     case DATA_TYPE_HRTIME: {
204         hrtime_t val;
205         VERIFY0(nvpair_value_hrtime(curr, &val));
206         FPRINTF(fp, "%llu", (unsigned long long)val);
207         break;
208     }

210     case DATA_TYPE_DOUBLE: {
211         double val;
212         VERIFY0(nvpair_value_double(curr, &val));
213         FPRINTF(fp, "%f", val);
214         break;
215     }

217     case DATA_TYPE_NVLIST: {
218         if (nvlist_print_json(fp,
219             fnvpair_value_nvlist(curr)) == -1)
220             return (-1);
221         break;
222     }

224     case DATA_TYPE_STRING_ARRAY: {
225         char **val;
226         uint_t valsz, i;
227         VERIFY0(nvpair_value_string_array(curr, &val, &valsz));
228         FPRINTF(fp, "[");
229         for (i = 0; i < valsz; i++) {
230             if (i > 0)
231                 FPRINTF(fp, ",");
232             if (nvlist_print_json_string(fp, val[i]) == -1)
233                 return (-1);
234         }
235         FPRINTF(fp, "];");
236         break;
237     }

239     case DATA_TYPE_NVLIST_ARRAY: {
240         nvlist_t **val;
241         uint_t valsz, i;
242         VERIFY0(nvpair_value_nvlist_array(curr, &val, &valsz));
243         FPRINTF(fp, "[");
244         for (i = 0; i < valsz; i++) {
245             if (i > 0)
246                 FPRINTF(fp, ",");
247             if (nvlist_print_json(fp, val[i]) == -1)
248                 return (-1);
249         }
250         FPRINTF(fp, "];");
251         break;
252     }

254     case DATA_TYPE_BOOLEAN_ARRAY: {
255         boolean_t *val;

```

```

256     uint_t valsz, i;
257     VERIFY0(nvpair_value_boolean_array(curr, &val, &valsz));
258     FPRINTF(fp, "[");
259     for (i = 0; i < valsz; i++) {
260         if (i > 0)
261             FPRINTF(fp, ",");
262         FPRINTF(fp, val[i] == B_TRUE ?
263             "true" : "false");
264     }
265     FPRINTF(fp, "];");
266     break;
267 }

269 case DATA_TYPE_BYTE_ARRAY: {
270     uchar_t *val;
271     uint_t valsz, i;
272     VERIFY0(nvpair_value_byte_array(curr, &val, &valsz));
273     FPRINTF(fp, "[");
274     for (i = 0; i < valsz; i++) {
275         if (i > 0)
276             FPRINTF(fp, ",");
277         FPRINTF(fp, "%hh", val[i]);
278     }
279     FPRINTF(fp, "];");
280     break;
281 }

283 case DATA_TYPE_UINT8_ARRAY: {
284     uint8_t *val;
285     uint_t valsz, i;
286     VERIFY0(nvpair_value_uint8_array(curr, &val, &valsz));
287     FPRINTF(fp, "[");
288     for (i = 0; i < valsz; i++) {
289         if (i > 0)
290             FPRINTF(fp, ",");
291         FPRINTF(fp, "%hu", val[i]);
292     }
293     FPRINTF(fp, "];");
294     break;
295 }

297 case DATA_TYPE_INT8_ARRAY: {
298     int8_t *val;
299     uint_t valsz, i;
300     VERIFY0(nvpair_value_int8_array(curr, &val, &valsz));
301     FPRINTF(fp, "[");
302     for (i = 0; i < valsz; i++) {
303         if (i > 0)
304             FPRINTF(fp, ",");
305         FPRINTF(fp, "%hd", val[i]);
306     }
307     FPRINTF(fp, "];");
308     break;
309 }

311 case DATA_TYPE_UINT16_ARRAY: {
312     uint16_t *val;
313     uint_t valsz, i;
314     VERIFY0(nvpair_value_uint16_array(curr, &val, &valsz));
315     FPRINTF(fp, "[");
316     for (i = 0; i < valsz; i++) {
317         if (i > 0)
318             FPRINTF(fp, ",");
319         FPRINTF(fp, "%hu", val[i]);
320     }
321     FPRINTF(fp, "];");

```

```

322         break;
323     }

325 case DATA_TYPE_INT16_ARRAY: {
326     int16_t *val;
327     uint_t valsz, i;
328     VERIFY0(nvpair_value_int16_array(curr, &val, &valsz));
329     FPRINTF(fp, "[");
330     for (i = 0; i < valsz; i++) {
331         if (i > 0)
332             FPRINTF(fp, ",");
333         FPRINTF(fp, "%hd", val[i]);
334         FPRINTF(fp, "%hh", val[i]);
335     }
336     FPRINTF(fp, "];");
337     break;
338 }

339 case DATA_TYPE_UINT32_ARRAY: {
340     uint32_t *val;
341     uint_t valsz, i;
342     VERIFY0(nvpair_value_uint32_array(curr, &val, &valsz));
343     FPRINTF(fp, "[");
344     for (i = 0; i < valsz; i++) {
345         if (i > 0)
346             FPRINTF(fp, ",");
347         FPRINTF(fp, "%u", val[i]);
348     }
349     FPRINTF(fp, "];");
350     break;
351 }

353 case DATA_TYPE_INT32_ARRAY: {
354     int32_t *val;
355     uint_t valsz, i;
356     VERIFY0(nvpair_value_int32_array(curr, &val, &valsz));
357     FPRINTF(fp, "[");
358     for (i = 0; i < valsz; i++) {
359         if (i > 0)
360             FPRINTF(fp, ",");
361         FPRINTF(fp, "%d", val[i]);
362     }
363     FPRINTF(fp, "];");
364     break;
365 }

367 case DATA_TYPE_UINT64_ARRAY: {
368     uint64_t *val;
369     uint_t valsz, i;
370     VERIFY0(nvpair_value_uint64_array(curr, &val, &valsz));
371     FPRINTF(fp, "[");
372     for (i = 0; i < valsz; i++) {
373         if (i > 0)
374             FPRINTF(fp, ",");
375         FPRINTF(fp, "%llu",
376             (unsigned long long)val[i]);
377     }
378     FPRINTF(fp, "];");
379     break;
380 }

382 case DATA_TYPE_INT64_ARRAY: {
383     int64_t *val;
384     uint_t valsz, i;
385     VERIFY0(nvpair_value_int64_array(curr, &val, &valsz));
386     FPRINTF(fp, "[");

```

```
387         for (i = 0; i < valsz; i++) {
388             if (i > 0)
389                 FPRINTF(fp, ",");
390             FPRINTF(fp, "%lld", (long long)val[i]);
391         }
392         FPRINTF(fp, "];");
393         break;
394     }
395
396     case DATA_TYPE_UNKNOWN:
397         return (-1);
398     }
399 }
400
401     FPRINTF(fp, "];");
402     return (0);
403 }
unchanged_portion_omitted
```

new/usr/src/pkg/manifests/system-test-utiltest.mf

1

2094 Mon Jul 21 17:22:59 2014

new/usr/src/pkg/manifests/system-test-utiltest.mf

5005 libnvpair JSON output broken by lint fixes

5006 libnvpair JSON cannot print int16 arrays

Reviewed by: Robert Mustacchi <rm@joyent.com>

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License ("CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #
11 #
12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 # Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
15 #
16 #
17 set name=pkg.fmri value=pkg:/system/test/utiltest@$(PKGVERS)
18 set name=pkg.description value="Miscellaneous Utility Unit Tests"
19 set name=pkg.summary value="Utility Unit Test Suite"
20 set name=info.classification \
21     value=org.opensolaris.category.2008:Development/System
22 set name=variant.arch value=$(ARCH)
23 dir path=opt/util-tests
24 dir path=opt/util-tests/bin
25 dir path=opt/util-tests/runfiles
26 dir path=opt/util-tests/tests
27 dir path=opt/util-tests/tests/libnvpair_json
28 file path=opt/util-tests/README mode=0444
29 file path=opt/util-tests/bin/print_json mode=0555
30 file path=opt/util-tests/bin/utiltest mode=0555
31 file path=opt/util-tests/runfiles/default.run mode=0444
32 file path=opt/util-tests/tests/allowed-ips mode=0555
33 file path=opt/util-tests/tests/libnvpair_json/json_00_blank mode=0555
34 file path=opt/util-tests/tests/libnvpair_json/json_01_boolean mode=0555
35 file path=opt/util-tests/tests/libnvpair_json/json_02_numbers mode=0555
36 file path=opt/util-tests/tests/libnvpair_json/json_03_empty_arrays mode=0555
37 file path=opt/util-tests/tests/libnvpair_json/json_04_number_arrays mode=0555
38 file path=opt/util-tests/tests/libnvpair_json/json_05_strings mode=0555
39 file path=opt/util-tests/tests/libnvpair_json/json_06_nested mode=0555
40 file path=opt/util-tests/tests/libnvpair_json/json_07_nested_arrays mode=0555
41 file path=opt/util-tests/tests/libnvpair_json/json_common mode=0555
42 file path=opt/util-tests/tests/print_test mode=0555
43 file path=opt/util-tests/tests/xargs_test mode=0555
44 license lic_CDDL license=lic_CDDL
45 depend fmri=system/test/testrunner type=require
```

new/usr/src/test/util-tests/runfiles/default.run

1

948 Mon Jul 21 17:22:59 2014

new/usr/src/test/util-tests/runfiles/default.run

5005 libnvpair JSON output broken by lint fixes

5006 libnvpair JSON cannot print int16 arrays

Reviewed by: Robert Mustacchi <rm@joyent.com>

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License ("CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #
11 #
12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
15 #
```

```
17 [DEFAULT]
18 pre =
19 verbose = False
20 quiet = False
21 timeout = 60
22 post =
23 outputdir = /var/tmp/test_results
```

```
25 [/opt/util-tests/tests/printf_test]
26 [/opt/util-tests/tests/allowed-ips]
```

```
28 [/opt/util-tests/tests/xargs_test]
```

```
30 [/opt/util-tests/tests/libnvpair_json]
31 tests = ['json_00_blank', 'json_01_boolean', 'json_02_numbers',
32         'json_03_empty_arrays', 'json_04_number_arrays', 'json_05_strings',
33         'json_06_nested', 'json_07_nested_arrays']
```

new/usr/src/test/util-tests/tests/Makefile

1

623 Mon Jul 21 17:22:59 2014

new/usr/src/test/util-tests/tests/Makefile

5005 libnvpair JSON output broken by lint fixes

5006 libnvpair JSON cannot print int16 arrays

Reviewed by: Robert Mustacchi <rm@joyent.com>

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License ("CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #
```

```
12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
15 #
```

```
17 SUBDIRS = dladm printf xargs
18 SUBDIRS = dladm libnvpair_json printf xargs
```

```
20 include $(SRC)/test/Makefile.com
```


new/usr/src/test/util-tests/tests/libnvpair_json/Makefile

1

1406 Mon Jul 21 17:22:59 2014

new/usr/src/test/util-tests/tests/libnvpair_json/Makefile

5005 libnvpair JSON output broken by lint fixes

5006 libnvpair JSON cannot print int16 arrays

Reviewed by: Robert Mustacchi <rm@joyent.com>

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License ("CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #
```

```
12 #
13 # Copyright (c) 2014 Joyent, Inc. All rights reserved.
14 #
```

```
16 include $(SRC)/Makefile.master
```

```
18 ROOTOPTPKG = $(ROOT)/opt/util-tests
19 TESTDIR = $(ROOTOPTPKG)/tests/libnvpair_json
20 ROOTBINDIR = $(ROOTOPTPKG)/bin
```

```
22 PROG = print_json
```

```
24 SCRIPTS = \
25     json_00_blank \
26     json_01_boolean \
27     json_02_numbers \
28     json_03_empty_arrays \
29     json_04_number_arrays \
30     json_05_strings \
31     json_06_nested \
32     json_07_nested_arrays \
33     json_common
```

```
35 include $(SRC)/cmd/Makefile.cmd
36 include $(SRC)/test/Makefile.com
```

```
38 OBJS = $(PROG:%=%.o)
39 SRCS = $(OBJS:%.o=%.c)
```

```
41 CMDS = $(PROG:%=$(ROOTBINDIR)/%) $(SCRIPTS:%=$(TESTDIR)/%)
42 $(CMDS) := FILEMODE = 0555
```

```
44 LDLIBS += -lnvpair
```

```
46 LINTFLAGS += -erroff=E_FUNC_ARG_UNUSED
```

```
48 all: $(PROG)
```

```
50 $(PROG): $(OBJS)
51     $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
52     $(POST_PROCESS)
```

```
54 install: all $(CMDS)
```

```
56 lint: lint_SRCS
```

```
58 clobber: clean
59     -$(RM) $(PROG)
```

new/usr/src/test/util-tests/tests/libnvpair_json/Makefile

2

```
61 clean:
62     -$(RM) $(OBJS)

64 $(CMDS): $(TESTDIR) $(PROG)

66 $(ROOTBINDIR):
67     $(INS.dir)

69 $(ROOTBINDIR)/%: %
70     $(INS.file)

72 $(TESTDIR):
73     $(INS.dir)

75 $(TESTDIR)/%: %.ksh
76     $(INS.rename)
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_00_blank.ksh

1

631 Mon Jul 21 17:22:59 2014

new/usr/src/test/util-tests/tests/libnvpair_json/json_00_blank.ksh

5005 libnvpair JSON output broken by lint fixes

5006 libnvpair JSON cannot print int16 arrays

Reviewed by: Robert Mustacchi <rm@joyent.com>

```
1 #!/bin/ksh
2 #
3 # This file and its contents are supplied under the terms of the
4 # Common Development and Distribution License ("CDDL"), version 1.0.
5 # You may only use this file in accordance with the terms of version
6 # 1.0 of the CDDL.
7 #
8 # A full copy of the text of the CDDL should have accompanied this
9 # source. A copy of the CDDL is also available via the Internet at
10 # http://www.illumos.org/license/CDDL.
11 #
12 #
13 #
14 # Copyright (c) 2014, Joyent, Inc.
15 #
16 #
17 DIR=$(dirname $(whence $0))
18 . ${DIR}/json_common
19 #
20 BASELINE="$(cat <<EOF
21 {\
22 }
23 EOF)"
24 #
25 OUTPUT="$((${DIR}/../bin/print_json <<'EOF'
26 /*
27 * Emit a blank object.
28 */
29 EOF)"
30 #
31 complete
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_01_boolean.ksh

1

966 Mon Jul 21 17:22:59 2014

new/usr/src/test/util-tests/tests/libnvpair_json/json_01_boolean.ksh

5005 libnvpair JSON output broken by lint fixes

5006 libnvpair JSON cannot print int16 arrays

Reviewed by: Robert Mustacchi <rm@joyent.com>

```
1 #!/bin/ksh
2 #
3 # This file and its contents are supplied under the terms of the
4 # Common Development and Distribution License ("CDDL"), version 1.0.
5 # You may only use this file in accordance with the terms of version
6 # 1.0 of the CDDL.
7 #
8 # A full copy of the text of the CDDL should have accompanied this
9 # source. A copy of the CDDL is also available via the Internet at
10 # http://www.illumos.org/license/CDDL.
11 #
12 #
13 #
14 # Copyright (c) 2014, Joyent, Inc.
15 #
16 #
17 DIR=$(dirname $(whence $0))
18 . ${DIR}/json_common
19 #
20 BASELINE="$(cat <<EOF
21 {\
22 "bool0":true,\
23 "a fact":true,\
24 "a fiction":false,\
25 "1":true,\
26 " ":true\
27 }
28 EOF)"
29 #
30 OUTPUT="$((${DIR}/../bin/print_json <<'EOF'
31 /*
32 * add_boolean calls nvlist_add_boolean(), which the JSON formatter
33 * will emit as a true-valued boolean.
34 */
35 add_boolean "bool0";
36 add_boolean_value "a fact" "true";
37 add_boolean_value "a fiction" "false";
38 add_boolean "1";
39 #
40 /*
41 * Test a key with a whitespace-only name:
42 */
43 add_boolean " ";
44 EOF)"
45 #
46 complete
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_02_numbers.ksh 1

```
*****
1411 Mon Jul 21 17:22:59 2014
new/usr/src/test/util-tests/tests/libnvpair_json/json_02_numbers.ksh
5005 libnvpair JSON output broken by lint fixes
5006 libnvpair JSON cannot print int16 arrays
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
1 #!/bin/ksh
2 #
3 # This file and its contents are supplied under the terms of the
4 # Common Development and Distribution License ("CDDL"), version 1.0.
5 # You may only use this file in accordance with the terms of version
6 # 1.0 of the CDDL.
7 #
8 # A full copy of the text of the CDDL should have accompanied this
9 # source. A copy of the CDDL is also available via the Internet at
10 # http://www.illumos.org/license/CDDL.
11 #
12 #
13 #
14 # Copyright (c) 2014, Joyent, Inc.
15 #
16 #
17 DIR=$(dirname $(whence $0))
18 . ${DIR}/json_common
19 #
20 BASELINE="$(cat <<EOF
21 {\
22 "byte":255,\
23 "uint8_0":0,\
24 "uint8_100":100,\
25 "uint8_255":255,\
26 "uint16":12345,\
27 "uint32":23423423,\
28 "uint64":19850709000000,\
29 "int16_small":-32768,\
30 "int8_neg":-128,\
31 "int8_pos":127,\
32 "int16_big":32767,\
33 "int32":-1270000,\
34 "int64":-12700000000001,\
35 "double_small":0.000023,\
36 "double_big":2342300000000.000000\
37 }
38 EOF)"
39 #
40 OUTPUT="$( ${DIR}/../bin/print_json <<'EOF'
41 add_byte "byte" "0";
42 add_byte "byte" "255";
43 #
44 add_uint8 "uint8_0" "0";
45 add_uint8 "uint8_100" "100";
46 add_uint8 "uint8_255" "255";
47 #
48 add_uint16 "uint16" "12345";
49 add_uint32 "uint32" "23423423";
50 add_uint64 "uint64" "19850709000000";
51 #
52 add_int16 "int16_small" "-32768";
53 add_int8 "int8_neg" "-128";
54 add_int8 "int8_pos" "127";
55 add_int16 "int16_big" "32767";
56 #
57 add_int32 "int32" "-1270000";
58 add_int64 "int64" "-12700000000001";
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_02_numbers.ksh 2

```
60 add_double "double_small" "0.000023423";
61 add_double "double_big" "0.000023423e17";
62 EOF)"
63 #
64 complete
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_03_empty_arrays.ksh 1

```
*****
1419 Mon Jul 21 17:22:59 2014
new/usr/src/test/util-tests/tests/libnvpair_json/json_03_empty_arrays.ksh
5005 libnvpair JSON output broken by lint fixes
5006 libnvpair JSON cannot print int16 arrays
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
1 #!/bin/ksh
2 #
3 # This file and its contents are supplied under the terms of the
4 # Common Development and Distribution License ("CDDL"), version 1.0.
5 # You may only use this file in accordance with the terms of version
6 # 1.0 of the CDDL.
7 #
8 # A full copy of the text of the CDDL should have accompanied this
9 # source. A copy of the CDDL is also available via the Internet at
10 # http://www.illumos.org/license/CDDL.
11 #
12 #
13 #
14 # Copyright (c) 2014, Joyent, Inc.
15 #
16 #
17 DIR=$(dirname $(whence $0))
18 . ${DIR}/json_common
19 #
20 BASELINE="$(cat <<EOF
21 {\
22 "boolean_array":[],\
23 "byte_array":[],\
24 "uint8_array":[],\
25 "uint16_array":[],\
26 "uint32_array":[],\
27 "uint64_array":[],\
28 "int8_array":[],\
29 "int16_array":[],\
30 "int32_array":[],\
31 "int64_array":[],\
32 "string_array":[],\
33 "object_array":[{}]\
34 }
35 EOF)"
36 #
37 OUTPUT="$((${DIR}/../bin/print_json <<'EOF'
38 add_boolean_array "boolean_array";
39 #
40 add_byte_array "byte_array";
41 #
42 add_uint8_array "uint8_array";
43 add_uint16_array "uint16_array";
44 add_uint32_array "uint32_array";
45 add_uint64_array "uint64_array";
46 #
47 add_int8_array "int8_array";
48 add_int16_array "int16_array";
49 add_int32_array "int32_array";
50 add_int64_array "int64_array";
51 #
52 add_string_array "string_array";
53 #
54 /*
55 * The testing DSL does not presently support the generation of a completely
56 * empty object array. Thus, the following directive will produce an array
57 * with a single keyless object:
58 */
59 add_object_array "object_array";
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_03_empty_arrays.ksh 2

```
60 end;
61 EOF)"
62 #
63 complete
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_04_number_arrays.ksh 1

```
*****
1959 Mon Jul 21 17:23:00 2014
new/usr/src/test/util-tests/tests/libnvpair_json/json_04_number_arrays.ksh
5005 libnvpair JSON output broken by lint fixes
5006 libnvpair JSON cannot print int16 arrays
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
1 #!/bin/ksh
2 #
3 # This file and its contents are supplied under the terms of the
4 # Common Development and Distribution License ("CDDL"), version 1.0.
5 # You may only use this file in accordance with the terms of version
6 # 1.0 of the CDDL.
7 #
8 # A full copy of the text of the CDDL should have accompanied this
9 # source. A copy of the CDDL is also available via the Internet at
10 # http://www.illumos.org/license/CDDL.
11 #
12 #
13 #
14 # Copyright (c) 2014, Joyent, Inc.
15 #
16 #
17 DIR=$(dirname $(whence $0))
18 . ${DIR}/json_common
19 #
20 BASELINE="$(cat <<EOF
21 {\
22 "byte_array":[0,1,2,10,15,100,103,127,128,254,255],\
23 "uint8_array":[128,254,255,10,15,100,103,127,0,1,2],\
24 "uint16_array":[0,1000,2000,3210,4321,5432,10000,15000,16384,\
25 17992,35012,65535,0],\
26 "uint32_array":[0,4294967295,4026531855,1,2,1000,501],\
27 "uint64_array":[19850907,0,18446744073709551615],\
28 "int8_array":[39,39,39,39,39,39,39,-128,-127,0,127],\
29 "int16_array":[7532,-32768,0,32767,0,-32768,100],\
30 "int32_array":[-2147483648,0,32767,-32768,2147483647],\
31 "int64_array":[0,0,9223372036854775807,1,1,1,-9223372036854775808,0]\
32 }
33 EOF)"
34 #
35 OUTPUT="$( ${DIR}/../bin/print_json <<'EOF'
36 add_byte_array "byte_array"
37 "0" "1" "2" "10" "15" "100" "103" "127" "128" "254" "255";
38 #
39 add_uint8_array "uint8_array"
40 "128" "254" "255" "10" "15" "100" "103" "127" "0" "1" "2";
41 #
42 add_uint16_array "uint16_array"
43 "0" "1000" "2000" "3210" "4321" "5432" "10000" "15000" "16384"
44 "17992" "35012" "65535" "0";
45 #
46 add_uint32_array "uint32_array"
47 "0" "4294967295" "4026531855" "1" "2" "1000" "501";
48 #
49 add_uint64_array "uint64_array"
50 "19850907" "0" "18446744073709551615";
51 #
52 add_int8_array "int8_array"
53 "39" "39" "39" "39" "39" "39" "39" "39" "-128" "-127" "0" "127";
54 #
55 add_int16_array "int16_array"
56 "7532" "-32768" "0" "32767" "0" "-32768" "100";
57 #
58 add_int32_array "int32_array"
59 "-2147483648" "0" "32767" "-32768" "2147483647";
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_04_number_arrays.ksh 2

```
61 add_int64_array "int64_array"
62 "0" "0" "9223372036854775807" "1" "1" "1" "-9223372036854775808" "0";
63 EOF)"
64 #
65 complete
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_05_strings.ksh

1

```
*****
1425 Mon Jul 21 17:23:00 2014
new/usr/src/test/util-tests/tests/libnvpair_json/json_05_strings.ksh
5005 libnvpair JSON output broken by lint fixes
5006 libnvpair JSON cannot print int16 arrays
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
1  #!/bin/ksh
2  #
3  # This file and its contents are supplied under the terms of the
4  # Common Development and Distribution License ("CDDL"), version 1.0.
5  # You may only use this file in accordance with the terms of version
6  # 1.0 of the CDDL.
7  #
8  # A full copy of the text of the CDDL should have accompanied this
9  # source. A copy of the CDDL is also available via the Internet at
10 # http://www.illumos.org/license/CDDL.
11 #
12 #
13 #
14 # Copyright (c) 2014, Joyent, Inc.
15 #
16 #
17 DIR=$(dirname $(whence $0))
18 . ${DIR}/json_common
19 #
20 #
21 # This test checks UTF-8 parsing behaviour
22 #
23 export LC_ALL="en_US.UTF-8"
24 export LANG="${LANG}"
25 #
26 BASELINE="$(cat <<EOF
27 {\
28 "blank": "", \
29 " ": "blank key", \
30 " ": "whitespace key", \
31 "\t\t\t\t\t": "tab key", \
32 "escapes": "escape \x001b newline \n cr \r backslash \\ quote \"", \
33 "escape array": [ \
34 "escape \u001b", \
35 "alarm \u0007", \
36 "backspace \b", \
37 "formfeed \f", \
38 "newline \n", \
39 "return \r", \
40 "tab \t", \
41 "vertical tab \u000b", \
42 "black circle (UTF-8) \u25cf" \
43 ] \
44 }
45 EOF)"
46 #
47 OUTPUT="$((${DIR}/../bin/print_json <<'EOF'
48 add_string "blank" "";
49 add_string " " "blank key";
50 add_string " " "whitespace key";
51 add_string " \t\t\t\t\t" "tab key";
52 add_string "escapes" "escape \x1b newline \n cr \r backslash \\ quote \"";
53 add_string_array "escape array"
54 "escape \x1b"
55 "alarm \a"
56 "backspace \b"
57 "formfeed \f"
58 "newline \n"
59 "return \r"
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_05_strings.ksh

2

```
60 "tab \t"
61 "vertical tab \v"
62 "black circle (UTF-8) \xe2\x97\x8f";
63 EOF)"
64 #
65 complete
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_06_nested.ksh

1

```
1300 Mon Jul 21 17:23:00 2014
new/usr/src/test/util-tests/tests/libnvpair_json/json_06_nested.ksh
5005 libnvpair JSON output broken by lint fixes
5006 libnvpair JSON cannot print int16 arrays
Reviewed by: Robert Mustacchi <rm@joyent.com>
```

```
1 #!/bin/ksh
2 #
3 # This file and its contents are supplied under the terms of the
4 # Common Development and Distribution License ("CDDL"), version 1.0.
5 # You may only use this file in accordance with the terms of version
6 # 1.0 of the CDDL.
7 #
8 # A full copy of the text of the CDDL should have accompanied this
9 # source. A copy of the CDDL is also available via the Internet at
10 # http://www.illumos.org/license/CDDL.
11 #
12 #
13 #
14 # Copyright (c) 2014, Joyent, Inc.
15 #
```

```
17 DIR=$(dirname $(whence $0))
18 . ${DIR}/json_common
```

```
20 BASELINE="$(cat <<EOF
21 {\
22 "a":{\},\
23 "b":{\
24 "name":"Roger","age":35\
25 },\
26 "c":{\
27 "d":{\
28 "name":"Stephen","age":27},\
29 "e":{\
30 "name":"Roberta","age":43,"pet":{\
31 "name":"Mister Bumberscratch",\
32 "species":"cat",\
33 "alive":true,\
34 "available_legs":[1,2,3,4]\
35 }\
36 }\
37 }\
38 }
39 EOF)"
```

```
41 OUTPUT="$( ${DIR}/../bin/print_json <<'EOF'
42 add_object "a";
43 end;
```

```
45 add_object "b";
46     add_string "name" "Roger";
47     add_uint16 "age" "35";
48 end;
```

```
50 add_object "c";
51     add_object "d";
52         add_string "name" "Stephen";
53         add_uint16 "age" "27";
54     end;
55     add_object "e";
56         add_string "name" "Roberta";
57         add_uint16 "age" "43";
58         add_object "pet";
59             add_string "name" "Mister Bumberscratch";
```

new/usr/src/test/util-tests/tests/libnvpair_json/json_06_nested.ksh

2

```
60     add_string "species" "cat";
61     add_boolean_value "alive" "true";
62     add_uint8_array "available_legs" "1" "2" "3" "4";
63     end;
64     end;
65 end;
66 EOF)"

68 complete
```



```

*****
1892 Mon Jul 21 17:23:00 2014
new/usr/src/test/util-tests/tests/libnvpair_json/json_07_nested_arrays.ksh
5005 libnvpair JSON output broken by lint fixes
5006 libnvpair JSON cannot print int16 arrays
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
1 #!/bin/ksh
2 #
3 # This file and its contents are supplied under the terms of the
4 # Common Development and Distribution License ("CDDL"), version 1.0.
5 # You may only use this file in accordance with the terms of version
6 # 1.0 of the CDDL.
7 #
8 # A full copy of the text of the CDDL should have accompanied this
9 # source. A copy of the CDDL is also available via the Internet at
10 # http://www.illumos.org/license/CDDL.
11 #
12 #
13 #
14 # Copyright (c) 2014, Joyent, Inc.
15 #
16 #
17 DIR=$(dirname $(whence $0))
18 . ${DIR}/json_common
19 #
20 BASELINE="$(cat <<EOF
21 {\
22 "event_store":{\
23 "name":"Occurrences",\
24 "events":[\
25 {"time":489715200,"desc":"inception"},\
26 {"time":1057708800,"desc":"maturation"},\
27 {"time":1344816000,"desc":"migration"},\
28 {"time":1405296000,"desc":"integration"},\
29 ]\
30 }\
31 },\
32 "first_level":[\
33 {"second_level_0":[\
34 "s10_a":true,\
35 "s10_b":"aaaa"\
36 ],\
37 {"x":1234}\
38 ],\
39 "second_level_1":[{}],\
40 "second_level_2":[\
41 {"alpha":"a"},\
42 {"beta":"b"},\
43 {"gamma":"c"},\
44 {"delta":"d"},\
45 {"order":["a","b","c","d"]}\
46 ]\
47 ]\
48 ]\
49 }
50 EOF)"
51 #
52 OUTPUT="$((${DIR}/../bin/print_json <<'EOF'
53 add_object "event_store";
54     add_string "name" "Occurrences";
55     add_object_array "events";
56         add_uint32 "time" "489715200";
57         add_string "desc" "inception";
58     next;

```

```

60         add_uint32 "time" "1057708800";
61         add_string "desc" "maturation";
62     next;
63 #
64         add_uint32 "time" "1344816000";
65         add_string "desc" "migration";
66     next;
67 #
68         add_uint32 "time" "1405296000";
69         add_string "desc" "integration";
70     next;
71     end;
72 end;
73 add_object_array "first_level";
74     add_object_array "second_level_0";
75         add_boolean "s10_a";
76         add_string "s10_b" "aaaa";
77     next;
78         add_int32 "x" "1234";
79     end;
80     add_object_array "second_level_1";
81     end;
82     add_object_array "second_level_2";
83         add_string "alpha" "a";
84     next;
85         add_string "beta" "b";
86     next;
87         add_string "gamma" "c";
88     next;
89         add_string "delta" "d";
90     next;
91         add_string_array "order" "a" "b" "c" "d";
92     end;
93 end;
94 EOF)"
95 #
96 complete

```

new/usr/src/test/util-tests/tests/libnvpair_json/json_common.ksh

1

808 Mon Jul 21 17:23:00 2014

new/usr/src/test/util-tests/tests/libnvpair_json/json_common.ksh

5005 libnvpair JSON output broken by lint fixes

5006 libnvpair JSON cannot print int16 arrays

Reviewed by: Robert Mustacchi <rm@joyent.com>

```
1 #!/bin/ksh
2 #
3 # This file and its contents are supplied under the terms of the
4 # Common Development and Distribution License ("CDDL"), version 1.0.
5 # You may only use this file in accordance with the terms of version
6 # 1.0 of the CDDL.
7 #
8 # A full copy of the text of the CDDL should have accompanied this
9 # source. A copy of the CDDL is also available via the Internet at
10 # http://www.illumos.org/license/CDDL.
11 #
12 #
13 #
14 # Copyright (c) 2014, Joyent, Inc.
15 #
16 #
17 function complete {
18     if [[ "${PRINT_OUTPUT}" ]]; then
19         printf "%s\n" "${OUTPUT}"
20         exit 0
21     elif [[ "${OUTPUT}" == "${BASELINE}" ]]; then
22         printf "TEST PASS: %s\n" "${basename $0}"
23         exit 0
24     else
25         printf "TEST FAIL: %s\n" "${basename $0}"
26         printf "EXPECTED: %s\n" "${BASELINE}"
27         printf "ACTUAL: %s\n" "${OUTPUT}"
28         exit 1
29     fi
30 }
```

```

*****
18272 Mon Jul 21 17:23:00 2014
new/usr/src/test/util-tests/tests/libnvpair_json/print_json.c
5005 libnvpair JSON output broken by lint fixes
5006 libnvpair JSON cannot print int16 arrays
Reviewed by: Robert Mustacchi <rm@joyent.com>
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2014 Joyent, Inc.
14 */

16 /*
17  * This program implements a small domain-specific language (DSL) for the
18  * generation of nvlists, and subsequent printing in JSON-formatted output.
19  * The test suite uses this tool to drive the JSON formatting routines in
20  * libnvpair(3LIB) for testing.
21 */

23 #include <stdlib.h>
24 #include <stdio.h>
25 #include <errno.h>
26 #include <string.h>
27 #include <ctype.h>
28 #include <limits.h>
29 #include <locale.h>

31 #include <libnvpair.h>

33 #define MAX_ARGS      100
34 #define CMD_NAME_LEN  50

36 /*
37  * As we are parsing a language that allows the creation of arbitrarily nested
38  * state, i.e. both nested nvlists and arrays of nested nvlists, we store that
39  * state in a stack. The top frame in the stack represents the nested nvlst
40  * (or nvlists, for an array) that we are currently building.
41  *
42  * When creating an array, the "next" directive advances lw_pos and allocates a
43  * new nvlst. The "end" directive commits either the nvlst, or array of
44  * nvlists, into the parent nvlst. It then pops and frees the stack frame
45  * before returning control to the parser.
46  */

48 typedef struct list_wrap {
49     nvlst_t *lw_nvl[MAX_ARGS];
50     char *lw_name;
51     int lw_pos;
52     boolean_t lw_array;
53     struct list_wrap *lw_next;
54 } list_wrap_t;

56 int
57 list_wrap_depth(list_wrap_t *lw)
58 {
59     int d = 0;

```

```

61     while (lw != NULL) {
62         d++;
63         lw = lw->lw_next;
64     }

66     return (d);
67 }

69 list_wrap_t *
70 list_wrap_alloc(list_wrap_t *next)
71 {
72     list_wrap_t *out = calloc(1, sizeof (list_wrap_t));

74     if (out == NULL)
75         abort();

77     out->lw_next = next;

79     return (out);
80 }

82 list_wrap_t *
83 list_wrap_pop_and_free(list_wrap_t *lw)
84 {
85     list_wrap_t *next = lw->lw_next;

87     free(lw->lw_name);
88     free(lw);

90     return (next);
91 }

93 /*
94  * Generic integer and floating point parsing routines:
95  */

97 int
98 parse_int(char *in, int64_t *val, int64_t min, int64_t max)
99 {
100     int64_t t;
101     char *end = NULL;

103     errno = 0;
104     t = strtoll(in, &end, 10);
105     if (errno != 0 || end == in || *end != '\0') {
106         if (errno == ERANGE) {
107             (void) fprintf(stderr, "ERROR: integer %s not in "
108                 "range [%lld,%lld]\n", in, min, max);
109             return (-1);
110         }
111         (void) fprintf(stderr, "ERROR: could not parse \"%s\" as "
112             "signed integer (%s)\n", in, strerror(errno));
113         return (-1);
114     }

116     if (t < min || t > max) {
117         (void) fprintf(stderr, "ERROR: integer %lld not in range "
118             "[%lld,%lld]\n", t, min, max);
119         return (-1);
120     }

122     *val = t;
123     return (0);
124 }

```

```

126 int
127 parse_uint(char *in, uint64_t *val, uint64_t min, uint64_t max)
128 {
129     uint64_t t;
130     char *end = NULL;
131
132     errno = 0;
133     t = strtoull(in, &end, 10);
134     if (errno != 0 || end == in || *end != '\0') {
135         if (errno == ERANGE) {
136             (void) fprintf(stderr, "ERROR: integer %s not in "
137                 "range [%llu,%llu]\n", in, min, max);
138             return (-1);
139         }
140         (void) fprintf(stderr, "ERROR: could not parse \"%s\" as "
141             "unsigned integer (%s)\n", in, strerror(errno));
142         return (-1);
143     }
144
145     if (t < min || t > max) {
146         (void) fprintf(stderr, "ERROR: integer %llu not in range "
147             "[%llu,%llu]\n", t, min, max);
148         return (-1);
149     }
150
151     *val = t;
152     return (0);
153 }
154
155 int
156 parse_double(char *in, double *val)
157 {
158     double t;
159     char *end = NULL;
160
161     errno = 0;
162     t = strtod(in, &end);
163     if (errno != 0 || end == in || *end != '\0') {
164         (void) fprintf(stderr, "ERROR: could not parse \"%s\" as "
165             "double\n", in);
166         return (-1);
167     }
168
169     *val = t;
170     return (0);
171 }
172
173 /*
174  * Command-specific handlers for directives specified in the DSL input:
175  */
176
177 typedef int (*command_handler_t)(list_wrap_t **, boolean_t, int,
178     char **);
179
180 static int
181 ch_add_string(list_wrap_t **lw, boolean_t array, int argc, char **argv)
182 {
183     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
184
185     if (array) {
186         if (nvlist_add_string_array(nvl, argv[0], &argv[1],
187             argc - 1) != 0) {
188             (void) fprintf(stderr, "fail at "
189                 "nvlist_add_string_array\n");
190             return (-1);
191         }

```

```

192     } else {
193         if (nvlist_add_string(nvl, argv[0], argv[1]) != 0) {
194             (void) fprintf(stderr, "fail at nvlist_add_string\n");
195             return (-1);
196         }
197     }
198
199     return (0);
200 }
201
202 static int
203 ch_add_boolean(list_wrap_t **lw, boolean_t array, int argc, char **argv)
204 {
205     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
206
207     if (array)
208         abort();
209
210     if (nvlist_add_boolean(nvl, argv[0]) != 0) {
211         (void) fprintf(stderr, "fail at nvlist_add_boolean\n");
212         return (-1);
213     }
214     return (0);
215 }
216
217 static int
218 ch_add_boolean_value(list_wrap_t **lw, boolean_t array, int argc, char **argv)
219 {
220     int i;
221     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
222     boolean_t arrval[MAX_ARGS];
223
224     for (i = 1; i < argc; i++) {
225         if (strcmp(argv[i], "true") == 0) {
226             arrval[i - 1] = B_TRUE;
227         } else if (strcmp(argv[i], "false") == 0) {
228             arrval[i - 1] = B_FALSE;
229         } else {
230             (void) fprintf(stderr, "invalid boolean value: %s\n",
231                 argv[i]);
232             return (-1);
233         }
234     }
235
236     if (array) {
237         if (nvlist_add_boolean_array(nvl, argv[0], arrval,
238             argc - 1) != 0) {
239             (void) fprintf(stderr, "fail at "
240                 "nvlist_add_boolean_array\n");
241             return (-1);
242         }
243     } else {
244         if (nvlist_add_boolean_value(nvl, argv[0], arrval[0]) != 0) {
245             (void) fprintf(stderr, "fail at "
246                 "nvlist_add_boolean_value\n");
247             return (-1);
248         }
249     }
250
251     return (0);
252 }
253
254
255 /*
256  * The confluence of a strongly typed C API for libnvpair(3LIB) and the
257  * combinatorial explosion of both sizes and signedness is unfortunate. Rather

```

```

258 * than reproduce the same code over and over, this macro parses an integer,
259 * checks applicable bounds based on size and signedness, and stores the value
260 * (or array of values).
261 */
262 #define DO_CMD_NUMBER(typ, nam, min, max, ptyp, func) \
263     ptyp val; \
264     typ ## _t arrval[MAX_ARGS]; \
265     int i; \
266     for (i = 1; i < argc; i++) { \
267         if (func(argv[i], &val, min, max) != 0) { \
268             return (-1); \
269         } \
270         arrval[i - 1] = (typ ## _t) val; \
271     } \
272     if (array) { \
273         if (nvlist_add_## nam ## _array(nvl, argv[0], \
274             arrval, argc - 1) != 0) { \
275             (void) fprintf(stderr, "fail at " \
276                 "nvlist_add_#nam "_array"\n"); \
277             return (-1); \
278         } \
279     } else { \
280         if (nvlist_add_## nam(nvl, argv[0], \
281             arrval[0]) == -1) { \
282             (void) fprintf(stderr, "fail at " \
283                 "nvlist_add_#nam "\n"); \
284             return (-1); \
285         } \
286     } \
287     return (0);
289 static int
290 ch_add_byte(list_wrap_t **lw, boolean_t array, int argc, char **argv)
291 {
292     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
293
294     DO_CMD_NUMBER(uchar, byte, 0, UCHAR_MAX, uint64_t, parse_uint)
295 }
297 static int
298 ch_add_int8(list_wrap_t **lw, boolean_t array, int argc, char **argv)
299 {
300     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
301
302     DO_CMD_NUMBER(int8, int8, INT8_MIN, INT8_MAX, int64_t, parse_int)
303 }
305 static int
306 ch_add_uint8(list_wrap_t **lw, boolean_t array, int argc, char **argv)
307 {
308     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
309
310     DO_CMD_NUMBER(uint8, uint8, 0, UINT8_MAX, uint64_t, parse_uint)
311 }
313 static int
314 ch_add_int16(list_wrap_t **lw, boolean_t array, int argc, char **argv)
315 {
316     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
317
318     DO_CMD_NUMBER(int16, int16, INT16_MIN, INT16_MAX, int64_t, parse_int)
319 }
321 static int
322 ch_add_uint16(list_wrap_t **lw, boolean_t array, int argc, char **argv)
323 {

```

```

324     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
325
326     DO_CMD_NUMBER(uint16, uint16, 0, UINT16_MAX, uint64_t, parse_uint)
327 }
329 static int
330 ch_add_int32(list_wrap_t **lw, boolean_t array, int argc, char **argv)
331 {
332     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
333
334     DO_CMD_NUMBER(int32, int32, INT32_MIN, INT32_MAX, int64_t, parse_int)
335 }
337 static int
338 ch_add_uint32(list_wrap_t **lw, boolean_t array, int argc, char **argv)
339 {
340     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
341
342     DO_CMD_NUMBER(uint32, uint32, 0, UINT32_MAX, uint64_t, parse_uint)
343 }
345 static int
346 ch_add_int64(list_wrap_t **lw, boolean_t array, int argc, char **argv)
347 {
348     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
349
350     DO_CMD_NUMBER(int64, int64, INT64_MIN, INT64_MAX, int64_t, parse_int)
351 }
353 static int
354 ch_add_uint64(list_wrap_t **lw, boolean_t array, int argc, char **argv)
355 {
356     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
357
358     DO_CMD_NUMBER(uint64, uint64, 0, UINT64_MAX, uint64_t, parse_uint)
359 }
361 static int
362 ch_add_double(list_wrap_t **lw, boolean_t array, int argc, char **argv)
363 {
364     nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
365     double val;
366
367     if (array)
368         abort();
369
370     if (parse_double(argv[1], &val) != 0) {
371         return (-1);
372     }
373
374     if (nvlist_add_double(nvl, argv[0], val) != 0) {
375         (void) fprintf(stderr, "fail at nvlist_add_double_value\n");
376         return (-1);
377     }
378
379     return (0);
380 }
382 static int
383 ch_end(list_wrap_t **lw, boolean_t array, int argc, char **argv)
384 {
385     nvlist_t *parent;
386     char *name;
387
388     if (list_wrap_depth(*lw) < 2) {
389         (void) fprintf(stderr, "ERROR: not nested, cannot end.\n");

```

```

390         return (-1);
391     }

393     parent = (*lw)->lw_next->lw_nvlist[(*lw)->lw_next->lw_pos];
394     name = (*lw)->lw_name;
395     if ((*lw)->lw_array) {
396         /*
397          * This was an array of objects.
398          */
399         nvlist_t **children = (*lw)->lw_nvlist;
400         int nelems = (*lw)->lw_pos + 1;

402         if (nvlist_add_nvlist_array(parent, name, children,
403             nelems) != 0) {
404             (void) fprintf(stderr, "fail at "
405                 "nvlist_add_nvlist_array\n");
406             return (-1);
407         }
408     } else {
409         /*
410          * This was a single object.
411          */
412         nvlist_t *child = (*lw)->lw_nvlist[0];

414         if ((*lw)->lw_pos != 0)
415             abort();

417         if (nvlist_add_nvlist(parent, name, child) != 0) {
418             (void) fprintf(stderr, "fail at nvlist_add_nvlist\n");
419             return (-1);
420         }
421     }

423     *lw = list_wrap_pop_and_free(*lw);

425     return (0);
426 }

428 static int
429 ch_next(list_wrap_t **lw, boolean_t array, int argc, char **argv)
430 {
431     if (!(*lw)->lw_array) {
432         (void) fprintf(stderr, "ERROR: cannot use 'next' outside an "
433             "object array.\n");
434         return (-1);
435     }

437     if ((*lw)->lw_pos++ >= MAX_ARGS) {
438         (void) fprintf(stderr, "ERROR: object array too long\n");
439         return (-1);
440     }

442     if (nvlist_alloc(&(*lw)->lw_nvlist[(*lw)->lw_pos], NV_UNIQUE_NAME,
443         0) != 0) {
444         (void) fprintf(stderr, "ERROR: failed at nvlist_alloc\n");
445         return (-1);
446     }

448     return (0);
449 }

451 static int
452 ch_add_object(list_wrap_t **lw, boolean_t array, int argc, char **argv)
453 {
454     *lw = list_wrap_alloc(*lw);

```

```

456     (*lw)->lw_name = strdup(argv[0]);
457     (*lw)->lw_array = array;

459     if (nvlist_alloc(&(*lw)->lw_nvlist[0], NV_UNIQUE_NAME, 0) != 0) {
460         (void) fprintf(stderr, "fail at nvlist_alloc\n");
461         return (-1);
462     }

464     return (0);
465 }

467 typedef struct command {
468     char cmd_name[CMD_NAME_LEN];
469     command_handler_t cmd_func;
470     int cmd_min_args;
471     int cmd_max_args;
472     boolean_t cmd_array_mode;
473 } command_t;

475 /*
476  * These are the commands we support in the testing DSL, and their
477  * handling functions:
478  */
479 command_t command_handlers[] = {
480     { "add boolean", ch_add_boolean, 1, 1, B_FALSE },
481     { "add boolean_value", ch_add_boolean_value, 2, 2, B_FALSE },
482     { "add byte", ch_add_byte, 2, 2, B_FALSE },
483     { "add int8", ch_add_int8, 2, 2, B_FALSE },
484     { "add uint8", ch_add_uint8, 2, 2, B_FALSE },
485     { "add int16", ch_add_int16, 2, 2, B_FALSE },
486     { "add uint16", ch_add_uint16, 2, 2, B_FALSE },
487     { "add int32", ch_add_int32, 2, 2, B_FALSE },
488     { "add uint32", ch_add_uint32, 2, 2, B_FALSE },
489     { "add int64", ch_add_int64, 2, 2, B_FALSE },
490     { "add uint64", ch_add_uint64, 2, 2, B_FALSE },
491     { "add double", ch_add_double, 2, 2, B_FALSE },
492     { "add string", ch_add_string, 2, 2, B_FALSE },
493     { "add object", ch_add_object, 1, 1, B_FALSE },
494     { "add boolean_array", ch_add_boolean_value, 1, MAX_ARGS, B_TRUE },
495     { "add byte_array", ch_add_byte, 1, MAX_ARGS, B_TRUE },
496     { "add int8_array", ch_add_int8, 1, MAX_ARGS, B_TRUE },
497     { "add uint8_array", ch_add_uint8, 1, MAX_ARGS, B_TRUE },
498     { "add int16_array", ch_add_int16, 1, MAX_ARGS, B_TRUE },
499     { "add uint16_array", ch_add_uint16, 1, MAX_ARGS, B_TRUE },
500     { "add int32_array", ch_add_int32, 1, MAX_ARGS, B_TRUE },
501     { "add uint32_array", ch_add_uint32, 1, MAX_ARGS, B_TRUE },
502     { "add int64_array", ch_add_int64, 1, MAX_ARGS, B_TRUE },
503     { "add uint64_array", ch_add_uint64, 1, MAX_ARGS, B_TRUE },
504     { "add string_array", ch_add_string, 1, MAX_ARGS, B_TRUE },
505     { "add object_array", ch_add_object, 1, 1, B_TRUE },
506     { "end", ch_end, 0, 0, B_FALSE },
507     { "next", ch_next, 0, 0, B_FALSE },
508     { 0 }
509 };

511 /*
512  * This function determines which command we are executing, checks argument
513  * counts, and dispatches to the appropriate handler:
514  */
515 static int
516 command_call(list_wrap_t **lw, char *command, int argc, char **argv)
517 {
518     int ch;

520     for (ch = 0; command_handlers[ch].cmd_name[0] != '\0'; ch++) {
521         if (strcmp(command, command_handlers[ch].cmd_name) != 0)

```

```

522         continue;
524         if (argc > command_handlers[ch].cmd_max_args ||
525             argc < command_handlers[ch].cmd_min_args) {
527             (void) fprintf(stderr, "ERROR: command \"%s\"
528                 \" expects between %d and %d arguments,\"
529                 \" but %d were provided.\n\", command,
530                 command_handlers[ch].cmd_min_args,
531                 command_handlers[ch].cmd_max_args,
532                 argc);
534             return (-1);
535         }
537         return (command_handlers[ch].cmd_func(lw,
538             command_handlers[ch].cmd_array_mode, argc, argv));
539     }
541     (void) fprintf(stderr, "ERROR: invalid command: \"%s\"\n", command);
543     return (-1);
544 }
546 /*
547  * The primary state machine for parsing the input DSL is implemented in
548  * this function:
549  */
551 typedef enum state {
552     STATE_REST = 1,
553     STATE_COMMAND,
554     STATE_ARG_FIND,
555     STATE_ARG,
556     STATE_ARG_ESCAPE,
557     STATE_ARG_ESCAPE_HEX,
558     STATE_C_COMMENT_0,
559     STATE_C_COMMENT_1,
560     STATE_C_COMMENT_2
561 } state_t;
563 int
564 parse(FILE *in, list_wrap_t **lw)
565 {
566     char b[8192];
567     int bp;
568     state_t st = STATE_REST;
569     int argc = 0;
570     char *argv[MAX_ARGS];
571     int line = 1;
572     char hex[3];
573     int nhex = 0;
575     b[0] = '\0';
576     bp = 0;
578     for (;;) {
579         int c = fgetc(in);
581         /*
582          * Signal an error if the file ends part way through a
583          * construct:
584          */
585         if (st != STATE_REST && c == EOF) {
586             (void) fprintf(stderr, "ERROR: unexpected end of \"
587                 \"file\n");

```

```

588         return (-1);
589     } else if (c == EOF) {
590         return (0);
591     }
593     if (c == '\n')
594         line++;
596     switch (st) {
597     case STATE_REST:
598         if (isalpha(c) || c == '_') {
599             argc = 0;
600             bp = 0;
601             b[bp++] = c;
602             b[bp] = '\0';
603             st = STATE_COMMAND;
604             continue;
605         } else if (c == ' ' || c == '\t' || c == '\n') {
606             /*
607              * Ignore whitespace.
608              */
609             continue;
610         } else if (c == '/') {
611             st = STATE_C_COMMENT_0;
612             continue;
613         } else {
614             goto unexpected;
615         }
617     case STATE_C_COMMENT_0:
618         if (c != '*') {
619             goto unexpected;
620         }
621         st = STATE_C_COMMENT_1;
622         continue;
624     case STATE_C_COMMENT_1:
625         if (c == '*') {
626             st = STATE_C_COMMENT_2;
627         }
628         continue;
630     case STATE_C_COMMENT_2:
631         if (c == '/') {
632             st = STATE_REST;
633         } else if (c != '*') {
634             st = STATE_C_COMMENT_1;
635         }
636         continue;
638     case STATE_COMMAND:
639         if (isalnum(c) || c == '_') {
640             b[bp++] = c;
641             b[bp] = '\0';
642             st = STATE_COMMAND;
644             continue;
646         } else if (isspace(c)) {
647             /*
648              * Start collecting arguments into 'b'
649              * after the command.
650              */
651             st = STATE_ARG_FIND;
652             bp++;

```

```

654         continue;
655     } else if (c == ';' ) {
656         /*
657          * This line was _just_ a command,
658          * so break out and process now:
659          */
660         goto execute;
661     } else {
662         goto unexpected;
663     }
664
665     case STATE_ARG_FIND:
666         if (isspace(c)) {
667             /*
668              * Whitespace, ignore.
669              */
670             continue;
671
672         } else if (c == ';' ) {
673             /*
674              * Break out to process command.
675              */
676             goto execute;
677
678         } else if (c == '"') {
679             st = STATE_ARG;
680
681             argv[argc] = &b[bp];
682             b[bp] = '\0';
683
684             continue;
685         } else {
686             goto unexpected;
687         }
688
689     case STATE_ARG:
690         if (c == '"') {
691             if (argc++ >= MAX_ARGS) {
692                 (void) fprintf(stderr, "ERROR: too "
693                     "many args\n");
694                 return (-1);
695             }
696             st = STATE_ARG_FIND;
697             continue;
698         } else if (c == '\n') {
699             (void) fprintf(stderr, "ERROR: line not "
700                 "finished\n");
701             return (-1);
702         } else if (c == '\\') {
703             st = STATE_ARG_ESCAPE;
704             continue;
705         } else {
706             b[bp++] = c;
707             b[bp] = '\0';
708             continue;
709         }
710
711     case STATE_ARG_ESCAPE:
712         if (c == 'a') {
713             c = '\a';
714         } else if (c == 'b') {
715             c = '\b';
716         } else if (c == 'f') {
717             c = '\f';
718         } else if (c == 'n') {
719             c = '\n';

```

```

720         } else if (c == 'r') {
721             c = '\r';
722         } else if (c == 't') {
723             c = '\t';
724         } else if (c == 'v') {
725             c = '\v';
726         } else if (c == 'x') {
727             st = STATE_ARG_ESCAPE_HEX;
728             hex[0] = hex[1] = hex[2] = '\0';
729             nhex = 0;
730             continue;
731         } else if (c != '\\') {
732             goto unexpected;
733         }
734
735         b[bp++] = c;
736         b[bp] = '\0';
737         st = STATE_ARG;
738         continue;
739
740     case STATE_ARG_ESCAPE_HEX:
741         if (!isxdigit(c)) {
742             goto unexpected;
743         }
744         hex[nhex] = c;
745         if (nhex++ >= 1) {
746             /*
747              * The hex escape pair is complete, parse
748              * the integer and insert it as a character:
749              */
750             int x;
751             errno = 0;
752             if ((x = strtoul(hex, NULL, 16)) == 0 ||
753                 errno != 0) {
754                 goto unexpected;
755             }
756             b[bp++] = (char)x;
757             b[bp] = '\0';
758             st = STATE_ARG;
759         }
760         continue;
761     }
762
763     /*
764     * We do not ever expect to break out of the switch block
765     * above.  If we do, it's a programmer error.
766     */
767     abort();
768
769     execute:
770         if (command_call(lw, b, argc, argv) == -1)
771             return (-1);
772
773         st = STATE_REST;
774         continue;
775
776     unexpected:
777         (void) fprintf(stderr, "ERROR: (line %d) unexpected "
778             "character: %c\n", line, c);
779         return (-1);
780     }
781 }
782
783 /*
784  * Entry point:
785  */

```



```
786 int
787 main(int argc, char **argv)
788 {
789     int rc = EXIT_FAILURE;
790     list_wrap_t *lw;
791
792     /*
793      * Be locale-aware. The JSON output functions will process multibyte
794      * characters in the current locale, and emit a correct JSON encoding
795      * for unprintable characters.
796      */
797     if (setlocale(LC_ALL, "") == NULL) {
798         (void) fprintf(stderr, "Could not set locale: %s\n",
799             strerror(errno));
800         goto out;
801     }
802
803     lw = list_wrap_alloc(NULL);
804
805     if (nvlist_alloc(&lw->lw_nvlist[0], NV_UNIQUE_NAME, 0) != 0)
806         goto out;
807
808     /*
809      * Generate the list from the commands passed to us on stdin:
810      */
811     if (parse(stdin, &lw) != 0)
812         goto out;
813
814     /*
815      * Print the resultant list, and a terminating newline:
816      */
817     if (nvlist_print_json(stdout, lw->lw_nvlist[0]) != 0 ||
818         fprintf(stdout, "\n") < 0)
819         goto out;
820
821     rc = EXIT_SUCCESS;
822
823 out:
824     (void) list_wrap_pop_and_free(lw);
825
826     return (rc);
827 }
```