```
**********************************************************
    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>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright (c) 2014, Joyent, Inc. All rights reserved.
# Copyright (c) 2013, Joyent, Inc. All rights reserved.
#

LIBRARY=        libnvpair.a
VERS=           .1

OBJECTS=        libnvpair.o \
                nvpair_alloc_system.o \
                nvpair_alloc_fixed.o \
                nvpair.o \
                fnvpair.o \
                nvpair_json.o

include ../../Makefile.lib
include ../../Makefile.rootfs

SRCS=           ../libnvpair.c \
                ../nvpair_alloc_system.c \
                ../nvpair_json.c \
                $(SRC)/common/nvpair/nvpair_alloc_fixed.c \
                $(SRC)/common/nvpair/nvpair.c \
                $(SRC)/common/nvpair/fnvpair.c

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

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

# turn off warning caused by lint bug: not understanding SCNi8 "hhi"
```

```
LINTFLAGS +=    -erroff=E_BAD_FORMAT_STR2
LINTFLAGS +=    -erroff=E_INVALID_TOKEN_IN_DEFINE_MACRO
LINTFLAGS +=    -erroff=E_RET_INT_IMPLICITLY
LINTFLAGS +=    -erroff=E_FUNC_USED_VAR_ARG2
LINTFLAGS +=    -erroff=E_CONSTANT_CONDITION
LINTFLAGS64 +=  -erroff=E_BAD_FORMAT_STR2
LINTFLAGS64 +=  -erroff=E_INVALID_TOKEN_IN_DEFINE_MACRO
LINTFLAGS64 +=  -erroff=E_RET_INT_IMPLICITLY
LINTFLAGS64 +=  -erroff=E_FUNC_USED_VAR_ARG2
LINTFLAGS64 +=  -erroff=E_CONSTANT_CONDITION

CERRWARN +=     -_gcc=-Wno-type-limits
CERRWARN +=     -_gcc=-Wno-parentheses
CERRWARN +=     -_gcc=-Wno-uninitialized

CFLAGS +=       $(CCVERBOSE)
CPPFLAGS +=     -D_REENTRANT

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

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

.KEEP_STATE:

all: $(LIBS)

lint: lintcheck

include ../../Makefile.targ

pics/%.o: $(SRC)/common/nvpair/%.c
        $(COMPILE.c) -o $@ $<
        $(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>
**********************************************************
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */
/*
 * Copyright (c) 2014, Joyent, Inc. All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 */

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <wchar.h>
#include <sys/debug.h>

#include "libnvpair.h"

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

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

        bzero(&mbr, sizeof (mbr));

        FPRINTF(fp, "\"");
        while ((sz = mbrtowc(&c, input, MB_CUR_MAX, &mbr)) > 0) {
                switch (c) {
                case '"':
```

```
                        FPRINTF(fp, "\\\"");
                        break;
                case '\n':
                        FPRINTF(fp, "\\n");
                        break;
                case '\r':
                        FPRINTF(fp, "\\r");
                        break;
                case '\\':
                        FPRINTF(fp, "\\\\");
                        break;
                case '\f':
                        FPRINTF(fp, "\\f");
                        break;
                case '\t':
                        FPRINTF(fp, "\\t");
                        break;
                case '\b':
                        FPRINTF(fp, "\\b");
                        break;
                default:
                        if ((c >= 0x00 && c <= 0x1f) ||
                            (c > 0x7f && c <= 0xffff)) {
                                /*
                                 * Render both Control Characters and Unicode
                                 * characters in the Basic Multilingual Plane
                                 * as JSON-escaped multibyte characters.
                                 */
                                FPRINTF(fp, "\\u%04x", (int)(0xffff & c));
                        } else if (c >= 0x20 && c <= 0x7f) {
                                /*
                                 * Render other 7-bit ASCII characters directly
                                 * and drop other, unrepresentable characters.
                                 */
                                FPRINTF(fp, "%c", (int)(0xff & c));
                        }
                        break;
                }
                input += sz;
        }

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

        FPRINTF(fp, "\"");
        return (0);
}

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

        FPRINTF(fp, "{");
        for (curr = nvlist_next_nvpair(nvl, NULL); curr;
            curr = nvlist_next_nvpair(nvl, curr)) {
                data_type_t type = nvpair_type(curr);

                if (!first)
                        FPRINTF(fp, ",");
                else
                        first = B_FALSE;

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

                switch (type) {
                case DATA_TYPE_STRING: {
                        char *string = fnvpair_value_string(curr);
                        if (nvlist_print_json_string(fp, string) == -1)
                                return (-1);
                        break;
                }

                case DATA_TYPE_BOOLEAN: {
                        FPRINTF(fp, "true");
                        break;
                }

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

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

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

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

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

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

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

                case DATA_TYPE_UINT32: {
                        FPRINTF(fp, "%u", fnvpair_value_uint32(curr));
                        break;
                }
```

```
                case DATA_TYPE_INT64: {
                        FPRINTF(fp, "%lld",
                            (long long)fnvpair_value_int64(curr));
                        break;
                }

                case DATA_TYPE_UINT64: {
                        FPRINTF(fp, "%llu",
                            (unsigned long long)fnvpair_value_uint64(curr));
                        break;
                }

                case DATA_TYPE_HRTIME: {
                        hrtime_t val;
                        VERIFY0(nvpair_value_hrtime(curr, &val));
                        FPRINTF(fp, "%llu", (unsigned long long)val);
                        break;
                }

                case DATA_TYPE_DOUBLE: {
                        double val;
                        VERIFY0(nvpair_value_double(curr, &val));
                        FPRINTF(fp, "%f", val);
                        break;
                }

                case DATA_TYPE_NVLIST: {
                        if (nvlist_print_json(fp,
                            fnvpair_value_nvlist(curr)) == -1)
                                return (-1);
                        break;
                }

                case DATA_TYPE_STRING_ARRAY: {
                        char **val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_string_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                if (nvlist_print_json_string(fp, val[i]) == -1)
                                        return (-1);
                        }
                        FPRINTF(fp, "]");
                        break;
                }

                case DATA_TYPE_NVLIST_ARRAY: {
                        nvlist_t **val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_nvlist_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                if (nvlist_print_json(fp, val[i]) == -1)
                                        return (-1);
                        }
                        FPRINTF(fp, "]");
                        break;
                }

                case DATA_TYPE_BOOLEAN_ARRAY: {
                        boolean_t *val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_boolean_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                FPRINTF(fp, val[i] == B_TRUE ?
                                    "true" : "false");
                        }
                        FPRINTF(fp, "]");
                        break;
                }

                case DATA_TYPE_BYTE_ARRAY: {
                        uchar_t *val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_byte_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                FPRINTF(fp, "%hhu", val[i]);
                        }
                        FPRINTF(fp, "]");
                        break;
                }

                case DATA_TYPE_UINT8_ARRAY: {
                        uint8_t *val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_uint8_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                FPRINTF(fp, "%hhu", val[i]);
                        }
                        FPRINTF(fp, "]");
                        break;
                }

                case DATA_TYPE_INT8_ARRAY: {
                        int8_t *val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_int8_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                FPRINTF(fp, "%hd", val[i]);
                        }
                        FPRINTF(fp, "]");
                        break;
                }

                case DATA_TYPE_UINT16_ARRAY: {
                        uint16_t *val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_uint16_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                FPRINTF(fp, "%hu", val[i]);
                        }
                        FPRINTF(fp, "]");
```

```
                        break;
                }

                case DATA_TYPE_INT16_ARRAY: {
                        int16_t *val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_int16_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                FPRINTF(fp, "%hd", val[i]);
                                FPRINTF(fp, "%hhd", val[i]);
                        }
                        FPRINTF(fp, "]");
                        break;
                }

                case DATA_TYPE_UINT32_ARRAY: {
                        uint32_t *val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_uint32_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                FPRINTF(fp, "%u", val[i]);
                        }
                        FPRINTF(fp, "]");
                        break;
                }

                case DATA_TYPE_INT32_ARRAY: {
                        int32_t *val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_int32_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                FPRINTF(fp, "%d", val[i]);
                        }
                        FPRINTF(fp, "]");
                        break;
                }

                case DATA_TYPE_UINT64_ARRAY: {
                        uint64_t *val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_uint64_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                FPRINTF(fp, "%llu",
                                    (unsigned long long)val[i]);
                        }
                        FPRINTF(fp, "]");
                        break;
                }

                case DATA_TYPE_INT64_ARRAY: {
                        int64_t *val;
                        uint_t valsz, i;
                        VERIFY0(nvpair_value_int64_array(curr, &val, &valsz));
                        FPRINTF(fp, "[");
                        for (i = 0; i < valsz; i++) {
                                if (i > 0)
                                        FPRINTF(fp, ",");
                                FPRINTF(fp, "%lld", (long long)val[i]);
                        }
                        FPRINTF(fp, "]");
                        break;
                }

                case DATA_TYPE_UNKNOWN:
                        return (-1);
                }
        }

        FPRINTF(fp, "}");
        return (0);
}
```
______***unchanged_portion_omitted_***

****
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>
****

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

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
#

set name=pkg.fmri value=pkg:/system/test/utiltest@$(PKGVERS)
set name=pkg.description value="Miscellaneous Utility Unit Tests"
set name=pkg.summary value="Utility Unit Test Suite"
set name=info.classification \
    value=org.opensolaris.category.2008:Development/System
set name=variant.arch value=$(ARCH)
dir path=opt/util-tests
dir path=opt/util-tests/bin
dir path=opt/util-tests/runfiles
dir path=opt/util-tests/tests
dir path=opt/util-tests/tests/libnvpair_json
file path=opt/util-tests/README mode=0444
file path=opt/util-tests/bin/print_json mode=0555
file path=opt/util-tests/bin/utiltest mode=0555
file path=opt/util-tests/runfiles/default.run mode=0444
file path=opt/util-tests/tests/allowed-ips mode=0555
file path=opt/util-tests/tests/libnvpair_json/json_00_blank mode=0555
file path=opt/util-tests/tests/libnvpair_json/json_01_boolean mode=0555
file path=opt/util-tests/tests/libnvpair_json/json_02_numbers mode=0555
file path=opt/util-tests/tests/libnvpair_json/json_03_empty_arrays mode=0555
file path=opt/util-tests/tests/libnvpair_json/json_04_number_arrays mode=0555
file path=opt/util-tests/tests/libnvpair_json/json_05_strings mode=0555
file path=opt/util-tests/tests/libnvpair_json/json_06_nested mode=0555
file path=opt/util-tests/tests/libnvpair_json/json_07_nested_arrays mode=0555
file path=opt/util-tests/tests/libnvpair_json/json_common mode=0555
file path=opt/util-tests/tests/printf_test mode=0555
file path=opt/util-tests/tests/xargs_test mode=0555
license lic_CDDL license=lic_CDDL
depend fmri=system/test/testrunner type=require
```

****

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>

****

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

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

[DEFAULT]
pre =
verbose = False
quiet = False
timeout = 60
post =
outputdir = /var/tmp/test_results

[/opt/util-tests/tests/printf_test]
[/opt/util-tests/tests/allowed-ips]

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

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

****

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
```

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

#
# Copyright (c) 2014 Joyent, Inc. All rights reserved.
#

include $(SRC)/Makefile.master

ROOTOPTPKG = $(ROOT)/opt/util-tests
TESTDIR = $(ROOTOPTPKG)/tests/libnvpair_json
ROOTBINDIR = $(ROOTOPTPKG)/bin

PROG = print_json

SCRIPTS = \
        json_00_blank \
        json_01_boolean \
        json_02_numbers \
        json_03_empty_arrays \
        json_04_number_arrays \
        json_05_strings \
        json_06_nested \
        json_07_nested_arrays \
        json_common

include $(SRC)/cmd/Makefile.cmd
include $(SRC)/test/Makefile.com

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

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

LDLIBS += -lnvpair

LINTFLAGS += -erroff=E_FUNC_ARG_UNUSED

all: $(PROG)

$(PROG): $(OBJS)
        $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
        $(POST_PROCESS)

install: all $(CMDS)

lint: lint_SRCS

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

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

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

$(ROOTBINDIR):
        $(INS.dir)

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

$(TESTDIR):
        $(INS.dir)

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

```
**********************************************************
     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>
**********************************************************
```

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

#
# Copyright (c) 2014, Joyent, Inc.
#

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

BASELINE="$(cat <<EOF
{\
}
EOF)"

OUTPUT="$(${DIR}/../../bin/print_json <<'EOF'
/*
 * Emit a blank object.
 */
EOF)"

complete
```

```
**********************************************************
     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>
**********************************************************
```

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

#
# Copyright (c) 2014, Joyent, Inc.
#

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

BASELINE="$(cat <<EOF
{\
"bool0":true,\
"a fact":true,\
"a fiction":false,\
"1":true,\
" ":true\
}
EOF)"

OUTPUT="$(${DIR}/../../bin/print_json <<'EOF'
/*
 * add_boolean calls nvlist_add_boolean(), which the JSON formatter
 * will emit as a true-valued boolean.
 */
add_boolean "bool0";
add_boolean_value "a fact" "true";
add_boolean_value "a fiction" "false";
add_boolean "1";

/*
 * Test a key with a whitespace-only name:
 */
add_boolean " ";
EOF)"

complete
```

```
**********************************************************
    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>
**********************************************************
```

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

#
# Copyright (c) 2014, Joyent, Inc.
#

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

BASELINE="$(cat <<EOF
{\
"byte":255,\
"uint8_0":0,\
"uint8_100":100,\
"uint8_255":255,\
"uint16":12345,\
"uint32":23423423,\
"uint64":19850709000000,\
"int16_small":-32768,\
"int8_neg":-128,\
"int8_pos":127,\
"int16_big":32767,\
"int32":-1270000,\
"int64":-12700000000001,\
"double_small":0.000023,\
"double_big":2342300000000.000000\
}
EOF)"

OUTPUT="$(${DIR}/../../bin/print_json <<'EOF'
add_byte "byte" "0";
add_byte "byte" "255";

add_uint8 "uint8_0" "0";
add_uint8 "uint8_100" "100";
add_uint8 "uint8_255" "255";

add_uint16 "uint16" "12345";
add_uint32 "uint32" "23423423";
add_uint64 "uint64" "19850709000000";

add_int16 "int16_small" "-32768";
add_int8 "int8_neg" "-128";
add_int8 "int8_pos" "127";
add_int16 "int16_big" "32767";

add_int32 "int32" "-1270000";
add_int64 "int64" "-12700000000001";
```

```
add_double "double_small" "0.000023423";
add_double "double_big" "0.000023423e17";
EOF)"

complete
```

```
**********************************************************
    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>
**********************************************************
#!/bin/ksh
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2014, Joyent, Inc.
#

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

BASELINE="$(cat <<EOF
{\
"boolean_array":[],\
"byte_array":[],\
"uint8_array":[],\
"uint16_array":[],\
"uint32_array":[],\
"uint64_array":[],\
"int8_array":[],\
"int16_array":[],\
"int32_array":[],\
"int64_array":[],\
"string_array":[],\
"object_array":[{}]\
}
EOF)"

OUTPUT="$(${DIR}/../../bin/print_json <<'EOF'
add_boolean_array "boolean_array";

add_byte_array "byte_array";

add_uint8_array "uint8_array";
add_uint16_array "uint16_array";
add_uint32_array "uint32_array";
add_uint64_array "uint64_array";

add_int8_array "int8_array";
add_int16_array "int16_array";
add_int32_array "int32_array";
add_int64_array "int64_array";

add_string_array "string_array";

/*
 * The testing DSL does not presently support the generation of a completely
 * empty object array.  Thus, the following directive will produce an array
 * with a single keyless object:
 */
add_object_array "object_array";
```

```
end;
EOF)"

complete
```

****
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>
****

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

#
# Copyright (c) 2014, Joyent, Inc.
#

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

BASELINE="$(cat <<EOF
{\
"byte_array":[0,1,2,10,15,100,103,127,128,254,255],\
"uint8_array":[128,254,255,10,15,100,103,127,0,1,2],\
"uint16_array":[0,1000,2000,3210,4321,5432,10000,15000,16384,\
17992,35012,65535,0],\
"uint32_array":[0,4294967295,4026531855,1,2,1000,501],\
"uint64_array":[19850907,0,18446744073709551615],\
"int8_array":[39,39,39,39,39,39,39,-128,-127,0,127],\
"int16_array":[7532,-32768,0,32767,0,-32768,100],\
"int32_array":[-2147483648,0,32767,-32768,2147483647],\
"int64_array":[0,0,9223372036854775807,1,1,1,-9223372036854775808,0]\
}
EOF)"

OUTPUT="$(${DIR}/../../bin/print_json <<'EOF'
add_byte_array "byte_array"
  "0" "1" "2" "10" "15" "100" "103" "127" "128" "254" "255";

add_uint8_array "uint8_array"
  "128" "254" "255" "10" "15" "100" "103" "127" "0" "1" "2";

add_uint16_array "uint16_array"
  "0" "1000" "2000" "3210" "4321" "5432" "10000" "15000" "16384"
  "17992" "35012" "65535" "0";

add_uint32_array "uint32_array"
  "0" "4294967295" "4026531855" "1" "2" "1000" "501";

add_uint64_array "uint64_array"
  "19850907" "0" "18446744073709551615";

add_int8_array "int8_array"
  "39" "39" "39" "39" "39" "39" "39" "-128" "-127" "0" "127";

add_int16_array "int16_array"
  "7532" "-32768" "0" "32767" "0" "-32768" "100";

add_int32_array "int32_array"
  "-2147483648" "0" "32767" "-32768" "2147483647";
```

```
add_int64_array "int64_array"
  "0" "0" "9223372036854775807" "1" "1" "1" "-9223372036854775808" "0";
EOF)"

complete
```

```
**********************************************************
    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>
**********************************************************
#!/bin/ksh
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2014, Joyent, Inc.
#

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

#
# This test checks UTF-8 parsing behaviour
#
export LC_ALL="en_US.UTF-8"
export LANG="${LANG}"

BASELINE="$(cat <<EOF
{\
"blank":"",\
"":"blank key",\
" ":"whitespace key",\
"\ttab\t":"tab key",\
"escapes":"escape \u001b newline \n cr \r backslash \\\\ quote \"",\
"escape array":[\
"escape \u001b",\
"alarm \u0007",\
"backspace \b",\
"formfeed \f",\
"newline \n",\
"return \r",\
"tab \t",\
"vertical tab \u000b",\
"black circle (UTF-8) \u25cf"\
]\
}
EOF)"

OUTPUT="$(${DIR}/../../bin/print_json <<'EOF'
add_string "blank" "";
add_string "" "blank key";
add_string " " "whitespace key";
add_string "	tab	" "tab key";
add_string "escapes" "escape \x1b newline \n cr \r backslash \\ quote \"";
add_string_array "escape array"
    "escape \x1b"
    "alarm \a"
    "backspace \b"
    "formfeed \f"
    "newline \n"
    "return \r"
```

```
    "tab \t"
    "vertical tab \v"
    "black circle (UTF-8) \xe2\x97\x8f";
EOF)"

complete
```

****
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>
****

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

#
# Copyright (c) 2014, Joyent, Inc.
#

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

BASELINE="$(cat <<EOF
{\
"a":{},\
"b":{\
"name":"Roger","age":35\
},\
"c":{\
"d":{\
"name":"Stephen","age":27},\
"e":{\
"name":"Roberta","age":43,"pet":{\
"name":"Mister Bumberscratch",\
"species":"cat",\
"alive":true,\
"available_legs":[1,2,3,4]\
}\
}\
}\
}
EOF)"

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

add_object "b";
        add_string "name" "Roger";
        add_uint16 "age" "35";
end;

add_object "c";
        add_object "d";
                add_string "name" "Stephen";
                add_uint16 "age" "27";
        end;
        add_object "e";
                add_string "name" "Roberta";
                add_uint16 "age" "43";
                add_object "pet";
                        add_string "name" "Mister Bumberscratch";
```

```
                        add_string "species" "cat";
                        add_boolean_value "alive" "true";
                        add_uint8_array "available_legs" "1" "2" "3" "4";
                end;
        end;
end;
EOF)"

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>
****

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

#
# Copyright (c) 2014, Joyent, Inc.
#

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

BASELINE="$(cat <<EOF
{\
"event_store":{\
"name":"Occurences",\
"events":[\
{"time":489715200,"desc":"inception"},\
{"time":1057708800,"desc":"maturation"},\
{"time":1344816000,"desc":"migration"},\
{"time":1405296000,"desc":"integration"},\
{}\
]\
},\
"first level":[\
{"second_level_0":[{\
"sl0_a":true,\
"sl0_b":"aaaa"\
},\
{"x":1234}\
],\
"second_level_1":[{}],\
"second_level_2":[\
{"alpha":"a"},\
{"beta":"b"},\
{"gamma":"c"},\
{"delta":"d"},\
{"order":["a","b","c","d"]}\
]\
}\
]\
}
EOF)"

OUTPUT="$(${DIR}/../../bin/print_json <<'EOF'
add_object "event_store";
        add_string "name" "Occurences";
        add_object_array "events";
                add_uint32 "time" "489715200";
                add_string "desc" "inception";
                next;
```

```
                add_uint32 "time" "1057708800";
                add_string "desc" "maturation";
                next;

                add_uint32 "time" "1344816000";
                add_string "desc" "migration";
                next;

                add_uint32 "time" "1405296000";
                add_string "desc" "integration";
                next;
        end;
end;
add_object_array "first level";
        add_object_array "second_level_0";
                add_boolean "sl0_a";
                add_string "sl0_b" "aaaa";
                next;
                add_int32 "x" "1234";
        end;
        add_object_array "second_level_1";
        end;
        add_object_array "second_level_2";
                add_string "alpha" "a";
                next;
                add_string "beta" "b";
                next;
                add_string "gamma" "c";
                next;
                add_string "delta" "d";
                next;
                add_string_array "order" "a" "b" "c" "d";
        end;
end;
EOF)"

complete
```

****
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>
****

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

#
# Copyright (c) 2014, Joyent, Inc.
#

function complete {
  if [[ "${PRINT_OUTPUT}" ]]; then
    printf "%s\n" "${OUTPUT}"
    exit 0
  elif [[ "${OUTPUT}" == "${BASELINE}" ]]; then
    printf "TEST PASS: %s\n" "$(basename $0)"
    exit 0
  else
    printf "TEST FAIL: %s\n" "$(basename $0)"
    printf "EXPECTED: %s\n" "${BASELINE}"
    printf "ACTUAL:   %s\n" "${OUTPUT}"
    exit 1
  fi
}
```

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

/*
 * Copyright 2014 Joyent, Inc.
 */

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

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include <locale.h>

#include <libnvpair.h>

#define MAX_ARGS        100
#define CMD_NAME_LEN    50

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

typedef struct list_wrap {
        nvlist_t *lw_nvl[MAX_ARGS];
        char *lw_name;
        int lw_pos;
        boolean_t lw_array;
        struct list_wrap *lw_next;
} list_wrap_t;

int
list_wrap_depth(list_wrap_t *lw)
{
        int d = 0;
```

```
        while (lw != NULL) {
                d++;
                lw = lw->lw_next;
        }

        return (d);
}

list_wrap_t *
list_wrap_alloc(list_wrap_t *next)
{
        list_wrap_t *out = calloc(1, sizeof (list_wrap_t));

        if (out == NULL)
                abort();

        out->lw_next = next;

        return (out);
}

list_wrap_t *
list_wrap_pop_and_free(list_wrap_t *lw)
{
        list_wrap_t *next = lw->lw_next;

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

        return (next);
}

/*
 * Generic integer and floating point parsing routines:
 */

int
parse_int(char *in, int64_t *val, int64_t min, int64_t max)
{
        int64_t t;
        char *end = NULL;

        errno = 0;
        t = strtoll(in, &end, 10);
        if (errno != 0 || end == in || *end != '\0') {
                if (errno == ERANGE) {
                        (void) fprintf(stderr, "ERROR: integer %s not in "
                            "range [%lld,%lld]\n", in, min, max);
                        return (-1);
                }
                (void) fprintf(stderr, "ERROR: could not parse \"%s\" as "
                    "signed integer (%s)\n", in, strerror(errno));
                return (-1);
        }

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

        *val = t;
        return (0);
}
```

```
int
parse_uint(char *in, uint64_t *val, uint64_t min, uint64_t max)
{
        uint64_t t;
        char *end = NULL;

        errno = 0;
        t = strtoull(in, &end, 10);
        if (errno != 0 || end == in || *end != '\0') {
                if (errno == ERANGE) {
                        (void) fprintf(stderr, "ERROR: integer %s not in "
                            "range [%llu,%llu]\n", in, min, max);
                        return (-1);
                }
                (void) fprintf(stderr, "ERROR: could not parse \"%s\" as "
                    "unsigned integer (%s)\n", in, strerror(errno));
                return (-1);
        }

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

        *val = t;
        return (0);
}

int
parse_double(char *in, double *val)
{
        double t;
        char *end = NULL;

        errno = 0;
        t = strtod(in, &end);
        if (errno != 0 || end == in || *end != '\0') {
                (void) fprintf(stderr, "ERROR: could not parse \"%s\" as "
                    "double\n", in);
                return (-1);
        }

        *val = t;
        return (0);
}

/*
 * Command-specific handlers for directives specified in the DSL input:
 */

typedef int (*command_handler_t)(list_wrap_t **, boolean_t, int,
    char **);

static int
ch_add_string(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];

        if (array) {
                if (nvlist_add_string_array(nvl, argv[0], &argv[1],
                    argc - 1) != 0) {
                        (void) fprintf(stderr, "fail at "
                            "nvlist_add_string_array\n");
                        return (-1);
                }
```

```
        } else {
                if (nvlist_add_string(nvl, argv[0], argv[1]) != 0) {
                        (void) fprintf(stderr, "fail at nvlist_add_string\n");
                        return (-1);
                }
        }

        return (0);
}

static int
ch_add_boolean(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];

        if (array)
                abort();

        if (nvlist_add_boolean(nvl, argv[0]) != 0) {
                (void) fprintf(stderr, "fail at nvlist_add_boolean\n");
                return (-1);
        }
        return (0);
}

static int
ch_add_boolean_value(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        int i;
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
        boolean_t arrval[MAX_ARGS];

        for (i = 1; i < argc; i++) {
                if (strcmp(argv[i], "true") == 0) {
                        arrval[i - 1] = B_TRUE;
                } else if (strcmp(argv[i], "false") == 0) {
                        arrval[i - 1] = B_FALSE;
                } else {
                        (void) fprintf(stderr, "invalid boolean value: %s\n",
                            argv[i]);
                        return (-1);
                }
        }

        if (array) {
                if (nvlist_add_boolean_array(nvl, argv[0], arrval,
                    argc - 1) != 0) {
                        (void) fprintf(stderr, "fail at "
                            "nvlist_add_boolean_array\n");
                        return (-1);
                }
        } else {
                if (nvlist_add_boolean_value(nvl, argv[0], arrval[0]) != 0) {
                        (void) fprintf(stderr, "fail at "
                            "nvlist_add_boolean_value\n");
                        return (-1);
                }
        }

        return (0);
}


/*
 * The confluence of a strongly typed C API for libnvpair(3LIB) and the
 * combinatorial explosion of both sizes and signedness is unfortunate.  Rather
 * than reproduce the same code over and over, this macro parses an integer,
 * checks applicable bounds based on size and signedness, and stores the value
 * (or array of values).
 */
#define DO_CMD_NUMBER(typ, nam, min, max, ptyp, func)                   \
        ptyp val;                                                       \
        typ ## _t arrval[MAX_ARGS];                                     \
        int i;                                                          \
        for (i = 1; i < argc; i++) {                                    \
                if (func(argv[i], &val, min, max) != 0) {               \
                        return (-1);                                    \
                }                                                       \
                arrval[i - 1] = (typ ## _t) val;                        \
        }                                                               \
        if (array) {                                                    \
                if (nvlist_add_ ## nam ## _array(nvl, argv[0],          \
                    arrval, argc - 1) != 0) {                           \
                        (void) fprintf(stderr, "fail at "               \
                            "nvlist_add_" #nam "_array\n");             \
                        return (-1);                                    \
                }                                                       \
        } else {                                                        \
                if (nvlist_add_ ## nam(nvl, argv[0],                    \
                    arrval[0]) == -1) {                                 \
                        (void) fprintf(stderr, "fail at "               \
                            "nvlist_add_" #nam "\n");                   \
                        return (-1);                                    \
                }                                                       \
        }                                                               \
        return (0);

static int
ch_add_byte(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];

        DO_CMD_NUMBER(uchar, byte, 0, UCHAR_MAX, uint64_t, parse_uint)
}

static int
ch_add_int8(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];

        DO_CMD_NUMBER(int8, int8, INT8_MIN, INT8_MAX, int64_t, parse_int)
}

static int
ch_add_uint8(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];

        DO_CMD_NUMBER(uint8, uint8, 0, UINT8_MAX, uint64_t, parse_uint)
}

static int
ch_add_int16(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];

        DO_CMD_NUMBER(int16, int16, INT16_MIN, INT16_MAX, int64_t, parse_int)
}

static int
ch_add_uint16(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
```

```
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];

        DO_CMD_NUMBER(uint16, uint16, 0, UINT16_MAX, uint64_t, parse_uint)
}

static int
ch_add_int32(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];

        DO_CMD_NUMBER(int32, int32, INT32_MIN, INT32_MAX, int64_t, parse_int)
}

static int
ch_add_uint32(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];

        DO_CMD_NUMBER(uint32, uint32, 0, UINT32_MAX, uint64_t, parse_uint)
}

static int
ch_add_int64(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];

        DO_CMD_NUMBER(int64, int64, INT64_MIN, INT64_MAX, int64_t, parse_int)
}

static int
ch_add_uint64(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];

        DO_CMD_NUMBER(uint64, uint64, 0, UINT64_MAX, uint64_t, parse_uint)
}

static int
ch_add_double(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *nvl = (*lw)->lw_nvl[(*lw)->lw_pos];
        double val;

        if (array)
                abort();

        if (parse_double(argv[1], &val) != 0) {
                return (-1);
        }

        if (nvlist_add_double(nvl, argv[0], val) != 0) {
                (void) fprintf(stderr, "fail at nvlist_add_double_value\n");
                return (-1);
        }

        return (0);
}

static int
ch_end(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
        nvlist_t *parent;
        char *name;

        if (list_wrap_depth(*lw) < 2) {
                (void) fprintf(stderr, "ERROR: not nested, cannot end.\n");
```

```
		return (-1);
	}

	parent = (*lw)->lw_next->lw_nvl[(*lw)->lw_next->lw_pos];
	name = (*lw)->lw_name;
	if ((*lw)->lw_array) {
		/*
		 * This was an array of objects.
		 */
		nvlist_t **children = (*lw)->lw_nvl;
		int nelems = (*lw)->lw_pos + 1;

		if (nvlist_add_nvlist_array(parent, name, children,
		    nelems) != 0) {
			(void) fprintf(stderr, "fail at "
			    "nvlist_add_nvlist_array\n");
			return (-1);
		}
	} else {
		/*
		 * This was a single object.
		 */
		nvlist_t *child = (*lw)->lw_nvl[0];

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

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

	*lw = list_wrap_pop_and_free(*lw);

	return (0);
}

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

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

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

	return (0);
}

static int
ch_add_object(list_wrap_t **lw, boolean_t array, int argc, char **argv)
{
	*lw = list_wrap_alloc(*lw);
```

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

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

	return (0);
}

typedef struct command {
	char cmd_name[CMD_NAME_LEN];
	command_handler_t cmd_func;
	int cmd_min_args;
	int cmd_max_args;
	boolean_t cmd_array_mode;
} command_t;

/*
 * These are the commands we support in the testing DSL, and their
 * handling functions:
 */
command_t command_handlers[] = {
	{ "add_boolean", ch_add_boolean, 1, 1, B_FALSE },
	{ "add_boolean_value", ch_add_boolean_value, 2, 2, B_FALSE },
	{ "add_byte", ch_add_byte, 2, 2, B_FALSE },
	{ "add_int8", ch_add_int8, 2, 2, B_FALSE },
	{ "add_uint8", ch_add_uint8, 2, 2, B_FALSE },
	{ "add_int16", ch_add_int16, 2, 2, B_FALSE },
	{ "add_uint16", ch_add_uint16, 2, 2, B_FALSE },
	{ "add_int32", ch_add_int32, 2, 2, B_FALSE },
	{ "add_uint32", ch_add_uint32, 2, 2, B_FALSE },
	{ "add_int64", ch_add_int64, 2, 2, B_FALSE },
	{ "add_uint64", ch_add_uint64, 2, 2, B_FALSE },
	{ "add_double", ch_add_double, 2, 2, B_FALSE },
	{ "add_string", ch_add_string, 2, 2, B_FALSE },
	{ "add_object", ch_add_object, 1, 1, B_FALSE },
	{ "add_boolean_array", ch_add_boolean_value, 1, MAX_ARGS, B_TRUE },
	{ "add_byte_array", ch_add_byte, 1, MAX_ARGS, B_TRUE },
	{ "add_int8_array", ch_add_int8, 1, MAX_ARGS, B_TRUE },
	{ "add_uint8_array", ch_add_uint8, 1, MAX_ARGS, B_TRUE },
	{ "add_int16_array", ch_add_int16, 1, MAX_ARGS, B_TRUE },
	{ "add_uint16_array", ch_add_uint16, 1, MAX_ARGS, B_TRUE },
	{ "add_int32_array", ch_add_int32, 1, MAX_ARGS, B_TRUE },
	{ "add_uint32_array", ch_add_uint32, 1, MAX_ARGS, B_TRUE },
	{ "add_int64_array", ch_add_int64, 1, MAX_ARGS, B_TRUE },
	{ "add_uint64_array", ch_add_uint64, 1, MAX_ARGS, B_TRUE },
	{ "add_string_array", ch_add_string, 1, MAX_ARGS, B_TRUE },
	{ "add_object_array", ch_add_object, 1, 1, B_TRUE },
	{ "end", ch_end, 0, 0, B_FALSE },
	{ "next", ch_next, 0, 0, B_FALSE },
	{ 0 }
};

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

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

```
                        continue;

                if (argc > command_handlers[ch].cmd_max_args ||
                    argc < command_handlers[ch].cmd_min_args) {

                        (void) fprintf(stderr, "ERROR: command \"%s\""
                            " expects between %d and %d arguments,"
                            " but %d were provided.\n", command,
                            command_handlers[ch].cmd_min_args,
                            command_handlers[ch].cmd_max_args,
                            argc);

                        return (-1);
                }

                return (command_handlers[ch].cmd_func(lw,
                    command_handlers[ch].cmd_array_mode, argc, argv));
        }

        (void) fprintf(stderr, "ERROR: invalid command: \"%s\"\n", command);

        return (-1);
}

/*
 * The primary state machine for parsing the input DSL is implemented in
 * this function:
 */

typedef enum state {
        STATE_REST = 1,
        STATE_COMMAND,
        STATE_ARG_FIND,
        STATE_ARG,
        STATE_ARG_ESCAPE,
        STATE_ARG_ESCAPE_HEX,
        STATE_C_COMMENT_0,
        STATE_C_COMMENT_1,
        STATE_C_COMMENT_2
} state_t;

int
parse(FILE *in, list_wrap_t **lw)
{
        char b[8192];
        int bp;
        state_t st = STATE_REST;
        int argc = 0;
        char *argv[MAX_ARGS];
        int line = 1;
        char hex[3];
        int nhex = 0;

        b[0] = '\0';
        bp = 0;

        for (;;) {
                int c = fgetc(in);

                /*
                 * Signal an error if the file ends part way through a
                 * construct:
                 */
                if (st != STATE_REST && c == EOF) {
                        (void) fprintf(stderr, "ERROR: unexpected end of "
                            "file\n");
```

```
                        return (-1);
                } else if (c == EOF) {
                        return (0);
                }

                if (c == '\n')
                        line++;

                switch (st) {
                case STATE_REST:
                        if (isalpha(c) || c == '_') {
                                argc = 0;
                                bp = 0;
                                b[bp++] = c;
                                b[bp] = '\0';
                                st = STATE_COMMAND;
                                continue;
                        } else if (c == ' ' || c == '\t' || c == '\n') {
                                /*
                                 * Ignore whitespace.
                                 */
                                continue;
                        } else if (c == '/') {
                                st = STATE_C_COMMENT_0;
                                continue;
                        } else {
                                goto unexpected;
                        }

                case STATE_C_COMMENT_0:
                        if (c != '*') {
                                goto unexpected;
                        }
                        st = STATE_C_COMMENT_1;
                        continue;

                case STATE_C_COMMENT_1:
                        if (c == '*') {
                                st = STATE_C_COMMENT_2;
                        }
                        continue;

                case STATE_C_COMMENT_2:
                        if (c == '/') {
                                st = STATE_REST;
                        } else if (c != '*') {
                                st = STATE_C_COMMENT_1;
                        }
                        continue;

                case STATE_COMMAND:
                        if (isalnum(c) || c == '_') {
                                b[bp++] = c;
                                b[bp] = '\0';
                                st = STATE_COMMAND;

                                continue;

                        } else if (isspace(c)) {
                                /*
                                 * Start collecting arguments into 'b'
                                 * after the command.
                                 */
                                st = STATE_ARG_FIND;
                                bp++;
                                continue;
                        } else if (c == ';') {
                                /*
                                 * This line was _just_ a command,
                                 * so break out and process now:
                                 */
                                goto execute;
                        } else {
                                goto unexpected;
                        }

                case STATE_ARG_FIND:
                        if (isspace(c)) {
                                /*
                                 * Whitespace, ignore.
                                 */
                                continue;

                        } else if (c == ';') {
                                /*
                                 * Break out to process command.
                                 */
                                goto execute;

                        } else if (c == '"') {
                                st = STATE_ARG;

                                argv[argc] = &b[++bp];
                                b[bp] = '\0';

                                continue;
                        } else {
                                goto unexpected;
                        }

                case STATE_ARG:
                        if (c == '"') {
                                if (argc++ >= MAX_ARGS) {
                                        (void) fprintf(stderr, "ERROR: too "
                                            "many args\n");
                                        return (-1);
                                }
                                st = STATE_ARG_FIND;
                                continue;
                        } else if (c == '\n') {
                                (void) fprintf(stderr, "ERROR: line not "
                                    "finished\n");
                                return (-1);
                        } else if (c == '\\') {
                                st = STATE_ARG_ESCAPE;
                                continue;
                        } else {
                                b[bp++] = c;
                                b[bp] = '\0';
                                continue;
                        }

                case STATE_ARG_ESCAPE:
                        if (c == 'a') {
                                c = '\a';
                        } else if (c == 'b') {
                                c = '\b';
                        } else if (c == 'f') {
                                c = '\f';
                        } else if (c == 'n') {
                                c = '\n';
```

```
                        } else if (c == 'r') {
                                c = '\r';
                        } else if (c == 't') {
                                c = '\t';
                        } else if (c == 'v') {
                                c = '\v';
                        } else if (c == 'x') {
                                st = STATE_ARG_ESCAPE_HEX;
                                hex[0] = hex[1] = hex[2] = '\0';
                                nhex = 0;
                                continue;
                        } else if (c != '\\' && c != '"') {
                                goto unexpected;
                        }

                        b[bp++] = c;
                        b[bp] = '\0';
                        st = STATE_ARG;
                        continue;

                case STATE_ARG_ESCAPE_HEX:
                        if (!isxdigit(c)) {
                                goto unexpected;
                        }
                        hex[nhex] = c;
                        if (nhex++ >= 1) {
                                /*
                                 * The hex escape pair is complete, parse
                                 * the integer and insert it as a character:
                                 */
                                int x;
                                errno = 0;
                                if ((x = strtol(hex, NULL, 16)) == 0 ||
                                    errno != 0) {
                                        goto unexpected;
                                }
                                b[bp++] = (char)x;
                                b[bp] = '\0';
                                st = STATE_ARG;
                        }
                        continue;
                }

                /*
                 * We do not ever expect to break out of the switch block
                 * above.  If we do, it's a programmer error.
                 */
                abort();

execute:
                if (command_call(lw, b, argc, argv) == -1)
                        return (-1);

                st = STATE_REST;
                continue;

unexpected:
                (void) fprintf(stderr, "ERROR: (line %d) unexpected "
                    "character: %c\n", line, c);
                return (-1);
        }
}

/*
 * Entry point:
 */
```

```
int
main(int argc, char **argv)
{
        int rc = EXIT_FAILURE;
        list_wrap_t *lw;

        /*
         * Be locale-aware.  The JSON output functions will process multibyte
         * characters in the current locale, and emit a correct JSON encoding
         * for unprintable characters.
         */
        if (setlocale(LC_ALL, "") == NULL) {
                (void) fprintf(stderr, "Could not set locale: %s\n",
                    strerror(errno));
                goto out;
        }

        lw = list_wrap_alloc(NULL);

        if (nvlist_alloc(&lw->lw_nvl[0], NV_UNIQUE_NAME, 0) != 0)
                goto out;

        /*
         * Generate the list from the commands passed to us on stdin:
         */
        if (parse(stdin, &lw) != 0)
                goto out;

        /*
         * Print the resultant list, and a terminating newline:
         */
        if (nvlist_print_json(stdout, lw->lw_nvl[0]) != 0 ||
            fprintf(stdout, "\n") < 0)
                goto out;

        rc = EXIT_SUCCESS;

out:
        (void) list_wrap_pop_and_free(lw);

        return (rc);
}
```