```
**********************************************************
   79277 Tue Jun 12 09:01:23 2012
new/usr/src/cmd/mdb/common/mdb/mdb_cmds.c
2574 mdb needs ::printf
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Approved by: ?
**********************************************************
_____unchanged_portion_omitted_

/*
 * Table of built-in dcmds associated with the root 'mdb' module.  Future
 * expansion of this program should be done here, or through the external
 * loadable module interface.
 */
const mdb_dcmd_t mdb_dcmd_builtins[] = {

        /*
         * dcmds common to both mdb and kmdb
         */
        { ">", "variable-name", "assign variable", cmd_assign_variable },
        { "/", "fmt-list", "format data from virtual as", cmd_print_core },
        { "\\", "fmt-list", "format data from physical as", cmd_print_phys },
        { "@", "fmt-list", "format data from physical as", cmd_print_phys },
        { "=", "fmt-list", "format immediate value", cmd_print_value },
        { "$<", "macro-name", "replace input with macro",
            cmd_exec_file, srcexec_file_help },
        { "$<<", "macro-name", "source macro",
            cmd_src_file, srcexec_file_help},
        { "$%", NULL, NULL, cmd_quit },
        { "$?", NULL, "print status and registers", cmd_notsup },
        { "$a", NULL, NULL, cmd_algol },
        { "$b", "[-av]", "list traced software events",
            cmd_events, events_help },
        { "$c", "?[cnt]", "print stack backtrace", cmd_notsup },
        { "$C", "?[cnt]", "print stack backtrace", cmd_notsup },
        { "$d", NULL, "get/set default output radix", cmd_radix },
        { "$D", "?[mode,...]", NULL, cmd_dbmode },
        { "$e", NULL, "print listing of global symbols", cmd_globals },
        { "$f", NULL, "print listing of source files", cmd_files },
        { "$m", "?[name]", "print address space mappings", cmd_mappings },
        { "$M", NULL, "list macro aliases", cmd_macalias_list },
        { "$P", "[prompt]", "set debugger prompt string", cmd_prompt },
        { "$q", NULL, "quit debugger", cmd_quit },
        { "$Q", NULL, "quit debugger", cmd_quit },
        { "$r", NULL, "print general-purpose registers", cmd_notsup },
        { "$s", NULL, "get/set symbol matching distance", cmd_symdist },
        { "$v", NULL, "print non-zero variables", cmd_nzvars },
        { "$V", "[mode]", "get/set disassembly mode", cmd_dismode },
        { "$w", NULL, "get/set output page width", cmd_pgwidth },
        { "$W", NULL, "re-open target in write mode", cmd_reopen },
        { ":a", ":[cmd...]", "set read access watchpoint", cmd_oldwpr },
        { ":b", ":[cmd...]", "breakpoint at the specified address", cmd_oldbp },
        { ":d", "?[id|all]", "delete traced software events", cmd_delete },
        { ":p", ":[cmd...]", "set execute access watchpoint", cmd_oldwpx },
        { ":S", NULL, NULL, cmd_step },
        { ":w", ":[cmd...]", "set write access watchpoint", cmd_oldwpw },
        { ":z", NULL, "delete all traced software events", cmd_zapall },
        { "array", ":[type count] [variable]", "print each array element's "
            "address", cmd_array },
        { "bp", "?[+/-dDestT] [-c cmd] [-n count] sym ...", "breakpoint at the "
            "specified addresses or symbols", cmd_bp, bp_help },
        { "dcmds", NULL, "list available debugger commands", cmd_dcmds },
        { "delete", "?[id|all]", "delete traced software events", cmd_delete },
        { "dis", "?[-abfw] [-n cnt] [addr]", "disassemble near addr", cmd_dis },
```



```
        { "disasms", NULL, "list available disassemblers", cmd_disasms },
        { "dismode", "[mode]", "get/set disassembly mode", cmd_dismode },
        { "dmods", "[-l] [mod]", "list loaded debugger modules", cmd_dmods },
        { "dump", "?[-eqrstu] [-f|-p] [-g bytes] [-w paragraphs]",
            "dump memory from specified address", cmd_dump, dump_help },
        { "echo", "args ...", "echo arguments", cmd_echo },
        { "enum", "?[-ex] enum [name]", "print an enumeration", cmd_enum,
            enum_help },
        { "eval", "command", "evaluate the specified command", cmd_eval },
        { "events", "[-av]", "list traced software events",
            cmd_events, events_help },
        { "evset", "?[+/-dDestT] [-c cmd] [-n count] id ...",
            "set software event specifier attributes", cmd_evset, evset_help },
        { "files", "[object]", "print listing of source files", cmd_files },
#ifdef __sparc
        { "findsym", "?[-g] [symbol|addr ...]", "search for symbol references "
            "in all known functions", cmd_findsym, NULL },
#endif
        { "formats", NULL, "list format specifiers", cmd_formats },
        { "grep", "?expr", "print dot if expression is true", cmd_grep },
        { "head", "-num|-n num", "limit number of elements in pipe", cmd_head,
            head_help },
        { "help", "[cmd]", "list commands/command help", cmd_help },
        { "list", "?type member [variable]",
            "walk list using member as link pointer", cmd_list, NULL,
            mdb_tab_complete_mt },
        { "map", "?expr", "print dot after evaluating expression", cmd_map },
        { "mappings", "?[name]", "print address space mappings", cmd_mappings },
        { "nm", "?[-DPdghnopuvx] [-f format] [-t types] [object]",
            "print symbols", cmd_nm, nm_help },
        { "nmadd", ":[-fo] [-e end] [-s size] name",
            "add name to private symbol table", cmd_nmadd, nmadd_help },
        { "nmdel", "name", "remove name from private symbol table", cmd_nmdel },
        { "obey", NULL, NULL, cmd_obey },
        { "objects", "[-v]", "print load objects information", cmd_objects },
        { "offsetof", "type member", "print the offset of a given struct "
            "or union member", cmd_offsetof, NULL, mdb_tab_complete_mt },
        { "print", "?[-aCdhiLptx] [-c lim] [-l lim] [type] [member|offset ...]",
            "print the contents of a data structure", cmd_print, print_help,
            cmd_print_tab },
        { "printf", "?format type member ...", "print and format the "
            "member(s) of a data structure", cmd_printf, printf_help },
        { "regs", NULL, "print general purpose registers", cmd_notsup },
        { "set", "[-wF] [+/-o opt] [-s dist] [-I path] [-L path] [-P prompt]",
            "get/set debugger properties", cmd_set },
        { "showrev", "[-pv]", "print version information", cmd_showrev },
        { "sizeof", "type", "print the size of a type", cmd_sizeof, NULL,
            cmd_sizeof_tab },
        { "stack", "?[cnt]", "print stack backtrace", cmd_notsup },
        { "stackregs", "?", "print stack backtrace and registers",
            cmd_notsup },
        { "status", NULL, "print summary of current target", cmd_notsup },
        { "term", NULL, "display current terminal type", cmd_term },
        { "typeset", "[+/-t] var ...", "set variable attributes", cmd_typeset },
        { "unset", "[name ...]", "unset variables", cmd_unset },
        { "vars", "[-npt]", "print listing of variables", cmd_vars },
        { "version", NULL, "print debugger version string", cmd_version },
        { "vtop", ":[-a as]", "print physical mapping of virtual address",
            cmd_vtop },
        { "walk", "?name [variable]", "walk data structure", cmd_walk, NULL,
            cmd_walk_tab },
        { "walkers", NULL, "list available walkers", cmd_walkers },
        { "whatis", ":[-aikqv]", "given an address, return information",
            cmd_whatis, whatis_help },
        { "whence", "[-v] name ...", "show source of walk or dcmd", cmd_which },
        { "which", "[-v] name ...", "show source of walk or dcmd", cmd_which },
```

```
2993         { "xdata", NULL, "print list of external data buffers", cmd_xdata },

2995 #ifdef _KMDB
2996         /*
2997          * dcmds specific to kmdb, or which have kmdb-specific arguments
2998          */
2999         { "?", "fmt-list", "format data from virtual as", cmd_print_core },
3000         { ":c", NULL, "continue target execution", cmd_cont },
3001         { ":e", NULL, "step target over next instruction", cmd_next },
3002         { ":s", NULL, "single-step target to next instruction", cmd_step },
3003         { ":u", NULL, "step target out of current function", cmd_step_out },
3004         { "cont", NULL, "continue target execution", cmd_cont },
3005         { "load", "[-sd] module", "load debugger module", cmd_load, load_help },
3006         { "next", NULL, "step target over next instruction", cmd_next },
3007         { "quit", "[-u]", "quit debugger", cmd_quit, quit_help },
3008         { "step", "[ over | out ]",
3009             "single-step target to next instruction", cmd_step },
3010         { "unload", "[-d] module", "unload debugger module", cmd_unload,
3011             unload_help },
3012         { "wp", ":[+/-dDelstT] [-rwx] [-pi] [-c cmd] [-n count] [-L size]",
3013             "set a watchpoint at the specified address", cmd_wp, wp_help },

3015 #else
3016         /*
3017          * dcmds specific to mdb, or which have mdb-specific arguments
3018          */
3019         { "?", "fmt-list", "format data from object file", cmd_print_object },
3020         { "$>", "[file]", "log session to a file", cmd_old_log },
3021         { "$g", "?", "get/set C++ demangling options", cmd_demflags },
3022         { "$G", NULL, "enable/disable C++ demangling support", cmd_demangle },
3023         { "$i", NULL, "print signals that are ignored", cmd_notsup },
3024         { "$l", NULL, "print the representative thread's lwp id", cmd_notsup },
3025         { "$p", ":", "change debugger target context", cmd_context },
3026         { "$x", NULL, "print floating point registers", cmd_notsup },
3027         { "$X", NULL, "print floating point registers", cmd_notsup },
3028         { "$y", NULL, "print floating point registers", cmd_notsup },
3029         { "$Y", NULL, "print floating point registers", cmd_notsup },
3030         { ":A", "?[core|pid]", "attach to process or core file", cmd_notsup },
3031         { ":c", "[SIG]", "continue target execution", cmd_cont },
3032         { ":e", "[SIG]", "step target over next instruction", cmd_next },
3033         { ":i", ":", "ignore signal (delete all matching events)", cmd_notsup },
3034         { ":k", NULL, "forcibly kill and release target", cmd_notsup },
3035         { ":t", "?[+/-dDestT] [-c cmd] [-n count] SIG ...", "stop on delivery "
3036             "of the specified signals", cmd_sigbp, sigbp_help },
3037         { ":r", "[ args ... ]", "run a new target process", cmd_run },
3038         { ":R", NULL, "release the previously attached process", cmd_notsup },
3039         { ":s", "[SIG]", "single-step target to next instruction", cmd_step },
3040         { ":u", "[SIG]", "step target out of current function", cmd_step_out },
3041         { "attach", "?[core|pid]",
3042             "attach to process or core file", cmd_notsup },
3043         { "cat", "[file ...]", "concatenate and display files", cmd_cat },
3044         { "cont", "[SIG]", "continue target execution", cmd_cont },
3045         { "context", ":", "change debugger target context", cmd_context },
3046         { "dem", "name ...", "demangle C++ symbol names", cmd_demstr },
3047         { "fltbp", "?[+/-dDestT] [-c cmd] [-n count] fault ...",
3048             "stop on machine fault", cmd_fltbp, fltbp_help },
3049         { "fpregs", NULL, "print floating point registers", cmd_notsup },
3050         { "kill", NULL, "forcibly kill and release target", cmd_notsup },
3051         { "load", "[-s] module", "load debugger module", cmd_load, load_help },
3052         { "log", "[-d | [-e] file]", "log session to a file", cmd_log },
3053         { "next", "[SIG]", "step target over next instruction", cmd_next },
3054         { "quit", NULL, "quit debugger", cmd_quit },
3055         { "release", NULL,
3056             "release the previously attached process", cmd_notsup },
3057         { "run", "[ args ... ]", "run a new target process", cmd_run },
3058         { "sigbp", "?[+/-dDestT] [-c cmd] [-n count] SIG ...", "stop on "
```



```
3059             "delivery of the specified signals", cmd_sigbp, sigbp_help },
3060         { "step", "[ over | out ] [SIG]",
3061             "single-step target to next instruction", cmd_step },
3062         { "sysbp", "?[+/-dDestT] [-io] [-c cmd] [-n count] syscall ...",
3063             "stop on entry or exit from system call", cmd_sysbp, sysbp_help },
3064         { "unload", "module", "unload debugger module", cmd_unload },
3065         { "wp", ":[+/-dDelstT] [-rwx] [-c cmd] [-n count] [-L size]",
3066             "set a watchpoint at the specified address", cmd_wp, wp_help },
3067 #endif

3069         { NULL }
3070 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   75827 Tue Jun 12 09:01:24 2012
new/usr/src/cmd/mdb/common/mdb/mdb_print.c
2574 mdb needs ::printf
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Approved by: ?
**********************************************************
```

```
/*
 * 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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright (c) 2012 Joyent, Inc. All rights reserved.
 */

#include <mdb/mdb_modapi.h>
#include <mdb/mdb_target.h>
#include <mdb/mdb_argvec.h>
#include <mdb/mdb_string.h>
#include <mdb/mdb_stdlib.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb_fmt.h>
#include <mdb/mdb_ctf.h>
#include <mdb/mdb_ctf_impl.h>
#include <mdb/mdb.h>
#include <mdb/mdb_tab.h>

#include <sys/isa_defs.h>
#include <sys/param.h>
#include <sys/sysmacros.h>
#include <netinet/in.h>
#include <strings.h>
#include <libctf.h>
#include <ctype.h>

typedef struct holeinfo {
        ulong_t hi_offset;              /* expected offset */
        uchar_t hi_isunion;             /* represents a union */
} holeinfo_t;
_____unchanged_portion_omitted_
```



```
/*
 * Special semantics for pipelines.
 */
static int
pipe_print(mdb_ctf_id_t id, ulong_t off, void *data)
{
        printarg_t *pap = data;
        ssize_t size;
        static const char *const fsp[] = { "%#r", "%#r", "%#r", "%#llr" };
        uintptr_t value;
        uintptr_t addr = pap->pa_addr + off / NBBY;
        mdb_ctf_id_t base;
        ctf_encoding_t e;

        union {
                uint64_t i8;
                uint32_t i4;
                uint16_t i2;
                uint8_t i1;
        } u;

        if (mdb_ctf_type_resolve(id, &base) == -1) {
                mdb_warn("could not resolve type");
                mdb_warn("could not resolve type\n");
                return (-1);
        }

        /*
         * If the user gives -a, then always print out the address of the
         * member.
         */
        if ((pap->pa_flags & PA_SHOWADDR)) {
                mdb_printf("%#lr\n", addr);
                return (0);
        }

again:
        switch (mdb_ctf_type_kind(base)) {
        case CTF_K_POINTER:
                if (mdb_tgt_aread(pap->pa_tgt, pap->pa_as,
                    &value, sizeof (value), addr) != sizeof (value)) {
                        mdb_warn("failed to read pointer at %p", addr);
                        return (-1);
                }
                mdb_printf("%#lr\n", value);
                break;

        case CTF_K_INTEGER:
        case CTF_K_ENUM:
                if (mdb_ctf_type_encoding(base, &e) != 0) {
                        mdb_printf("could not get type encoding\n");
                        return (-1);
                }

                /*
                 * For immediate values, we just print out the value.
                 */
                size = e.cte_bits / NBBY;
                if (size > 8 || (e.cte_bits % NBBY) != 0 ||
                    (size & (size - 1)) != 0) {
                        return (print_bitfield(off, pap, &e));
                }

                if (mdb_tgt_aread(pap->pa_tgt, pap->pa_as, &u.i8, size,
                    addr) != size) {
                        mdb_warn("failed to read %lu bytes at %p",
```

```
1734                            (ulong_t)size, pap->pa_addr);
1735                    return (-1);
1736            }

1738            switch (size) {
1739            case sizeof (uint8_t):
1740                    mdb_printf(fsp[0], u.i1);
1741                    break;
1742            case sizeof (uint16_t):
1743                    mdb_printf(fsp[1], u.i2);
1744                    break;
1745            case sizeof (uint32_t):
1746                    mdb_printf(fsp[2], u.i4);
1747                    break;
1748            case sizeof (uint64_t):
1749                    mdb_printf(fsp[3], u.i8);
1750                    break;
1751            }
1752            mdb_printf("\n");
1753            break;

1755        case CTF_K_FUNCTION:
1756        case CTF_K_FLOAT:
1757        case CTF_K_ARRAY:
1758        case CTF_K_UNKNOWN:
1759        case CTF_K_STRUCT:
1760        case CTF_K_UNION:
1761        case CTF_K_FORWARD:
1762            /*
1763             * For these types, always print the address of the member
1764             */
1765            mdb_printf("%#lr\n", addr);
1766            break;

1768        default:
1769            mdb_warn("unknown type %d", mdb_ctf_type_kind(base));
1770            break;
1771        }

1773        return (0);
1774 }
```

unchanged_portion_omitted_

```
1837 static int
1838 parse_member(printarg_t *pap, const char *str, mdb_ctf_id_t id,
1839     mdb_ctf_id_t *idp, ulong_t *offp, int *last_deref)
1840 {
1841        int delim;
1842        char member[64];
1843        char buf[128];
1844        uint_t index;
1845        char *start = (char *)str;
1846        char *end;
1847        ulong_t off = 0;
1848        mdb_ctf_arinfo_t ar;
1849        mdb_ctf_id_t rid;
1850        int kind;
1851        ssize_t size;
1852        int non_array = FALSE;

1854        /*
1855         * id always has the unresolved type for printing error messages
1856         * that include the type; rid always has the resolved type for
1857         * use in mdb_ctf_* calls.  It is possible for this command to fail,
1858         * however, if the resolved type is in the parent and it is currently
1859         * unavailable.  Note that we also can't print out the name of the
```



```
1860         * type, since that would also rely on looking up the resolved name.
1861         */
1862        if (mdb_ctf_type_resolve(id, &rid) != 0) {
1863                mdb_warn("failed to resolve type");
1864                return (-1);
1865        }

1867        delim = parse_delimiter(&start);
1868        /*
1869         * If the user fails to specify an initial delimiter, guess -> for
1870         * pointer types and . for non-pointer types.
1871         */
1872        if (delim == MEMBER_DELIM_ERR)
1873                delim = (mdb_ctf_type_kind(rid) == CTF_K_POINTER) ?
1874                    MEMBER_DELIM_PTR : MEMBER_DELIM_DOT;

1876        *last_deref = FALSE;

1878        while (delim != MEMBER_DELIM_DONE) {
1879                switch (delim) {
1880                case MEMBER_DELIM_PTR:
1881                        kind = mdb_ctf_type_kind(rid);
1882                        if (kind != CTF_K_POINTER) {
1883                                mdb_warn("%s is not a pointer type\n",
1884                                    mdb_ctf_type_name(id, buf, sizeof (buf)));
1885                                return (-1);
1886                        }

1888                        size = mdb_ctf_type_size(id);
1889                        if (deref(pap, size) != 0)
1890                                return (-1);

1892                        (void) mdb_ctf_type_reference(rid, &id);
1893                        (void) mdb_ctf_type_resolve(id, &rid);

1895                        off = 0;
1896                        break;

1898                case MEMBER_DELIM_DOT:
1899                        kind = mdb_ctf_type_kind(rid);
1900                        if (kind != CTF_K_STRUCT && kind != CTF_K_UNION) {
1901                                mdb_warn("%s is not a struct or union type\n",
1902                                    mdb_ctf_type_name(id, buf, sizeof (buf)));
1903                                return (-1);
1904                        }
1905                        break;

1907                case MEMBER_DELIM_LBR:
1908                        end = strchr(start, ']');
1909                        if (end == NULL) {
1910                                mdb_warn("no trailing ']'\n");
1911                                return (-1);
1912                        }

1914                        (void) mdb_snprintf(member, end - start + 1, "%s",
1915                            start);

1917                        index = mdb_strtoull(member);

1919                        switch (mdb_ctf_type_kind(rid)) {
1920                        case CTF_K_POINTER:
1921                                size = mdb_ctf_type_size(rid);

1923                                if (deref(pap, size) != 0)
1924                                        return (-1);
```

```
1926                                (void) mdb_ctf_type_reference(rid, &id);
1927                                (void) mdb_ctf_type_resolve(id, &rid);

1929                                size = mdb_ctf_type_size(id);
1930                                if (size <= 0) {
1931                                        mdb_warn("cannot dereference void "
1932                                            "type\n");
1933                                        return (-1);
1934                                }

1936                                pap->pa_addr += index * size;
1937                                off = 0;

1939                                if (index == 0 && non_array)
1940                                        *last_deref = TRUE;
1941                                break;

1943                        case CTF_K_ARRAY:
1944                                (void) mdb_ctf_array_info(rid, &ar);

1946                                if (index >= ar.mta_nelems) {
1947                                        mdb_warn("index %r is outside of "
1948                                            "array bounds [0 .. %r]\n",
1949                                            index, ar.mta_nelems - 1);
1950                                }

1952                                id = ar.mta_contents;
1953                                (void) mdb_ctf_type_resolve(id, &rid);

1955                                size = mdb_ctf_type_size(id);
1956                                if (size <= 0) {
1957                                        mdb_warn("cannot dereference void "
1958                                            "type\n");
1959                                        return (-1);
1960                                }

1962                                pap->pa_addr += index * size;
1963                                off = 0;
1964                                break;

1966                        default:
1967                                mdb_warn("cannot index into non-array, "
1968                                    "non-pointer type\n");
1969                                return (-1);
1970                        }

1972                        start = end + 1;
1973                        delim = parse_delimiter(&start);
1974                        continue;

1976                case MEMBER_DELIM_ERR:
1977                default:
1978                        mdb_warn("'%c' is not a valid delimiter\n", *start);
1979                        return (-1);
1980                }

1982                *last_deref = FALSE;
1983                non_array = TRUE;

1985                /*
1986                 * Find the end of the member name; assume that a member
1987                 * name is at least one character long.
1988                 */
1989                for (end = start + 1; isalnum(*end) || *end == '_'; end++)
1990                        continue;
```



```
1992                (void) mdb_snprintf(member, end - start + 1, "%s", start);

1994                if (mdb_ctf_member_info(rid, member, &off, &id) != 0) {
1995                        mdb_warn("failed to find member %s of %s", member,
1996                            mdb_ctf_type_name(id, buf, sizeof (buf)));
1997                        return (-1);
1998                }
1999                (void) mdb_ctf_type_resolve(id, &rid);

2001                pap->pa_addr += off / NBBY;

2003                start = end;
2004                delim = parse_delimiter(&start);
2005        }

2007        *idp = id;
2008        *offp = off;

2010        return (0);
2011 }
______unchanged_portion_omitted_

2439 void
2440 print_help(void)
2441 {
2442        mdb_printf(
2443            "-a         show address of object\n"
2444            "-C         unlimit the length of character arrays\n"
2445            "-c limit   limit the length of character arrays\n"
2446            "-d         output values in decimal\n"
2447            "-h         print holes in structures\n"
2448            "-i         interpret address as data of the given type\n"
2449            "-L         unlimit the length of standard arrays\n"
2450            "-l limit   limit the length of standard arrays\n"
2451            "-n         don't print pointers as symbol offsets\n"
2452            "-p         interpret address as a physical memory address\n"
2453            "-s depth   limit the recursion depth\n"
2454            "-T         show type and <<base type>> of object\n"
2455            "-t         show type of object\n"
2456            "-x         output values in hexadecimal\n"
2457            "\n"
2458            "type may be omitted if the C type of addr can be inferred.\n"
2459            "\n"
2460            "Members may be specified with standard C syntax using the\n"
2461            "array indexing operator \"[index]\", structure member\n"
2462            "operator \".\", or structure pointer operator \"->\".\n"
2463            "\n"
2464            "Offsets must use the $[ expression ] syntax\n");
2465 }

2467 static int
2468 printf_signed(mdb_ctf_id_t id, uintptr_t addr, ulong_t off, char *fmt,
2469     boolean_t sign)
2470 {
2471        ssize_t size;
2472        mdb_ctf_id_t base;
2473        ctf_encoding_t e;

2475        union {
2476                uint64_t ui8;
2477                uint32_t ui4;
2478                uint16_t ui2;
2479                uint8_t ui1;
2480                int64_t i8;
2481                int32_t i4;
```

```
2482                int16_t i2;
2483                int8_t i1;
2484        } u;

2486        if (mdb_ctf_type_resolve(id, &base) == -1) {
2487                mdb_warn("could not resolve type");
2488                return (DCMD_ABORT);
2489        }

2491        if (mdb_ctf_type_kind(base) != CTF_K_INTEGER) {
2492                mdb_warn("expected integer type\n");
2493                return (DCMD_ABORT);
2494        }

2496        if (mdb_ctf_type_encoding(base, &e) != 0) {
2497                mdb_warn("could not get type encoding");
2498                return (DCMD_ABORT);
2499        }

2501        if (sign)
2502                sign = e.cte_format & CTF_INT_SIGNED;

2504        size = e.cte_bits / NBBY;

2506        /*
2507         * Check to see if our life has been complicated by the presence of
2508         * a bitfield.  If it has, we will print it using logic that is only
2509         * slightly different than that found in print_bitfield(), above.  (In
2510         * particular, see the comments there for an explanation of the
2511         * endianness differences in this code.)
2512         */
2513        if (size > 8 || (e.cte_bits % NBBY) != 0 ||
2514            (size & (size - 1)) != 0) {
2515                uint64_t mask = (1ULL << e.cte_bits) - 1;
2516                uint64_t value = 0;
2517                uint8_t *buf = (uint8_t *)&value;
2518                uint8_t shift;

2520                /*
2521                 * Round our size up one byte.
2522                 */
2523                size = (e.cte_bits + (NBBY - 1)) / NBBY;

2525                if (e.cte_bits > sizeof (value) * NBBY - 1) {
2526                        mdb_printf("invalid bitfield size %u", e.cte_bits);
2527                        return (DCMD_ABORT);
2528                }

2530 #ifdef _BIG_ENDIAN
2531                buf += sizeof (value) - size;
2532                off += e.cte_bits;
2533 #endif

2535                if (mdb_vread(buf, size, addr) == -1) {
2536                        mdb_warn("failed to read %lu bytes at %p", size, addr);
2537                        return (DCMD_ERR);
2538                }

2540                shift = off % NBBY;
2541 #ifdef _BIG_ENDIAN
2542                shift = NBBY - shift;
2543 #endif

2545                /*
2546                 * If we have a bit offset within the byte, shift it down.
2547                 */
```



```
2548                if (off % NBBY != 0)
2549                        value >>= shift;
2550                value &= mask;

2552                if (sign) {
2553                        int sshift = sizeof (value) * NBBY - e.cte_bits;
2554                        value = ((int64_t)value << sshift) >> sshift;
2555                }

2557                mdb_printf(fmt, value);
2558                return (0);
2559        }

2561        if (mdb_vread(&u.i8, size, addr) == -1) {
2562                mdb_warn("failed to read %lu bytes at %p", (ulong_t)size, addr);
2563                return (DCMD_ERR);
2564        }

2566        switch (size) {
2567        case sizeof (uint8_t):
2568                mdb_printf(fmt, (uint64_t)(sign ? u.i1 : u.ui1));
2569                break;
2570        case sizeof (uint16_t):
2571                mdb_printf(fmt, (uint64_t)(sign ? u.i2 : u.ui2));
2572                break;
2573        case sizeof (uint32_t):
2574                mdb_printf(fmt, (uint64_t)(sign ? u.i4 : u.ui4));
2575                break;
2576        case sizeof (uint64_t):
2577                mdb_printf(fmt, (uint64_t)(sign ? u.i8 : u.ui8));
2578                break;
2579        }

2581        return (0);
2582 }

2584 static int
2585 printf_int(mdb_ctf_id_t id, uintptr_t addr, ulong_t off, char *fmt)
2586 {
2587        return (printf_signed(id, addr, off, fmt, B_TRUE));
2588 }

2590 static int
2591 printf_uint(mdb_ctf_id_t id, uintptr_t addr, ulong_t off, char *fmt)
2592 {
2593        return (printf_signed(id, addr, off, fmt, B_FALSE));
2594 }

2596 /*ARGSUSED*/
2597 static int
2598 printf_uint32(mdb_ctf_id_t id, uintptr_t addr, ulong_t off, char *fmt)
2599 {
2600        mdb_ctf_id_t base;
2601        ctf_encoding_t e;
2602        uint32_t value;

2604        if (mdb_ctf_type_resolve(id, &base) == -1) {
2605                mdb_warn("could not resolve type\n");
2606                return (DCMD_ABORT);
2607        }

2609        if (mdb_ctf_type_kind(base) != CTF_K_INTEGER ||
2610            mdb_ctf_type_encoding(base, &e) != 0 ||
2611            e.cte_bits / NBBY != sizeof (value)) {
2612                mdb_warn("expected 32-bit integer type\n");
2613                return (DCMD_ABORT);
```

```
2614         }

2616         if (mdb_vread(&value, sizeof (value), addr) == -1) {
2617                 mdb_warn("failed to read 32-bit value at %p", addr);
2618                 return (DCMD_ERR);
2619         }

2621         mdb_printf(fmt, value);

2623         return (0);
2624 }

2626 /*ARGSUSED*/
2627 static int
2628 printf_ptr(mdb_ctf_id_t id, uintptr_t addr, ulong_t off, char *fmt)
2629 {
2630         uintptr_t value;
2631         mdb_ctf_id_t base;

2633         if (mdb_ctf_type_resolve(id, &base) == -1) {
2634                 mdb_warn("could not resolve type\n");
2635                 return (DCMD_ABORT);
2636         }

2638         if (mdb_ctf_type_kind(base) != CTF_K_POINTER) {
2639                 mdb_warn("expected pointer type\n");
2640                 return (DCMD_ABORT);
2641         }

2643         if (mdb_vread(&value, sizeof (value), addr) == -1) {
2644                 mdb_warn("failed to read pointer at %llx", addr);
2645                 return (DCMD_ERR);
2646         }

2648         mdb_printf(fmt, value);

2650         return (0);
2651 }

2653 /*ARGSUSED*/
2654 static int
2655 printf_string(mdb_ctf_id_t id, uintptr_t addr, ulong_t off, char *fmt)
2656 {
2657         mdb_ctf_id_t base;
2658         mdb_ctf_arinfo_t r;
2659         char buf[1024];
2660         ssize_t size;

2662         if (mdb_ctf_type_resolve(id, &base) == -1) {
2663                 mdb_warn("could not resolve type");
2664                 return (DCMD_ABORT);
2665         }

2667         if (mdb_ctf_type_kind(base) == CTF_K_POINTER) {
2668                 uintptr_t value;

2670                 if (mdb_vread(&value, sizeof (value), addr) == -1) {
2671                         mdb_warn("failed to read pointer at %llx", addr);
2672                         return (DCMD_ERR);
2673                 }

2675                 if (mdb_readstr(buf, sizeof (buf) - 1, value) < 0) {
2676                         mdb_warn("failed to read string at %llx", value);
2677                         return (DCMD_ERR);
2678                 }
```



```
2680                 mdb_printf(fmt, buf);
2681                 return (0);
2682         }

2684         if (mdb_ctf_type_kind(base) != CTF_K_ARRAY) {
2685                 mdb_warn("exepected pointer or array type\n");
2686                 return (DCMD_ABORT);
2687         }

2689         if (mdb_ctf_array_info(base, &r) == -1 ||
2690             mdb_ctf_type_resolve(r.mta_contents, &base) == -1 ||
2691             (size = mdb_ctf_type_size(base)) == -1) {
2692                 mdb_warn("can't determine array type");
2693                 return (DCMD_ABORT);
2694         }

2696         if (size != 1) {
2697                 mdb_warn("string format specifier requires "
2698                     "an array of characters\n");
2699                 return (DCMD_ABORT);
2700         }

2702         bzero(buf, sizeof (buf));

2704         if (mdb_vread(buf, MIN(r.mta_nelems, sizeof (buf) - 1), addr) == -1) {
2705                 mdb_warn("failed to read array at %p", addr);
2706                 return (DCMD_ERR);
2707         }

2709         mdb_printf(fmt, buf);

2711         return (0);
2712 }

2714 /*ARGSUSED*/
2715 static int
2716 printf_ipv6(mdb_ctf_id_t id, uintptr_t addr, ulong_t off, char *fmt)
2717 {
2718         mdb_ctf_id_t base;
2719         mdb_ctf_id_t ipv6_type, ipv6_base;
2720         in6_addr_t ipv6;

2722         if (mdb_ctf_lookup_by_name("in6_addr_t", &ipv6_type) == -1) {
2723                 mdb_warn("could not resolve in6_addr_t type\n");
2724                 return (DCMD_ABORT);
2725         }

2727         if (mdb_ctf_type_resolve(id, &base) == -1) {
2728                 mdb_warn("could not resolve type\n");
2729                 return (DCMD_ABORT);
2730         }

2732         if (mdb_ctf_type_resolve(ipv6_type, &ipv6_base) == -1) {
2733                 mdb_warn("could not resolve in6_addr_t type\n");
2734                 return (DCMD_ABORT);
2735         }

2737         if (mdb_ctf_type_cmp(base, ipv6_base) != 0) {
2738                 mdb_warn("requires argument of type in6_addr_t\n");
2739                 return (DCMD_ABORT);
2740         }

2742         if (mdb_vread(&ipv6, sizeof (ipv6), addr) == -1) {
2743                 mdb_warn("couldn't read in6_addr_t at %p", addr);
2744                 return (DCMD_ERR);
2745         }
```

```
2747         mdb_printf(fmt, &ipv6);

2749         return (0);
2750 }

2752 /*
2753  * To validate the format string specified to ::printf, we run the format
2754  * string through a very simple state machine that restricts us to a subset
2755  * of mdb_printf() functionality.
2756  */
2757 enum {
2758         PRINTF_NOFMT = 1,               /* no current format specifier */
2759         PRINTF_PERC,                    /* processed '%' */
2760         PRINTF_FMT,                     /* processing format specifier */
2761         PRINTF_LEFT,                    /* processed '-', expecting width */
2762         PRINTF_WIDTH,                   /* processing width */
2763         PRINTF_QUES                     /* processed '?', expecting format */
2764 };

2766 int
2767 cmd_printf(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
2768 {
2769         char type[MDB_SYM_NAMLEN];
2770         int i, nfmts = 0, ret;
2771         mdb_ctf_id_t id;
2772         const char *fmt, *member;
2773         char **fmts, *last, *dest, f;
2774         int (**funcs)(mdb_ctf_id_t, uintptr_t, ulong_t, char *);
2775         int state = PRINTF_NOFMT;
2776         printarg_t pa;

2778         if (!(flags & DCMD_ADDRSPEC))
2779                 return (DCMD_USAGE);

2781         bzero(&pa, sizeof (pa));
2782         pa.pa_as = MDB_TGT_AS_VIRT;
2783         pa.pa_realtgt = pa.pa_tgt = mdb.m_target;

2785         if (argc == 0 || argv[0].a_type != MDB_TYPE_STRING) {
2786                 mdb_warn("expected a format string\n");
2787                 return (DCMD_USAGE);
2788         }

2790         /*
2791          * Our first argument is a format string; rip it apart and run it
2792          * through our state machine to validate that our input is within the
2793          * subset of mdb_printf() format strings that we allow.
2794          */
2795         fmt = argv[0].a_un.a_str;
2796         /*
2797          * 'dest' must be large enough to hold a copy of the format string,
2798          * plus a NUL and up to 2 additional characters for each conversion
2799          * in the format string.  This gives us a bloat factor of 5/2 ~= 3.
2800          *   e.g. "%d" (strlen of 2) --> "%lld\0" (need 5 bytes)
2801          */
2802         dest = mdb_zalloc(strlen(fmt) * 3, UM_SLEEP | UM_GC);
2803         fmts = mdb_zalloc(strlen(fmt) * sizeof (char *), UM_SLEEP | UM_GC);
2804         funcs = mdb_zalloc(strlen(fmt) * sizeof (void *), UM_SLEEP | UM_GC);
2805         last = dest;

2807         for (i = 0; fmt[i] != '\0'; i++) {
2808                 *dest++ = f = fmt[i];

2810                 switch (state) {
2811                 case PRINTF_NOFMT:
```



```
2812                         state = f == '%' ? PRINTF_PERC : PRINTF_NOFMT;
2813                         break;

2815                 case PRINTF_PERC:
2816                         state = f == '-' ? PRINTF_LEFT :
2817                             f >= '0' && f <= '9' ? PRINTF_WIDTH :
2818                             f == '?' ? PRINTF_QUES :
2819                             f == '%' ? PRINTF_NOFMT : PRINTF_FMT;
2820                         break;

2822                 case PRINTF_LEFT:
2823                         state = f >= '0' && f <= '9' ? PRINTF_WIDTH :
2824                             f == '?' ? PRINTF_QUES : PRINTF_FMT;
2825                         break;

2827                 case PRINTF_WIDTH:
2828                         state = f >= '0' && f <= '9' ? PRINTF_WIDTH :
2829                             PRINTF_FMT;
2830                         break;

2832                 case PRINTF_QUES:
2833                         state = PRINTF_FMT;
2834                         break;
2835                 }

2837                 if (state != PRINTF_FMT)
2838                         continue;

2840                 dest--;

2842                 /*
2843                  * Now check that we have one of our valid format characters.
2844                  */
2845                 switch (f) {
2846                 case 'a':
2847                 case 'A':
2848                 case 'p':
2849                         funcs[nfmts] = printf_ptr;
2850                         break;

2852                 case 'd':
2853                 case 'q':
2854                 case 'R':
2855                         funcs[nfmts] = printf_int;
2856                         *dest++ = 'l';
2857                         *dest++ = 'l';
2858                         break;

2860                 case 'I':
2861                         funcs[nfmts] = printf_uint32;
2862                         break;

2864                 case 'N':
2865                         funcs[nfmts] = printf_ipv6;
2866                         break;

2868                 case 'o':
2869                 case 'r':
2870                 case 'u':
2871                 case 'x':
2872                 case 'X':
2873                         funcs[nfmts] = printf_uint;
2874                         *dest++ = 'l';
2875                         *dest++ = 'l';
2876                         break;
```

```
2878                case 's':
2879                        funcs[nfmts] = printf_string;
2880                        break;

2882                case 'Y':
2883                        funcs[nfmts] = sizeof (time_t) == sizeof (int) ?
2884                            printf_uint32 : printf_uint;
2885                        break;

2887                default:
2888                        mdb_warn("illegal format string at or near "
2889                            "'%c' (position %d)\n", f, i + 1);
2890                        return (DCMD_ABORT);
2891                }

2893                *dest++ = f;
2894                *dest++ = '\0';
2895                fmts[nfmts++] = last;
2896                last = dest;
2897                state = PRINTF_NOFMT;
2898        }

2900        argc--;
2901        argv++;

2903        /*
2904         * Now we expect a type name.
2905         */
2906        if ((ret = args_to_typename(&argc, &argv, type, sizeof (type))) != 0)
2907                return (ret);

2909        argv++;
2910        argc--;

2912        if (mdb_ctf_lookup_by_name(type, &id) != 0) {
2913                mdb_warn("failed to look up type %s", type);
2914                return (DCMD_ABORT);
2915        }

2917        if (argc == 0) {
2918                mdb_warn("at least one member must be specified\n");
2919                return (DCMD_USAGE);
2920        }

2922        if (argc != nfmts) {
2923                mdb_warn("%s format specifiers (found %d, expected %d)\n",
2924                    argc > nfmts ? "missing" : "extra", nfmts, argc);
2925                return (DCMD_ABORT);
2926        }

2928        for (i = 0; i < argc; i++) {
2929                mdb_ctf_id_t mid;
2930                ulong_t off;
2931                int ignored;

2933                if (argv[i].a_type != MDB_TYPE_STRING) {
2934                        mdb_warn("expected only type member arguments\n");
2935                        return (DCMD_ABORT);
2936                }

2938                if (strcmp((member = argv[i].a_un.a_str), ".") == 0) {
2939                        /*
2940                         * We allow "." to be specified to denote the current
2941                         * value of dot.
2942                         */
2943                        if (funcs[i] != printf_ptr && funcs[i] != printf_uint &&
```



```
2944                            funcs[i] != printf_int) {
2945                                mdb_warn("expected integer or pointer format "
2946                                    "specifier for '.'\n");
2947                                return (DCMD_ABORT);
2948                        }

2950                        mdb_printf(fmts[i], mdb_get_dot());
2951                        continue;
2952                }

2954                pa.pa_addr = addr;

2956                if (parse_member(&pa, member, id, &mid, &off, &ignored) != 0)
2957                        return (DCMD_ABORT);

2959                if ((ret = funcs[i](mid, pa.pa_addr, off, fmts[i])) != 0) {
2960                        mdb_warn("failed to print member '%s'\n", member);
2961                        return (ret);
2962                }
2963        }

2965        mdb_printf("%s", last);

2967        return (DCMD_OK);
2968 }

2970 static char _mdb_printf_help[] =
2971 "The format string argument is a printf(3C)-like format string that is a\n"
2972 "subset of the format strings supported by mdb_printf().  The type argument\n"
2973 "is the name of a type to be used to interpret the memory referenced by dot.\n"
2974 "The member should either be a field in the specified structure, or the\n"
2975 "special member '.', denoting the value of dot (and treated as a pointer).\n"
2976 "The number of members must match the number of format specifiers in the\n"
2977 "format string.\n"
2978 "\n"
2979 "The following format specifiers are recognized by ::printf:\n"
2980 "\n"
2981 "  %%    Prints the '%' symbol.\n"
2982 "  %a    Prints the member in symbolic form.\n"
2983 "  %d    Prints the member as a decimal integer.  If the member is a signed\n"
2984 "        integer type, the output will be signed.\n"
2985 "  %I    Prints the member a IPv4 address (must be a 32-bit integer type).\n"
2986 "  %N    Prints the member an IPv6 address (must be of type in6_addr_t).\n"
2987 "  %o    Prints the member as an unsigned octal integer.\n"
2988 "  %p    Prints the member as a pointer, in hexadecimal.\n"
2989 "  %q    Prints the member in signed octal.  Honk if you ever use this!\n"
2990 "  %r    Prints the member as an unsigned value in the current output radix.\n"
2991 "  %R    Prints the member as a signed value in the current output radix.\n"
2992 "  %s    Prints the member as a string (requires a pointer or an array of\n"
2993 "        characters).\n"
2994 "  %u    Prints the member as an unsigned decimal integer.\n"
2995 "  %x    Prints the member in hexadecimal.\n"
2996 "  %X    Prints the member in hexadecimal, using the characters A-F as the\n"
2997 "        digits for the values 10-15.\n"
2998 "  %Y    Prints the member as a time_t as the string "
2999             "'year month day HH:MM:SS'.\n"
3000 "\n"
3001 "The following field width specifiers are recognized by ::printf:\n"
3002 "\n"
3003 "  %n    Field width is set to the specified decimal value.\n"
3004 "  %?    Field width is set to the maximum width of a hexadecimal pointer\n"
3005 "        value.  This is 8 in an ILP32 environment, and 16 in an LP64\n"
3006 "        environment.\n"
3007 "\n"
3008 "The following flag specifers are recognized by ::printf:\n"
3009 "\n"
```

```
3010 "  %-    Left-justify the output within the specified field width.  If the\n"
3011 "        width of the output is less than the specified field width, the\n"
3012 "        output will be padded with blanks on the right-hand side.  Without\n"
3013 "        %-, values are right-justified by default.\n"
3014 "\n"
3015 "  %0    Zero-fill the output field if the output is right-justified and the\n"
3016 "        width of the output is less than the specified field width.  Without\n"
3017 "        %0, right-justified values are prepended with blanks in order to\n"
3018 "        fill the field.\n"
3019 "\n"
3020 "Examples: \n"
3021 "\n"
3022 "  ::walk proc | "
3023         "::printf \"%-6d %s\\n\" proc_t p_pidp->pid_id p_user.u_psargs\n"
3024 "  ::walk thread | "
3025         "::printf \"%?p %3d %a\\n\" kthread_t . t_pri t_startpc\n"
3026 "  ::walk zone | "
3027         "::printf \"%-40s %20s\\n\" zone_t zone_name zone_nodename\n"
3028 "  ::walk ire | "
3029         "::printf \"%Y %I\\n\" ire_t ire_create_time ire_u.ire4_u.ire4_addr\n"
3030 "\n";

3032 void
3033 printf_help(void)
3034 {
3035         mdb_printf("%s", _mdb_printf_help);
3036 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    1948 Tue Jun 12 09:01:24 2012
new/usr/src/cmd/mdb/common/mdb/mdb_print.h
2574 mdb needs ::printf
Reviewed by: Joshua M. Clulow <josh@sysmgr.org>
Reviewed by: Eric Schrock <eric.schrock@delphix.com>
Reviewed by: Adam Leventhal <ahl@delphix.com>
Approved by: ?
**********************************************************
```

```
/*
 * 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 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright (c) 2012 Joyent, Inc. All rights reserved.
 */

#ifndef _MDB_PRINT_H
#define _MDB_PRINT_H

#include <mdb/mdb_tab.h>

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _MDB

extern int cmd_enum(uintptr_t, uint_t, int, const mdb_arg_t *);
extern void enum_help(void);
extern int cmd_sizeof(uintptr_t, uint_t, int, const mdb_arg_t *);
extern int cmd_sizeof_tab(mdb_tab_cookie_t *, uint_t, int, const mdb_arg_t *);
extern int cmd_offsetof(uintptr_t, uint_t, int, const mdb_arg_t *);
extern int cmd_list(uintptr_t, uint_t, int, const mdb_arg_t *);
extern int cmd_array(uintptr_t, uint_t, int, const mdb_arg_t *);
extern int cmd_print(uintptr_t, uint_t, int, const mdb_arg_t *);
extern int cmd_print_tab(mdb_tab_cookie_t *, uint_t, int, const mdb_arg_t *);
extern void print_help(void);
extern int cmd_printf(uintptr_t, uint_t, int, const mdb_arg_t *);
extern void printf_help(void);

#endif  /* _MDB */

#ifdef  __cplusplus
```



```
}
```

*unchanged_portion_omitted_*