NAME

csio_build, **csio_build_visit**, **csio_decode**, **csio_decode_visit**, **buff_decode**, **buff_decode_visit**, **csio_encode**, **csio_encode_visit**, **buff_encode_visit** - CAM user library SCSI buffer parsing routines

LIBRARY

Common Access Method User Library (libcam, -lcam)

SYNOPSIS

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

int
csio_build(*struct ccb_scsiio *csio, uint8_t *data_ptr, uint32_t dxfer_len, uint32_t flags, int retry_count, int timeout, const char *cmd_spec, ...*);

int
csio_build_visit(*struct ccb_scsiio *csio, uint8_t *data_ptr, uint32_t dxfer_len, uint32_t flags, int retry_count, int timeout, const char *cmd_spec, int (*arg_get)(void *hook, char *field_name), void *gethook*);

int
csio_decode(*struct ccb_scsiio *csio, const char *fmt, ...*);

int
csio_decode_visit(*struct ccb_scsiio *csio, const char *fmt, void (*arg_put)(void *hook, int letter, void *val, int count, char *name), void *puthook*);

int
buff_decode(*uint8_t *buff, size_t len, const char *fmt, ...*);

int
buff_decode_visit(*uint8_t *buff, size_t len, const char *fmt, void (*arg_put)(void *, int, void *, int, char *), void *puthook*);

int
csio_encode(*struct ccb_scsiio *csio, const char *fmt, ...*);

int
csio_encode_visit(*struct ccb_scsiio *csio, const char *fmt, int (*arg_get)(void *hook, char *field_name), void *gethook*);

int
buff_encode_visit(*uint8_t *buff, size_t len, const char *fmt,*
*int (*arg_get)(void *hook, char *field_name), void *gethook*);

DESCRIPTION

The CAM buffer/CDB encoding and decoding routines provide a relatively easy migration path for userland SCSI applications written with the similarly-named *scsireq_** functions from the old FreeBSD SCSI layer.

These functions may be used in new applications, but users may find it easier to use the various SCSI CCB building functions included with the cam(3) library, e.g., **cam_fill_csio**(), **scsi_start_stop**(), and **scsi_read_write**().

csio_build() builds up a *ccb_scsiio* structure based on the information provided in the variable argument list. It gracefully handles a NULL *data_ptr* argument passed to it.

dxfer_len is the length of the data phase; the data transfer direction is determined by the *flags* argument.

data_ptr is the data buffer used during the SCSI data phase. If no data is to be transferred for the SCSI command in question, this should be set to NULL. If there is data to transfer for the command, this buffer must be at least *dxfer_len* long.

flags are the flags defined in <*cam/cam_ccb.h*>:

```
/* Common CCB header */
/* CAM CCB flags */
typedef enum {
    CAM_CDB_POINTER       = 0x00000001,/* The CDB field is a pointer    */
    CAM_SCATTER_VALID     = 0x00000010,/* Scatter/gather list is valid  */
    CAM_DIS_AUTOSENSE     = 0x00000020,/* Disable autosense feature     */
    CAM_DIR_RESV          = 0x00000000,/* Data direction (00:reserved)  */
    CAM_DIR_IN            = 0x00000040,/* Data direction (01:DATA IN)   */
    CAM_DIR_OUT           = 0x00000080,/* Data direction (10:DATA OUT)  */
    CAM_DIR_NONE          = 0x000000C0,/* Data direction (11:no data)   */
    CAM_DIR_MASK          = 0x000000C0,/* Data direction Mask                */
    CAM_DEV_QFRZDIS       = 0x00000400,/* Disable DEV Q freezing             */
    CAM_DEV_QFREEZE       = 0x00000800,/* Freeze DEV Q on execution     */
    CAM_HIGH_POWER        = 0x00001000,/* Command takes a lot of power  */
    CAM_SENSE_PTR         = 0x00002000,/* Sense data is a pointer  */
    CAM_SENSE_PHYS        = 0x00004000,/* Sense pointer is physical addr*/
```

```
    CAM_TAG_ACTION_VALID  = 0x00008000,/* Use the tag action in this ccb*/
    CAM_PASS_ERR_RECOVER  = 0x00010000,/* Pass driver does err. recovery*/
    CAM_DIS_DISCONNECT    = 0x00020000,/* Disable disconnect                 */
    CAM_SG_LIST_PHYS      = 0x00040000,/* SG list has physical addrs.   */
    CAM_DATA_PHYS         = 0x00200000,/* SG/Buffer data ptrs are phys. */
    CAM_CDB_PHYS          = 0x00400000,/* CDB pointer is physical            */

/* Host target Mode flags */
    CAM_SEND_SENSE        = 0x08000000,/* Send sense data with status   */
    CAM_SEND_STATUS       = 0x80000000,/* Send status after data phase  */
} ccb_flags;
```

Multiple flags should be ORed together. Any of the CCB flags may be used, although it is worth noting several important ones here:

CAM_DIR_IN	This indicates that the operation in question is a read operation. i.e., data is being read from the SCSI device to the user-supplied buffer.
CAM_DIR_OUT	This indicates that the operation is a write operation. i.e., data is being written from the user-supplied buffer to the device.
CAM_DIR_NONE	This indicates that there is no data to be transferred for this command.
CAM_DEV_QFRZDIS	This flag disables device queue freezing as an error recovery mechanism.
CAM_PASS_ERR_RECOVER	This flag tells the pass(4) driver to enable error recovery. The default is to not perform error recovery, which means that the retry count will not be honored without this flag, among other things.
CAM_DATA_PHYS	This indicates that the address contained in *data_ptr* is a physical address, not a virtual address.

The *retry_count* tells the kernel how many times to retry the command in question. The retry count is ignored unless the pass(4) driver is told to enable error recovery via the CAM_PASS_ERR_RECOVER flag.

The *timeout* tells the kernel how long to wait for the given command to complete. If the timeout expires and the command has not completed, the CCB will be returned from the kernel with an appropriate error status.

cmd_spec is a CDB format specifier used to build up the SCSI CDB. This text string is made up of a list of field specifiers. Field specifiers specify the value for each CDB field (including indicating that the value be taken from the next argument in the variable argument list), the width of the field in bits or bytes, and an optional name. White space is ignored, and the pound sign ('#') introduces a comment that ends at the end of the current line.

The optional name is the first part of a field specifier and is in curly braces. The text in curly braces in this example are the names:

{PS} v:b1 {Reserved} 0:b1 {Page Code} v:b6 # Mode select page

This field specifier has two one bit fields and one six bit field. The second one bit field is the constant value 0 and the first one bit field and the six bit field are taken from the variable argument list. Multi byte fields are swapped into the SCSI byte order in the CDB and white space is ignored.

When the field is a hex value or the letter v, (e.g., *1A* or *v*) then a single byte value is copied to the next unused byte of the CDB. When the letter *v* is used the next integer argument is taken from the variable argument list and that value used.

A constant hex value followed by a field width specifier or the letter *v* followed by a field width specifier (e.g., *3:4*, *3:b4*, *3:i3*, *v:i3*) specifies a field of a given bit or byte width. Either the constant value or (for the V specifier) the next integer value from the variable argument list is copied to the next unused bits or bytes of the CDB.

A decimal number or the letter *b* followed by a decimal number field width indicates a bit field of that width. The bit fields are packed as tightly as possible beginning with the high bit (so that it reads the same as the SCSI spec), and a new byte of the CDB is started whenever a byte fills completely or when an *i* field is encountered.

A field width specifier consisting of the letter *i* followed by either 1, 2, 3 or 4 indicates a 1, 2, 3 or 4 byte integral value that must be swapped into SCSI byte order (MSB first).

For the *v* field specifier the next integer argument is taken from the variable argument list and that value is used swapped into SCSI byte order.

csio_build_visit() operates similarly to **csio_build**(), except that the values to substitute for variable arguments in *cmd_spec* are retrieved via the **arg_get**() function passed in to **csio_build_visit**() instead of via stdarg(3). The **arg_get**() function takes two arguments:

gethook is passed into the **arg_get**() function at each invocation. This enables the **arg_get**() function to keep some state in between calls without using global or static variables.

field_name is the field name supplied in *fmt*, if any.

csio_decode() is used to decode information from the data in phase of the SCSI transfer.

The decoding is similar to the command specifier processing of **csio_build**() except that the data is extracted from the data pointed to by *csio->data_ptr*. The stdarg list should be pointers to integers instead of integer values. A seek field type and a suppression modifier are added. The * suppression modifier (e.g., **i3* or **b4*) suppresses assignment from the field and can be used to skip over bytes or bits in the data, without having to copy them to a dummy variable in the arg list.

The seek field type *s* permits you to skip over data. This seeks to an absolute position (*s3*) or a relative position (*s+3*) in the data, based on whether or not the presence of the '+' sign. The seek value can be specified as *v* and the next integer value from the argument list will be used as the seek value.

csio_decode_visit() operates like **csio_decode**() except that instead of placing the decoded contents of the buffer in variadic arguments, the decoded buffer contents are returned to the user via the **arg_put**() function that is passed in. The **arg_put**() function takes several arguments:

hook The "hook" is a mechanism to allow the **arg_put**() function to save state in between calls.

letter is the letter describing the format of the argument being passed into the function.

val is a void pointer to the value being passed into the function.

count
is the size of the value being passed into the **arg_put**() function. The argument format determines the unit of measure.

name
This is a text description of the field, if one was provided in the *fmt*.

buff_decode() decodes an arbitrary data buffer using the method described above for **csio_decode**().

buff_decode_visit() decodes an arbitrary data buffer using the method described above for **csio_decode_visit**().

csio_encode() encodes the *data_ptr* portion (not the CDB!) of a *ccb_scsiio* structure, using the method described above for **csio_build**().

csio_encode_visit() encodes the *data_ptr* portion (not the CDB!) of a *ccb_scsiio* structure, using the

method described above for **csio_build_visit**().

buff_encode_visit() encodes an arbitrary data pointer, using the method described above for **csio_build_visit**().

RETURN VALUES

csio_build(), **csio_build_visit**(), **csio_encode**(), **csio_encode_visit**(), and **buff_encode_visit**() return the number of fields processed.

csio_decode(), **csio_decode_visit**(), **buff_decode**(), and **buff_decode_visit**() return the number of assignments performed.

SEE ALSO

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

HISTORY

The CAM versions of these functions are based upon similar functions implemented for the old FreeBSD SCSI layer. The encoding/decoding functions in the old SCSI code were written by Peter Dufault <*dufault@hda.com*>.

Many systems have comparable interfaces to permit a user to construct a SCSI command in user space.

The old *scsireq* data structure was almost identical to the SGI /dev/scsi data structure. If anyone knows the name of the authors it should go here; Peter Dufault first read about it in a 1989 Sun Expert magazine.

The new CCB data structures are derived from the CAM-2 and CAM-3 specifications.

Peter Dufault implemented a clone of SGI's interface in 386BSD that led to the original FreeBSD SCSI library and the related kernel ioctl. If anyone needs that for compatibility, contact *dufault@hda.com*.

AUTHORS

Kenneth Merry <*ken@FreeBSD.org*> implemented the CAM versions of these encoding and decoding functions. This current work is based upon earlier work by Peter Dufault <*dufault@hda.com*>.

BUGS

There should probably be a function that encodes both the CDB and the data buffer portions of a SCSI CCB. I discovered this while implementing the arbitrary command execution code in camcontrol(8), but I have not yet had time to implement such a function.

Some of the CCB flag descriptions really do not belong here. Rather they belong in a generic CCB man page. Since that man page has not yet been written, the shorter descriptions here will have to suffice.