7  SCSI Reference Pages
SCSI Reference Pages

SCSI Services is a set of commonly used SCSI functions that allow device and interface drivers to be much smaller and more supportable. In addition to providing most commonly used SCSI functions, WSIO SCSI Services also provides a supported pass-through mechanism.
SCSI Functions
NAME

dd_close(SCSI_DRV) SCSI Function - Driver-specific processing during close

SYNOPSIS

void dd_close (dev_t dev);

PARAMETERS

dev The device number.

DESCRIPTION

The dd_close() SCSI function is provided by the driver writer. It can have any unique name. You pass the name to SCSI Services by specifying it in the dd_close field of the scsi_ddsw structure.

See HP-UX Driver Development Guide for details;

RETURN VALUES

dd_close() does not return a value.

CONSTRAINTS

SEE ALSO

scsi_lun_close(SCSI3), scsi_ddsw(SCSI4)
NAME

\texttt{dd\_done} (SCSI\_DRV) SCSI Function - Driver-specific post-I/O processing

SYNOPSIS

\begin{verbatim}
int dd_done (struct buf *bp);
\end{verbatim}

PARAMETERS

bp \hspace{1cm} buf structure.

DESCRIPTION

The \texttt{dd\_done}() routine is provided by the driver writer. It can have any
unique name. You pass the name to SCSI Services by specifying it in the
\texttt{dd\_done} field of the \texttt{scsi\_ddsw} structure.

See HP-UX Driver Development Guide for details;

RETURN VALUES

\texttt{dd\_done}() is declared as returning int; however, its return is not used by
SCSI services.

SEE ALSO

\texttt{biodone(KER2)}, \texttt{scsi\_action(SCSI3)}, \texttt{scsi\_ddsw(SCSI4)}
NAME

dd_ioctl (SCSI_DRV) SCSI Function – Driver-specific ioctl processing

SYNOPSIS

int dd_ioctl (dev_t dev, int cmd, caddr_t data, int flags);

PARAMETERS

dev Device number of the associated device

cmd Driver command to execute

data Pointer to the command arguments

flags The file-access flags.

DESCRIPTION

The dd_ioctl() routine is provided by the driver writer. It can have any unique name. You pass the name to SCSI Services by specifying it in the dd_ioctl field of the scsi_ddsw structure.

See HP-UX Driver Development Guide for details;

RETURN VALUES

dd_ioctl() is expected to return the following values:

0 Successful completion.

<>0 Error. The value is expected to be an errno value.

CONSTRAINTS

SEE ALSO

scsi_cmd(SCSI3), scsi_init_inquiry_data(SCSI3),
scsi_ioctl(SCSI3)
NAME

`dd_ioctl_okay` (SCSI_DRV) SCSI Function – Disallow ioctl commands through the pass-through driver

SYNOPSIS

```c
int dd_ioctl_okay (dev_t dev, int cmd, caddr_t data, int flags);
```

PARAMETERS

- `dev` Device number of the associate device
- `cmd` Driver command to execute
- `data` Pointer to command parameter
- `flags` The file-access flags

DESCRIPTION

The `dd_ioctl_okay()` SCSI function is provided by the driver writer. It can have any unique name. You pass the name to SCSI Services by specifying it in the `dd_ioctl_okay` field of the `scsi_ddsw` structure. See HP-UX Driver Development Guide for details;

RETURN VALUES

- `dd_ioctl_okay()` is expected to return the following values:
  - `PT_OKAY` Successful completion.
  - `0` Error.

CONSTRAINTS

SEE ALSO

`scsi_ioctl(SCSI3)`
NAME

dd_open (SCSI_DRV) SCSI Function – Driver-specific open processing

SYNOPSIS

dd_open (dev, oflags);

PARAMETERS

    dev        The device number of the device to be opened
    oflags     The flags passed in the open call

DESCRIPTION

The dd_open() SCSI function is provided by the driver writer. It can have any unique name. You pass the name to SCSI Services by specifying it in the dd_open field of the scsi_ddsw structure.

See HP-UX Driver Development Guide for details;

RETURN VALUES

dd_open() is expected to return the following values:

    0          Successful completion.
    <>0        Error. The value is expected to be an errno value.

CONSTRAINTS

SEE ALSO

m_scsi_lun(SCSI3), major(KER2), scsi_cmdx(SCSI3),
scsi_init_inquiry_data(SCSI3), scsi_lun_open(SCSI3)
NAME

\texttt{dd\_pass\_thru\_done} (SCSI_DRV) SCSI Function - Driver-specific notation of pass-through I/O

SYNOPSIS

\begin{verbatim}
int dd_pass_thru_done (struct buf * bp);
\end{verbatim}

PARAMETERS

bp \hspace{1cm} buf structure.

DESCRIPTION

The \texttt{dd\_pass\_thru\_done}() routine is provided by the driver writer. It can have any unique name. You pass the name to SCSI Services by specifying it in the \texttt{dd\_pass\_thru\_done} field of the \texttt{scsi\_ddsw} structure. See HP-UX Driver Development Guide for details;

CONSTRAINTS

RETURN VALUES

\texttt{dd\_pass\_thru\_done}() is declared as returning \texttt{int}; however, the return value is not used by SCSI services.
NAME

`dd_pass_thru_okay` (SCSI_DRV) SCSI Function – Driver-specific control of pass-through I/O

SYNOPSIS

`dd_pass_thru_okay (dev_t dev, struct sctl_io * sctl_io);`

PARAMETERS

- `dev` The device number
- `sctl_io` Struct containing ioctl information

DESCRIPTION

The `dd_pass_thru_okay()` routine is provided by the driver writer. It can have any unique name. You pass the name to SCSI Services by specifying it in the `dd_pass_thru_okay` field of the `scsi_ddsw` structure. See HP-UX Driver Development Guide for details;

RETURN VALUES

`dd_pass_thru_okay()` is expected to return the following values:

- `PT_OKAY` Successful completion.
- `0` Error.

CONSTRAINTS
NAME

dd_read(SCSI_DRV) SCSI Function - Driver-specific read function

SYNOPSIS

int dd_read (dev_t dev, struct uio * uio);

PARAMETERS

dev The device number
uio Struct containing transfer information

DESCRIPTION

The dd_read() routine is provided by the driver writer. It can have any unique name. You pass the name to SCSI Services by specifying it in the dd_read field of the scsi_ddsw structure.

See HP-UX Driver Development Guide for details;

RETURN VALUES

dd_read() is expected to return the following values:

0 Successful completion.
<>0 Error. The value is expected to be an errno value.

CONSTRAINTS

SEE ALSO

scsi_read(SCSI3)
NAME

dd_start (SCSI_DRV) SCSI Function - Driver-specific start routine

SYNOPSIS

struct buf * (d_start) dd_start (struct scsi_lun *lp,
        struct scb *scb);

PARAMETERS

lp       The open LUN structure
scb      Struct containing state information for I/O

DESCRIPTION

The dd_start() routine is provided by the driver writer. It can have any
unique name. You pass the name to SCSI Services by specifying it in the
dd_start field of the scsi_ddsw structure.

See HP-UX Driver Development Guide for details;

RETURN VALUES

dd_start() is expected to return the following values:

struct buf *bp          Successful completion.
        NULL              Error.

CONSTRAINTS
NAME

\texttt{dd\_strategy} (SCSI\_DRV) SCSI Function \textendash{} Driver-specific I/O routine

SYNOPSIS

\begin{verbatim}
int dd_strategy (struct buf *bp struct scsi_lun *lp);
\end{verbatim}

PARAMETERS

\begin{itemize}
\item \texttt{struct buf *bp} \textendash{} transfer buf header
\end{itemize}

DESCRIPTION

The \texttt{dd\_strategy()} routine is provided by the driver writer. It can have any unique name. You pass the name to SCSI Services by specifying it in the \texttt{dd\_strategy} field of the \texttt{scsi\_ddsw} structure.

See HP-UX Driver Development Guide for details;

RETURN VALUES

\texttt{dd\_strategy()} is expected to return the following values:

\begin{itemize}
\item 0 \textendash{} Successful completion.
\item -1 \textendash{} Error.
\end{itemize}

WARNINGS

\texttt{dd\_strategy()} must exist (be defined as non-NULL in the \texttt{scsi\_ddsw} structure) if your driver calls \texttt{scsi\_strategy()}. \texttt{scsi\_strategy()} calls \texttt{dd\_strategy} while holding \texttt{lun\_lock}.

SEE ALSO

\begin{itemize}
\item \texttt{physio(KER2) dd\_read(SCSI\_DRV), dd\_write(SCSI\_DRV),}
\item \texttt{scsi\_enqueue(SCSI3), scsi\_strategy(SCSI3)}
\end{itemize}
NAME

dd_write (SCSI3) SCSI Function – dd_write – driver-specific write routine

SYNOPSIS

int dd_write (dev_t dev, struct uio * uio);

PARAMETERS

dev The device number
uio Structure containing transfer information

DESCRIPTION

The dd_write() routine is provided by the driver writer. It can have any unique name. You pass the name to SCSI Services by specifying it in the dd_write field of the scsi_ddsw structure.

See HP-UX Driver Development Guide for details;

RETURN VALUES

dd_write() is expected to return the following values:

0 Successful completion.
<0 Error. The value is expected to be an errno value.

SEE ALSO

physio(KER2), scsi_write(SCSI3)
NAME

m_scsi_lun(SCSI13) SCSI Function - Get scsi_lun pointer

SYNOPSIS

m_scsi_lun (dev_t dev);

PARAMETERS

dev The device number

DESCRIPTION

m_scsi_lun() returns the scsi_lun pointer corresponding to dev.

RETURN VALUES

m_scsi_lun() returns the following values:

0 Error.

<>0 Pointer to the SCSI logical unit associated with dev.
NAME

scsi_action (SCSI3) SCSI Function - Give I/O completion information to SCSI Services

SYNOPSIS

#include <sys/scsi_ctl.h>

scsi_action (struct buf *bp, int flags, int error, int msecs);

PARAMETERS

bp Transfer buf header

flags The following bit values can be combined for flags:

SA_ANY Wild card entry for matching parameters.

SA_DISABLE_TAGS Initiate the transition to nontagged operation for the device. This is used to recover from tagged queuing problems.

SA_DONE Call dd_done() and biodone().

SA_IGNORE_MAX_RETRIES Retry I/O independently of scb->max_retries. This is used when a command fails for a reason unrelated to the command, such as unit attention, power-on, or reset.

SA_LOG_IT_ALWAYS Always log an I/O attempt record to dmesg.

SA_LOG_IT_NEVER Never log an I/O attempt record to dmesg.

SA_LOG_IT_SOMETIMES Log an I/O attempt record to dmesg if !SCB_DONT_PRINT is true.

SA_NONE Value used for undefined fields.
SA_PANIC   Execute panic(error).
SA_REINIT  Go to reinitialization state.
SA_RETRY   Retry the I/O if scb->max_retries has not been exceeded.

The default is SA_DONE + SA_LOG_IT.Never.

DESCRIPTION

scsi_action() must ultimately be called after all I/O attempt completions (as in a retry situation). It exists only because there is too much information needed by SCSI Services from the device driver’s action routine to encode easily in a single integer return value. The arguments determine: whether or not the I/O attempt record is logged to the dmesg buffer, whether tags should be disabled, and whether to retry the I/O, consider it to be completed, or panic. It is either entered directly into the device driver’s status action list or called at the end of the function that is in the status action list.

scsi_action() is called by device drivers and SCSI services internally in both interrupt and process contexts.

The scsi_action() function appears not to have any real protection issues itself; it mainly operates on the request structures (buf and scb). The dd_done() function is called from scsi_action() and some of the device driver dd_done() functions do need the protection.

The constants and data structures used are specified in the header file, ../wsio/scsi_ctl.h.

Logging, as a result of SA_LOG_IT_ALWAYS or SA_LOG_IT_SOMETIMES sent to scsi_status(), causes scsi_log_io() to be invoked. This routine records the I/O attempt and its results in the dmesg buffer. Output is controlled by scsi_log_mask and scsi_log_nbytes. An attempt is made to conserve dmesg buffer space by outputting only differences between successive retries of the same I/O.

RETURN VALUES

scsi_action() returns the following values:
msecs if (flags & SA_RETRY)
-1 Normal Completion
panic() if (flags & SA_PANIC)

SEE ALSO

biodone(KER2), panic(KER2)
NAME

**scsi_cmd** (SCSI3) SCSI Function – Prepare driver-generated I/O requests

SYNOPSIS

```c
#include <sys/scsi_ctl.h>

scsi_cmdx (dev_t dev, ubit32 flags, int cdb_len,
        ubit8 * cdb, int nbytes, void * addr,
        ubit32 msecs ubit32 retries, int * pErr);
```

PARAMETERS

- **dev**: Device used to find correct LUN and target.
- **flags**: Read, 6-, 10-, or 12-byte **cdb**, or action.
- **cdb_len**: Length of the cdb 6,10,12.
- **cdb**: SCSI command data block.
- **nbytes**: If zero, there is no data phase.
- **addr**: Buffer for read data return.
- **msecs**: Assigned to sctl_io->max_msecs. Zero means no timeout.
- **retries**: Number of retries.
- **pErr**: If not NULL, then contains the error returned by the operation (in bp->b_error).

DESCRIPTION

The **scsi_cmd()** SCSI function is used for driver-generated I/O requests. It is a wrapper for **scsi_cmdx()** which it calls setting the two additional parameters to NULL and 0.

Used by device drivers and SCSI services internally, this function must be called in the process context and may block. The function is not called from within any critical section.

Refer to “NAME” for details.
RETURN VALUES

`s CSI_cmd()` returns the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Number of bytes transferred.</td>
</tr>
<tr>
<td>-1</td>
<td>Error</td>
</tr>
</tbody>
</table>

SEE ALSO

`biowait(KER2), scsi_ctl(7), scsi_init_inquiry_data(SCSI3), scsi_cmdx(SCSI3), scsi_strategy(SCSI3)`
NAME

scsi_cmdx (SCSI3) SCSI Function - Prepare driver-generated I/O requests

SYNOPSIS

```
#include <sys/scsi_ctl.h>

scsi_cmdx (dev_t dev, int flags, int cdb_len, u_char * cdb,
           int nbytes, void * addr, u_int msecs,
           u_int retries, int * Err,
           struct status_action *sa, int n);
```

PARAMETERS

- **addr**: Buffer for read data return.
- **cdb**: SCSI command data block.
- **cdb_len**: Length of the cdb 6,10,12.
- **dev**: Device used to find correct LUN and target.
- **Err**: If not NULL, then contains the error returned by the operation (in bp->b_error)
- **flags**: Read, 6-, 10-, or 12-byte cdb, or action.
- **msecs**: Assigned to sctl_io->max_msecs. Zero means no timeout.
- **n**: Status count.
- **nbytes**: If zero, there is no data phase.
- **sa**: If NULL, there is no action to match or take.
- **retries**: Number of retries.

DESCRIPTION

The `scsi_cmdx()` SCSI function is used for driver-generated I/O requests. It creates and builds a `sctl_io` and a `bp`, attaches the `sctl_io` to the `bp`, forwards the `bp` to the `scsi_strategy()` routine, and cleans up when the I/O is completed.
The `scsi_cmdx()` routine is used by drivers to perform initialization or `ioctl` types of operations. It is also used within SCSI Services to perform `scsi_init_inquiry_data()`, `scsi_mode_sense()`, and `scsi_mode_select()`.

Used by device drivers and SCSI services internally, this function must be called in the process context and may block. The function is not called from within any critical section.

`scsi_cmdx()` allocates a `bp` structure and a `sctl_io` structure. It sets `B_SCSI_CMD` in `bp->b_flags` and places a pointer to the `sctl_io` structure into `bp->b_offset`. For a detailed discussion, see the `sctl_io` portion of the SCSI pass-through driver in `scsi_ctl(7)`.

The parameter `max_msecs` is assigned to `sctl_io->max_msecs`, which itself is assigned to `scb->msecs`. Similarly, `max_retries` is assigned to `sctl_io->max_retries`, which itself is assigned to `scb->max_retries`.

To perform the I/O, `scsi_cmdx()` calls `scsi_strategy()`, then `scsi_iowait()`. Upon completion, it releases the `bp` and `sctl_io` structures, prior to returning to the caller.

**RETURN VALUES**

`scsi_cmdx()` returns the following values:

- **N** Number of bytes transferred.
- **-1** Error.

**SEE ALSO**

`biowait(KER2)`, `scsi_ctl(7)`, `scsi_init_inquiry_data(SCSI3)`, `scsi_strategy(SCSI3)`
NAME

scsi_ddsw (SCSI4) SCSI Structure - SCSI device-switch structure

SYNOPSIS

#include <sys/scsi_ctl.h>

struct scsi_ddsw
{
    u_char blk_major;
    u_char raw_major;
    int dd_lun_size;
    int (*dd_open)();
    void (*dd_close)();
    int (*dd_strategy)();
    int (*dd_read)();
    int (*dd_write)();
    int (*dd_ioctl)();
    struct buf (*dd_start)();
    int (*dd_done)();
    int (*dd_pass_thru_okay)();
    int (*dd_pass_thru_done)();
    int (*dd_ioctl_okay)();
    struct status_action *dd_status_list;
    int dd_status_cnt;
    ubit32 (*dd_flags);
    wsio_drv_info_t *wsio_drv;
};

PARAMETERS

blk_major   Obsolete field, not initialized.
raw_major   Obsolete field, not initialized.
dd_lun_size The number of bytes to be allocated and attached to the open device tree when driver_open() is first executed.

dd_open()  Pointer to driver supplied routine.
dd_close() Pointer to driver supplied routine.
dd_strategy() Pointer to driver supplied routine.
**DESCRIPTION**

In order to use SCSI Services effectively, a SCSI driver must define its `scsi_ddsw` device-switch structure. This structure contains pointers to special `dd` routines, some of which are executed indirectly by the standard driver routines, such as `driver_read`. The structure is passed to SCSI Services routines from the `driver_open` routine, which calls the `scsi_lun_open()` SCSI Services routine.

SCSI Services has been set up to control the housekeeping and other processing in the SCSI interface. Therefore, you should have the standard driver routines restrict their operation to calling the appropriate SCSI Services routine. Special processing and customization should all be handled in the special `dd` routines.

**EXAMPLE**

Here is an example of an initialized declaration of the `scsi_ddsw`:

The first example is the declaration of your driver's version of the `dd` routines that can be called by SCSI Services. The routine names are arbitrary. The names in comments are the field names of the `scsi_ddsw` structure.

```c
int mydriver_dd_open(); /* dd_open */
```
The following example shows the `scsi_ddsw` structure. Specify `NULL` for routines that are not defined (that is, that you are not providing). The first two fields specify the block and character major numbers; they are filled in by the call in `driver_dev_init()` to the SCSI Services routine `scsi_ddsw_init()`. The last field points to the `wsio_drv_info_t` structure. The first name in each comment is the field name of the `scsi_ddsw` structure element.

```c
struct scsi_ddsw mydriver_ddsw =
{
    NODEV, /* blk_major - mydriver_dev_init sets */
    NODEV, /* raw_major - mydriver_dev_init sets */
    sizeof(struct mydriver_lun), /* dd_lun_size */
    mydriver_dd_open, /* dd_open */
    mydriver_dd_close, /* dd_close */
    mydriver_dd_strategy, /* dd_strategy */
    NULL,  /* dd_read */
    NULL,  /* dd_write */
    mydriver_dd_ioctl, /* dd_ioctl */
    mydriver_dd_start, /* dd_start */
    mydriver_dd_done, /* dd_done */
    mydriver_dd_pass_thru_okay, /* dd_pass_thru_okay */
    mydriver_dd_pass_thru_done, /* dd_pass_thru_done */
    mydriver_dd_ioctl_okay, /* dd_ioctl_okay */
    mydriver_dd_status_list, /* dd_status_list */
    sizeof(mydriver_dd_status_list)/sizeof(mydriver_dd_status_list[0]),
    /* dd_status_cnt */
    mydriver_dd_flags, /* dd_flag bits DD_DDG */
    &mydriver_wsio_info
    /* For Diagnostics Logging; NULL means errors print in
    * dmesg */
};
```
SEE ALSO

scsi_lun_open(SCSI3)
NAME

scsi_dequeue (SCSI 3) SCSI Function – Remove I/O requests from queues maintained by SCSI Services

SYNOPSIS

struct buf * scsi_dequeue (struct buf ** qp, int where);

PARAMETERS

qp Pointer to the head of a circular list.
where Location to extract from.

RETURN VALUES

scsi_dequeue() returns the following values:
<>NULL Successful completion. The value is the removed struct address.
NULL Failure.

EXAMPLE

struct buf *
sf_start(lp, scb)
struct scsi_lun *lp;
struct scb *scb;
{
    struct sf_lun *llp = lp->dd_lun;
    struct buf *bp;

    /* Protect scb_q */
    scsi_lun_lock(lp);
    bp = scsi_dequeue(&lp->scb_q, HEAD);
    scsi_lun_unlock(lp);

    if (bp == NULL)
        return NULL;

    if (bp->b_flags & B_READ)
        {
            scb->cdb[0] = CMDread;
}  
else  
{  
    scb->cdb[0] = CMDwrite;  
}  
...  
return bp;  
}
NAME

**scsi_dequeue_bp**(SCI3) SCSI Function – Dequeue a specific element from a circular list

SYNOPSIS

```c
scsi_dequeue_bp (struct buf ** qp, struct buf * bp);
```

PARAMETERS

- **qp** Pointer to circular list.
- **bp** Specific buf to remove from the list.

DESCRIPTION

The `scsi_dequeue_bp()` is externally available to dequeue a particular `bp` from a circular list. It is intended for use with LVM’s `B_PFTIMEOUT`.

This is primarily for the `scsi_c700` interface driver to "abort" an I/O from its `select_q`.

`scsi_dequeue_bp()` may be called in either the process or interrupt context. The function must be called within a critical section.

RETURN VALUES

- `<>NULL` Successful completion. The value is the removed struct address.
- `NULL` Failure.
NAME

scsi_enqueue (SCSI3) SCSI Function – Add buffer bp to a circular list

SYNOPSIS

#include <sys/scsi_ctl.h>

void scsi_enqueue (struct buf **qp, struct buf *bp, int where);

PARAMETERS

qp A pointer to the list header which is a pointer to the
    head of the list. The tail of the list is (*qp)->av_back.

bp The struct to be added to the list

where If where equals HEAD, bp is added to the tail of the list;
        otherwise, it is inserted at the head of the list.

DESCRIPTION

The scsi_enqueue() routine is used to queue a buf buffer bp (and an
scb which has its link in the same location as the bp) to a number of
linked lists maintained by SCSI Services: nexus_q, scb free lists,
retry_list, tag_q, busp->select_q. Note that scsi_disk uses
disksort_enqueue() rather than this routine for its list of pending bps.

scsi_enqueue() may be called in either the process or interrupt context.
It must be called within a critical section. For scsi_lun list
manipulation, scsi_lun_lock() and scsi_lun_unlock() must be
called around this function.

RETURN VALUES

scsi_enqueue() is a void function.

SEE ALSO

scsi_dequeue(SCSI3)
NAME

scsi_init_inquiry_data (SCSI 3) SCSI Function - Perform the first Inquiry request on a device

SYNOPSIS

scsi_init_inquiry_data ();

PARAMETERS

None.

DESCRIPTION

The scsi_init_inquiry_data () SCSI routine is called by a device driver from its dd_open () routine to perform the first SCSI Inquiry request on the device. It returns the SCSI Inquiry data from the device to the lp->inquiry_data buffer. It may return an error. However, success does not imply that there is no more pending sense data. In fact, the SCSI-2 standard encourages devices not to give Check Condition status on Inquiry, but to defer it until a subsequent command. Also, if the inquiry data had already been cached as a result of a pass-through driver open or SIOC_INQUIRY, this may not even result in I/O.

Used by device drivers, this function must be called in the process context and may block. The function is not called from within any critical section. It verifies that no spinlocks are held with SD_ASSERT ( ).

It uses lun lock to protect lp->state while testing for L_INIT_INQUIRY. It calls scsi_sleep () until this state flag is cleared, at which time it sets the flag. When the inquiry is completed, it clears the flag and calls wakeup ( ).

RETURN VALUES

scsi_init_inquiry_data () returns the following values:

0 Successful completion.
<>0 Error.
NAME

scsi_ioctl(SCSI3) SCSI Function - Standard SCSI ioctl routine

SYNOPSIS

#include "../h/scsi.h"

scsi_ioctl (dev_t dev, int cmd, caddr_t data, [, int flags]);

PARAMETERS

cmd The ioctl command. It can be one of the commands listed in DESCRIPTION or it can be one that is supported by the driver's dd_ioctl() routine.

data Pointer to the command argument

devo Device number of the associate device

flags The file access flags

DESCRIPTION

The scsi_ioctl() SCSI routine simplifies the job of the device driver. Ioctls which are supported by all device drivers are implemented here to insure consistency from one driver to the next, and to minimize maintenance costs.

Note that other ioctl commands may be supported by a particular driver's dd_ioctl() routine, which is invoked if the command is one that scsi_ioctl() does not recognize.

Used by device drivers, this function must be called in the process context and may block. The function is not called from within any critical section.

RETURN VALUES

scsi_ioctl() returns the following values:

0 Successful completion.

-1 Error.

scsi_ioctl() supports the following ioctl commands (defined in the
SCSI Reference Pages

SCSI Functions

Chapter 7

../h/scsi.h header file):

**SIOC_INQUIRY**

ioctl(fd, SIOC_INQUIRY, &inquiry_data)

The SCSI standard inquiry information for the device is copied to the passed inquiry data structure. The structures inquiry, inquiry_2, and inquiry_data are defined in the ../h/scsi.h header file.

ioctl(fd, SIOC_EXCLUSIVE, &int)

Gain/release exclusive access mode.

The int parameter is an integer that may contain one of the following values:

<table>
<thead>
<tr>
<th>int</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Release exclusive access to logical unit</td>
</tr>
<tr>
<td>1</td>
<td>Gain exclusive access to logical unit</td>
</tr>
<tr>
<td>2</td>
<td>Release exclusive access to target</td>
</tr>
<tr>
<td>3</td>
<td>Gain exclusive access to target</td>
</tr>
<tr>
<td>4</td>
<td>Release exclusive access to bus</td>
</tr>
<tr>
<td>5</td>
<td>Gain exclusive access to bus</td>
</tr>
</tbody>
</table>

**Error Returns**

- [EBUSY] Other opens are active on the level for which exclusive access is desired (lun, target, bus).
- [EINVAL] int is not in the range 0 to 5.

**SIOC_XSENSE**

ioctl(fd, SIOC_XSENSE, &sense_data)

The last available sense data is copied to the passed sense_data structure. It may be used by either a device driver or the pass-through driver.

Sense in the "lun" structure utilizes an array of 2 so that command-mode applications can get data associated with command-mode I/Os and not get misleading data from non-command-mode I/Os that happen to get interleaved with command-mode I/Os. x[CMD_MODE_ONLY] is updated for command-mode only I/Os while x[EVERYTHING] is updated for all I/Os including command-mode I/Os. This is managed automatically by SCSI Services depending upon whether the request is originating from the dev_t set to SCSI_CMD_MODE.
The sense_2, sense_2_aligned, sense_data, xsense, and xsense_aligned structures and other data are defined in the ../h/scsi.h header file.

**Error Returns**

- **[EINVAL]** The data size is not equal to 128 bytes.

**SIOC_IO**

```c
ioctl(fd, SIOC_IO, &sctl_io)
```

Used for "pass-through" I/Os. The bp->b2_flags is used for B2_FIRST, for `disksort()`, and B2_LOWPRI, for `kmetrics`, used for `scsi_cmd()`, `ioctl(SIOC_IO)` and `ioctl(SIOC_CMD_MODE)` type I/Os.

See the `scsi_ctl(7)` manpage.

It uses the lun open/close blocking semaphore to prevent a change in the status of concurrent non-pass-through opens to this LUN. This semaphore protects the lp->ddsw and lp->state value L_DD_PASS_THRU_VALID from being modified.

It must be noted that the sctl_io data structure used by SIOC_IO is passed into the driver by way of `physio()`. This data structure itself contains pointers to other data buffers. This violates the `ioctl(2)` manpage, which states, "Note: Any data structure referenced by arg must not contain any pointers." This "violation" is allowable because the driver and the SCSI Subsystem expect these pointers to exist in this structure. It must be noted that a future implementation for which `physio()` needs to map pointers — to other hosts' memory for example — will not work for this case. Currently, there are no plans for such implementations.

Data transfer size maximum is `SCSI_MAXPHYS`, 1024 * 1024, or 1 megabyte.

**Error Returns**

- **[EACCES]** The user is not superuser or there is no write access permission.

**SIOC_PRIORITY_MODE**

```c
ioctl(fd, SIOC_PRIORITY_MODE, &int)
```

See the `scsi_ctl(7)` manpage.

A device can only be put into priority-mode from the pass-through driver. Once in priority-mode, all pass-through driver SIOC_IO requests to the device are priority-mode I/Os; all other I/Os (not yet queued by SCSI Services in its scb queue) are blocked until the device is taken out of priority-mode. Also while in priority-mode, all device open attempts via the pass-through driver fail. Priority-mode poses a potential deadlock
problem. If the process which has a device in priority-mode blocks waiting for a non-priority-mode I/O to that same device, the result is deadlock. No other I/O to that device will occur — ever. Therefore, the process simply cannot do non-priority-mode I/O to the priority-mode device. Nor can the process allow the system to block it waiting for a page-fault or swap I/O to the device.

If not superuser, it returns [EACCES].

The int parameter is an integer: The value 1 enables priority mode. The value 0 disables priority mode.

Error Returns

[EBUSY] The pass-through driver open count is not one.

[EINVAL] int is invalid, or the command was not invoked by the pass-through major number.

SIOC_CMD_MODE ioctl(fd, SIOC_CMD_MODE, &int)

This may be used by either the device driver or the pass-through driver, sctl.

The int parameter can be either 0 (off) or 1 (on).

The bp->b2_flags is used for B2_FIRST, for disksort(), and B2_LOWPRIO, for kmetrics, used for scsi_cmd(), ioctl(SIOC_IOC) and ioctl(SIOC_CMD_MODE) type I/Os.

Error Returns

[EACCESS] The caller is not superuser and the open was not with FWRITE.

[EBUSY]

• The pass-through driver is the caller and either it currently has more than one open or the device driver is already open.

• A device driver (raw) is the caller and it has more than one open currently.

• Any driver is the caller, command mode is already on, and this is not the driver that turned it on.

[EINVAL] int is invalid.

SIOC_SET_CMD ioctl(fd, SIOC_SET_CMD, &scsi_cmd_parms)

This command may be used by either a device driver or the pass-through driver, sctl, so long as the lp->cmd_mode_major is the calling driver.
SCSI Reference Pages

SCSI Functions

The structure `scsi_cmd_parms` is defined in `../h/scsi.h`.

**Error Returns**

- **[EACCES]** The command was not invoked by the "command code major".
- **[EINVAL]** The parameter, `scsi_cmd_parms->cmd_type` is less than 1 or greater than `SCSI_MAX_CDB_LEN`.

**SIOC_RETURN_STATUS**

`ioctl(fd, SIOC_RETURN_STATUS, &int)`

May be used by either device driver or `sctl` pass-through driver, whether in command mode or not. The SCSI status will be that of the last I/O [EVERYTHING], or that of the last cmd_mode_major originated I/O [CMD_MODE_ONLY], unless "STINGRAY" (includes SCSI Cascade), for which `if2_x_status` is returned.

- `cdb_status[EVERYTHING]` set at every I/O completion
- `cdb_status[CMD_MODE_ONLY]` set at cmd_mode_major I/Os only
- `if2_last_status` set at every I/O completion
- `if2_scsi_status` set at every I/O when `cdb_status` least significant byte has bits on (i.e., is SCSI status; doesn't include SCTL_xxx).

This has special support for the STINGRAY/SCSI Cascade disk arrays, which rely upon SCSI Common Access Method (CAM) status for their utilities. In IF2, `SIOC_RETURN_STATUS` returned a combination of CAM status and SCSI status. The CAM status was from the most recent I/O and the SCSI status was from the most recent I/O with CAM status of `CS_REQ_COMP_WERROR`. The DMD/Stingray folks have been adamant about preserving this behavior; the mechanism is (lp->state & L_IF2_STATUS).

For `SIOC_RETURN_STATUS` `ioctl`, status in the "lun" structure utilizes an array of 2 so that command-mode applications can get data associated with command-mode I/Os and not get misleading data from non-command-mode I/Os that happen to get interleaved with command-mode I/Os. `x[CMD_MODE_ONLY]` is updated for command-mode only I/Os while `x[EVERYTHING]` is updated for all I/Os including command-mode I/Os. This is managed automatically by SCSI Services, depending upon whether the request is originating from the `dev_t` set to `SCSI_CMD_MODE`.

The following is the code which supports this functionality in the SCSI Services:
/*
** CAM status values for backward compatibility.
** Pre-shifted for convenience.
** From wsio/scsi_ctl.h.
*/
#define CS_GOOD (0x01 << 8)
#define CS_ABORTED_BY_HOST (0x02 << 8)
#define CS_REQ_COMP_WERROR (0x04 << 8)
#define CS_INVALID_REQUEST (0x06 << 8)
#define CS_SELECT_TIMEOUT (0x0a << 8)

if (!pass_thru_major(dev) && (lp->state & L_IF2_STATUS))
{
  i = lp->if2_last_status;
  k = j | lp->if2_scsi_status;
}
else
{
  /* Non "STINGRAY" case */
  i = major(dev) == lp->cmd_mode_major ? CMD_MODE_ONLY : EVERYTHING;
  /* 1:0 */
  k = lp->cdb_status[i];
  lp->cdb_status[i] = -1;
}

* (u_int *) data = k;
return 0;

Error Returns
None.

** SIOC_GET_LUN_PARMS **
ioctl(fd, SIOC_GET_LUN_PARMS, &sioc_lun_parms)
The structure sioc_lun_parms is defined in ../h/scsi.h. See scsi_ctl(7).

** Error Returns **
None.

** SIOC_GET_TGT_PARMS **
ioctl(fd, SIOC_GET_TGT_PARMS, &sioc_tgt_parms)
SCSI Reference Pages

SCSI Functions

The structure `sioc_tgt_parms` is defined in `../h/scsi.h`. See `scsi_ctl(7).

Error Returns
None.

**SIOC_GET_BUS_P ARMS**

`ioctl(fd, SIOC_GET_BUS_PARMS, &sioc_bus_parms)`

The structure `sioc_bus_parms` is defined in `../h/scsi.h`. See `scsi_ctl(7).

Error Returns
None.

**SIOC_GET_LUN_ LIMITS**

`ioctl(fd, SIOC_GET_LUN_LIMITS, &sioc_lun_limits)`

If limits have not been set, the act of getting them, sets them.
The structure `sioc_lun_limits` is defined in `../h/scsi.h`. See `scsi_ctl(7).

Error Returns
None.

**SIOC_GET_TGT_ LIMITS**

`ioctl(fd, SIOC_GET_TGT_LIMITS, &sioc_tgt_limits)`

If limits have not been set, the act of getting them, sets them.
The structure `sioc_tgt_limits` is defined in `../h/scsi.h`. See `scsi_ctl(7).

Error Returns
None.

**SIOC_GET_BUS_ LIMITS**

`ioctl(fd, SIOC_GET_BUS_LIMITS, &sioc_bus_limits)`

If limits have not been set, the act of getting them, sets them.
The structure `sioc_bus_limits` is defined in `../h/scsi.h`. See `scsi_ctl(7).

Error Returns
None.

**SIOC_SET_LUN_ LIMITS**

`ioctl(fd, SIOC_SET_LUN_LIMITS, &sioc_lun_limits)`
The structure `sioc_lun_limits` is defined in `../h/scsi.h`. See `scsi_ctl(7).

Error Returns

- `[EACCES]` If not superuser or write permission.
- `[EINVAL]` If reserved fields are not zero.
**SCSI Reference Pages**

**SCSI Functions**

---

**SIOC_SET_TGT_LIMITS**

`ioctl(fd, SIOC_SET_TGT_LIMITS, &sioc_tgt_limits)`

The structure `sioc_tgt_limits` is defined in `../h/scsi.h`. See `scsi_ctl(7)`.

**Error Returns**

- `[EACCES]` If not superuser or write permission.
- `[EINVAL]` If reserved fields are not zero.

---

**SIOC_SET_BUS_LIMITS**

`ioctl(fd, SIOC_SET_BUS_LIMITS, &sioc_bus_limits)`

The structure `sioc_bus_limits` is defined in `../h/scsi.h`. See `scsi_ctl(7)`.

**Error Returns**

- `[EACCES]` If not superuser or write permission.
- `[EINVAL]` If reserved fields are not zero.

---

**SIOC_RESET_DEV**

`ioctl(fd, SIOC_RESET_DEV)`

This command causes a SCSI Bus Device Reset to be sent to the target device by calling the Interface Driver’s `if_bdr()` routine.

**Error Returns**

- `[EACCES]` The user is not superuser.
- `[EINVAL]` `if_bdr()` is NULL.

---

**SIOC_RESET_BUS**

`ioctl(fd, SIOC_RESET_BUS)`

This command causes the SCSI RST line to be "pulled," by calling the Interface Driver’s `if_reset_bus()` routine.

**Error Returns**

- `[EACCES]` The user is not superuser.
- `[EINVAL]` `if_reset_bus()` is NULL.
NAME

`scsi_lun_close` (SCSI3) SCSI Function – Close a device

SYNOPSIS

```c
scsi_lun_close(dev_t dev);
```

PARAMETERS

- `dev` The device number

DESCRIPTION

The `scsi_lun_close()` function is called to close a device. It must be called in the process context and may block. It is not called from within any critical section. It uses the lun open/close semaphore.

The `scsi_lun_close()` function performs the following algorithm:

- Acquire the logical unit open/close semaphore.
  - If this is the last nonpass-through close, wait for all nonpass-through I/O's to complete.
  - If this is any nonpass-through close, call device driver close routine, `dd_close()`.
  - If this is the last nonpass-through close, `clear (lp->state & L_NPT_DD_BITS)`.
  - Attempt to honor the new state with respect to tagged queuing.
  - Call the interface driver close routine, `ifsw->if_close`.
  - Update logical unit open counts.
  - Exit command-mode if closing process neglected to do so.
  - Exit priority-mode if closing process neglected to do so.
  - If this is the last nonpass-through close:
    - `Free *lp->dd_lun and clear lp->dd_lun` if necessary.
    - `lp->ddsw = NULL`.
  - If this is the last close:
— Free any logical unit SCBs that may be hanging around.
— Free *lp->if_lun and *lp and clear tp->lun[lun_id].

• Release the logical unit semaphore.
• Update the ancestor portion of the open device tree via `scsi_tgt_close()`.

This routine assumes that `dev` is open. If it is not, the system will panic in `m_scsi_lun()` when it attempts to dereference a NULL pointer.

**RETURN VALUES**

`scsi_lun_close()` returns an indeterminate value. Treat it as if it was a `void` function.
NAME

\texttt{scsi\_lun\_open} (SCSI) SCSI Function - Open the elements of the hardware path of a SCSI lun

SYNOPSIS

\texttt{scsi\_lun\_open (dev\_t dev, struct scsi\_ddsw * ddswn, int oflags);}

PARAMETERS

- \texttt{dev} The device number
- \texttt{ddsw} Pointer to the non pass_thru driver descriptor
- \texttt{oflags} File access flags

DESCRIPTION

Usually called from the device driver's \texttt{driver\_dev\_init()} routine. The \texttt{scsi\_lun\_open()} SCSI function performs necessary open operations down the hardware path upon which this SCSI LUN resides, including the invocation of the calling driver's \texttt{ddsw->dd\_open()} routine. It opens the appropriate target if necessary. Also used by the pass-through driver.

Used by device drivers, this function must be called in the process context and may block. The function is not called from within any critical section. It verifies that no spinlocks are held with call to \texttt{SD\_ASSERT}. Does use the lun open/close semaphore.

Uses \texttt{kmalloc()} to allocate memory for the \texttt{scsi\_lun} structure.

Checks if \texttt{major(dev) == sctl\_ddsw.raw\_major}.

RETURN VALUES

\texttt{scsi\_lun\_open()} returns the following values:

- 0 Successful completion.
- \texttt{[EBUSY]} The LUN is already opened EXCLUSIVE by another or the open is incomplete and this isn't a pass-through or command mode open attempt.
- \texttt{[EINVAL]} The open request major number doesn't make sense.
[ENXIO] The LUN requested is greater than SCSI_MAX_LUN_ID.

Other errors may be returned from `ddsw->dd_open()`, `if_open()`, `scsi_bus_open()`, or `scsi_tgt_open()`, if they are called from here.
NAME

`scsi_read` (SCSI 3) SCSI Function - Read from device

SYNOPSIS

```c
scsi_read (dev_t dev, struct uio *uio);
```

PARAMETERS

- **dev**: The device number
- **uio**: struct containing transfer information

DESCRIPTION

The `scsi_read()` SCSI function is used for normal (synchronous) reads, and for command mode I/Os for which the ioctl, `SCSI_CMD_MODE`, has been set previously. For normal I/Os, if the driver has defined a `dd_read()` routine in the `scsi_ddsw` structure, it is called; otherwise, `physio()` is called directly.

Used by device drivers, this function must be called in the process context and may block. The function is not called from within any critical section.

RETURN VALUES

`scsi_read()` returns the following values:

- **0**: Successful completion.
- **<>0**: Error. The value is expected to be an `errno` value.

SEE ALSO

`physio(KER2), scsi_ddsw(SCSI4)`
NAME

`scsi_sense_action` (SCSI) SCSI Function - Decode SCSI sense information

SYNOPSIS

```c
scsi_sense_action (struct buf *bp,
                   struct sense_action *sense_list, int n)
```

PARAMETERS

- `bp`: Pointer to the I/O buf structure
- `n`: Number of actions in the list.
- `sense_list`: List of actions to take.

DESCRIPTION

The `scsi_sense_action` SCSI function decodes SCSI sense information. It traverses the functions in a driver's sense action list trying to find a match, and calls the associated action function. It provides the very valuable service of interpreting sense data with regard to SCSI, CCS, or SCSI-2 compliance, so the device driver doesn't need to worry about such things.

`scsi_sense_action` should be called only in the interrupt context by device drivers and by SCSI services on behalf of a device driver.

This function only operates on the request. Sense information does not appear to have any real protection concerns.

The inquiry data for the device must be initialized with `scsi_init_inquiry_data` before it can be interpreted. If an I/O completes before the inquiry data is initialized, `scsi_sense_action` will not match anything other than wild card entries. It will panic if there is no matching entry.

RETURN VALUES

`scsi_sense_action` returns the following values:

- 0: Successful completion.
SCSI Reference Pages
SCSI Functions

<>0  Error. The value is provided by the sense action called.

SEE ALSO

scsi_init_inquiry_data(SCSI3)
NAME

scsi_strategy (SCSI 3) SCSI Function – scsi_strategy –

SYNOPSIS

void scsi_strategy (struct buf *bp)

PARAMETERS

bp The pointer to the I/O buf structure.

DESCRIPTION

The scsi_strategy() SCSI function primarily enqueues the bp to await the necessary resources to allow the request to be sent to the interface driver, and thus, the hardware.

Another purpose is to record the fact that an I/O has been enqueued so the device is not closed while unfinished I/Os exist.

This routine is the first place in the I/O path that all I/Os have in common.

scsi_strategy() is usually called in the process context; it may be invoked on the interrupt context (possibly in the case of a bp->b_call used by the biodone() of a previous I/O completion). Regardless, scsi_strategy() cannot block. Verifies no spinlocks are held by calling SD_ASSERT.

scsi_strategy() must be invoked with a valid bp. If a “special” request (i.e., either B_SIOC_IO or B_SCSI_CMD), it calls scsi_enqueue() to place bp in the lp->priority_scb_q (if lp->pri_mode_major) or lp->special_scb_q. Otherwise, it calls the dd_strategy() routine so that the device driver can manage request order, etc. In this last (normal I/O) case, bp->b2_flags has B2_LOWPRIO cleared for kmetrics' support. Finally, it calls scsi_start().

NOTE

scsi_strategy() calls dd_strategy(), if present, holding the lun_lock.
RETURN VALUES

scsi_strategy() is a void function.

SEE ALSO

biodone(KER2), scsi_enqueue(SCSI3)
NAME

`scsi_write(SCSI3)` SCSI Function - Write to device

SYNOPSIS

```c
scsi_write (dev_t dev, struct uio * uio);
```

PARAMETERS

- `dev` The device number
- `uio` struct containing transfer information

DESCRIPTION

The `scsi_write()` SCSI function is used for normal (synchronous) writes and for command mode I/Os for which the ioctl, `SCSI_CMD_MODE`, has been set previously. For normal I/Os, if the driver has defined a `dd_write()` routine in the `scsi_ddsw` structure, it is called; otherwise, `physio()` is called directly.

Used by device drivers, this function must be called in the process context and may block. The function is not called from within any critical section.

RETURN VALUES

- `0` Successful completion.
- `<>0` Error. The value is expected to be an `errno` value.

SEE ALSO

`physio(KER2), scsi_ddsw(SCSI4)`