NAME

cam_open_device, **cam_open_spec_device**, **cam_open_btl**, **cam_open_pass**, **cam_close_device**, **cam_close_spec_device**, **cam_getccb**, **cam_send_ccb**, **cam_freeccb**, **cam_path_string**, **cam_device_dup**, **cam_device_copy**, **cam_get_device** - CAM user library

LIBRARY

Common Access Method User Library (libcam, -lcam)

SYNOPSIS

#include <stdio.h>
#include <camlib.h>

*struct cam_device **
cam_open_device(*const char *path, int flags*);

*struct cam_device **
cam_open_spec_device(*const char *dev_name, int unit, int flags, struct cam_device *device*);

*struct cam_device **
cam_open_btl(*path_id_t path_id, target_id_t target_id, lun_id_t target_lun, int flags,*
*struct cam_device *device*);

*struct cam_device **
cam_open_pass(*const char *path, int flags, struct cam_device *device*);

void
cam_close_device(*struct cam_device *dev*);

void
cam_close_spec_device(*struct cam_device *dev*);

*union ccb **
cam_getccb(*struct cam_device *dev*);

int
cam_send_ccb(*struct cam_device *device, union ccb *ccb*);

void
cam_freeccb(*union ccb *ccb*);

*char **
cam_path_string(*struct cam_device *dev, char *str, int len*);

*struct cam_device **
cam_device_dup(*struct cam_device *device*);

void
cam_device_copy(*struct cam_device *src, struct cam_device *dst*);

int
cam_get_device(*const char *path, char *dev_name, int devnamelen, int *unit*);

DESCRIPTION

The CAM library consists of a number of functions designed to aid in programming with the CAM subsystem described in cam(4). This man page covers the basic set of library functions. More functions are documented in the man pages listed below.

Many of the CAM library functions use the *cam_device* structure:

```
struct cam_device {
        char                    device_path[MAXPATHLEN+1];/*
                                                          * Pathname of the
                                                          * device given by the
                                                          * user. This may be
                                                          * null if the user
                                                          * states the device
                                                          * name and unit number
                                                          * separately.
                                                          */
        char                    given_dev_name[DEV_IDLEN+1];/*
                                                            * Device name given by
                                                            * the user.
                                                            */
        uint32_t  given_unit_number;            /*
                                                            * Unit number given by
                                                            * the user.
                                                            */
        char                    device_name[DEV_IDLEN+1];/*
                                                         * Name of the device,
                                                         * e.g., 'pass'
```

```
                                                        */
        uint32_t  dev_unit_num;       /* Unit number of the passthrough
                                       * device associated with this
                                       * particular device.
                                       */

        char                  sim_name[SIM_IDLEN+1];/*
                                       * Controller name, e.g., 'ahc'
                                       */
        uint32_t  sim_unit_number; /* Controller unit number */
        uint32_t  bus_id;              /* Controller bus number */
        lun_id_t  target_lun;          /* Logical Unit Number */
        target_id_t           target_id;  /* Target ID */
        path_id_t path_id;    /* System SCSI bus number */
        uint16_t  pd_type;    /* type of peripheral device */
        struct scsi_inquiry_data inq_data;  /* SCSI Inquiry data */
        uint8_t               serial_num[252]; /* device serial number */
        uint8_t               serial_num_len;  /* length of the serial number */
        uint8_t               sync_period;        /* Negotiated sync period */
        uint8_t               sync_offset;        /* Negotiated sync offset */
        uint8_t               bus_width;          /* Negotiated bus width */
        int                   fd;                 /* file descriptor for device */
};
```

cam_open_device() takes as arguments a string describing the device it is to open, and *flags* suitable for passing to open(2). The "path" passed in may actually be most any type of string that contains a device name and unit number to be opened. The string will be parsed by **cam_get_device**() into a device name and unit number. Once the device name and unit number are determined, a lookup is performed to determine the passthrough device that corresponds to the given device.

cam_open_spec_device() opens the pass(4) device that corresponds to the device name and unit number passed in. The *flags* should be flags suitable for passing to open(2). The *device* argument is optional. The user may supply pre-allocated space for the *cam_device* structure. If the *device* argument is NULL, **cam_open_spec_device**() will allocate space for the *cam_device* structure using malloc(3).

cam_open_btl() is similar to **cam_open_spec_device**(), except that it takes a SCSI bus, target and logical unit instead of a device name and unit number as arguments. The *path_id* argument is the CAM equivalent of a SCSI bus number. It represents the logical bus number in the system. The *flags* should be flags suitable for passing to open(2). As with **cam_open_spec_device**(), the *device* argument is optional.

cam_open_pass() takes as an argument the *path* of a pass(4) device to open. No translation or lookup is performed, so the path passed in must be that of a CAM pass(4) device. The *flags* should be flags suitable for passing to open(2). The *device* argument, as with **cam_open_spec_device**() and **cam_open_btl**(), should be NULL if the user wants the CAM library to allocate space for the *cam_device* structure.

cam_close_device() frees the *cam_device* structure allocated by one of the above open(2) calls, and closes the file descriptor to the passthrough device. This routine should not be called if the user allocated space for the *cam_device* structure. Instead, the user should call **cam_close_spec_device**().

cam_close_spec_device() merely closes the file descriptor opened in one of the open(2) routines described above. This function should be called when the *cam_device* structure was allocated by the caller, rather than the CAM library.

cam_getccb() allocates a prezeroed CCB using calloc(3) and sets fields in the CCB header using values from the *cam_device* structure.

cam_send_ccb() sends the given *ccb* to the *device* described in the *cam_device* structure.

cam_freeccb() frees CCBs allocated by **cam_getccb**(). If *ccb* is NULL, no action is taken.

cam_path_string() takes as arguments a *cam_device* structure, and a string with length *len*. It creates a colon-terminated printing prefix string similar to the ones used by the kernel. e.g.: "(cd0:ahc1:0:4:0): ". **cam_path_string**() will place at most *len*-1 characters into *str*. The *len*'th character will be the terminating '\0'.

cam_device_dup() operates in a fashion similar to strdup(3). It allocates space for a *cam_device* structure and copies the contents of the passed-in *device* structure to the newly allocated structure.

cam_device_copy() copies the *src* structure to *dst*.

cam_get_device() takes a *path* argument containing a string with a device name followed by a unit number. It then breaks the string down into a device name and unit number, and passes them back in *dev_name* and *unit*, respectively. **cam_get_device**() can handle strings of the following forms, at least:

/dev/foo1
foo0
nsa2

cam_get_device() is provided as a convenience function for applications that need to provide

functionality similar to **cam_open_device**().

RETURN VALUES

cam_open_device(), **cam_open_spec_device**(), **cam_open_btl**(), and **cam_open_pass**() return a pointer to a *cam_device* structure, or NULL if there was an error.

cam_getccb() returns an allocated and partially initialized CCB, or NULL if allocation of the CCB failed.

cam_send_ccb() returns a value of -1 if an error occurred, and *errno* is set to indicate the error.

cam_path_string() returns a filled printing prefix string as a convenience. This is the same *str* that is passed into **cam_path_string**().

cam_device_dup() returns a copy of the *device* passed in, or NULL if an error occurred.

cam_get_device() returns 0 for success, and -1 to indicate failure.

If an error is returned from one of the base CAM library functions described here, the reason for the error is generally printed in the global string *cam_errbuf* which is CAM_ERRBUF_SIZE characters long.

SEE ALSO

cam_cdbparse(3), pass(4), camcontrol(8)

HISTORY

The CAM library first appeared in FreeBSD 3.0.

AUTHORS

Kenneth Merry <*ken@FreeBSD.org*>

BUGS

cam_open_device() does not check to see if the *path* passed in is a symlink to something. It also does not check to see if the *path* passed in is an actual pass(4) device. The former would be rather easy to implement, but the latter would require a definitive way to identify a device node as a pass(4) device.

Some of the functions are possibly misnamed or poorly named.