****
52404 Mon May 5 01:02:55 2014
new/usr/src/cmd/hal/hald/solaris/devinfo_storage.c
4846 HAL partition names don’t match real partition names
****

```
/***************************************************************************
 *
 * devinfo_storage.c : storage devices
 *
 * Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2013 Garrett D’Amore <garrett@damore.org>
 * Copyright 2014 Andrew Stormont.
#endif /* ! codereview */
 *
 * Licensed under the Academic Free License version 2.1
 *
 **************************************************************************/

#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <ctype.h>
#include <libdevinfo.h>
#include <sys/types.h>
#include <sys/mkdev.h>
#include <sys/stat.h>
#include <sys/mntent.h>
#include <sys/mnttab.h>

#include "../osspec.h"
#include "../logger.h"
#include "../hald.h"
#include "../hald_dbus.h"
#include "../device_info.h"
#include "../util.h"
#include "../hald_runner.h"
#include "hotplug.h"
#include "devinfo.h"
#include "devinfo_misc.h"
#include "devinfo_storage.h"
#include "osspec_solaris.h"

#ifdef sparc
#define WHOLE_DISK      "s2"
#define DOS_SEPERATOR   ":"
#define DOS_FORMAT      "s2:%d"
#define DOS_TEMPLATE    ":NN"
#endif /* ! codereview */
#else
#define WHOLE_DISK      "p0"
#define DOS_SEPERATOR   "p"
#define DOS_FORMAT      "p%d"
#define DOS_TEMPLATE    "pNN"
#endif /* ! codereview */
#endif

/* some devices,especially CDROMs, may take a while to be probed (values in ms)
#define DEVINFO_PROBE_STORAGE_TIMEOUT   60000
#define DEVINFO_PROBE_VOLUME_TIMEOUT    60000

typedef struct devinfo_storage_minor {
        char    *devpath;
```

```
        char    *devlink;
        char    *slice;
        dev_t   dev;
        int     dosnum; /* dos disk number or -1 */
} devinfo_storage_minor_t;

HalDevice *devinfo_ide_add(HalDevice *parent, di_node_t node, char *devfs_path,
static HalDevice *devinfo_ide_host_add(HalDevice *parent, di_node_t node, char *
static HalDevice *devinfo_ide_device_add(HalDevice *parent, di_node_t node, char
static HalDevice *devinfo_ide_storage_add(HalDevice *parent, di_node_t node, cha
HalDevice *devinfo_scsi_add(HalDevice *parent, di_node_t node, char *devfs_path,
static HalDevice *devinfo_scsi_storage_add(HalDevice *parent, di_node_t node, ch
HalDevice *devinfo_blkdev_add(HalDevice *parent, di_node_t node, char *devfs_pat
static HalDevice *devinfo_blkdev_storage_add(HalDevice *parent, di_node_t node,
HalDevice *devinfo_floppy_add(HalDevice *parent, di_node_t node, char *devfs_pat
static void devinfo_floppy_add_volume(HalDevice *parent, di_node_t node);
static HalDevice *devinfo_lofi_add(HalDevice *parent, di_node_t node, char *devf
static void devinfo_lofi_add_minor(HalDevice *parent, di_node_t node, char *mino
static void devinfo_storage_minors(HalDevice *parent, di_node_t node, gchar *dev
static struct devinfo_storage_minor *devinfo_storage_new_minor(char *maindev_pat
    char *devlink, dev_t dev, int dosnum);
static void devinfo_storage_free_minor(struct devinfo_storage_minor *m);
HalDevice *devinfo_volume_add(HalDevice *parent, di_node_t node, devinfo_storage
static void devinfo_volume_preprobing_done(HalDevice *d, gpointer userdata1, gpo
static void devinfo_volume_hotplug_begin_add (HalDevice *d, HalDevice *parent, D
static void devinfo_storage_hotplug_begin_add (HalDevice *d, HalDevice *parent,
static void devinfo_storage_probing_done (HalDevice *d, guint32 exit_type, gint
const gchar *devinfo_volume_get_prober (HalDevice *d, int *timeout);
const gchar *devinfo_storage_get_prober (HalDevice *d, int *timeout);

static char *devinfo_scsi_dtype2str(int dtype);
static char *devinfo_volume_get_slice_name (char *devlink);
static gboolean dos_to_dev(char *path, char **devpath, int *partnum);
static gboolean is_dos_path(char *path, int *partnum);

static void devinfo_storage_set_nicknames (HalDevice *d);

DevinfoDevHandler devinfo_ide_handler = {
        devinfo_ide_add,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL
};
DevinfoDevHandler devinfo_scsi_handler = {
        devinfo_scsi_add,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL
};
DevinfoDevHandler devinfo_blkdev_handler = {
        devinfo_blkdev_add,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL
};
DevinfoDevHandler devinfo_floppy_handler = {
        devinfo_floppy_add,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL
};
DevinfoDevHandler devinfo_lofi_handler = {
        devinfo_lofi_add,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL
};
DevinfoDevHandler devinfo_storage_handler = {
        NULL,
        NULL,
        devinfo_storage_hotplug_begin_add,
        NULL,
        devinfo_storage_probing_done,
        devinfo_storage_get_prober
};
DevinfoDevHandler devinfo_volume_handler = {
        NULL,
        NULL,
        devinfo_volume_hotplug_begin_add,
        NULL,
        NULL,
        devinfo_volume_get_prober
};

/* IDE */

HalDevice *
devinfo_ide_add(HalDevice *parent, di_node_t node, char *devfs_path, char *devic
{
        char    *s;

        if ((device_type != NULL) && (strcmp(device_type, "ide") == 0)) {
                return (devinfo_ide_host_add(parent, node, devfs_path));
        }

        if ((di_prop_lookup_strings (DDI_DEV_T_ANY, node, "class", &s) > 0) &&
            (strcmp (s, "dada") == 0)) {
                return (devinfo_ide_device_add(parent, node, devfs_path));
        }

        return (NULL);
}

static HalDevice *
devinfo_ide_host_add(HalDevice *parent, di_node_t node, char *devfs_path)
{
        HalDevice *d;

        d = hal_device_new ();

        devinfo_set_default_properties (d, parent, node, devfs_path);
        hal_device_property_set_string (d, "info.product", "IDE host controller"
        hal_device_property_set_string (d, "info.subsystem", "ide_host");
        hal_device_property_set_int (d, "ide_host.number", 0); /* XXX */

        devinfo_add_enqueue (d, devfs_path, &devinfo_ide_handler);

        return (d);
}

static HalDevice *
devinfo_ide_device_add(HalDevice *parent, di_node_t node, char *devfs_path)
```

```
{
        HalDevice *d;

        d = hal_device_new();

        devinfo_set_default_properties (d, parent, node, devfs_path);
        hal_device_property_set_string (parent, "info.product", "IDE device");
        hal_device_property_set_string (parent, "info.subsystem", "ide");
        hal_device_property_set_int (parent, "ide.host", 0); /* XXX */
        hal_device_property_set_int (parent, "ide.channel", 0);

        devinfo_add_enqueue (d, devfs_path, &devinfo_ide_handler);

        return (devinfo_ide_storage_add (d, node, devfs_path));
}

static HalDevice *
devinfo_ide_storage_add(HalDevice *parent, di_node_t node, char *devfs_path)
{
        HalDevice *d;
        char    *s;
        int     *i;
        char    *driver_name;
        char    udi[HAL_PATH_MAX];

        if ((driver_name = di_driver_name (node)) == NULL) {
                return (NULL);
        }

        d = hal_device_new ();

        devinfo_set_default_properties (d, parent, node, devfs_path);
        hal_device_property_set_string (d, "info.category", "storage");

        hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
                "%s/%s%d", hal_device_get_udi (parent), driver_name, di_instance
        hal_device_set_udi (d, udi);
        hal_device_property_set_string (d, "info.udi", udi);
        PROP_STR(d, node, s, "devid", "info.product");

        hal_device_add_capability (d, "storage");
        hal_device_property_set_string (d, "storage.bus", "ide");
        hal_device_property_set_int (d, "storage.lun", 0);
        hal_device_property_set_string (d, "storage.drive_type", "disk");

        PROP_BOOL(d, node, i, "hotpluggable", "storage.hotpluggable");
        PROP_BOOL(d, node, i, "removable-media", "storage.removable");

        hal_device_property_set_bool (d, "storage.media_check_enabled", FALSE);

        /* XXX */
        hal_device_property_set_bool (d, "storage.requires_eject", FALSE);

        hal_device_add_capability (d, "block");

        devinfo_storage_minors (d, node, (char *)devfs_path, FALSE);

        return (d);
}

/* SCSI */

HalDevice *
devinfo_scsi_add(HalDevice *parent, di_node_t node, char *devfs_path, char *devi
{
        int     *i;
        char    *driver_name;
        HalDevice *d;
        char    udi[HAL_PATH_MAX];

        driver_name = di_driver_name (node);
        if ((driver_name == NULL) || (strcmp (driver_name, "sd") != 0)) {
                return (NULL);
        }

        d = hal_device_new ();

        devinfo_set_default_properties (d, parent, node, devfs_path);
        hal_device_property_set_string (d, "info.subsystem", "scsi");

        hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
                "%s/%s%d", hal_device_get_udi (parent), di_node_name(node), di_i
        hal_device_set_udi (d, udi);
        hal_device_property_set_string (d, "info.udi", udi);

        hal_device_property_set_int (d, "scsi.host",
                hal_device_property_get_int (parent, "scsi_host.host"));
        hal_device_property_set_int (d, "scsi.bus", 0);
        PROP_INT(d, node, i, "target", "scsi.target");
        PROP_INT(d, node, i, "lun", "scsi.lun");
        hal_device_property_set_string (d, "info.product", "SCSI Device");

        devinfo_add_enqueue (d, devfs_path, &devinfo_scsi_handler);

        return (devinfo_scsi_storage_add (d, node, devfs_path));
}

static HalDevice *
devinfo_scsi_storage_add(HalDevice *parent, di_node_t node, char *devfs_path)
{
        HalDevice *d;
        int     *i;
        char    *s;
        char    udi[HAL_PATH_MAX];

        d = hal_device_new ();

        devinfo_set_default_properties (d, parent, node, devfs_path);
        hal_device_property_set_string (d, "info.category", "storage");

        hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
                "%s/sd%d", hal_device_get_udi (parent), di_instance (node));
        hal_device_set_udi (d, udi);
        hal_device_property_set_string (d, "info.udi", udi);
        PROP_STR(d, node, s, "inquiry-product-id", "info.product");

        hal_device_add_capability (d, "storage");

        hal_device_property_set_int (d, "storage.lun",
                hal_device_property_get_int (parent, "scsi.lun"));
        PROP_BOOL(d, node, i, "hotpluggable", "storage.hotpluggable");
        PROP_BOOL(d, node, i, "removable-media", "storage.removable");
        hal_device_property_set_bool (d, "storage.requires_eject", FALSE);

        /*
         * We have to enable polling not only for drives with removable media,
         * but also for hotpluggable devices, because when a disk is
         * unplugged while busy/mounted, there is not sysevent generated.
         * Instead, the HBA driver (scsa2usb, scsa1394) will notify sd driver
         * and the latter will report DKIO_DEV_GONE via DKIOCSTATE ioctl.
         * So we have to enable media check so that hald-addon-storage notices
         * the "device gone" condition and unmounts all associated volumes.
```

```
         */
        hal_device_property_set_bool (d, "storage.media_check_enabled",
            ((di_prop_lookup_ints(DDI_DEV_T_ANY, node, "removable-media", &i) >=
            (di_prop_lookup_ints(DDI_DEV_T_ANY, node, "hotpluggable", &i) >= 0))

        if (di_prop_lookup_ints(DDI_DEV_T_ANY, node, "inquiry-device-type",
            &i) > 0) {
                s = devinfo_scsi_dtype2str (*i);
                hal_device_property_set_string (d, "storage.drive_type", s);

                if (strcmp (s, "cdrom") == 0) {
                        hal_device_add_capability (d, "storage.cdrom");
                        hal_device_property_set_bool (d, "storage.no_partitions_
                        hal_device_property_set_bool (d, "storage.requires_eject
                }
        }

        hal_device_add_capability (d, "block");

        devinfo_storage_minors (d, node, devfs_path, FALSE);

        return (d);
}

static char *
devinfo_scsi_dtype2str(int dtype)
{
        char *dtype2str[] = {
                "disk"  ,         /* DTYPE_DIRECT         0x00 */
                "tape"  ,         /* DTYPE_SEQUENTIAL     0x01 */
                "printer",         /* DTYPE_PRINTER        0x02 */
                "processor",         /* DTYPE_PROCESSOR      0x03 */
                "worm"  ,         /* DTYPE_WORM           0x04 */
                "cdrom" ,         /* DTYPE_RODIRECT       0x05 */
                "scanner",         /* DTYPE_SCANNER        0x06 */
                "cdrom" ,         /* DTYPE_OPTICAL        0x07 */
                "changer",         /* DTYPE_CHANGER        0x08 */
                "comm"  ,         /* DTYPE_COMM           0x09 */
                "scsi"  ,         /* DTYPE_???            0x0A */
                "scsi"  ,         /* DTYPE_???            0x0B */
                "array_ctrl",         /* DTYPE_ARRAY_CTRL     0x0C */
                "esi"   ,         /* DTYPE_ESI            0x0D */
                "disk"            /* DTYPE_RBC            0x0E */
        };

        if (dtype < NELEM(dtype2str)) {
                return (dtype2str[dtype]);
        } else {
                return ("scsi");
        }

}

/* blkdev */

HalDevice *
devinfo_blkdev_add(HalDevice *parent, di_node_t node, char *devfs_path, char *de
{
        int     *i;
        char    *driver_name;
        HalDevice *d;
        char    udi[HAL_PATH_MAX];

        driver_name = di_driver_name (node);
        if ((driver_name == NULL) || (strcmp (driver_name, "blkdev") != 0)) {
                return (NULL);
        }

        d = hal_device_new ();

        devinfo_set_default_properties (d, parent, node, devfs_path);
        hal_device_property_set_string (d, "info.subsystem", "pseudo");

        hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
                "%s/%s%d", hal_device_get_udi (parent), di_node_name(node), di_i
        hal_device_set_udi (d, udi);
        hal_device_property_set_string (d, "info.udi", udi);
        hal_device_property_set_string (d, "info.product", "Block Device");

        devinfo_add_enqueue (d, devfs_path, &devinfo_blkdev_handler);

        return (devinfo_blkdev_storage_add (d, node, devfs_path));
}

static HalDevice *
devinfo_blkdev_storage_add(HalDevice *parent, di_node_t node, char *devfs_path)
{
        HalDevice *d;
        char    *driver_name;
        int     *i;
        char    *s;
        char    udi[HAL_PATH_MAX];

        d = hal_device_new ();

        devinfo_set_default_properties (d, parent, node, devfs_path);
        hal_device_property_set_string (d, "info.category", "storage");

        hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
                "%s/blkdev%d", hal_device_get_udi (parent), di_instance (node));
        hal_device_set_udi (d, udi);
        hal_device_property_set_string (d, "info.udi", udi);

        hal_device_add_capability (d, "storage");

        hal_device_property_set_int (d, "storage.lun", 0);

        PROP_BOOL(d, node, i, "hotpluggable", "storage.hotpluggable");
        PROP_BOOL(d, node, i, "removable-media", "storage.removable");

        hal_device_property_set_bool (d, "storage.requires_eject", FALSE);
        hal_device_property_set_bool (d, "storage.media_check_enabled", TRUE);
        hal_device_property_set_string (d, "storage.drive_type", "disk");

        hal_device_add_capability (d, "block");

        devinfo_storage_minors (d, node, devfs_path, FALSE);

        return (d);
}

/* floppy */

HalDevice *
devinfo_floppy_add(HalDevice *parent, di_node_t node, char *devfs_path, char *de
{
        char    *driver_name;
        char    *raw;
        char    udi[HAL_PATH_MAX];
        di_devlink_handle_t devlink_hdl;
        int     major;
        di_minor_t minor;
```

```
        dev_t   dev;
        HalDevice *d = NULL;
        char    *minor_path = NULL;
        char    *devlink = NULL;

        driver_name = di_driver_name (node);
        if ((driver_name == NULL) || (strcmp (driver_name, "fd") != 0)) {
                return (NULL);
        }

        /*
         * The only minor node we’re interested in is /dev/diskette*
         */
        major = di_driver_major(node);
        if ((devlink_hdl = di_devlink_init(NULL, 0)) == NULL) {
                return (NULL);
        }
        minor = DI_MINOR_NIL;
        while ((minor = di_minor_next(node, minor)) != DI_MINOR_NIL) {
                dev = di_minor_devt(minor);
                if ((major != major(dev)) ||
                    (di_minor_type(minor) != DDM_MINOR) ||
                    (di_minor_spectype(minor) != S_IFBLK) ||
                    ((minor_path = di_devfs_minor_path(minor)) == NULL)) {
                        continue;
                }
                if ((devlink = get_devlink(devlink_hdl, "diskette.+" , minor_pat
                        break;
                }
                di_devfs_path_free (minor_path);
                minor_path = NULL;
                free(devlink);
                devlink = NULL;
        }
        di_devlink_fini (&devlink_hdl);

        if ((devlink == NULL) || (minor_path == NULL)) {
                HAL_INFO (("floppy devlink not found %s", devfs_path));
                goto out;
        }

        d = hal_device_new ();

        devinfo_set_default_properties (d, parent, node, devfs_path);
        hal_device_property_set_string (d, "info.category", "storage");
        hal_device_add_capability (d, "storage");
        hal_device_property_set_string (d, "storage.bus", "platform");
        hal_device_property_set_bool (d, "storage.hotpluggable", FALSE);
        hal_device_property_set_bool (d, "storage.removable", TRUE);
        hal_device_property_set_bool (d, "storage.requires_eject", TRUE);
        hal_device_property_set_bool (d, "storage.media_check_enabled", FALSE);
        hal_device_property_set_string (d, "storage.drive_type", "floppy");

        hal_device_add_capability (d, "block");
        hal_device_property_set_bool (d, "block.is_volume", FALSE);
        hal_device_property_set_int (d, "block.major", major(dev));
        hal_device_property_set_int (d, "block.minor", minor(dev));
        hal_device_property_set_string (d, "block.device", devlink);
        raw = dsk_to_rdsk (devlink);
        hal_device_property_set_string (d, "block.solaris.raw_device", raw);
        free (raw);

        devinfo_add_enqueue (d, devfs_path, &devinfo_storage_handler);

        /* trigger initial probe-volume */
        devinfo_floppy_add_volume(d, node);
```

```
out:
        di_devfs_path_free (minor_path);
        free(devlink);

        return (d);
}

static void
devinfo_floppy_add_volume(HalDevice *parent, di_node_t node)
{
        char    *devlink;
        char    *devfs_path;
        int     minor, major;
        dev_t   dev;
        struct devinfo_storage_minor *m;

        devfs_path = (char *)hal_device_property_get_string (parent, "solaris.de
        devlink = (char *)hal_device_property_get_string (parent, "block.device"
        major = hal_device_property_get_int (parent, "block.major");
        minor = hal_device_property_get_int (parent, "block.minor");
        dev = makedev (major, minor);

        m = devinfo_storage_new_minor (devfs_path, WHOLE_DISK, devlink, dev, -1)
        devinfo_volume_add (parent, node, m);
        devinfo_storage_free_minor (m);
}

/*
 * After reprobing storage, reprobe its volumes.
 */
static void
devinfo_floppy_rescan_probing_done (HalDevice *d, guint32 exit_type, gint return
    char **error, gpointer userdata1, gpointer userdata2)
{
        void *end_token = (void *) userdata1;
        const char *devfs_path;
        di_node_t node;
        HalDevice *v;

        if (!hal_device_property_get_bool (d, "storage.removable.media_available
                HAL_INFO (("no floppy media", hal_device_get_udi (d)));

                /* remove child (can only be single volume) */
                if (((v = hal_device_store_match_key_value_string (hald_get_gdl(
                    "info.parent", hal_device_get_udi (d))) != NULL) &&
                    ((devfs_path = hal_device_property_get_string (v,
                    "solaris.devfs_path")) != NULL)) {
                        devinfo_remove_enqueue ((char *)devfs_path, NULL);
                }
        } else {
                HAL_INFO (("floppy media found", hal_device_get_udi (d)));

                if ((devfs_path = hal_device_property_get_string(d, "solaris.dev
                        HAL_INFO (("no devfs_path", hal_device_get_udi (d)));
                        hotplug_event_process_queue ();
                        return;
                }
                if ((node = di_init (devfs_path, DINFOCPYALL)) == DI_NODE_NIL) {
                        HAL_INFO (("di_init %s failed %d", devfs_path, errno));
                        hotplug_event_process_queue ();
                        return;
                }

                devinfo_floppy_add_volume (d, node);
```

```
                di_fini (node);
        }

        hotplug_event_process_queue ();
}

/* lofi */

HalDevice *
devinfo_lofi_add(HalDevice *parent, di_node_t node, char *devfs_path, char *devi
{
        return (devinfo_lofi_add_major(parent,node, devfs_path, device_type, FAL
}

HalDevice *
devinfo_lofi_add_major(HalDevice *parent, di_node_t node, char *devfs_path, char
    gboolean rescan, HalDevice *lofi_d)
{
        char    *driver_name;
        HalDevice *d = NULL;
        char    udi[HAL_PATH_MAX];
        di_devlink_handle_t devlink_hdl;
        int     major;
        di_minor_t minor;
        dev_t   dev;
        char    *minor_path = NULL;
        char    *devlink = NULL;

        driver_name = di_driver_name (node);
        if ((driver_name == NULL) || (strcmp (driver_name, "lofi") != 0)) {
                return (NULL);
        }

        if (!rescan) {
                d = hal_device_new ();

                devinfo_set_default_properties (d, parent, node, devfs_path);
                hal_device_property_set_string (d, "info.subsystem", "pseudo");

                hal_util_compute_udi (hald_get_gdl (), udi, sizeof (udi),
                        "%s/%s%d", hal_device_get_udi (parent), di_node_name(nod
                hal_device_set_udi (d, udi);
                hal_device_property_set_string (d, "info.udi", udi);

                devinfo_add_enqueue (d, devfs_path, &devinfo_lofi_handler);
        } else {
                d = lofi_d;
        }

        /*
         * Unlike normal storage, as in devinfo_storage_minors(), where
         * sd instance -> HAL storage, sd minor node -> HAL volume,
         * lofi always has one instance, lofi minor -> HAL storage.
         * lofi storage never has slices, but it can have
         * embedded pcfs partitions that fstyp would recognize
         */
        major = di_driver_major(node);
        if ((devlink_hdl = di_devlink_init(NULL, 0)) == NULL) {
                return (d);
        }
        minor = DI_MINOR_NIL;
        while ((minor = di_minor_next(node, minor)) != DI_MINOR_NIL) {
                dev = di_minor_devt(minor);
                if ((major != major(dev)) ||
                    (di_minor_type(minor) != DDM_MINOR) ||
                    (di_minor_spectype(minor) != S_IFBLK) ||
                    ((minor_path = di_devfs_minor_path(minor)) == NULL)) {
                        continue;
                }
                if ((devlink = get_devlink(devlink_hdl, NULL, minor_path)) == NU
                        di_devfs_path_free (minor_path);
                        continue;
                }

                if (!rescan ||
                    (hal_device_store_match_key_value_string (hald_get_gdl (),
                    "solaris.devfs_path", minor_path) == NULL)) {
                        devinfo_lofi_add_minor(d, node, minor_path, devlink, dev
                }

                di_devfs_path_free (minor_path);
                free(devlink);
        }
        di_devlink_fini (&devlink_hdl);

        return (d);
}

static void
devinfo_lofi_add_minor(HalDevice *parent, di_node_t node, char *minor_path, char
{
        HalDevice *d;
        char    *raw;
        char    *doslink;
        char    dospath[64];
        struct devinfo_storage_minor *m;
        int     i;

        /* add storage */
        d = hal_device_new ();

        devinfo_set_default_properties (d, parent, node, minor_path);
        hal_device_property_set_string (d, "info.category", "storage");
        hal_device_add_capability (d, "storage");
        hal_device_property_set_string (d, "storage.bus", "lofi");
        hal_device_property_set_bool (d, "storage.hotpluggable", TRUE);
        hal_device_property_set_bool (d, "storage.removable", FALSE);
        hal_device_property_set_bool (d, "storage.requires_eject", FALSE);
        hal_device_property_set_string (d, "storage.drive_type", "disk");
        hal_device_add_capability (d, "block");
        hal_device_property_set_int (d, "block.major", major(dev));
        hal_device_property_set_int (d, "block.minor", minor(dev));
        hal_device_property_set_string (d, "block.device", devlink);
        raw = dsk_to_rdsk (devlink);
        hal_device_property_set_string (d, "block.solaris.raw_device", raw);
        free (raw);
        hal_device_property_set_bool (d, "block.is_volume", FALSE);

        devinfo_add_enqueue (d, minor_path, &devinfo_storage_handler);

        /* add volumes: one on main device and a few pcfs candidates */
        m = devinfo_storage_new_minor(minor_path, WHOLE_DISK, devlink, dev, -1);
        devinfo_volume_add (d, node, m);
        devinfo_storage_free_minor (m);

        doslink = (char *)calloc (1, strlen (devlink) + sizeof (DOS_TEMPLATE) +
        doslink = (char *)calloc (1, strlen (devlink) + sizeof (":NNN") + 1);
        if (doslink != NULL) {
                for (i = 1; i < 16; i++) {
                        snprintf(dospath, sizeof (dospath), DOS_FORMAT, i);
                        sprintf(doslink, "%s"DOS_SEPERATOR"%d", devlink, i);
                        snprintf(dospath, sizeof (dospath), WHOLE_DISK":%d", i);
```

```
                        sprintf(doslink, "%s:%d", devlink, i);
                        m = devinfo_storage_new_minor(minor_path, dospath, dosli
                        devinfo_volume_add (d, node, m);
                        devinfo_storage_free_minor (m);
                }
                free (doslink);
        }
}
______unchanged_portion_omitted_

/*
 * Storage minor nodes are potential "volume" objects.
 * This function also completes building the parent object (main storage device)
 */
static void
devinfo_storage_minors(HalDevice *parent, di_node_t node, gchar *devfs_path, gbo
{
        di_devlink_handle_t devlink_hdl;
        gboolean is_cdrom;
        const char *whole_disk;
        int     major;
        di_minor_t minor;
        dev_t   dev;
        char    *minor_path = NULL;
        char    *maindev_path = NULL;
        char    *devpath, *devlink;
        int     doslink_len;
        char    *doslink;
        char    dospath[64];
        char    *slice;
        int     pathlen;
        int     i;
        char    *raw;
        boolean_t maindev_is_d0;
        GQueue  *mq;
        HalDevice *volume;
        struct devinfo_storage_minor *m;
        struct devinfo_storage_minor *maindev = NULL;

        /* for cdroms whole disk is always s2 */
        is_cdrom = hal_device_has_capability (parent, "storage.cdrom");
        whole_disk = is_cdrom ? "s2" : WHOLE_DISK;

        major = di_driver_major(node);

        /* the "whole disk" p0/s2/d0 node must come first in the hotplug queue
         * so we put other minor nodes on the local queue and move to the
         * hotplug queue up in the end
         */
        if ((mq = g_queue_new()) == NULL) {
                goto err;
        }
        if ((devlink_hdl = di_devlink_init(NULL, 0)) == NULL) {
                g_queue_free (mq);
                goto err;
        }
        minor = DI_MINOR_NIL;
        while ((minor = di_minor_next(node, minor)) != DI_MINOR_NIL) {
                dev = di_minor_devt(minor);
                if ((major != major(dev)) ||
                    (di_minor_type(minor) != DDM_MINOR) ||
                    (di_minor_spectype(minor) != S_IFBLK) ||
                    ((minor_path = di_devfs_minor_path(minor)) == NULL)) {
                        continue;
                }
                if ((devlink = get_devlink(devlink_hdl, NULL, minor_path)) == NU
                        di_devfs_path_free (minor_path);
                        continue;
                }

                slice = devinfo_volume_get_slice_name (devlink);
                if (strlen (slice) < 2) {
                        free (devlink);
                        di_devfs_path_free (minor_path);
                        continue;
                }

                /* ignore p1..N - we'll use p0:N instead */
                if ((strlen (slice) > 1) && (slice[0] == 'p') && isdigit(slice[1
                    ((atol(&slice[1])) > 0)) {
                        free (devlink);
                        di_devfs_path_free (minor_path);
                        continue;
                }

                m = devinfo_storage_new_minor(minor_path, slice, devlink, dev, -
                if (m == NULL) {
                        free (devlink);
                        di_devfs_path_free (minor_path);
                        continue;
                }

                /* main device is either s2/p0 or d0, the latter taking preceden
                if ((strcmp (slice, "d0") == 0) ||
                    (((strcmp (slice, whole_disk) == 0) && (maindev == NULL))))
                        if (maindev_path != NULL) {
                                di_devfs_path_free (maindev_path);
                        }
                        maindev_path = minor_path;
                        maindev = m;
                        g_queue_push_head (mq, maindev);
                } else {
                        di_devfs_path_free (minor_path);
                        g_queue_push_tail (mq, m);
                }

                free (devlink);
        }
        di_devlink_fini (&devlink_hdl);

        if (maindev == NULL) {
                /* shouldn't typically happen */
                while (!g_queue_is_empty (mq)) {
                        devinfo_storage_free_minor (g_queue_pop_head (mq));
                }
                goto err;
        }

        /* first enqueue main storage device */
        if (!rescan) {
                hal_device_property_set_int (parent, "block.major", major);
                hal_device_property_set_int (parent, "block.minor", minor(mainde
                hal_device_property_set_string (parent, "block.device", maindev-
                raw = dsk_to_rdsk (maindev->devlink);
                hal_device_property_set_string (parent, "block.solaris.raw_devic
                free (raw);
                hal_device_property_set_bool (parent, "block.is_volume", FALSE);
                hal_device_property_set_string (parent, "solaris.devfs_path", ma
                devinfo_add_enqueue (parent, maindev_path, &devinfo_storage_hand
        }

        /* add virtual dos volumes to enable pcfs probing */
```

```
        if (!is_cdrom) {
                doslink_len = strlen (maindev->devlink) + sizeof (DOS_TEMPLATE)
                doslink_len = strlen (maindev->devlink) + sizeof (":NNN") + 1;
                if ((doslink = (char *)calloc (1, doslink_len)) != NULL) {
                        for (i = 1; i < 16; i++) {
                                snprintf(dospath, sizeof (dospath), "%s"DOS_SEPE
                                snprintf(doslink, doslink_len, "%s"DOS_SEPERATOR
                                snprintf(dospath, sizeof (dospath), "%s:%d", mai
                                snprintf(doslink, doslink_len, "%s:%d", maindev-
                                m = devinfo_storage_new_minor(maindev_path, dosp
                                g_queue_push_tail (mq, m);
                        }
                        free (doslink);
                }
        }

        maindev_is_d0 = (strcmp (maindev->slice, "d0") == 0);

        /* enqueue all volumes */
        while (!g_queue_is_empty (mq)) {
                m = g_queue_pop_head (mq);

                /* if main device is d0, we'll throw away s2/p0 */
                if (maindev_is_d0 && (strcmp (m->slice, whole_disk) == 0)) {
                        devinfo_storage_free_minor (m);
                        continue;
                }
                /* don't do p0 on cdrom */
                if (is_cdrom && (strcmp (m->slice, "p0") == 0)) {
                        devinfo_storage_free_minor (m);
                        continue;
                }
                if (rescan) {
                        /* in rescan mode, don't reprobe existing volumes */
                        /* XXX detect volume removal? */
                        volume = hal_device_store_match_key_value_string (hald_g
                            "solaris.devfs_path", m->devpath);
                        if ((volume == NULL) || !hal_device_has_capability(volum
                                devinfo_volume_add (parent, node, m);
                        } else {
                                HAL_INFO(("rescan volume exists %s", m->devpath)
                        }
                } else {
                        devinfo_volume_add (parent, node, m);
                }
                devinfo_storage_free_minor (m);
        }

        if (maindev_path != NULL) {
                di_devfs_path_free (maindev_path);
        }

        return;

err:
        if (maindev_path != NULL) {
                di_devfs_path_free (maindev_path);
        }
        if (!rescan) {
                devinfo_add_enqueue (parent, devfs_path, &devinfo_storage_handle
        }
}
```

______***unchanged_portion_omitted_***

```
static gboolean
is_dos_path(char *path, int *partnum)
```

```
{
        char *p;

        if ((p = strrchr (path, DOS_SEPERATOR)) == NULL) {
        if ((p = strrchr (path, ':')) == NULL) {
                return (FALSE);
        }
        return ((*partnum = atoi(p + 1)) != 0);
}

static gboolean
dos_to_dev(char *path, char **devpath, int *partnum)
{
        char *p;

        if ((p = strrchr (path, DOS_SEPERATOR)) == NULL) {
        if ((p = strrchr (path, ':')) == NULL) {
                return (FALSE);
        }
        if ((*partnum = atoi(p + 1)) == 0) {
                return (FALSE);
        }
        p[0] = '\0';
        *devpath = strdup(path);
        p[0] = DOS_SEPERATOR;
        p[0] = ':';
        return (*devpath != NULL);
}
```
______unchanged_portion_omitted_