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 provide a supported pass-through mechanism.
SCSI Functions
NAME

dd_close(SCSI_DRV) – SCSI driver entry point to handle device close.

SYNOPSIS

void dd_close (dev_t dev);

PARAMETERS

dev

The device number of the file to be closed. The dd_close() routine can extract the major and minor numbers from the device number (see major (WSIO3) and minor (WSIO3).

DESCRIPTION

The dd_close() SCSI function is provided by the driver writer. It can have any unique name. 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

None.

CONSTRAINTS

SEE ALSO

scsi_lun_close(SCSI3), scsi_ddsw(SCSI4)
NAME

dd_done(SCSI_DRV) – SCSI driver entry point to handle post-I/O processing

SYNOPSIS

int dd_done (struct buf *bp);

PARAMETERS

bp Pointer to a buf structure.

DESCRIPTION

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

See HP-UX Driver Development Guide for details;

RETURN VALUES

dd_done() is declared as returning int; however, its return value is not used by SCSI services.

SEE ALSO

biodone(KER2), scsi_action(SCSI3), scsi_ddsw(SCSI4)
NAME

**dd_ioctl**(SCSI_DRV) – SCSI driver entry point to handle device I/O controls.

SYNOPSIS

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

PARAMETERS

- **dev**: Device number
- **cmd**: Command word
- **data**: Pointer to the command parameters
- **flags**: File access flags.

DESCRIPTION

The **dd_ioctl()** SCSI function is provided by the driver writer. It can have any unique name. 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

- 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 driver entry point to allow/disallow ioctl commands sent 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
- `cmd` Command word
- `data` Pointer to command parameter
- `flags` File access flags

DESCRIPTION

The `dd_ioctl_okay`() SCSI function is provided by the driver writer. It can have any unique name. 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

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

CONSTRAINTS

SEE ALSO

`scsi_ioctl(SCSI3)`
NAME

\texttt{dd\_open} (SCSI\_DRV) – SCSI driver entry point to handle device open.

SYNOPSIS

\texttt{dd\_open (dev\_t dev, int oflags);} \\

PARAMETERS

\texttt{dev} Device number of the device to be opened \\
\texttt{oflags} Flags passed in the open call

DESCRIPTION

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

See \textit{HP-UX Driver Development Guide} for details;

RETURN VALUES

0 Successful completion. \\
<>0 Error. The value is expected to be an \texttt{errno} value.

CONSTRAINTS

SEE ALSO

\texttt{m\_scsi\_lun(SCSI3), major(KER2), scsi\_cmdx(SCSI3),} \\
\texttt{scsi\_init\_inquiry\_data(SCSI3), scsi\_lun\_open(SCSI3)}
NAME

dd_pass_thru_done(SCSI_DRV) – SCSI driver entry point to handle post-pass-through I/O processing.

SYNOPSIS

int dd_pass_thru_done (struct buf * bp);

PARAMETERS

bp Pointer to a buf structure

DESCRIPTION

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

CONSTRAINTS

RETURN VALUES

dd_pass_thru_done() is declared as returning int; however, the return value is not used by SCSI services.
NAME

`dd_pass_thru_okay(SCSI_DRV)` – SCSI driver entry point to control pass-through I/O requests.

SYNOPSIS

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

PARAMETERS

- `dev` Device number
- `sctl_io` Struct containing ioctl information

DESCRIPTION

The `dd_pass_thru_okay()` SCSI function is provided by the driver writer. It can have any unique name. 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

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

CONSTRAINTS
NAME

`dd_read(SCSI_DRV)` – SCSI driver entry point to handle device read operations.

SYNOPSIS

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

PARAMETERS

- `dev` Device number
- `uio` Pointer to a `uio` structure.

DESCRIPTION

The `dd_read()` SCSI function is provided by the driver writer. It can have any unique name. 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

- `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 driver entry point to start an I/O request.

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

PARAMETERS
   lp       Pointer to the scsi_lun structure.
   scbp     Pointer to the SCSI control block.

DESCRIPTION
The dd_start() SCSI function is provided by the driver writer. It can have any unique name. 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
<>NULL       Successful completion.
NULL         Error.

CONSTRAINTS
NAME

dd_strategy(SCSI_DRV) – SCSI driver entry point to handle buf requests.

SYNOPSIS

int dd_strategy (struct buf *bp, struct scsi_lun *lp);

PARAMETERS

*bp Pointer to a buf structure

*lp Pointer to a scsi_lun structure.

DESCRIPTION

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

RETURN VALUES

0 Successful completion.

-1 Error.

WARNINGS

dd_strategy() must exist (be defined as non-NULL in the scsi_ddsw structure) if your driver calls scsi_strategy(). scsi_strategy() calls dd_strategy while holding lun_lock.

SEE ALSO

physio(KER2) dd_read(SCSI_DRV), dd_write(SCSI_DRV), scsi_enqueue(SCSI3), scsi_strategy(SCSI3)
NAME

*dd_write*(SCSI3) – SCSI driver entry point to handle device write operations.

SYNOPSIS

```c
int dd_write (dev_t dev, struct uio * uiop);
```

PARAMETERS

- `dev`    Device number
- `uiop`   Pointer to a `uio` structure.

DESCRIPTION

The `dd_write()` SCSI function is provided by the driver writer. It can have any unique name. 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

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

SEE ALSO

`physio(KER2), scsi_write(SCSI3)`
NAME

driver_if_abort(SCSI_DRV2) – Interface driver specific SCSI abort function.

SYNOPSIS

int driver_if_abort (dev_t dev);

PARAMETERS

dev The device number

DESCRIPTION

The SCSI subsystem allows, but does not require, the interface driver to specify an abort function. It can have any unique name. Pass the name SCSI services by specifying it in the if_abort field of the scsi_ifsw structure. Commonly, the driver is replaced by your driver’s name.

It is intended to serve as a way for the SCSI subsystem to direct the interface driver to send a SCSI ABORT message to the indicated logical unit. The SCSI subsystem makes this call only in response to an SIOC_ABORT ioctl request.

See the HP-UX Driver Development Guide for details.

RETURN VALUES

0 Success.
<>0 Error.

CONSTRAINTS

SEE ALSO

scsi_ifsw(SCSI3)
NAME

\texttt{driver\_if\_bdr} (SCSI\_DRV2) – Interface driver specific SCSI Bus Device Reset function.

SYNOPSIS

\begin{verbatim}
int driver\_if\_bdr (dev\_t dev);
\end{verbatim}

PARAMETERS

\begin{description}
\item[dev] The device number
\end{description}

DESCRIPTION

The SCSI subsystem allows, but does not require, the interface driver to specify a Bus Device Reset (BDR) function. It can have any unique name. Pass the name SCSI services by specifying it in the \texttt{if\_bdr} field of the \texttt{scsi\_ifsw} structure. Commonly, the driver is replaced by your driver's name.

It is intended to serve as a way for the SCSI subsystem to direct the interface driver to send a SCSI BDR message to the indicated target. The SCSI subsystem makes this call only in response to an \texttt{SIOC\_RESET\_DEV} ioctl request.

See the \textit{HP-UX Driver Development Guide} for details.

RETURN VALUES

\begin{description}
\item[0] Success.
\item[\textless 0] Error.
\end{description}

CONSTRAINTS

SEE ALSO

\texttt{scsi\_ifsw} (SCSI3)
NAME

**driver_if_close**(SCSI_DRV2) – Interface driver specific logical unit close processing

SYNOPSIS

```c
int driver_if_close (dev_t dev);
```

PARAMETERS

- `dev` The device number

DESCRIPTION

The SCSI subsystem allows, but does not require, the interface driver to specify a logical unit close function. It can have any unique name. You pass the name SCSI services by specifying it in the `if_close` field of the `scsi_ifsw` structure. Commonly, the driver is replaced by your driver's name.

On all logical closes, the SCSI subsystem checks the `if_close` field of the `scsi_ifsw` structure for the SCSI bus. If the `if_close` field is not `NULL`, the SCSI subsystem calls it with the device number of the device being opened as its sole argument.

See the *HP-UX Driver Development Guide* for details.

RETURN VALUES

- `0` Success.
- `<>0` Error.

CONSTRAINTS

It is never called under interrupt context. It is allowed to sleep. The SCSI subsystem provides protection that blocks all other opens and closes to the same logical unit until it returns.

SEE ALSO

```
scsi_ifsw(SCSI3)
```
NAME

driver_if_open(SCSI3) – Interface driver specific logical unit open processing

SYNOPSIS

int driver_if_open (dev_t dev);

PARAMETERS

dev The device number

DESCRIPTION

The SCSI subsystem allows, but does not require, the interface driver to specify a logical unit open function. It can have any unique name. You pass the name SCSI services by specifying it in the if_open field of the scsi_ifsw structure. Commonly, driver is replaced by your driver’s name.

On all logical opens, the SCSI subsystem checks the if_open field of the scsi_ifsw structure for the SCSI bus. If the if_open field is not NULL, the SCSI subsystem calls it with the device number of the device being opened as its sole argument.

See the HP-UX Driver Development Guide for details.

RETURN VALUES

0 Success.
<>0 Error.

CONSTRAINTS

It is never called under interrupt context. It is allowed to sleep. The SCSI subsystem provides protection that blocks all other opens and closes to the same logical unit until it returns.

SEE ALSO

scsi_ifsw(SCSI3)
NAME

driver_if_reset_bus(SCSI_DRV2) – Interface driver specific SCSI bus reset function.

SYNOPSIS

void driver_if_reset_bus (dev_t dev);

PARAMETERS

dev The device number

DESCRIPTION

The SCSI subsystem allows, but does not require, the interface driver to specify a bus reset function. It can have any unique name. You pass the name SCSI services by specifying it in the \texttt{if\_reset\_bus} field of the \texttt{scsi\_ifsw} structure. Commonly, the driver is replaced by your driver's name.

When the SCSI subsystem wants to reset a bus, it checks the \texttt{if\_reset\_bus} field of the \texttt{scsi\_ifsw} structure for the bus. If the \texttt{if\_reset\_bus} is not \texttt{NULL}, it is called with a device number identifying the bus as its sole argument. When \texttt{if\_reset\_bus} returns, the SCSI bus should have been reset.

I/Os that are disconnected and the I/Os that are connected with the bus (if any) at the time of the reset should be returned to the SCSI subsystem with the appropriate status field set to \texttt{SCTL\_INCOMPLETE}. That is, if it was the Request Sense resulting from a check condition that was terminated by the reset, then \texttt{scb->sense\_action} should be set to \texttt{SCTL\_INCOMPLETE}. Otherwise, \texttt{scb->cdb\_status} should be set to \texttt{SCTL\_INCOMPLETE}. “struct scb” is described under data structures later in this section.

The SCSI subsystem makes this call only in response to \texttt{SIOC\_RESET\_BUS ioctl} request.

See \textit{HP-UX Driver Development Guide} for details.
RETURN VALUES

0       Success.
<0      Error

CONSTRAINTS

SEE ALSO

scsi_ifsw(SCSI3)
NAME

driver_if_start(SCSI_DRV2) – Interface driver specific start function

SYNOPSIS

void driver_if_start (struct isc_table_type *isc);

PARAMETERS

isc A pointer to structure isc_table_type

DESCRIPTION

The SCSI subsystem requires the interface driver to specify a start function. Its purpose is to ensure the SCSI subsystem does not hang by providing a way to inform the interface driver that it has work to do.

It can have any unique name. You pass the name SCSI services by specifying it in the if_start field of the scsi_ifsw structure. Commonly, the driver is replaced by your driver’s name.

Whenever the SCSI subsystem enqueues an I/O on the select queue of a dormant bus, the function pointed by the <if_start> field of the scsi_ifsw structure for the bus is always called with a pointer to the isc_table_type structure as its sole argument.

The SCSI subsystem may call if_start at any time, i.e., when the bus is dormant or not, and in a process’ context or under interrupt. In all cases, the interface driver must continue to execute I/Os that are on the select queue until the bus becomes dormant.

A bus is considered dormant if it has no active I/Os. An I/O is considered to be active from the time it is enqueued on the select queue until scsi_cbfn is called for the I/O.

See the HP-UX Driver Development Guide for details.

RETURN VALUES

None
CONRAINTS

\texttt{if\_start} is not permitted to sleep under any circumstances.

SEE ALSO

\texttt{scsi\_ifsw(SCSI3)}
NAME

m_bus_id(SCSI3) – Returns SCSI Bus ID.

SYNOPSIS

#include<sys/scsi_ctl.h>

m_bus_id (dev_t dev);

PARAMETERS

dev The device number

DESCRIPTION

m_bus_id() macro evaluates to the bus ID of the SCSI bus corresponding to dev.

RETURN VALUES

m_bus_id() does not return any values. It is a macro.

CONSTRAINTS
EXAMPLES

#include <sys/scsi_ctl.h>

static int
mydriver_lun_open(dev_t dev)
{
    struct isc_table_type * isc;
    int bus_id, tgt_id, lun_id;
    ....

    /* Get the SCSI bus ID */
    bus_id = m_bus_id(dev);

    /* Get the SCSI target ID */
    tgt_id = m_tgt_id(dev);

    /* Get the SCSI LUN ID */
    lun_id = m_lun_id(dev);

    ....
}

SEE ALSO
NAME

*m_lun_id*(SCSI3) – Returns SCSI LUN ID.

SYNOPSIS

```c
#include<sys/scsi_ctl.h>
m_lun_id (dev_t dev);
```

PARAMETERS

`dev` The device number

DESCRIPTION

`m_lun_id()` macro evaluates to the LUN ID of the SCSI LUN corresponding to `dev`.

RETURN VALUES

`m_lun_id()` does not return any values. It is a macro.

CONSTRAINTS
EXAMPLES

#include <sys/scsi_ctl.h>

static int
mydriver_lun_open(dev_t dev)
{
    struct isc_table_type * isc;
    int bus_id, tgt_id, lun_id;
    ....

    /* Get the SCSI bus ID */
    bus_id = m_bus_id(dev);

    /* Get the SCSI target ID */
    tgt_id = m_tgt_id(dev);

    /* Get the SCSI LUN ID */
    lun_id = m_lun_id(dev);

    ....
}

SEE ALSO
NAME

m_tgt_id(SCSI3) – Returns SCSI target ID.

SYNOPSIS

#include<sys/scsi_ctl.h>

m_tgt_id (dev_t dev);

PARAMETERS

dev The device number

DESCRIPTION

m_tgt_id() macro evaluates to the target ID of the SCSI target corresponding to dev.

RETURN VALUES

m_tgt_id() does not return any values. It is a macro.

CONSTRAINTS
EXAMPLES

#include <sys/scsi_ctl.h>

static int
mydriver_lun_open(dev_t dev)
{
    struct isc_table_type * isc;
    int bus_id, tgt_id, lun_id;
    ....

    /* Get the SCSI bus ID */
    bus_id = m_bus_id(dev);

    /* Get the SCSI target ID */
    tgt_id = m_tgt_id(dev);

    /* Get the SCSI LUN ID */
    lun_id = m_lun_id(dev);

    ....
}

SEE ALSO
NAME

m_scsi_bus(SCSI3) – Returns scsi_bus pointer

SYNOPSIS

#include <sys/scsi_ctl.h>
struct scsi_bus * m_scsi_bus (dev_t dev);

PARAMETERS

dev The device number

DESCRIPTION

m_scsi_bus() function returns the scsi_bus pointer corresponding to dev.

RETURN VALUES

NULL Error.
<>0 Pointer to the SCSI bus structure associated with dev.

CONSTRAINTS
EXAMPLES

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

static int
mydriver_if_abort(struct buf *bp)
{
    dev_t dev;
    struct scsi_bus *busp;
    struct scsi_tgt *tgtp;
    struct scsi_lun *lunp;
    ....

    /* Get the device number */
    dev = bp->b_dev;

    /* Get the pointer to scsi_bus structure */
    busp = m_scsi_bus(dev);

    /* Get the pointer to scsi_tgt structure */
    tgtp = m_scsi_tgt(dev);

    /* Get the pointer to scsi_lun structure */
    lunp = m_scsi_lun(dev);

    if(busp == NULL) {
        msg_printf("mydriver - a NULL scsi_bus pointer\n");
        return (ENXIO);
    }

    ....
}
```

SEE ALSO
NAME

m_scsi_isc (SCSI3) – Returns isc_table_type pointer

SYNOPSIS

#include<sys/scsi_ctl.h>
struct isc_table_type * m_scsi_isc (dev_t dev);

PARAMETERS

dev The device number

DESCRIPTION

m_scsi_isc() function returns the isc_table_type pointer corresponding to dev.

RETURN VALUES

NULL Error.
<>0 Pointer to the ISC structure associated with dev.

CONSTRAINTS
EXAMPLES

#include <sys/scsi_ctl.h>

static int
mydriver_lun_open(dev_t dev)
{
    struct isc_table_type *isc;
    ......

    /* Get the isc pointer */
    isc = m_scsi_isc(dev);

    if(isc == NULL) {
        msg_printf("mydriver - a NULL isc returned\n");
        return (ENXIO);
    }

    ...

}

SEE ALSO
NAME

\texttt{m\_scsi\_lun(SCSI3)} – Returns \texttt{scsi\_lun} pointer

SYNOPSIS

\begin{verbatim}
#include <sys/scsi_ctl.h>

struct scsi_lun * m_scsi_lun (dev_t dev);
\end{verbatim}

PARAMETERS

\begin{itemize}
  \item \texttt{dev} \quad The device number
\end{itemize}

DESCRIPTION

\texttt{m\_scsi\_lun()} function returns the \texttt{scsi\_lun} pointer corresponding to \texttt{dev}.

RETURN VALUES

\begin{itemize}
  \item \texttt{NULL} \quad Error.
  \item \texttt{<>0} \quad Pointer to the SCSI LUN structure associated with \texttt{dev}.
\end{itemize}

CONSTRAINTS
EXAMPLES

#include <sys/scsi_ctl.h>

static int
mydriver_if_abort(struct buf *bp)
{
    dev_t dev;
    struct scsi_bus *busp;
    struct scsi_tgt *tgtp;
    struct scsi_lun *lunp;
    ....

    /* Get the device number */
    dev = bp->b_dev;

    /* Get the pointer to scsi_bus structure */
    busp = m_scsi_bus(dev);

    /* Get the pointer to scsi_tgt structure */
    tgtp = m_scsi_tgt(dev);

    /* Get the pointer to scsi_lun structure */
    lunp = m_scsi_lun(dev);

    if(busp == NULL) {
        msg_printf("mydriver - a NULL scsi_bus pointer\n");
        return (ENXIO);
    }

    ....
}

SEE ALSO
NAME

m_scsi_tgt(SCSI3) – Returns scsi_tgt pointer

SYNOPSIS

#include <sys/scsi_ctl.h>

struct scsi_lun * m_scsi_tgt (dev_t dev);

PARAMETERS

dev The device number

DESCRIPTION

m_scsi_tgt() function returns the scsi_tgt pointer corresponding to dev.

RETURN VALUES

NULL Error.
<>0 Pointer to the SCSI target structure associated with dev.

CONSTRAINTS


EXAMPLES

#include <sys/scsi_ctl.h>

static int
mydriver_if_abort(struct buf *bp)
{
    dev_t dev;
    struct scsi_bus *busp;
    struct scsi_tgt *tgtp;
    struct scsi_lun *lunp;
    ....

    /* Get the device number */
    dev = bp->b_dev;

    /* Get the pointer to scsi_bus structure */
    busp = m_scsi_bus(dev);

    /* Get the pointer to scsi_tgt structure */
    tgtp = m_scsi_tgt(dev);

    /* Get the pointer to scsi_lun structure */
    lunp = m_scsi_lun(dev);

    if(busp == NULL) {
        msg_printf("mydriver - a NULL scsi_bus
        pointer\n");
        return (ENXIO);
    }
    ....
}

SEE ALSO
NAME
scb(SCSI3) – SCSI Control Block Structure

SYNOPSIS

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

DESCRIPTION

SCSI services allocate `scb` structure and associate with a `buf` structure. The fields in the `scb` structure hold temporary state information until an I/O is completed.

This structure is used by an interface driver to get the SCSI command to be issued to the HBA it controls and to report the I/O completion status to the SCSI subsystem. The `scb` structure has also fields to pass any sense data that an interface driver may fill to the SCSI services layer.

When an interface driver calls `scsi_cbfn()` on completion of an I/O request, the SCSI services free the `scb` structure if the I/O is not going to be retried.

STRUCTURE MEMBERS

The following is a list of driver accessible fields

<table>
<thead>
<tr>
<th>Table 7-1 Relevant scb Structure Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>void*</td>
</tr>
<tr>
<td>struct scsi_lun *</td>
</tr>
<tr>
<td>ubit32</td>
</tr>
<tr>
<td>ubit32</td>
</tr>
<tr>
<td>ubit8</td>
</tr>
<tr>
<td>ubit8</td>
</tr>
<tr>
<td>ubit32</td>
</tr>
<tr>
<td>ubit8</td>
</tr>
</tbody>
</table>
Table 7-1 Relevant scb Structure Fields (Continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubit32</td>
<td>cdb_status</td>
</tr>
<tr>
<td>ubit32</td>
<td>data_resid</td>
</tr>
<tr>
<td>ubit32</td>
<td>sense_status</td>
</tr>
<tr>
<td>ubit8</td>
<td>sense_bytes</td>
</tr>
<tr>
<td>ubit8*</td>
<td>sense_data</td>
</tr>
</tbody>
</table>

**if_scb**  
if_scb is a pointer to ifsw->if_scb_size bytes allocated by SCSI services and reserved for use by the interface driver. The pointer is initialized at scb creation time by services and the data area is bzero’ed by services for each I/O attempt prior to putting the I/O on the select queue. It is not touched by services at any other time. The if_scb area is later freed by the SCSI services along with the scb.

**lp**  
lp is a pointer to the scsi_lun structure in the open device tree with which this scb is associated. If the scb belongs to a per-lun pool of scb’s as opposed to a per-bus pool, then scb->lp is initialized at scb creation time, by SCSI services, and never changed. Otherwise, the scb belongs to a per-bus pool, and scp->lp is only valid while the scb is associated with a bp. Then scb->lp points to the scsi_lun structure associated with bp->b_dev.

**flags**  
The interface driver may check these bits in the flag for proper functionality

<table>
<thead>
<tr>
<th>Flag</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCB_NO_DISC</td>
<td>This bit indicates that the disconnect privilege should not be granted in the identify message.</td>
</tr>
<tr>
<td>SCB_SDTR</td>
<td>If this bit is set and SCB_WDTR is not set, the interface driver should initiate SDTR negotiation immediately following the Selection, Identify or tag message, whichever comes last, and before sending the CDB for the I/O.</td>
</tr>
<tr>
<td>SCB_WDTR</td>
<td>This bit directs the interface driver that a wide negotiation should be initiated immediately following the Selection, Identify or tag message, whichever comes last, and before sending any CDB for the I/O. If ((\text{tp-&gt;state} &amp; \text{T_ENABLE_SDTR})) or ((\text{scb-&gt;flags} &amp; \text{SCB_SDTR})) is also set, the interface driver should initiate SDTR negotiation immediately following the WDTR negotiation. The wide negotiation should always precede the synchronous negotiation, since a wide negotiation resets the link to asynchronous.</td>
</tr>
<tr>
<td>SCB_4BYTE</td>
<td>This bit is a hint to the interface driver that the target will never change phase while in data phase on other than a four-byte boundary (at the beginning of the data transfer) without subsequently restoring the data pointer (implicitly or explicitly) to a previously aligned value and re-transferring data up to and beyond the point of disconnection to an aligned boundary. The phase change at the end of the I/O need be considered only if the amount of data transferred may be less than that requested in (\text{bp-&gt;b_bcount}). Note that SCB_4BYTE does not imply that (\text{bp-&gt;b_count}) is a multiple of four or that (\text{bp-&gt;b_un.b_addr}) is four-byte aligned. Note also that the phase</td>
</tr>
</tbody>
</table>
change out of data phase if all bp->b_bcount bytes have been transferred is not subject to the alignment restructuring.

**SCB_2BYTE**
This bit is the same as SCB_4BYTE except that phase changes are only restricted to even boundaries.

**SCB_ORDERED_TAG**
Denotes that ordered tags are intended to be used for this device.

**max_msecs**
Minimum number of milliseconds the interface driver is to allow for this I/O from the time of Selection until Command Complete. If scb->max_msecs milliseconds elapses and the I/O has not completed, the interface driver is encouraged to abort the I/O with Abort or Abort Tag as appropriate. A value of zero indicates the interface driver should never abort this I/O based solely on the amount of time since Selection.

**cdb**
Holds the SCSI command bytes for this I/O.

**cdb_len**
The number of bytes in the cdb. This can be a maximum of \( \text{SCSI\_MAX\_CDB\_LEN} \).

**io_id**
A unique identifier for a SCSI I/O. It is initialized when the scb is associated with a bp and is unique across all SCSI buses.

**tag**
The tag value allocated for this I/O by the SCSI subsystem in accordance with the interface driver's direction via ifsw->if_max_tag. It is recommended that the interface driver use this value as the tag value for the I/O if the I/O will be tagged, but it is not required. Currently there can only be 256 tags per bus. The tag value may not remain the same for retried I/Os.

**cdb_status**
Indicates the status of the I/O command. If the I/O attempt completes with no phase sequencing errors and without being aborted or timing out, the interface driver sets scb->cdb_status to S_GOOD. If the selection phase times out, the interface driver sets cdb_status to SCTL_SELECT_TIMEOUT. If the I/O is not
even attempted because of bogus data in the bp or scb, the interface driver sets cdb_status to SCTL_INVALID_REQUEST. If the I/O is not attempted or does not complete for any other reason, cdb_status is set to SCTL_INCOMPLETE. If there is a Contingent Allegiance condition, the cdb_status is set to S_CHECK_CONDITION to request an auto-sense request. scb->cdb_status must be set by the interface driver prior to returning the bp via scsi_cbfn. Refer to scsi.h for all the valid values for the cdb_status.

data_resid If the I/O attempt completes with no phase sequencing errors and without being aborted or timing out, the interface driver sets scb->data_resid such that bp->b_count - scb->data_resid is the offset from bp->b_un.b_addr of the first byte not transferred by the target, i.e., number of bytes transferred = bp->b_bcount - scb->data_resid. Even if the I/O attempt is failed for some reason, it is advisable to set the scb->data_resid to indicate the number of bytes not yet transferred. Setting this field will have no adverse affect. scb->data_resid must be set by the interface driver prior to returning the bp via scsi_cbfn.

sense_status Represents the status of the automatic request sense that is performed if scb->cdb_status is S_CHECK_CONDITION. If the Request Sense completes with no phase sequencing errors and without being aborted or timing out, the interface driver sets scb->sense_status. Otherwise, scb->sense_status is undefined and will not be referenced by the SCSI subsystem on callback. The possible values for scb->sense_status are the same as those for scb->cdb_status except SCTL_INVALID_REQUEST cannot be used. scb->sense_status represents the result of the automatic Request Sense in the same way that scb->cdb_status represents the result of attempting scb->cdb. It must be set by the interface driver before returning the bp via scsi_cbfn. If there is any sense data, the sense_status has to be set to S_GOOD.
sense_bytes  Number of bytes of data received in response to the automatic request sense if one was performed. It is valid only if sense_status is valid and is neither SCTL_SELECT_TIMEOUT nor SCTL_INCOMPLETE. scb->sense_bytes is the offset from scb->sense_data of the first byte of sense data not transferred by the target. It must be set by the interface driver prior to returning the bp via scsi_cbfn.

sense_data  If scb->cdb_status is Check Condition and the resulting Request Sense completes with no phase sequencing errors and without being aborted or timing out, and if scb->sense_status is not zero, the interface driver sets scb->sense_data. Otherwise, scb->sense_data is undefined and will not be referenced by the SCSI subsystem on callback. The interface driver sets scb->sense_data to point to a KERNELSPACE buffer containing the sense data; its size must be at least scb->sense_bytes. It must be set prior to returning the bp via scsi_cbfn and the interface driver must not modify the buffer for the duration of scsi_cbfn. When scsi_cbfn returns, and not until, the interface driver can reuse the buffer.

SEE ALSO

buf(KER4), scsi_ifsw(SCSI3)
NAME

`scsi_action(SCSI3)` – Give I/O completion information to SCSI Services

SYNOPSIS

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

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

PARAMETERS

- `bp`  
  Pointer to a `buf` structure

- `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 queueing 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.

error  errno value.
msecs  ???

DESCRIPTION

csci_action() function 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.

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

The csci_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 csci_action() and some of the device driver dd_done() functions do need the protection.

Logging, as a result of SA_LOG_IT_ALWAYS or SA_LOG_IT_SOMETIMES sent to csci_status(), causes csci_log_io() to be invoked. This routine records the I/O attempt and its results in the dmesg buffer. Output is controlled by csci_log_mask and csci_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

-1 Do not retry. Call biodone()
SEE ALSO

biodone(KER2), panic(KER2)
NAME

`scsi_bus_lock'(SCSI3) – Acquire SCSI bus lock.

SYNOPSIS

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

void scsi_bus_lock(struct scsi_bus * busp);
```

PARAMETERS

`busp`  
Pointer to the `scsi_bus` structure.

DESCRIPTION

The `scsi_bus_lock()` and `scsi_bus_unlock()` functions are used to provide exclusive access to the `scsi_bus` structure. Calls to manipulate the I/O requests queues maintained by the SCSI services are bounded by `scsi_bus_lock()` and `scsi_bus_unlock()`. This can be called under interrupt context.

RETURN VALUES

None.

CONSTRAINTS

Must not be called while holding a spinlock with lock order $\geq$ SCSI_BUS_LOCK_ORDER.
EXAMPLE

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

static int
mydriver_enqueue(struct scsi_bus *busp, struct buf *bp,
                  int where)
{
    ....

    /* Enqueue the I/O request bp to the select_q */
    scsi_bus_lock(busp);
    scsi_enqueue(&busp->select_q, bp);
    scsi_bus_unlock(busp);

    ....
}
```

SEE ALSO

`scsi_dequeue(SCSI3), scsi_dequeue_bp(SCSI3),
scsi_enqueue(SCSI3)`
NAME

`scsi_bus_unlock(SCSI3)` – Release SCSI bus lock.

SYNOPSIS

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

void scsi_bus_unlock(struct scsi_bus * busp);
```

PARAMETERS

`busp`  
Pointer to the scsi_bus structure.

DESCRIPTION

The `scsi_bus_lock()` and `scsi_bus_unlock()` functions are used to provide exclusive access to the `scsi_bus` structure. Calls to manipulate the I/O requests queues maintained by the SCSI services are bounded by `scsi_bus_lock()` and `scsi_bus_unlock()`.

RETURN VALUES

None.

CONSTRAINTS
EXAMPLE

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

static int
mydriver_enqueue(struct scsi_bus *busp, struct buf * bp,
                   int where)
{
    ....

    /* Enqueue the I/O request bp to the select_q */
    scsi_bus_lock(busp);
    scsi_enqueue(&busp->select_q, bp);
    scsi_bus_unlock(busp);

    ....
}
```

SEE ALSO

scsi_dequeue(SCSI3), scsi_dequeue_bp(SCSI3),
scsi_enqueue(SCSI3)
NAME

`scsi_cbfn` (SCSI3) — SCSI subsystem callback function.

SYNOPSIS

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

void scsi_cbfn(struct buf *bp);
```

PARAMETERS

`bp`Buffer pointer to the I/O request that is completed.

DESCRIPTION

The `scsi_cbfn` function is called by the interface driver on I/O attempt completion. When the interface driver finishes with an I/O, it returns the I/O to the SCSI subsystem by calling `scsi_cbfn` with the bp as its sole argument.

The interface driver relinquishes all rights to access bp, scb and `*scb->if_scb` once it calls `scsi_cbfn()`. Of course, the bp may be reused later for another I/O, and similarly for the scb and `*scb->if_scb`, although they will not necessarily be related in subsequent I/O's.

If the interface driver has attached a sense buffer to `scb->sense_data`, the `sense_data` buffer must be valid till `scsi_cbfn()` returns. The interface driver is forbidden from accessing it until `scsi_cbfn()` returns. It is important to note that the allocation and management of this buffer for holding `sense_data` is the responsibility of the interface driver.

This can be called either in process or interrupt context. This must not be called with any locks held.

RETURN VALUES

None

SEE ALSO

`scb` (SCSI3), `buf` (KER4)
NAME

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

SYNOPSIS

#include <wsio/scsi_ctl.h>

int scsi_cmd (dev_t dev, ubit32 flags, int cdb_len,
             ubit8 * cdb, int nbytes, void * addr,
             ubit32 msecs, ubit32 retries, int * errp);

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.
retries    Number of retries.
errp       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.
RETURN VALUES

<table>
<thead>
<tr>
<th></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) – Prepare driver-generated I/O requests

SYNOPSIS

#include <wsio/scsi_ctl.h>

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

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.
retries Number of retries.
errp If not NULL, then contains the error returned by the operation (in bp->b_error)
sa If NULL, there is no action to match or take.
n Status count.

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**

- `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 device switch structure

SYNOPSIS

#include <wsio/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.

dd_read() Pointer to driver supplied routine.
dd_write()  Pointer to driver supplied routine.

dd_ioctl()  Pointer to driver supplied routine.

dd_start()  Pointer to driver supplied routine.

dd_done()  Pointer to driver supplied routine.

dd_pass_thru_okay()  Pointer to driver supplied routine.

dd_pass_thru_done()  Pointer to driver supplied routine.

dd_ioctl_okay()  Pointer to driver supplied routine.

dd_flags  Flag bits, currently only DD_DDG defined.

dd_status_list  Table of device driver status/action pairs.

dd_status_cnt  Count of device driver status/action pairs in the table.

wsio_drv  A pointer to your drivers wsio_drv_info structure.

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 customizing should all be handled in the special dd routines.
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 */
void mydriver_dd_close();  /* dd_close */
int mydriver_dd_strategy(); /* dd_strategy */
int mydriver_dd_read();  /* dd_read */
int mydriver_dd_write(); /* dd_write */
int mydriver_dd_ioctl(); /* dd_ioctl */
struct buf mydriver_dd_start(); /* dd_start */
int mydriver_dd_done(); /* dd_done */
int mydriver_dd_pass_thru_okay(); /* dd_pass_thru_okay */
int mydriver_dd_pass_thru_done(); /* dd_pass_thru_done */
int mydriver_dd_ioctl_okay(); /* dd_ioctl_okay */
```

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 */
};
```
SEE ALSO

scsi_lun_open(SCSI3)
NAME

scsi_dequeue(SCSI3) – Remove I/O requests from queues maintained by SCSI Services

SYNOPSIS

#include <sys/scsi_ctl.h>

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

PARAMETERS

qp Pointer to the head of a list of I/O requests.
where Location to extract from.

DESCRIPTION

The scsi_dequeue() function extracts the I/O request at HEAD or TAIL of the list. Parameter *qp is based on the value of where and return the bp. This returns NULL when the queue is empty.

RETURN VALUES

NULL Error.
<>0 Pointer to struct buf (I/O request).

CONSTRAINTS

This must be called with scsi_bus lock held.
EXAMPLE

#include <sys/scsi_ctl.h>

static int
mydriver_if_start(struct isc_table_type * isc)
{
    struct scsi_bus *busp;
    struct buf *bp;
    ....
    /* Get the SCSI bus pointer */
    busp = (struct scsi_bus *) isc->if_drv_data;
    bp = scsi_dequeue(&busp->select_q, HEAD);
    /* Dequeue an I/O request from HEAD of the select_q */
    scsi_bus_unlock(busp);
    ....
}

SEE ALSO

scsi_dequeue_bp(SCSI3), scsi_enqueue(SCSI3)
NAME

`scsi_dequeue_bp(SCSI3)` – Remove a specific I/O request from a specified queue maintained by SCSI services.

SYNOPSIS

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

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

PARAMETERS

- `qp` Pointer to the head of a list of I/O requests.
- `bp` Specific `buf` to remove from the list.

DESCRIPTION

The `scsi_dequeue_bp()` function tries to dequeue `bp` from wherever it may be in the queue `*qp`. Returns `bp` when found on the queue. Returns `NULL` when not found on the queue.

RETURN VALUES

- `NULL` Error.
- `<>NULL` Pointer to struct `buf` (I/O request).

CONSTRAINTS

This must be called with `scsi_bus` lock held.
EXAMPLE

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

static int
mydriver_dequeue_bp(struct scsi_bus *busp, struct buf * bp)
{
    ....
    struct buf *rhp;
    ....
    /* Dequeue the I/O request bp from the select_q */
    scsi_bus_lock(busp);
    rbp = scsi_dequeue_bp(&busp->select_q, bp);
    scsi_bus_unlock(busp);
    ....
}
```

SEE ALSO

`scsi_dequeue(SCSI3), scsi_enqueue(SCSI3)`
NAME

scsi_enqueue(SCSI3) – Add buffer bp to a specified queue maintained by SCSI services.

SYNOPSIS

#include <sys/scsi_ctl.h>

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

PARAMETERS

qp A pointer to the head of a list of I/O requests.
bp Pointer to struct buf to be added to the list
where Location to insert the I/O request.

DESCRIPTION

The scsi_enqueue function enqueues bp at the HEAD or TAIL of an I/O requests list; qp is a pointer to the list header which is a pointer to the head of the list. If where is HEAD, the bp is inserted ahead of the list, otherwise it is added to the tail of the list. There are different linked lists maintained by SCSI services: nexus_q, scb free lists, retry list, tag_q and busp->select_q.

RETURN VALUES

None.

CONSTRAINTS

This must be called with scsi_bus lock held.
EXAMPLE

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

static int
mydriver_enqueue(struct scsi_bus *busp, struct buf *bp,
    int where)
{
    ....

    /* Enqueue the I/O request bp to the select_q */
    scsi_bus_lock(busp);
    scsi_enqueue(&busp->select_q, bp);
    scsi_bus_unlock(busp);

    ....
}
```

SEE ALSO

`scsi_dequeue(SCSI3), scsi_dequeue_bp(SCSI3)`
NAME

scsi_ifsw(SCSI3) – SCSI interface driver switch structure

SYNOPSIS

#include <sys/scsi_ctl.h>

DESCRIPTION

The interface driver switch structure exports an interface driver's entry points and operational parameters to the SCSI services layer. The interface driver's attach routine must initialize the ifsw field of the isc_table_type entry to point to a scsi_isw structure.

STRUCTURE MEMBERS

The scb structure is defined in <sys/scsi_ctl.h. The following are some important fields in the scb structure. Their types are given in the following table. Care should be taken when an interface driver modifies some of these fields.

The following is a list of driver accessible fields:

Table 7-2 Relevant scsi_ifsw Structure

<table>
<thead>
<tr>
<th>Type</th>
<th>Field Name</th>
<th>Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubit8</td>
<td>if_flags;</td>
<td>int (*) ()</td>
<td>if_open</td>
</tr>
<tr>
<td>ubit8</td>
<td>if_max_tag;</td>
<td>void (*) ()</td>
<td>if_close</td>
</tr>
<tr>
<td>unsigned int</td>
<td>if_scb_size;</td>
<td>void (*) ()</td>
<td>if_start</td>
</tr>
<tr>
<td>unsigned int</td>
<td>if_lun_size;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned int</td>
<td>if_tgt_size;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned int</td>
<td>if_bus_size;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
If_flags

Interface driver flags convey information to the SCSI services on what it supports and what not. The possible flags are:

- **IF_BUS_TAGS**: A default flag.
- **IF_NO_TAGS**: The interface driver does not support tags.
- **IF_B2_LIST**: The interface driver supports handling of disksort merge buffers.
- **IF_OWNS_TAGS**: The interface driver owns tagged queueing.

If_max_tag

One less than the number of per-bus tags supported by the interface driver. A tag is used to differentiate I/O requests. The SCSI subsystem will use tags from zero through ifsw->if_max_tag, inclusive. Actually, the interface driver is not required to use the tags allocated by the SCSI subsystem, but the SCSI subsystem will not allow more than ifsw->if_max_tag+1 active I/O’s to the bus at any given time (this includes untagged I/O’s). Currently the maximum value of a tag can be 254.
if_scb_size  The number of bytes the SCSI subsystem shall allocate and attach to each scb for use by the interface driver. The if_scb field of scb structure is initialized at scb creation time by services and the data area is bzero’ed by services for each I/O attempt prior to putting the I/O on the select queue. It is not touched by services at any other time.

if_lun_size  The number of bytes the SCSI subsystem shall allocate and attach to each scsi_lun structure for use by the interface driver. The if_lun field of scsi_lun structure is a pointer to ifsw->if_lun_size bytes for the use of interface driver.

if_tgt_size  The number of bytes the SCSI subsystem shall allocate and attach to each scsi_tgt structure for use by the interface driver. The if_tgt field of scsi_tgt structure is a pointer to ifsw->if_tgt_size bytes for the use of the interface driver.

if_bus_size  The number of bytes the SCSI subsystem shall allocate and attach to each scsi_bus structure for use by the interface driver. The if_bus field of scsi_bus structure is a pointer to ifsw

if_open  Pointer to the interface driver’s logical unit open function. This is optional for an interface driver.

if_close  Pointer to the interface driver’s logical unit close function. This is optional for an interface driver.

if_start  Pointer to the interface driver’s start function.

if_reset_bus  Pointer to the interface driver’s Bus Reset function. This is optional for an interface driver.

if_bdr  Pointer to the interface driver’s Bus Device Reset function. This is optional for an interface driver.

if_abort  Pointer to the interface driver’s Abort function. This is optional for an interface driver.

if_io_max_size  Maximum size of I/O request supported by the interface driver. A value of 0 specifies no limit. If set, I/O requests for more than the supported size will be error’ed back by the SCSI services.
if_beg_align, if_end_align
Interface driver data buffer alignment requirement. These fields must be set to \((n - 1)\) where \(n\) is a power of two. SCSI services will ensure the data buffer (bp->b_un.b_addr) is \(n\)-byte aligned. The maximum of both fields is used for buffer alignment.

SEE ALSO

buf(KER4), isc_table_type(KER4), scb(SCSI3)
NAME

`scsi_init_inquiry_data(SCSI3)` – Perform the first Inquiry request on a device

SYNOPSIS

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

int scsi_init_inquiry_data (dev_t dev);
```

PARAMETERS

- **dev**
  - Device number

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

- 0: Successful completion.
- <>0: Error.
NAME

scsi_ioctl(SCSI3) – Standard SCSI ioctl routine

SYNOPSIS

#include <wsio/scsi_ctl.h>

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

PARAMETERS

dev               Device number of the associate device

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

flags             The file access flags

DESCRIPTION

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

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.

scsi_ioctl() supports the following ioctl commands (defined in the <sys/scsi.h> header file):
DIOC_CAPACITY

ioctl(fd, DIOC_CAPACITY, &capacity)

Returns device size in DEV_BSIZE (1024) blocks. Returns information from data saved in LUN structure during the open. The structure capacity is defined in ../sys/diskio.h.

DIOC_DESCRIBE

ioctl(fd, DIOC_DESCRIBE, &describe)

Returns information about the device. The flags field within the describe_type structure contains a write-protect flag for detection of physical write-protection on MO and WORM media. If the size of the describe data is not 32 bytes, EINVAL is returned. The returns information from data is saved in LUN structure during the open. The structure describe is defined in ../sys/diskio.h.

DIOC_GET_PFTIMEOUT

ioctl(fd, DIOC_GET_PFTIMEOUT, &msec)

Returns integer value (msecs) used for timing all LVM requests within bp->b_flags & B_PFTIMEOUT. The structure msec is of type INT and defined in ../sys/diskio.h.

DIOC_RSTCLR

ioctl(fd, DIOC_RSTCLR)

Perform reset on device (Bus Device Reset for SCSI). Defined in the ../sys/diskio.h.

DIOC_SET_PFTIMEOUT

ioctl(fd, DIOC_SET_PFTIMEOUT, &msecs)

Sets integer value (msecs) used for timing all LVM requests within bp->bflags & B_PFTIMEOUT; zero (0) means reset to driver's default. The structure msecs is of the type INT and is defined in ../sys/diskio.h.
• 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.

• int is invalid.

• The caller is not superuser and the open was not with FWRITE.

Error Returns:

[SIOC_FORMAT]

ioctl(fd, SIOC_FORMAT, &sioc_format)

Format device media capacity and block size. Must be superuser and have write access permissions or EACCES returned. Must have exclusive access or EBUSY returned. The structure sioc_format is defined in ..sys/scsi.h.
- **SIOC_GET_BUS_LIMITS**
  
  ```c
  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 `../sys/scsi.h`. See `scsi_ctl (7)`.
  
  Error Returns: None

- **SIOC_GET_BUS_PARMS**
  
  ```c
  ioctl(fd, SIOC_GET_BUS_PARMS, &sioc_bus_parms)
  ```
  
  The structure `sioc_bus_parms` is defined in `../sys/scsi.h`. See `scsi_ctl (7)`.
  
  Error Returns: None.

- **SIOC_GET_LUN_LIMITS**
  
  ```c
  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 `../sys/scsi.h`. See `scsi_ctl (7)`.
  
  Error Returns: None.

- **SIOC_GET_LUN_PARMS**
  
  ```c
  ioctl(fd, SIOC_GET_LUN_PARMS, &sioc_lun_parms)
  ```
  
  The structure `sioc_lun_parms` is defined in `../sys/scsi.h`. See `scsi_ctl (7)`.
  
  Error Returns: None
- **SIOC_GET_TGT_LIMITS**

  ```c
  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 `../sys/scsi.h`. See `scsi_ctl(7)`.

  Error Returns: None

- **SIOC_GET_TGT_PARMS**

  ```c
  ioctl(fd, SIOC_GET_TGT_PARMS, &sioc_tgt_parms)
  ```

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

  Error Returns: None

- **SIOC_INQUIRY**

  ```c
  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 `../sys/scsi.h`.

- **SIOC_EXCLUSIVE**

  ```c
  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_IO**

  ioctl(fd, SIOC_IO, &sctl_io)

  This is used for pass-through I/Os.

  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:

  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.

  Parameter comments:

  The flags bit, SCTL_READ, specifies that the command is expected to have a data-in phase. If `data_length` is greater than zero and SCTL_READ is not set, the command is expected to have a data-out phase. SCTL_INIT_WDTR specifies the SCSI wide data transfer negotiation should be initiated. SCTL_INIT_SDTR specifies that SCSI synchronous data transfer negotiation should be initiated. The status values are the same as those for SIOC_RETURN_STATUS.

  There are several bits in `sctl->flags` that are interesting to the Interface Driver. SCTL_NO_ATN directs the Interface Driver to not assert ATN on selection for the I/O. If the target requests a message out anyway, the Interface Driver should send a No Op.
**SCTL_INIT_WDTR** directs the Interface Driver to initiate WDTR negotiation immediately following the Selection, identity or tag message, whichever comes last, and before sending the cdb for the I/O. If \((\text{tp->state} \& \text{T_ENABLE_SDTR})\) or \((\text{scb->flags} \& \text{SCB_INIT_SDTR})\) is also set, the Interface Driver should initiate SDTR negotiation immediately following the WDTR negotiation.

If **SCTL_INIT_SDTR** is set and **SCTL_INIT_WDTR** is not, the Interface Driver should initiate SDTR negotiation immediately following the Selection, Identify or tag message, whichever comes last, and before sending the cdb for the I/O.

**SCTL_4BYTE** is a hint to the Interface Driver that the target will never change phase while in data phase on other than a 4-byte boundary (with regard to the beginning of the data transfer) without subsequently restoring the data pointer (implicitly or explicitly) to a previously aligned value and retransferring the data up to and beyond the point of the unaligned phase change. The Interface Driver is absolved from ensuring data integrity for the I/O if this bit is set and the above rule is broken. Of course, if the Interface Driver can detect the problem without impacting performance, it should indicate the failure to the SCSI subsystem. Note that **SCTL_4BYTE** does not imply that \(\text{bp->b_bcount}\) is a multiple of four (4) nor that \(\text{bp->b_un.b_addr}\) is 4-byte aligned. Note also that the phase change out of data phase if all \(\text{bp->b_bcount}\) bytes have been transferred is not subject to the alignment restriction.

**SCTL_2BYTE** is the same as **SCB_4BYTE** except phase changes are only restricted to even boundaries.

The following is defined in \(\ldots/h/scsi.h\)

```c
struct sctl_io
{
    unsigned flags;        /* IN:SCTL_READ*/
    unsigned cdb_length;   /* IN*/
    unsigned char cdb[16]; /* IN*/
    void     *date;        /* IN*/
    unsigned data_length;  /* IN*/
    unsigned max_msecs;    /* IN: multi-seconds before abort*/
    unsigned data_xfer;    /* OUT*/
    unsigned cdb_status;   /* OUT: SCSI status*/
    unsigned char sense[256]; /* OUT: SCSI status*/
    unsigned sense_status; /* OUT*/
    unsigned sense_xfer;   /* OUT: bytes of sense data received*/
    unsigned reserved[16]; /* IN: Must be zero; OUT: undefined*/
};
```
#define SIOC_IO _IOWR('S', 22, struct sctl_io)

/*
** values for sctl_io->cdb_status and sctl_io->sense_status
*/
#define SCTL_INVALID_REQUEST  0X0100
#define SCTL_SELECT_TIMEOUT   0x0200
#define SCTL_INCOMPLETE       0x0400

/*
** sctl_io->flags bits
*/
#define SCTL_READ        0x00000001
#define SCTL_INIT_WDTR   0x00000002
#define SCTL_INIT_SDTR   0x00000004
#define SCTL_NO_ATN      0x00000008 /*select without ATN, no SCSI messages*/
#define SCTL_2BYTE       0x00000010 /*target maintains 2-byte alignment*/
#define SCTL_4BYTE       0x00000020 /*target maintains 4-byte alignment*/

Error Returns:

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

`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_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.


- **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_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`, or `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).

  For **SIOC_RETURN_STATUS** ioctl, status in the lun structure utilizes an array of two 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:

  ```c
  /*
   ** 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;
    j = i == SCTL_SELECT_TIMEOUT ? CS_SELECT_TIMEOUT :
       i == SCTL_INVALID_REQUEST ? CS_INVALID_REQUEST :
       i == S_GOOD ? CS_GOOD :
       CS_REQ_COMP_WERROR;
    k = j | lp->if2_scsi_status;
  }
else
  {
    /* Non "STINGRAY" case */
    i = major(dev) == lp->cmd_mode_major
        ? CMD_MODE_ONLY : EVERYTHING;
    k = lp->cdb_status[i];
    lp->cdb_status[i] = -1;
  }

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

Error Returns: None

- **SIIOC_SET_BUS_LIMITS**

  ioctl(fd, SIOC_SET_BUS_LIMITS, &sioc_bus_limits)

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

  Error Returns:

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

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

  **Error Returns:**
  - `[EACCESS]` 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_SET_LUN_LIMITS**

  `ioctl(fd, SIOC_SET_LUN_LIMITS, &sioc_lun_limits)`

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

  **Error Returns:**
  - `[EACCESS]` If not superuser or write permission.
  - `[EINVAL]` If reserved fields are not zero.

- **SIOC_SET_TGT_LIMITS**

  `ioctl(fd, SIOC_SET_TGT_LIMITS, &sioc_tgt_limits)`

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

  **Error Returns:**
  - `[EACCESS]` If not superuser or write permission.
  - `[EINVAL]` If reserved fields are not zero.
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 two 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 become 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 if the request originates 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 `<scsi.h>` header file.

Error Returns:

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

**RETURN VALUES**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>-1</td>
<td>Error.</td>
</tr>
</tbody>
</table>
NAME

`scsi_lun_close` (SCSI3) – Close a device

SYNOPSIS

```c
void 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.

`scsi_lun_close()` performs the following algorithm:

- Acquire the logical unit open/close semaphore.
  - If this is the last non-pass-through close, wait for all non-pass-through I/O's to complete.
  - If this is any non-pass-through close, call device driver close routine, `dd_close()`.
  - If this is the last non-pass-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 non-pass-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**

None
NAME

*scsi_lun_open*(SCSI3) – Open the elements of the hardware path of a SCSI lun

SYNOPSIS

```c
#include <wsio/scsi.ctl.h>

int scsi_lun_open (dev_t dev, struct scsi_ddsw *ddsw, int oflags);
```

PARAMETERS

- `dev` The device number
- `ddsw` Pointer to the non pass_thru driver descriptor
- `oflags` File access flags

DESCRIPTION

Usually called from the device driver's `driver_dev_init()` routine, The `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 `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 `SD_ASSERT`. Does use the lun open/close semaphore.

Uses `kmalloc()` to allocate memory for the `scsi_lun` structure.

Checks if `major(dev) == sctl_ddsw.raw_major`.
RETURN VALUES

0  Successful completion.

[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.

[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(SCSI3) – Read from device

SYNOPSIS

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

PARAMETERS

dev Device number
uio Pointer to a uio structure.

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

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

SEE ALSO

physio(KER2), scsi_ddsw(SCSI4)
NAME

\texttt{scsi\_sense\_action} (SCSI3) – Decode SCSI sense information

SYNOPSIS

\begin{verbatim}
#include <wsio/scsi_ctl.h>

scsi_sense_action (struct buf *bp,
                  struct sense_action *sense_list, size_t n)
\end{verbatim}

PARAMETERS

\begin{itemize}
\item \texttt{bp} Pointer to the I/O buf structure
\item \texttt{sense\_list} List of actions to take.
\item \texttt{n} Number of actions in the list.
\end{itemize}

DESCRIPTION

The \texttt{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.

\texttt{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 \texttt{scsi\_init\_inquiry\_data()} before it can be interpreted. If an I/O completes before the inquiry data is initialized, \texttt{scsi\_sense\_action()} will not match anything other than wild card entries. It will panic if there is no matching entry.
**RETURN VALUES**

0  Successful completion.

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

**SEE ALSO**

`scsi_init_inquiry_data(SCSI3)`
NAME

scsi_strategy(SCSI3) – Enqueue the bp to await resources

SYNOPSIS

#include <wsio/scsi_ctl.h>

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

None.

SEE ALSO

biodone(KER2), scsi_enqueue(SCSI3)
NAME

scsi_write(SCSI3) – Write to device

SYNOPSIS

#include <wsio/scsi_ctl.h>

int scsi_write (dev_t dev, struct uio * uiop);

PARAMETERS

dev The device number
uiop 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)