QNX® Neutrino® RTOS

Devctl and Ioctl Commands

BlackBerry | QNX

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

Electronic edition published: March 06, 2020

Contents

About This Reference

The QNX Neutrino *Devctl and Ioctl Commands* reference describes the main device- and I/O-control commands that you can pass to *devctl()* and *ioctl()*, or that your device driver might need to implement. For more information about these functions, see the *C Library Reference*.

In QNX Neutrino, *ioctl()* is built on top of *devctl()*, and both functions encode the direction in the command using the same two high-order bits. The commands are divided into *classes* to make organization easier. The class name is a single character, although for *devctl()* commands, the class name is defined as a constant that's usually in the form _DCMD_*. Note that some classes are actually part of another class, as indicated in parentheses below.

The following table may help you find information quickly:

Commands	Class	Description
AC97_DEVCTL_*	'Z'	Used internally
AFM_*	'F'	Used internally
ALTQ*, BLUE_*, CBQ_*, CDNR_*, FIFOQ_*, HFSC_*, JOBS_*, PRIQ_*, RED_*, RIO_*, WFQ_*	'Q', 'q'	Alternate queueing framework; see *altq* in the NetBSD documentation
BIOC*	'B'	Berkeley Packet Filter; see *bpf* in the NetBSD documentation
CAN_DEVCTL_*	_DCMD_MISC	Controller Area Networks
CIOC*, CRIO*	'c'	Cryptography; see *crypto* in the NetBSD documentation
DCMD_ALL_*	_DCMD_ALL	Common (all I/O servers)
DCMD_BLK_*	_DCMD_BLK	Block I/O managers
DCMD_BT_*	'Z'	Bluetooth audio
DCMD_CAM_*	_DCMD_CAM	Common Access Methods for devices such as disks or CD-ROMs; used internally
DCMD_CHR_*	_DCMD_CHR	Character devices
DCMD_CP210X_*	_DCMD_MISC	Hardware-specific; see **<hw/cp2103.h>**
DCMD_DSPMGR_*	_DCMD_DSPMGR (_DCMD_MISC)	Obsolete
DCMD_DUMPER_*	_DCMD_MISC	Postmortem dumps

Commands	Class	Description
DCMD_ETFS_*	_DCMD_MEM	Embedded Transaction Filesystem
DCMD_F3S_*	_DCMD_F3S (_DCMD_MEM)	Flash filesystems
DCMD_FSEVMGR_*	_DCMD_FSEVMGR	Filesystem Event Manager
DCMD_FSYS_*	_DCMD_FSYS (_DCMD_BLK)	Filesystem managers
DCMD_HAM_*	_DCMD_MISC	Used internally; see the *High Availability Framework Developer's Guide*
DCMD_I2C_*	_DCMD_I2C (_DCMD_MISC)	See the I2C (Inter-Integrated Circuit) chapter of *Customizing a BSP*
DCMD_INPUT_*	_DCMD_INPUT	Obsolete
DCMD_IO_NET_*	_DCMD_NET	Obsolete
DCMD_IP_*	_DCMD_IP	Internet Protocol stack
DCMD_MEDIA_*	_DCMD_MEDIA	Used internally
DCMD_MEM_*	_DCMD_MEM	Obsolete
DCMD_MEMCLASS_*	_DCMD_MEMCLASS	Used internally
DCMD_MEMMGR_*	_DCMD_PROC	Memory manager
DCMD_MISC_*	_DCMD_MISC	Miscellaneous commands
DCMD_MMCSD_*	_DCMD_CAM	MultiMedia Card/Secure Digital
DCMD_PHOTON_*	_DCMD_PHOTON	Obsolete
DCMD_PPPOE_*	_DCMD_NET	Obsolete
DCMD_PROC_*	_DCMD_PROC	Process manager; see “Controlling processes via the **/proc** filesystem” in the QNX Neutrino *Programmer's Guide*
DCMD_PROF_*	_DCMD_MISC	Profiler
DCMD_PTPD_*	_DCMD_NET	Precision Time Protocol
DCMD_RADIO_*	'Z'	Obsolete
DCMD_SDIO_*	_DCMD_MISC	Secure Digital Input/Output
DCMD_SDMMC_*	_DCMD_MISC	Secure Digital/MultiMedia Card
DCMD_SERCD_*	_DCMD_CAM	Used internally

Commands	Class	Description
DCMD_SIM_*	_DCMD_CAM	Used internally
DCMD_SPI_*	_DCMD_SPI (_DCMD_MISC)	Serial Peripheral Interface
DCMD_UCB1400_*	_DCMD_UCB1400	Obsolete
DIOC*	'D'	See the entry for **pf** in the *Utilities Reference*
FIO*	'f'	File I/O
GRES*	'i'	Generic Route Encapsulation; see *gre* in the NetBSD documentation
KFILTER_*	'k'	Kernel filters; see *kevent* in the NetBSD documentation
PART_*	_DCMD_PARTITION	Obsolete
RND*	'R'	Random number generation; see *rnd* in the NetBSD documentation
SIOC*	'i'	Socket I/O; see the NetBSD documentation at *http://netbsd.org/* (although the information about these commands there is limited)
SND_*	'A', 'C', 'K', 'L', 'R', 'S', 'W'	Used internally; client applications should use the **libasound** API (see the *Audio Developer's Guide*)
SRT_*	'e'	Source-based routing; see the NetBSD documentation at *http://netbsd.org/* (although the information about these commands there is limited)
TAP*	't'	Virtual Ethernet device; see *tap* in the NetBSD documentation
TC*	'T'	Terminals
TIOC*	't'	Terminals
TUN*	't'	Tunnel software network interface; see *tun* in the NetBSD documentation

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	`if( stream == NULL)`
Command options	`-lR`
Commands	`make`
Constants	NULL
Data types	`unsigned short`
Environment variables	*PATH*
File and pathnames	**/dev/null**
Function names	*exit()*
Keyboard chords	**Ctrl–Alt–Delete**
Keyboard input	`Username`
Keyboard keys	**Enter**
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
User-interface components	**Navigator**
Window title	**Options**

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective → Show View**.

We use notes, cautions, and warnings to highlight important messages:

Notes point out something important or useful.

CAUTION: Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

DANGER: Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we typically use a forward slash (/) as a delimiter in pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (*www.qnx.com*). You'll find a wide range of support options, including community forums.

Chapter 1
CAN_DEVCTL_*

This chapter describes the *devctl()* commands that apply to Controller Area Networks and the `dev-can-*` drivers.

These commands are used in Board Support Packages for targets that support CAN, and the details of some commands depend on the board; you can find additional information in **src/hardware/can/***board_name/board_name***can.readme** in your BSP. The `canctl` utility provides a command-line interface to the CAN_DEVCTL_* commands.

You need to use some commands with the file descriptor for a specific receive or transmit mailbox (e.g., **/dev/can1/tx3**).

CAN_DEVCTL_DEBUG_INFO

Get debugging information

Synopsis:

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

#define CAN_DEVCTL_DEBUG_INFO   __DION(_DCMD_MISC, CAN_CMD_CODE + 103)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	CAN_DEVCTL_DEBUG_INFO
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command asks the driver to display debugging information. The information depends on the type of device, but typically includes the registers and mailbox memory, and is sent to *stderr*. For more information, see **src/hardware/can/***board_name/board_name***can.readme** in your Board Support Package.

Input:

None.

Output:

None.

Example:

```
int      ret;

if( (fd = open( "/dev/can1/rx0", O_RDWR)) == -1 )
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}

if (EOK != (ret = devctl(fd, CAN_DEVCTL_DEBUG_INFO, NULL, 0, NULL)))
{
```

```
    fprintf(stderr, "devctl CAN_DEVCTL_DEBUG_INFO: %s\n", strerror(ret));
}
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_DEBUG_INFO2

Get specific debugging information

Synopsis:

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

#define CAN_DEVCTL_DEBUG_INFO2   __DIOT(_DCMD_MISC, CAN_CMD_CODE + 104, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	CAN_DEVCTL_DEBUG_INFO2
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command asks the driver to display specific debugging information. This information depends on the driver and the device; see **src/hardware/can/***board_name/board_name***can.readme** in your Board Support Package.

Input:

A driver-specific number that affects what information the driver displays.

Output:

None.

Example:

```
int      ret;
unsigned print_debug_all = 1;

if( (fd = open( "/dev/can1/rx0", O_RDWR)) == -1 )
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}

if (EOK != (ret = devctl(fd, CAN_DEVCTL_DEBUG_INFO2, &print_debug_all,
                         sizeof(print_debug_all), NULL)))
```

```
{
    fprintf(stderr, "devctl CAN_DEVCTL_DEBUG_INFO: %s\n", strerror(ret));
}
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_ERROR

Get error information

Synopsis:

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

#define CAN_DEVCTL_ERROR   __DIOF(_DCMD_MISC, CAN_CMD_CODE + 102, struct can_devctl_error)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	CAN_DEVCTL_ERROR
dev_data_ptr	A pointer to a `CAN_DEVCTL_ERROR` structure
n_bytes	`sizeof(CAN_DEVCTL_ERROR)`
dev_info_ptr	NULL

Description:

This command gets driver-specific error information. The driver might reset some of the information when you use this command; see **src/hardware/can/***board_name/board_name***can.readme** in your BSP.

Input:

None.

Output:

The error information. The `CAN_DEVCTL_ERROR` structure is defined as follows:

```
typedef struct can_devctl_error {
        uint32_t                drvr1;
        uint32_t                drvr2;
        uint32_t                drvr3;
        uint32_t                drvr4;
} CAN_DEVCTL_ERROR;
```

For details, see **src/hardware/can/***board_name/board_name***can.readme** in your Board Support Package.

Example:

```
int      ret;
struct can_devctl_error derror;

if( (fd = open( "/dev/can1/rx0", O_RDWR)) == -1 )
```

```
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}

if(EOK != (ret = devctl(fd, CAN_DEVCTL_ERROR, &derror, sizeof(derror), NULL)))
{
    fprintf(stderr, "devctl CAN_DEVCTL_ERROR: %s\n", strerror(ret));
} else {
    printf("ERROR drvr1 = 0x%X\n", derror.drvr1);
    printf("ERROR drvr2 = 0x%X\n", derror.drvr2);
    printf("ERROR drvr3 = 0x%X\n", derror.drvr3);
    printf("ERROR drvr4 = 0x%X\n", derror.drvr4);
}
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_GET_INFO

Get information about the CAN configuration

Synopsis:

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

#define CAN_DEVCTL_GET_INFO    __DIOF(_DCMD_MISC, CAN_CMD_CODE + 101,  struct can_devctl_info)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	CAN_DEVCTL_GET_INFO
dev_data_ptr	A pointer to a `struct can_devctl_info`
n_bytes	`sizeof(struct can_devctl_info)`
dev_info_ptr	NULL

Description:

This command gets information about the CAN configuration.

Input:

None.

Output:

A filled-in `struct can_devctl_info`, which is defined in **<sys/can_dcmd.h>** as follows:

```
typedef struct can_devctl_info
{
    char                description[64];    /* CAN device description */
    uint32_t            msgq_size;          /* Number of message queue objects */
    uint32_t            waitq_size;         /* Number of client wait queue objects */
    CANDEV_MODE         mode;               /* CAN driver mode - I/O or raw frames */
    uint32_t            bit_rate;           /* Bit rate */
    uint16_t            bit_rate_prescaler; /* Bit rate prescaler */
    uint8_t             sync_jump_width;    /* Time quantum Sync Jump Width */
    uint8_t             time_segment_1;     /* Time quantum Time Segment 1 */
    uint8_t             time_segment_2;     /* Time quantum Time Segment 2 */
    uint32_t            num_tx_mboxes;      /* Number of TX Mailboxes */
    uint32_t            num_rx_mboxes;      /* Number of RX Mailboxes */
    uint32_t            loopback_external;  /* External loopback is enabled */
    uint32_t            loopback_internal;  /* Internal loopback is enabled */
```

```
    uint32_t            autobus_on;         /* Auto timed bus on after bus off */
    uint32_t            silent;             /* Receiver only, no ack generation */
} CAN_DEVCTL_INFO;
```

The *mode* is one of the following:

CANDEV_MODE_IO

All mailboxes are set up to transmit or receive specific message IDs.

CANDEV_MODE_RAW_FRAME

All messages found on the bus are received and put into a single mailbox. Similarly, there is one transmit mailbox; the message ID and type of frame are defined within the message instead of in the mailbox.

See also:

CAN_DEVCTL_GET_STATS

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_GET_MFILTER

Get the current message filter

Synopsis:

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

#define CAN_DEVCTL_GET_MFILTER   __DIOF(_DCMD_MISC, CAN_CMD_CODE + 2, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device. This must be for a receive mailbox (e.g., **/dev/can1/rx2**).
dcmd	CAN_DEVCTL_GET_MFILTER
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command gets the current message filter, which is a mask that specifies which bits in the message IDs to ignore. The default mask is 0x0, so no bits are ignored.

Input:

None.

Output:

The message filter.

Example:

```
int      ret;
uint32_t val = 0;

if( (fd = open( "/dev/can1/rx2", O_RDWR)) == -1 )
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}

if (EOK != (ret = devctl(fd, CAN_DEVCTL_GET_MFILTER, &val, sizeof(val), NULL)))
{
```

```
    fprintf(stderr, "devctl CAN_DEVCTL_GET_MFILTER: %s\n", strerror(ret));
} else {
    printf("MFILTER = 0x%X\n", val);
}
```

See also:

CAN_DEVCTL_SET_MFILTER

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_GET_MID

Get a message ID

Synopsis:

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

#define CAN_DEVCTL_GET_MID   __DIOF(_DCMD_MISC, CAN_CMD_CODE + 0, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device. This must be for a transmit or receive mailbox (e.g., **/dev/can1/rx2**).
dcmd	CAN_DEVCTL_GET_MID
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command gets the message ID for the mailbox associated with the file descriptor.

Input:

None.

Output:

The message ID. The form depends on whether or not the driver is using extended MIDs:

- In standard 11-bit MIDs, bits 18–28 define the MID.
- In extended 29-bit MIDs, bits 0–28 define the MID.

Example:

```
int      ret;
uint32_t val;

if( (fd = open( "/dev/can1/rx2", O_RDWR)) == -1 )
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}
```

```
if(EOK != (ret = devctl(fd, CAN_DEVCTL_GET_MID, &val, sizeof(val), NULL)))
{
    fprintf(stderr, "devctl CAN_DEVCTL_GET_MID: %s\n", strerror(ret));
} else {
    printf("GET_MID = 0x%X\n", val);
}
```

See also:

CAN_DEVCTL_SET_MID

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_GET_PRIO

Get the priority for transmitted messages

Synopsis:

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

#define CAN_DEVCTL_GET_PRIO    __DIOF(_DCMD_MISC, CAN_CMD_CODE + 4, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device. This must be for a transmit mailbox (e.g., **/dev/can1/tx3**).
dcmd	CAN_DEVCTL_GET_PRIO
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command gets the priority of messages for a transmit mailbox.

Input:

None.

Output:

The priority.

Example:

```
int      ret;
uint32_t val;

if( (fd = open( "/dev/can1/tx3", O_RDWR)) == -1 )
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}

if(EOK != (ret = devctl(fd, CAN_DEVCTL_GET_PRIO, &val, sizeof(val), NULL)))
{
    fprintf(stderr, "devctl CAN_DEVCTL_GET_PRIO: %s\n", strerror(ret));
```

```
} else {
    printf("GET_PRIO = %u\n", val);
}
```

See also:

CAN_DEVCTL_SET_PRIO

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_GET_STATS

Get CAN statistics

Synopsis:

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

#define CAN_DEVCTL_GET_STATS    __DIOF(_DCMD_MISC, CAN_CMD_CODE + 100,  struct can_devctl_stats)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	CAN_DEVCTL_GET_STATS
dev_data_ptr	A pointer to a `struct can_devctl_stats`
n_bytes	`sizeof(struct can_devctl_stats)`
dev_info_ptr	NULL

Description:

This command gets CAN statistics, including some that you can use to monitor CAN bus issues.

Input:

None.

Output:

A filled-in `struct can_devctl_stats`, which is defined in **<sys/can_dcmd.h>** as follows:

```
typedef struct can_devctl_stats
{
    uint32_t                    transmitted_frames;
    uint32_t                    received_frames;
    uint32_t                    missing_ack;
    uint32_t                    total_frame_errors;
    uint32_t                    stuff_errors;
    uint32_t                    form_errors;
    uint32_t                    dom_bit_recess_errors;
    uint32_t                    recess_bit_dom_errors;
    uint32_t                    parity_errors;
    uint32_t                    crc_errors;
    uint32_t                    hw_receive_overflows;
    uint32_t                    sw_receive_q_full;
    uint32_t                    error_warning_state_count;
```

```
    uint32_t                    error_passive_state_count;
    uint32_t                    bus_off_state_count;
    uint32_t                    bus_idle_count;
    uint32_t                    power_down_count;
    uint32_t                    wake_up_count;
    uint32_t                    rx_interrupts;
    uint32_t                    tx_interrupts;
    uint32_t                    total_interrupts;
} CAN_DEVCTL_STATS;
```

See also:

CAN_DEVCTL_GET_INFO

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_GET_TIMESTAMP

Get the device timestamp

Synopsis:

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

#define CAN_DEVCTL_GET_TIMESTAMP   __DIOF(_DCMD_MISC, CAN_CMD_CODE + 6, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	CAN_DEVCTL_GET_TIMESTAMP
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command gets the timestamp.

Input:

None.

Output:

The timestamp.

Example:

```
int      ret;
uint32_t val;

if( (fd = open( "/dev/can1/rx0", O_RDWR)) == -1 )
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}

if(EOK != (ret = devctl(fd, CAN_DEVCTL_GET_TIMESTAMP, &val, sizeof(val), NULL)))
{
    fprintf(stderr, "devctl CAN_DEVCTL_GET_TIMESTAMP: %s\n", strerror(ret));
} else {
```

```
    printf("GET_TIMESTAMP = 0x%X\n", val);
}
```

See also:

CAN_DEVCTL_SET_TIMESTAMP

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_READ_CANMSG_EXT

Read an extended CAN message

Synopsis:

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

#define CAN_DEVCTL_READ_CANMSG_EXT   __DIOF(_DCMD_MISC, CAN_CMD_CODE + 8, struct can_msg)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	CAN_DEVCTL_READ_CANMSG_EXT
dev_data_ptr	A pointer to a `struct can_msg`
n_bytes	`sizeof(struct can_msg)`
dev_info_ptr	NULL

Description:

This command reads an extended CAN message, blocking if there's no message available unless you opened the device with O_NONBLOCK.

Input:

None.

Output:

A filled-in `can_msg` structure, which is defined in **<sys/can_dcmd.h>** as follows:

```
#define CAN_MSG_DATA_MAX  0x8 /* Max number of data bytes in a CAN message
                                 as defined by CAN spec */

/* Extended CAN Message */
typedef struct can_msg_ext {
    uint32_t        timestamp;         /* CAN message timestamp */
    uint32_t        is_extended_mid;   /* 1=29-bit MID, 0=11-bit MID */
    uint32_t        is_remote_frame;   /* 1=remote frame request, 0=data frame */
} CAN_MSG_EXT;

/* CAN Message */
typedef struct can_msg {
        /* Pre-allocate CAN messages to the max data size */
        uint8_t        dat[CAN_MSG_DATA_MAX];  /* CAN message data */
```

```
        uint8_t        len;                    /* Actual CAN message data length */
        uint32_t       mid;                    /* CAN message identifier */
        CAN_MSG_EXT    ext;                    /* Extended CAN message info */
} CAN_MSG;
```

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EAGAIN

There are no messages in the queue, and you opened the device with O_NONBLOCK.

EINVAL

The file descriptor doesn't correspond to a receive mailbox.

Example:

```
int      ret;
unsigned i;
uint8_t  dat[CAN_MSG_DATA_MAX];
char     dat_str[sizeof(dat) + 1]; /* max size of a CAN message + '\0' */

if( (fd = open( "/dev/can1/rx0", O_RDWR)) == -1 )
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}

if(EOK != (ret = devctl(fd, CAN_DEVCTL_READ_CANMSG_EXT, &canmsg, sizeof(canmsg),
                        NULL)))
{
    fprintf(stderr, "devctl CAN_DEVCTL_READ_CANMSG_EXT: %s\n", strerror(ret));
} else {
    printf("READ_CANMSG_EXT:\n");
    printf("mid = 0x%X\n", canmsg.mid);
    printf("timestamp = 0x%X\n", canmsg.ext.timestamp);
    printf("dat len = %d\n", canmsg.len);

    /* Copy the data */
    memcpy(dat, canmsg.dat, canmsg.len);

    /* Print the data */
    printf("dat =");
    for (i=0; i < canmsg.len; i++) {
        printf(" %.2x", dat[i]);
        dat_str[i] = isprint(dat[i]) ? dat[i] : '.';
    }
    dat_str[i] = '\0';
    printf("\t%s\n", dat_str);
}
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_RX_FRAME_RAW_BLOCK

Get a message in raw frame mode (blocking)

Synopsis:

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

#define CAN_DEVCTL_RX_FRAME_RAW_BLOCK   __DIOF(_DCMD_MISC, CAN_CMD_CODE + 10,   struct can_msg)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device. This must be for a receive mailbox (e.g., **/dev/can1/rx2**).
dcmd	CAN_DEVCTL_RX_FRAME_RAW_BLOCK
dev_data_ptr	A pointer to a `struct can_msg`
n_bytes	`sizeof(struct can_msg)`
dev_info_ptr	NULL

Description:

This command gets the next valid received message from the queue when the driver is in raw frame mode. If there are no messages, the command blocks.

Input:

None.

Output:

A filled-in `can_msg` structure, which is defined in **<sys/can_dcmd.h>** as follows:

```
#define CAN_MSG_DATA_MAX  0x8 /* Max number of data bytes in a CAN message
                                 as defined by CAN spec */

/* Extended CAN Message */
typedef struct can_msg_ext {
    uint32_t        timestamp;         /* CAN message timestamp */
    uint32_t        is_extended_mid;   /* 1=29-bit MID, 0=11-bit MID */
    uint32_t        is_remote_frame;   /* 1=remote frame request, 0=data frame */
} CAN_MSG_EXT;

/* CAN Message */
typedef struct can_msg {
        /* Pre-allocate CAN messages to the max data size */
```

```
        uint8_t         dat[CAN_MSG_DATA_MAX];  /* CAN message data */
        uint8_t         len;                    /* Actual CAN message data length */
        uint32_t        mid;                    /* CAN message identifier */
        CAN_MSG_EXT     ext;                    /* Extended CAN message info */
} CAN_MSG;
```

See also:

CAN_DEVCTL_RX_FRAME_RAW_NOBLOCK, CAN_DEVCTL_TX_FRAME_RAW

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_RX_FRAME_RAW_NOBLOCK

Get a message in raw frame mode (nonblocking)

Synopsis:

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

#define CAN_DEVCTL_RX_FRAME_RAW_NOBLOCK __DIOF(_DCMD_MISC, CAN_CMD_CODE + 9, struct can_msg)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device. This must be for a receive mailbox (e.g., **/dev/can1/rx2**).
dcmd	CAN_DEVCTL_RX_FRAME_RAW_NOBLOCK
dev_data_ptr	A pointer to a `struct can_msg`
n_bytes	`sizeof(struct can_msg)`
dev_info_ptr	NULL

Description:

This command gets the next valid received message from the queue when the driver is in raw frame mode. If there are no messages, the command doesn't block.

Input:

None.

Output:

A filled-in `can_msg` structure, which is defined in **<sys/can_dcmd.h>** as follows:

```
#define CAN_MSG_DATA_MAX  0x8 /* Max number of data bytes in a CAN message
                                 as defined by CAN spec */

/* Extended CAN Message */
typedef struct can_msg_ext {
    uint32_t        timestamp;         /* CAN message timestamp */
    uint32_t        is_extended_mid;   /* 1=29-bit MID, 0=11-bit MID */
    uint32_t        is_remote_frame;   /* 1=remote frame request, 0=data frame */
} CAN_MSG_EXT;

/* CAN Message */
typedef struct can_msg {
        /* Pre-allocate CAN messages to the max data size */
```

```
        uint8_t         dat[CAN_MSG_DATA_MAX];  /* CAN message data */
        uint8_t         len;                    /* Actual CAN message data length */
        uint32_t        mid;                    /* CAN message identifier */
        CAN_MSG_EXT     ext;                    /* Extended CAN message info */
} CAN_MSG;
```

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EAGAIN

There are no messages in the queue.

See also:

CAN_DEVCTL_RX_FRAME_RAW_BLOCK, CAN_DEVCTL_TX_FRAME_RAW

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_SET_MFILTER

Set the message filter

Synopsis:

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

#define CAN_DEVCTL_SET_MFILTER   __DIOT(_DCMD_MISC, CAN_CMD_CODE + 3, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device. This must be for a receive mailbox (e.g., **/dev/can1/rx2**).
dcmd	CAN_DEVCTL_SET_MFILTER
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command sets the message filter, which is a mask that specifies which bits in the message IDs to ignore. The default mask is 0x0, so no bits are ignored.

Input:

The new message filter.

Output:

None.

Example:

```
int      ret;
uint32_t val;

if( (fd = open( "/dev/can1/rx2", O_RDWR)) == -1 )
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}

/* Set val to the new filter. */
val = strtoul(optarg, NULL, 0);
```

```
if(EOK != (ret = devctl(fd, CAN_DEVCTL_SET_MFILTER, &val, sizeof(val), NULL)))
{
    fprintf(stderr, "devctl CAN_DEVCTL_SET_MFILTER: %s\n", strerror(ret));
}
```

See also:

CAN_DEVCTL_GET_MFILTER

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_SET_MID

Set the message ID

Synopsis:

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

#define CAN_DEVCTL_SET_MID   __DIOT(_DCMD_MISC, CAN_CMD_CODE + 1, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device. This must be for a transmit or receive mailbox (e.g., **/dev/can1/rx2**).
dcmd	CAN_DEVCTL_SET_MID
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command sets the message ID for the mailbox associated with the file descriptor.

Input:

The message ID. The form depends on whether or not the driver is using extended MIDs:

- In standard 11-bit MIDs, bits 18–28 define the MID.
- In extended 29-bit MIDs, bits 0–28 define the MID.

Output:

None.

Example:

```
int      ret;
uint32_t val;

if( (fd = open( "/dev/can1/rx2", O_RDWR)) == -1 )
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}
```

```
/* Set the new message ID. */
val = strtoul(optarg, NULL, 16);

if(EOK != (ret = devctl(fd, CAN_DEVCTL_SET_MID, &val, sizeof(val), NULL)))
{
    fprintf(stderr, "devctl CAN_DEVCTL_SET_MID: %s\n", strerror(ret));
}
```

See also:

CAN_DEVCTL_GET_MID, CAN_DEVCTL_SET_MFILTER

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_SET_PRIO

Set the priority for transmitted messages

Synopsis:

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

#define CAN_DEVCTL_SET_PRIO    __DIOT(_DCMD_MISC, CAN_CMD_CODE + 5, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device. This must be for a transmit mailbox (e.g., **/dev/can1/tx3**).
dcmd	CAN_DEVCTL_SET_PRIO
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command sets the priority of messages for a transmit mailbox. The hardware uses these priorities to determine which CAN message to transmit first if there are multiple messages waiting to be transmitted.

Some devices use fixed priorities for the mailboxes. For example, the first message object (i.e., the one that RX0 receives) might have the highest priority, and the last might have the lowest.

Input:

The new priority, which must not exceed the maximum for the device (usually defined as a constant whose name is in the form *board*`_CANMCF_TPL_MAXVAL`).

Output:

None.

Example:

```
int      ret;
uint32_t val;

if( (fd = open( "/dev/can1/tx3", O_RDWR)) == -1 )
```

```
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}

/* Get the new priority. */
val = strtoul(optarg, NULL, 0);

if(EOK != (ret = devctl(fd, CAN_DEVCTL_SET_PRIO, &val, sizeof(val), NULL)))
{
    fprintf(stderr, "devctl CAN_DEVCTL_SET_PRIO: %s\n", strerror(ret));
}
```

See also:

CAN_DEVCTL_GET_PRIO

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_SET_TIMESTAMP

Set the device timestamp

Synopsis:

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

#define CAN_DEVCTL_SET_TIMESTAMP   __DIOT(_DCMD_MISC, CAN_CMD_CODE + 7, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	CAN_DEVCTL_SET_TIMESTAMP
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command sets the timestamp.

Input:

The new timestamp.

Output:

None.

Example:

```
int      ret;
uint32_t val;

if( (fd = open( "/dev/can1/rx0", O_RDWR)) == -1 )
{
    printf("open of %s failed \n", devname);
    exit(EXIT_FAILURE);
}

/* Get the new value for the timestamp. */
val = strtoul(optarg, NULL, 0);

if(EOK != (ret = devctl(fd, CAN_DEVCTL_SET_TIMESTAMP, &val, sizeof(val), NULL)))
```

```
{
    fprintf(stderr, "devctl CAN_DEVCTL_SET_TIMESTAMP: %s\n", strerror(ret));
}
```

See also:

CAN_DEVCTL_GET_TIMESTAMP

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

CAN_DEVCTL_TX_FRAME_RAW

Transmit a message in raw frame mode

Synopsis:

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

#define CAN_DEVCTL_TX_FRAME_RAW   __DIOT(_DCMD_MISC, CAN_CMD_CODE + 11,   struct can_msg)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	CAN_DEVCTL_TX_FRAME_RAW
dev_data_ptr	A pointer to a `struct can_msg`
n_bytes	`sizeof(struct can_msg)`
dev_info_ptr	NULL

Description:

This command places a message on the outgoing queue, in raw frame mode.

Input:

A filled-in `can_msg` structure, which is defined in **<sys/can_dcmd.h>** as follows:

```
#define CAN_MSG_DATA_MAX  0x8 /* Max number of data bytes in a CAN message
                                 as defined by CAN spec */

/* Extended CAN Message */
typedef struct can_msg_ext {
    uint32_t        timestamp;         /* CAN message timestamp */
    uint32_t        is_extended_mid;   /* 1=29-bit MID, 0=11-bit MID */
    uint32_t        is_remote_frame;   /* 1=remote frame request, 0=data frame */
} CAN_MSG_EXT;

/* CAN Message */
typedef struct can_msg {
        /* Pre-allocate CAN messages to the max data size */
        uint8_t        dat[CAN_MSG_DATA_MAX];  /* CAN message data */
        uint8_t        len;                    /* Actual CAN message data length */
        uint32_t       mid;                    /* CAN message identifier */
        CAN_MSG_EXT    ext;                    /* Extended CAN message info */
} CAN_MSG;
```

Output:

None.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EAGAIN

The transmit queue is currently full.

EINVAL

The file descriptor doesn't correspond to a transmit mailbox.

See also:

CAN_DEVCTL_RX_FRAME_RAW_BLOCK, CAN_DEVCTL_RX_FRAME_RAW_NOBLOCK

devctl() in the QNX Neutrino *C Library Reference*

`canctl` in the *Utilities Reference*

Chapter 2
DCMD_ALL_*

This chapter describes the *devctl()* commands that are common to all I/O servers.

DCMD_ALL_FADVISE

Pass file advice to the filesystem

Synopsis:

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

#define DCMD_ALL_FADVISE __DIOT(_DCMD_ALL, 6, struct _fadvise)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_ALL_FADVISE
dev_data_ptr	A pointer to a `struct _fadvise` with the advice filled in (see below).
n_bytes	`sizeof(struct _fadvise)`
dev_info_ptr	NULL

Description:

This command passes file advice to the filesystem. Clients usually use the *posix_fadvise()* cover function to generate this request:

```
posix_fadvise(fd, offset, len, advice);
```

Input:

A pointer to a `struct _fadvise` with the advice filled in. This structure is defined as:

```
struct _fadvise {
        int                     advice;
        int                     spare;
        off64_t                 offset;
        off64_t                 len;
};
```

The members include the following:

advice

The advice you want to give; one of:

- POSIX_FADV_NORMAL — the application has no advice to give on its behavior with respect to the specified data. This is the default characteristic if no advice is given for an open file.

- POSIX_FADV_SEQUENTIAL — the application expects to access the specified data sequentially from lower offsets to higher offsets.
- POSIX_FADV_RANDOM — the application expects to access the specified data in a random order.
- POSIX_FADV_WILLNEED — the application expects to access the specified data in the near future.
- POSIX_FADV_DONTNEED — the application expects that it will not access the specified data in the near future.
- POSIX_FADV_NOREUSE — the application expects to access the specified data once and then not reuse it thereafter.

spare

Not used; set it to zero.

offset

The offset of the data that you want to provide advice about.

len

The length of the data.

Output:

None.

Example:

```
struct _fadvise  a;

a.advice = advice;
a.offset = offset;
a.len = len;
a.spare = 0;

if(devctl(fd, DCMD_ALL_FADVISE, &a, sizeof a, NULL) != EOK)
{
   /* Error */
}
```

See also:

devctl(), *posix_fadvise()* in the QNX Neutrino *C Library Reference*

DCMD_ALL_GETFLAGS

Get O_* *file status flags*

Synopsis:

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

#define DCMD_ALL_GETFLAGS    __DIOF(_DCMD_ALL, 1, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_ALL_GETFLAGS
dev_data_ptr	A pointer to an `int` where the device can store the flags.
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command gets O_* file status flags and file access modes, described in **<fcntl.h>**. Clients usually use the *fcntl()* cover function to generate this request:

```
fcntl(fd, F_GETFL);
```

Input:

None.

Output:

A bitwise OR of zero or more of the following status flags (identified by the QNX Neutrino O_SETFLAG mask):

- O_APPEND
- O_DSYNC
- O_LARGEFILE
- O_NONBLOCK
- O_RSYNC
- O_SYNC

and the following file access modes:

- O_RDONLY

- O_RDWR
- O_WRONLY

For more information, see *open()* in the QNX Neutrino *C Library Reference*.

Example:

```
int flags;

if(devctl(fd, DCMD_ALL_GETFLAGS, &flags, sizeof flags, NULL) != EOK)
{
   /* Error */
}
return flags;
```

See also:

DCMD_ALL_SETFLAGS

devctl(), *fcntl()*, *open()* in the QNX Neutrino *C Library Reference*

DCMD_ALL_GETMOUNTFLAGS

Get ST_* *mount flags*

Synopsis:

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

#define DCMD_ALL_GETMOUNTFLAGS  __DIOF(_DCMD_ALL, 3, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_ALL_GETMOUNTFLAGS
dev_data_ptr	A pointer to a `int` where the device can store the flags
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command gets the ST_* mount flags, described in **<sys/statvfs.h>**; for more information, see the entry for *statvfs()* in the *C Library Reference*. This command is sometimes used by *access()* to determine write or execute modes.

Input:

None.

Output:

A bitwise OR of the mount flags:

- ST_RDONLY
- ST_NOEXEC
- ST_NOSUID
- ST_NOCREAT
- ST_OFF32
- ST_NOATIME

Example:

```
int flags;
```

```
if (devctl(fd, DCMD_ALL_GETMOUNTFLAGS, &flags, sizeof flags, NULL) != EOK)
{
   /* Error */
}
```

See also:

access(), *devctl()*, *stavfs()* in the QNX Neutrino *C Library Reference*

DCMD_ALL_GETOWN

Get the owner of a file descriptor

Synopsis:

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

#define DCMD_ALL_GETOWN __DIOF(_DCMD_ALL, 4, pid_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_ALL_GETOWN
dev_data_ptr	A pointer to a `pid_t` where the device can store the owner.
n_bytes	`sizeof(pid_t)`
dev_info_ptr	NULL

Description:

This command gets the owner of the given file descriptor. Clients usually use the *fcntl()* cover function to generate this request:

```
fcntl(fd, F_GETOWN);
```

It's generally used for sockets where the owner will receive SIGIO and SIGURG signals.

Input:

None.

Output:

The process ID of the current owner.

Example:

```
if(devctl(fd, DCMD_ALL_GETOWN, &pid, sizeof pid, NULL) != EOK)
{
   /* Error */
}
```

See also:

DCMD_ALL_SETOWN

devctl(), *fcntl()* in the QNX Neutrino *C Library Reference*

DCMD_ALL_SETFLAGS

Set O_* *file status flags*

Synopsis:

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

#define DCMD_ALL_SETFLAGS    __DIOT(_DCMD_ALL, 2, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_ALL_SETFLAGS
dev_data_ptr	A pointer to a `int` containing the flags you want to set.
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command sets O_* file status flags, described in **<fcntl.h>**. You can't use this command to set the file access bits. Clients usually use the *fcntl()* cover function to generate this request:

```
fcntl(fd, F_SETFL, flags);
```

Input:

A bitwise OR of zero or more of the following status flags (identified by the QNX Neutrino O_SETFLAG mask):

- O_APPEND
- O_DSYNC
- O_LARGEFILE
- O_NONBLOCK
- O_RSYNC
- O_SYNC

For more information, see *open()* in the QNX Neutrino *C Library Reference*.

Output:

None.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EINVAL

You specified one of the *SYNC flags, but IOFUNC_PC_SYNC_IO isn't set in the device's mount configuration.

Example:

```
if (devctl(fd, DCMD_ALL_SETFLAGS, &flags, sizeof flags, NULL) != EOK)
{
   /* Error */
}
```

See also:

DCMD_ALL_GETFLAGS

devctl(), *fcntl()*, *open()* in the QNX Neutrino *C Library Reference*

DCMD_ALL_SETOWN

Set the owner of a file descriptor

Synopsis:

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

#define DCMD_ALL_SETOWN __DIOT(_DCMD_ALL, 5, pid_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_ALL_SETOWN
dev_data_ptr	A pointer to a `pid_t` that specifies the owner.
n_bytes	`sizeof(pid_t)`
dev_info_ptr	NULL

Description:

This command sets the owner of the given file descriptor. Clients usually use the *fcntl()* cover function to generate this request:

```
fcntl(fd, F_SETOWN, pid);
```

Input:

The process ID of the owner.

Output:

None.

Example:

```
if(devctl(fd, DCMD_ALL_SETOWN, &pid, sizeof pid, NULL) != EOK)
{
   /* Error */
}
```

See also:

DCMD_ALL_GETOWN

devctl(), *fcntl()* in the QNX Neutrino *C Library Reference*

Chapter 3
DCMD_BLK_*

This chapter describes the *devctl()* commands that apply to block I/O. Note that _DCMD_BLK and _DCMD_FSYS are the same value.

DCMD_BLK_FORCE_RELEARN, DCMD_FSYS_FORCE_RELEARN

Trigger a media reversioning and cache invalidation

Synopsis:

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

#define DCMD_BLK_FORCE_RELEARN      __DION(_DCMD_BLK, 2)
#define DCMD_FSYS_FORCE_RELEARN     DCMD_BLK_FORCE_RELEARN
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_BLK_FORCE_RELEARN or DCMD_FSYS_FORCE_RELEARN
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

These commands—which are identical—trigger a media reversioning and cache invalidation (for removable media). They're also used to sync-up a filesystem if utilities play with it "behind its back."

In order to use these commands, your process needs to have the `vfs/relearn` (BLK_ABILITY_RELEARN) custom ability enabled. For more information, see *procmgr_ability()* and *procmgr_ability_lookup()* in the *C Library Reference*.

Input:

None.

Output:

None.

Example:

```
if ( devctl( fd, DCMD_BLK_FORCE_RELEARN, NULL, 0, NULL ) != EOK)
{
   /* Error */
}
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_BLK_PART_DESCRIPTION

Get an extended description of a partition

Synopsis:

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

#define DCMD_BLK_PART_DESCRIPTION   __DIOF(_DCMD_BLK, 3, struct partition_description)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_BLK_PART_DESCRIPTION
dev_data_ptr	A pointer to a `struct partition_description` that the device can fill in (see below)
n_bytes	`sizeof(struct partition_description)`
dev_info_ptr	NULL

Description:

This command gets an extended description of the partition for the device associated with the given file descriptor.

Input:

None.

Output:

The `struct partition_description` structure is defined as follows:

```
struct partition_description {
        char                    scheme[4];
        uint32_t                index;
        uint64_t                header;
        char                    fsdll[16];
        uint32_t                sequence;
        char                    reserved[92];
        union {
                struct part_pc_entry {
                        uint8_t         boot_ind;
                        uint8_t         beg_head;
                        uint8_t         beg_sector;
                        uint8_t         beg_cylinder;
```

```
                        uint8_t         os_type;
                        uint8_t         end_head;
                        uint8_t         end_sector;
                        uint8_t         end_cylinder;
                        uint32_t        part_offset;
                        uint32_t        part_size;
                } pc;
                struct part_gpt_entry {
                        uint8_t         PartitionTypeGuid[16];
                        uint8_t         UniquePartitionGuid[16];
                        uint64_t        StartingLBA;
                        uint64_t        EndingLBA;
                        uint64_t        Attributes;
                        uint16_t        PartitionName[36];
                } gpt;
        } entry;
};
```

The members include:

scheme

A string that identifies the layout scheme for the partition:

Scheme	Constant	Value
Personal Computer-style	FS_PARTITION_PC	`"pc\x00\x00"`
Globally Unique ID Partition Table	FS_PARTITION_GPT	`"gpt\x00"`

index

The index in the partition table.

header

The location of the partition header.

fsdll

A shortened version of the name of the shared object that supports the partition (for example, `qnx6` for the Power-Safe filesystem's shared object, **fs-qnx6.so**).

sequence

The partition enumeration order.

entry

A union that contains partition-specific information:

- *pc* — the entry for a PC-style partition:

  boot_ind

  `0x80` if the partition is bootable, `0x00` if it isn't.

beg_head

The beginning head number.

beg_sector

The beginning sector number.

beg_cylinder

The beginning cylinder number.

os_type

The partition type; see "Partitions" in the Filesystems chapter of the *System Architecture* guide.

end_head

The end head number.

end_sector

The end sector number.

end_cylinder

The end cylinder number.

part_offset

The offset of the partition, in bytes.

part_size

The number of sectors in the partition.

- *gpt* — the entry for a GUID partition:

  PartitionTypeGuid

  The globally unique identifier for the partition type.

  UniquePartitionGuid

  The partition's globally unique identifier.

  StartingLBA

  The starting logical block address.

  EndingLBA

  The ending logical block address, inclusive.

  Attributes

  Flags that indicate attributes, such as read-only.

PartitionName

The name of the partition.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

ENOTTY

The file descriptor is for a raw block device, and hence there's no partition information.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <devctl.h>
#include <sys/dcmd_blk.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>

int main (void)
{
    int fd;
    int ret;
    struct partition_description pd;
    uint64_t slba, elba;

    fd = open ("/dev/hd0t179", O_RDONLY);
    if (fd == -1)
    {
        perror ("open() failed");
        return (EXIT_FAILURE);
    }

    /*  Determine the partition's start and end LBAs. */
    ret = devctl(fd, DCMD_BLK_PART_DESCRIPTION, &pd, sizeof pd, NULL);

    if (ret == EOK)
    {
        if (strcmp(pd.scheme, FS_PARTITION_PC) == 0) {
            printf ("PC: ");
            slba = pd.entry.pc.part_offset;
            elba = pd.entry.pc.part_offset + pd.entry.pc.part_size - 1;
        }
        else if (strcmp(pd.scheme, FS_PARTITION_GPT) == 0) {
            printf ("GPT: ");
            slba = pd.entry.gpt.StartingLBA;
            elba = pd.entry.gpt.EndingLBA;
        }
        printf ("start: %lld end: %lld\n", slba, elba);
```

```
    } else {
        printf ("DCMD_BLK_PART_DESCRIPTION failed: %s\n", strerror(ret) );
        return (EXIT_FAILURE);
    }

    return (EXIT_SUCCESS);
}
```

See also:

DCMD_BLK_PARTENTRY

devctl() in the QNX Neutrino *C Library Reference*

DCMD_BLK_PARTENTRY

Get the partition entry

Synopsis:

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

#define DCMD_BLK_PARTENTRY    __DIOF(_DCMD_BLK, 1, struct partition_entry)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_BLK_PARTENTRY
dev_data_ptr	A pointer to a `struct partition_entry` that the device can fill in (see below)
n_bytes	`sizeof(struct partition_entry)`
dev_info_ptr	NULL

Description:

This command gets the partition entry for the device associated with the given file descriptor. It's used by x86 disk partitions with harddisk-based filesystems.

Input:

None.

Output:

The `struct partition_entry` (or `partition_entry_t`) is defined in **<sys/disk.h>** as follows:

```
typedef struct partition_entry {
    unsigned char   boot_ind,
                    beg_head,
                    beg_sector,
                    beg_cylinder,
                    os_type,
                    end_head,
                    end_sector,
                    end_cylinder;
    uint32_t        part_offset,
                    part_size;
} partition_entry_t;
```

The members include:

boot_ind

`0x80` if the partition is bootable, `0x00` if it isn't.

beg_head

The beginning head number.

beg_sector

The beginning sector number.

beg_cylinder

The beginning cylinder number.

os_type

The partition type; see "Partitions" in the Filesystems chapter of the *System Architecture* guide.

end_head

The end head number.

end_sector

The end sector number.

end_cylinder

The end cylinder number.

part_offset

The offset of the partition, in bytes.

part_size

The number of sectors in the partition.

Example:

```
partition_entry_t *prt;

memset(&prt, 0, sizeof(prt));
if(devctl(fd, DCMD_BLK_PARTENTRY, prt, sizeof(*prt), 0) == EOK) {
    ...
}
```

See also:

DCMD_BLK_PART_DESCRIPTION

devctl() in the QNX Neutrino *C Library Reference*

`fdisk` in the QNX Neutrino *Utilities Reference*

Chapter 4
DCMD_BT_*

This chapter describes the *devctl()* commands that apply to Bluetooth and audio.

Not all audio drivers implement these commands.

DCMD_BT_ISOC_DISABLE

Disable Bluetooth connections for audio playback and capture

Synopsis:

```
#include <ado_pcm/dcmd_bluetooth.h>

#define DCMD_BT_ISOC_DISABLE   _IOW( 'Z', 0x02, struct _bt_isoch )
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_BT_ISOC_DISABLE
dev_data_ptr	A pointer to a `bt_isoch_t` structure
n_bytes	`sizeof(bt_isoch_t)`
dev_info_ptr	NULL

Description:

This command disables Bluetooth connections for audio playback and capture. For more information, see the entry for DCMD_BT_ISOC_ENABLE.

Input:

A filled-in `bt_isoch_t` structure:

```
typedef struct _bt_isoch {
        uint32_t                sco_hdl;
        uint32_t                rsvd[10];
} bt_isoch_t;
```

Set *sco_hdl* to the handle of the Bluetooth connection.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_BT_ISOC_ENABLE

Enable Bluetooth connections for audio playback and capture

Synopsis:

```
#include <ado_pcm/dcmd_bluetooth.h>

#define DCMD_BT_ISOC_ENABLE   _IOW( 'Z', 0x01, struct _bt_isoch )
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_BT_ISOC_ENABLE
dev_data_ptr	A pointer to a `bt_isoch_t` structure
n_bytes	`sizeof(bt_isoch_t)`
dev_info_ptr	NULL

Description:

This command enables Bluetooth connections for audio playback and capture. It tells the audio driver when it's allowed to accept users' connections to the device handle for audio playback/capture. The usage is:

1. The application codes establishes the Bluetooth connection.
2. Once the connection is established, issue the DCMD_BT_ISOC_ENABLE command to enable the audio driver interface for client use.
3. The client application can then successfully call *snd_pcm_plugin_params()* or *snd_pcm_channel_params()* against the audio device handle for the CSR BT USB-Audio device.
4. When the Bluetooth connection is dropped (or about to be disconnected), the application should issue the DCMD_BT_ISOC_DISABLE command to cause the driver to fail any further requests to use the audio driver until the connection is reestablished and the DCMD_BT_ISOC_ENABLE command is sent.

- If the audio interface is active when the DCDM_BT_ISOC_DISABLE command is issued, the driver makes the active stream fail with an error of SND_PCM_STATUS_ERROR (i.e., all further read/write calls to the audio interface will fail). The audio stream can't be recovered until the DCMD_BT_ISOC_ENABLE is sent.
- Any attempt to call *snd_pcm_plugin_param()* or *snd_pcm_channel_params()* the audio interface while not in the enabled state fails with an error of EAGAIN, and the *why_failed*

member of the `snd_pcm_channel_params_t` structure is set to SND_PCM_PARAMS_NO_CHANNEL.

Input:

A filled-in `bt_isoch_t` structure:

```
typedef struct _bt_isoch {
        uint32_t                sco_hdl;
        uint32_t                rsvd[10];
} bt_isoch_t;
```

Set *sco_hdl* to the handle of the Bluetooth connection.

Output:

None.

See also:

snd_pcm_channel_params(), `snd_pcm_channel_params_t`, *snd_pcm_plugin_params()* in the *Audio Developer's Guide*

devctl() in the QNX Neutrino *C Library Reference*

Chapter 5
DCMD_CHR_*

This chapter describes the *devctl()* commands that apply to character devices. Where possible, the *devctl()* commands are set up so as to match the corresponding *ioctl()* command:

- If there's a corresponding *ioctl()* command, the *devctl()* command is in this class:

  ```
  #define _CMD_IOCTL_TTY                  't'
  ```

  and the Synopsis includes the name of the *ioctl()* command as a comment.

- If there isn't a corresponding *ioctl()* command, the *devctl()* command is in this class:

  ```
  #define _DCMD_CHR               0x03
  ```

DCMD_CHR_DISABLE_LOGGING

Disable the logging of transmitted and/or received data

Synopsis:

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

#define DCMD_CHR_DISABLE_LOGGING    __DIOT(_DCMD_CHR, 37, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_DISABLE_LOGGING
dev_data_ptr	A pointer to a `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command disables the logging of transmitted and/or received data.

Input:

A bitwise OR of the flags that specify the type of data:

- _LOG_RX — received
- _LOG_TX — transmitted

Output:

None.

See also:

DCMD_CHR_ENABLE_LOGGING, *DCMD_CHR_FLUSH_LOG*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_ENABLE_LOGGING

Enable the logging of transmitted and/or received data

Synopsis:

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

#define DCMD_CHR_ENABLE_LOGGING     __DIOT(_DCMD_CHR, 36, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_ENABLE_LOGGING
dev_data_ptr	A pointer to a `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command enables the logging of transmitted and/or received data.

Input:

A bitwise OR of the flags that specify the type of data:

- _LOG_RX — received
- _LOG_TX — transmitted

Output:

None.

See also:

DCMD_CHR_DISABLE_LOGGING, *DCMD_CHR_FLUSH_LOG*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_FLUSH_LOG

Force a flush of the logged data to the log files

Synopsis:

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

#define DCMD_CHR_FLUSH_LOG      __DIOT(_DCMD_CHR, 38, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_FLUSH_LOG
dev_data_ptr	A pointer to a `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This forces a flush of the logged data to the log files.

Input:

A bitwise OR of the types of logged data to flush:

- _FLUSH_RX — received
- _FLUSH_TX — transmitted

Output:

None.

See also:

DCMD_CHR_DISABLE_LOGGING, *DCMD_CHR_ENABLE_LOGGING*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_FORCE_RTS

Force the RTS line to the specified level

Synopsis:

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

#define DCMD_CHR_FORCE_RTS      __DIOT(_DCMD_CHR, 34, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_FORCE_RTS
dev_data_ptr	A pointer to a `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command forces the RTS line to the specified level.

- Not all drivers pay attention to this command.
- Forcing RTS to low gives control of the RTS line back to **io-char** (or to the device for auto RTS). In other words, you could force it low, but it could immediately go back high.

Input:

The level of the RTS line: nonzero for high, or zero for low.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_GETOBAND

Get out-of-band data stored by a device

Synopsis:

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

#define DCMD_CHR_GETOBAND   __DIOF(_DCMD_CHR, 25, 0)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_GETOBAND
dev_data_ptr	A pointer to a `char`
n_bytes	`sizeof(char)`
dev_info_ptr	NULL

Description:

This command obtains out of band data stored by the device. This is usually associated with a serial driver.

Input:

None.

Output:

A bitwise OR of the following _OBAND_SER_* bits. This encoding depends on the individual device.

- _OBAND_SER_EXTENDED — the device has extended data that you can get with the DCMD_CHR_GETOBAND_EXTENDED command.
- _OBAND_SER_OE — overrun error.
- _OBAND_SER_PE — parity error.
- _OBAND_SER_FE — framing error.
- _OBAND_SER_BI — break.
- _OBAND_SER_SW_OE — software overrun error.
- _OBAND_SER_MS — change in modem status.
- _OBAND_SER_RESUME — the device has resumed.

See also:

DCMD_CHR_GETOBAND_EXTENDED, *DCMD_CHR_PUTOBAND*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_GETOBAND_EXTENDED

Get extended out-of-band data stored by a device

Synopsis:

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

#define DCMD_CHR_GETOBAND_EXTENDED   __DIOF(_DCMD_CHR, 39, unsigned int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_GETOBAND_EXTENDED
dev_data_ptr	A pointer to an `unsigned int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command obtains extended out-of-band data stored by the device. Not all drivers support this command. The _OBAND_SER_EXTENDED bit is set in the value returned from the DCMD_CHR_GETOBAND command if the device has extended data.

Input:

None.

Output:

The meaning of the extended data depends on the individual device.

See also:

DCMD_CHR_GETOBAND

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_GETSIZE

Get the size of a character device

Synopsis:

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

#define DCMD_CHR_GETSIZE   __DIOF(_CMD_IOCTL_TTY, 104, struct winsize)    /* TIOCGWINSZ */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_GETSIZE
dev_data_ptr	A pointer to a `struct winsize` (see below)
n_bytes	`sizeof(struct winsize)`
dev_info_ptr	NULL

Description:

This command gets the size of a character device. It's also implemented as the TIOCGSIZE *or* TIOCGWINSZ *ioctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcgetsize()* cover function.

Input:

None.

Output:

The `winsize` structure is defined in **<sys/ioctl.h>** as follows:

```
struct winsize {
        unsigned short  ws_row;
        unsigned short  ws_col;
        unsigned short  ws_xpixel;
        unsigned short  ws_ypixel;
};
```

See also:

DCMD_CHR_SETSIZE, TIOCGSIZE, TIOCGWINSZ

devctl(), *tcgetsize()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_GETVERBOSITY

*Get the verbosity of **io-char***

Synopsis:

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

#define DCMD_CHR_GETVERBOSITY   __DIOF(_DCMD_CHR, 29, unsigned)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_GETVERBOSITY
dev_data_ptr	A pointer to an unsigned integer
n_bytes	`sizeof(unsigned)`
dev_info_ptr	NULL

Description:

This command gets the current verbosity level of the **io-char** library.

Input:

None.

Output:

The current level of verbosity.

See also:

DCMD_CHR_SET_LOGGING_DIR, *DCMD_CHR_SETVERBOSITY*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_IDLE

Put the device into idle

Synopsis:

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

#define DCMD_CHR_IDLE   __DION(_DCMD_CHR, 32)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_IDLE
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command puts the device into idle.

Not all drivers pay attention to this command.

Input:

None.

Output:

None.

See also:

DCMD_CHR_RESET, *DCMD_CHR_RESUME*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_ISATTY

Test to see if a file descriptor is associated with a terminal

Synopsis:

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

#define DCMD_CHR_ISATTY   __DION(_DCMD_CHR, 24)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_ISATTY
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command tests to see if a file descriptor is associated with a terminal, returning EOK if it is, or ENOTTY if it isn't.

This command is for internal use, and you shouldn't use it directly. Instead use the *isatty()* cover function.

Input:

None.

Output:

None.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

ENOTTY

The file descriptor isn't associated with a terminal.

See also:

devctl(), *isatty()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_ISCHARS

Determine the number of characters waiting to be read

Synopsis:

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

#define DCMD_CHR_ISCHARS    __DIOF('f', 127, unsigned)   /* FIONREAD */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_ISCHARS
dev_data_ptr	A pointer to an `unsigned`
n_bytes	`sizeof(unsigned)`
dev_info_ptr	NULL

Description:

This command determine the number of characters waiting to be read. It's also implemented as the FIONREAD *ioctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcischars()* cover function.

Input:

None.

Output:

The number of characters.

See also:

DCMD_CHR_OSCHARS, FIONREAD

devctl(), *tcischars()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_ISSIZE

Determine the size of the device's input buffer

Synopsis:

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

#define DCMD_CHR_ISSIZE    __DIOF(_DCMD_CHR, 27, unsigned)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_ISSIZE
dev_data_ptr	A pointer to an `unsigned`
n_bytes	`sizeof(unsigned)`
dev_info_ptr	NULL

Description:

This command determines the size of the device's input buffer.

Input:

None.

Output:

The size of the device's input buffer.

See also:

DCMD_CHR_OSSIZE

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_LINESTATUS

Get line status information for the terminal device

Synopsis:

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

#define DCMD_CHR_LINESTATUS    __DIOF(_CMD_IOCTL_TTY, 106, int) /* TIOCMGET */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_LINESTATUS
dev_data_ptr	A pointer to an `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command gets line status information for the terminal device and is usually associated with the control lines of a serial port. It's also implemented as the TIOCMGET *ioctl()* command.

The contents of the returned data depend on the device. The `stty` utility calls this function and displays this information.

Input:

None.

Output:

The line status information, which is a combination of the following bits (depending on the type of device):

Device type	Bit	Description
Serial	_LINESTATUS_SER_DTR (TIOCM_DTR)	Data terminal ready
	_LINESTATUS_SER_RTS (TIOCM_RTS)	Request to send
	_LINESTATUS_SER_CTS (TIOCM_CTS)	Clear to send

Device type	Bit	Description
	_LINESTATUS_SER_DSR (TIOCM_DSR)	Data set ready
	_LINESTATUS_SER_RI (TIOCM_RI or TIOCM_RNG)	Ring
	_LINESTATUS_SER_CD (TIOCM_CAR or TIOCM_CD)	Carrier detect
Console	_LINESTATUS_CON_SCROLL	**Scroll Lock** is on
	_LINESTATUS_CON_NUM	**Num Lock** is on
	_LINESTATUS_CON_CAPS	**Caps Lock** is on
	_LINESTATUS_CON_SHIFT	**Shift** is pressed (not currently used)
	_LINESTATUS_CON_CTRL	**Ctrl** is pressed (not currently used)
	_LINESTATUS_CON_ALT	**Alt** is pressed (not currently used)
Parallel	_LINESTATUS_PAR_NOERROR	No error detected
	_LINESTATUS_PAR_SELECTED	Selected
	_LINESTATUS_PAR_PAPEROUT	No paper
	_LINESTATUS_PAR_NOTACK	Not acknowledge
	_LINESTATUS_PAR_NOTBUSY	Not busy

Example:

Check to see if RTS is set:

```
int data = 0, error;

if (error = devctl (fd, DCMD_CHR_LINESTATUS, &data, sizeof(data), NULL))
{
   fprintf(stderr, "Error getting RTS: %s\n", strerror ( error ));
   exit(EXIT_FAILURE);
}

if (data & _LINESTATUS_SER_RTS)
{
    printf("RTS is set.\n");
}
else
{
    printf("RTS isn't set.\n");
}
```

See also:

DCMD_CHR_SERCTL, TIOCMGET

devctl() in the QNX Neutrino *C Library Reference*

`stty` in the *Utilities Reference*

DCMD_CHR_OSCHARS

Determine the number of characters waiting to be sent

Synopsis:

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

#define DCMD_CHR_OSCHARS    __DIOF(_CMD_IOCTL_TTY, 115, unsigned) /* TIOCOUTQ */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_OSCHARS
dev_data_ptr	A pointer to an `unsigned`
n_bytes	`sizeof(unsigned)`
dev_info_ptr	NULL

Description:

This command determines the number of characters waiting to be sent. It's also implemented as the TIOCOUTQ *ioctl()* command.

Input:

None.

Output:

The number of characters.

See also:

DCMD_CHR_ISCHARS, TIOCOUTQ

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_OSSIZE

Determine the size of the device's output buffer

Synopsis:

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

#define DCMD_CHR_OSSIZE   __DIOF(_DCMD_CHR, 28, unsigned)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_OSSIZE
dev_data_ptr	A pointer to an `unsigned`
n_bytes	`sizeof(unsigned)`
dev_info_ptr	NULL

Description:

This command determines the size of the device's output buffer.

Input:

None.

Output:

The size of the device's output buffer.

See also:

DCMD_CHR_ISSIZE

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_PARCTL

Control a parallel device

Synopsis:

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

#define DCMD_CHR_PARCTL   __DIOT(_DCMD_CHR, 98, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_PARCTL
dev_data_ptr	A pointer to an `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command controls the control lines of a parallel device. It isn't currently used.

Input:

The action to take.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_PNPTEXT

Get the text associated with a plug and play device

Synopsis:

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

#define DCMD_CHR_PNPTEXT   __DIOF(_DCMD_CHR, 99, char)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_PNPTEXT
dev_data_ptr	A pointer to a `char` buffer
n_bytes	The size of the buffer
dev_info_ptr	NULL

Description:

This command gets the text associated with a plug and play device. This command is usually associated with printer devices.

Input:

None.

Output:

The plug and play device's text.

Example:

```
char buf[500];

if(devctl(fd, DCMD_CHR_PNPTEXT, buf, sizeof(buf), NULL) == EOK)
{
   /* Parse the plug and play text. */
}
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_PUTOBAND

Send out of band data to the device

Synopsis:

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

#define DCMD_CHR_PUTOBAND  __DIOT(_DCMD_CHR, 26, 0)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_PUTOBAND
dev_data_ptr	A pointer to a `char`
n_bytes	`sizeof(char)`
dev_info_ptr	NULL

Description:

This command sends out of band data to the device.

Input:

A pointer to an unsigned integer containing the out-of-band data; see *DCMD_CHR_GETOBAND*.

Output:

None.

See also:

DCMD_CHR_GETOBAND

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_RESET

Reset the device

Synopsis:

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

#define DCMD_CHR_RESET   __DION(_DCMD_CHR, 31)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_RESET
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command resets the device.

Not all drivers pay attention to this command.

Input:

None.

Output:

None.

See also:

DCMD_CHR_IDLE, *DCMD_CHR_RESUME*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_RESUME

Make the device resume from idle

Synopsis:

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

#define DCMD_CHR_RESUME   __DION(_DCMD_CHR, 33)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_RESUME
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command makes the device resume from idle.

Not all drivers pay attention to this command.

Input:

None.

Output:

None.

See also:

DCMD_CHR_IDLE, *DCMD_CHR_RESET*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_SERCTL

Control serial communication lines

Synopsis:

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

#define DCMD_CHR_SERCTL   __DIOT(_DCMD_CHR, 20, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_SERCTL
dev_data_ptr	A pointer to an `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command controls serial communication lines. The *tcsendbreak()*, *tcdropline()*, and *ioctl()* functions act as cover functions for many of the actions. If these cover functions don't provide suitable functionality, use this command directly.

Input:

The desired serial line control action. The `*_CHG` bits indicate which attributes you want to change, and the corresponding non-`*_CHG` bits are the "on" setting. For example, to turn CTS on, specify a value of `_CTL_CTS_CHG | _CTL_CTS`; to turn it off, specify a value of `_CTL_CTS_CHG`.

Device type	"On" value	"Change" bit	Description
General	_CTL_DTR	_CTL_DTR_CHG	Data terminal ready; you can OR in a duration
	_CTL_RTS	_CTL_RTS_CHG	Ready to send
	_CTL_BRK	_CTL_BRK_CHG	Break; you can OR in a duration
	_CTL_TIMED	_CTL_TIMED_CHG	Data ready timeout; you can OR in a duration

Device type	"On" value	"Change" bit	Description
	_CTL_DSR	_CTL_DSR_CHG	Data set ready; for use when DSR is an output (USB device side serial class driver). You can OR in a duration.
	_CTL_DCD	_CTL_DCD_CHG	Data carrier detect; for use when DCD is an output (USB device side serial class driver). You can OR in a duration.
	_CTL_CTS	_CTL_CTS_CHG	Clear to send; for use when CTS is an output (USB device side serial class driver)
	_CTL_MASK	_CTL_MASK_CHG	Mask the reporting of errors
Serial	_SERCTL_DTR	_SERCTL_DTR_CHG	Data terminal ready
	_SERCTL_RTS	_SERCTL_RTS_CHG	Ready to send
	_SERCTL_BRK	_SERCTL_BRK_CHG	Break
	_SERCTL_LOOP	_SERCTL_LOOP_CHG	Loopback
	_SERCTL_DSR	_SERCTL_DSR_CHG	Data send ready
	_SERCTL_DCD	_SERCTL_DCD_CHG	Data carrier detect
	_SERCTL_CTS	_SERCTL_CTS_CHG	Clear to send
Console	_CONCTL_BELL	_CONCTL_BELL_CHG	Ring the bell
	_CONCTL_SCROLL	_CONCTL_SCROLL_CHG	Scroll lock
	_CONCTL_NUM	_CONCTL_NUM_CHG	Num lock
	_CONCTL_CAPS	_CONCTL_CAPS_CHG	Caps lock
	_CONCTL_INVISIBLE	_CONCTL_INVISIBLE_CHG	Don't talk to video hardware

You can use the following macros to shift a duration so that you can OR it into a command:

```
#define _CTL_DURATION(__duration)    ((__duration) << 16)
#define _SERCTL_DURATION(__duration) _CTL_DURATION(__duration)
#define _CONCTL_DURATION(__duration) _CTL_DURATION(__duration)
```

Output:

None.

Example:

Turn RTS on:

```
int i_Data, i_fd;
int RTS_TOGGLE;

fd = open ("/dev/ser1", O_RDONLY);

RTS_TOGGLE = 1; // Turn RTS on.

i_Data = _CTL_RTS_CHG | (RTS_TOGGLE ? _CTL_RTS : 0);

if (devctl (i_fd, DCMD_CHR_SERCTL, &i_Data, sizeof(i_Data), NULL) != EOK)
{
  /* Error */
}
```

Here's how to include a duration of 300 milliseconds in a command:

```
int duration = 300;
int cmd;

cmd = _SERCTL_DTR_CHG | _SERCTL_DURATION (duration);
if (devctl(fd, DCMD_CHR_SERCTL, &cmd, sizeof cmd, NULL) != EOK)
{
  /* Error */
}
```

See also:

DCMD_CHR_LINESTATUS

devctl(), *ioctl()*, *tcdropline()*, *tcsendbreak()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_SET_LOGGING_DIR

Set the path to the logging directory

Synopsis:

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

#define DCMD_CHR_SET_LOGGING_DIR   __DIOT(_DCMD_CHR, 35, char*)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_SET_LOGGING_DIR
dev_data_ptr	A pointer to a `char`
n_bytes	`sizeof(char)`
dev_info_ptr	NULL

Description:

This command sets the path to the logging directory for **io-char**.

Input:

A string that contains the path to the logging directory.

Output:

None.

See also:

DCMD_CHR_GETVERBOSITY, *DCMD_CHR_SETVERBOSITY*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_SETSIZE

Set the size of a character device

Synopsis:

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

#define DCMD_CHR_SETSIZE    __DIOT(_CMD_IOCTL_TTY, 103, struct winsize) /* TIOCSWINSZ */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_SETSIZE
dev_data_ptr	A pointer to a `struct winsize` (see below)
n_bytes	`sizeof(struct winsize)`
dev_info_ptr	NULL

Description:

This command sets the size of a character device. It's also implemented as the TIOCSSIZE *or* TIOCSWINSZ *ioctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcsetsize()* cover function.

Input:

The `winsize` structure is defined in **<sys/ioctl.h>** as follows:

```
struct winsize {
        unsigned short  ws_row;
        unsigned short  ws_col;
        unsigned short  ws_xpixel;
        unsigned short  ws_ypixel;
};
```

Output:

None.

See also:

DCMD_CHR_GETSIZE, TIOCSSIZE *or* TIOCSWINSZ

devctl(), *tcsetsize()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_SETVERBOSITY

*Set the verbosity of **io-char***

Synopsis:

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

#define DCMD_CHR_SETVERBOSITY   __DIOT(_DCMD_CHR, 30, unsigned)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_SETVERBOSITY
dev_data_ptr	A pointer to an `unsigned`
n_bytes	`sizeof(unsigned)`
dev_info_ptr	NULL

Description:

This command sets the current verbosity level of the **io-char** library.

Input:

The new verbosity level.

Output:

None.

See also:

DCMD_CHR_GETVERBOSITY, *DCMD_CHR_SET_LOGGING_DIR*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_TCDRAIN

Wait until all output has been transmitted to a device

Synopsis:

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

#define DCMD_CHR_TCDRAIN    __DION(_CMD_IOCTL_TTY, 94)    /* TIOCDRAIN */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_TCDRAIN
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command waits until all output has been transmitted to a device. It's also implemented as the TIOCDRAIN *ioctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcdrain()* cover function.

Input:

None.

Output:

None.

See also:

devctl(), *tcdrain()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_TCFLOW

Perform a flow-control operation on a data stream

Synopsis:

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

#define DCMD_CHR_TCFLOW    __DIOT('T', 6, int)    /* TCXONC */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_TCFLOW
dev_data_ptr	A pointer to an `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command performs a flow-control operation on the data stream associated with the file descriptor passed to *devctl()*. It's also implemented as the TCXONC *ioctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcflow()* cover function.

Input:

The action you want to perform (defined in **<termios.h>**):

TCOOFF

Use software flow control to suspend output on the device associated with the file descriptor.

TCOOFFHW

Use hardware flow control to suspend output on the device associated with the file descriptor.

TCOON

Use software flow control to resume output on the device associated with the file descriptor.

TCOONHW

Use hardware flow control to resume output on the device associated with the file descriptor.

TCIOFF

Cause input to be flow-controlled by sending a STOP character immediately across the communication line associated with the file descriptor, (that is, software flow control).

TCIOFFHW

Cause input to be flow-controlled by using hardware control.

TCION

Resume input by sending a START character immediately across the communication line associated with the file descriptor (that is, software flow control).

TCIONHW

Cause input to be resumed by using hardware flow control.

Output:

None.

See also:

TCXONC

devctl(), *tcflow()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_TCFLUSH

Flush the input or output stream

Synopsis:

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

#define DCMD_CHR_TCFLUSH   __DIOT(_CMD_IOCTL_TTY, 16, int) /* TIOCFLUSH */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_TCFLUSH
dev_data_ptr	A pointer to an `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command flushes the input and/or output stream associated with the file descriptor. It's also implemented as the TIOCFLUSH *ioctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcflush()* cover function.

Input:

The queue selector; one of:

- TCIFLUSH — discard all data that's received, but not yet read, on the device associated with *fildes*.
- TCOFLUSH — discard all data that's written, but not yet transmitted, on the device associated with *fildes*.
- TCIOFLUSH — discard all data that's written, but not yet transmitted, as well as all data that's received, but not yet read, on the device associated with *fildes*.

Output:

None.

See also:

TIOCFLUSH

devctl(), *tcflush()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_TCGETATTR

Get the terminal properties

Synopsis:

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

#define DCMD_CHR_TCGETATTR   __DIOF(_CMD_IOCTL_TTY, 19, struct termios)  /* TIOCGETA */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_TCGETATTR
dev_data_ptr	A pointer to a `struct termios`
n_bytes	`sizeof(struct termios)`
dev_info_ptr	NULL

Description:

This command gets the current terminal control settings of a device. It's also implemented as the TIOCGETA *ioctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcgetattr()* cover function.

Input:

None.

Output:

A filled-in `termios` structure.

See also:

DCMD_CHR_TCSETATTR, DCMD_CHR_TCSETATTRD, DCMD_CHR_TCSETATTRF, TIOCGETA

devctl(), *tcgetattr()*, `termios` in the QNX Neutrino *C Library Reference*

DCMD_CHR_TCGETPGRP

Get the process group ID associated with a device

Synopsis:

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

#define DCMD_CHR_TCGETPGRP   __DIOF(_CMD_IOCTL_TTY, 119, pid_t) /* TIOCGPGRP */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_TCGETPGRP
dev_data_ptr	A pointer to a `pid_t`
n_bytes	`sizeof(pid_t)`
dev_info_ptr	NULL

Description:

This command gets the process group ID associated with a device. It's also implemented as the TIOCGPGRP *ioctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcgetpgrp()* cover function.

Input:

None.

Output:

The process group ID.

See also:

DCMD_CHR_TCSETPGRP, TIOCGPGRP

devctl(), *tcgetpgrp()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_TCGETSID

Get the process group ID of a controlling terminal's session leader

Synopsis:

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

#define DCMD_CHR_TCGETSID   __DIOF(_DCMD_CHR, 7, pid_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_TCGETSID
dev_data_ptr	A pointer to a `pid_t`
n_bytes	`sizeof(pid_t)`
dev_info_ptr	NULL

Description:

This command gets the process group ID of the session leader for a controlling terminal.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcgetsid()* cover function.

Input:

None.

Output:

The process group ID.

See also:

DCMD_CHR_TCSETSID

devctl(), *tcgetsid()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_TCINJECTC, DCMD_CHR_TCINJECTR

Inject characters into a device's input buffer

Synopsis:

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

#define DCMD_CHR_TCINJECTC   __DIOT(_DCMD_CHR, 22, 0) /*CANONICAL Input Buffer*/
#define DCMD_CHR_TCINJECTR   __DIOT(_DCMD_CHR, 23, 0) /*RAW Input Buffer*/
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_TCINJECTC or DCMD_CHR_TCINJECTR
dev_data_ptr	A pointer to a `char`
n_bytes	The number of characters to inject
dev_info_ptr	NULL

Description:

These commands inject characters into a device's canonical or raw input buffer.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcinject()* cover function.

Input:

A pointer to a character buffer.

Output:

None.

See also:

devctl(), *tcinject()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_TCSETATTR, DCMD_CHR_TCSETATTRD, DCMD_CHR_TCSETATTRF

Set terminal properties

Synopsis:

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

#define DCMD_CHR_TCSETATTR    __DIOT(_CMD_IOCTL_TTY, 20, struct termios)  /* TIOCSETA */
#define DCMD_CHR_TCSETATTRD   __DIOT(_CMD_IOCTL_TTY, 21, struct termios)  /* TIOCSETAW */
#define DCMD_CHR_TCSETATTRF   __DIOT(_CMD_IOCTL_TTY, 22, struct termios)  /* TIOCSETAF */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_TCSETATTR, DCMD_CHR_TCSETATTRD, or DCMD_CHR_TCSETATTRF
dev_data_ptr	A pointer to a `struct termios`
n_bytes	`sizeof(struct termios)`
dev_info_ptr	NULL

Description:

These commands change the current terminal control settings of a device:

- DCMD_CHR_TCSETATTR — make the change immediately
- DCMD_CHR_TCSETATTRD — don't make the change until all currently written data has been transmitted.
- DCMD_CHR_TCSETATTRF — don't make the change until all currently written data has been transmitted, at which point any received but unread data is also discarded.

They're also implemented as *devctl()* commands:

***devctl()* command**	***ioctl()* command**
DCMD_CHR_TCSETATTR	TIOCSETA
DCMD_CHR_TCSETATTRD	TIOCSETAW
DCMD_CHR_TCSETATTRF	TIOCSETAF

These commands are for internal use, and you shouldn't use them directly. Instead use the *tcsetattr()* cover function.

Input:

A `termios` structure.

Output:

None.

See also:

DCMD_CHR_TCGETATTR, TIOCSETA, TIOCSETAF, TIOCSETAW

devctl(), *tcsetattr()*, `termios` in the QNX Neutrino *C Library Reference*

DCMD_CHR_TCSETPGRP

Set the process group ID associated with a device

Synopsis:

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

#define DCMD_CHR_TCSETPGRP   __DIOT(_CMD_IOCTL_TTY, 118, pid_t) /* TIOCSPGRP */
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_TCSETPGRP
dev_data_ptr	A pointer to a `pid_t`
n_bytes	`sizeof(pid_t)`
dev_info_ptr	NULL

Description:

This command sets the process group ID associated with a device. It's also implemented as the TIOCSPGRP *ioctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcsetpgrp()* cover function.

Input:

The process group ID.

Output:

None.

See also:

DCMD_CHR_TCGETPGRP, TIOCSPGRP

devctl(), *tcsetpgrp()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_TCSETSID

Set the process group ID of a controlling terminal's session leader

Synopsis:

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

#define DCMD_CHR_TCSETSID    __DIOT(_DCMD_CHR, 8, pid_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_TCSETSID
dev_data_ptr	A pointer to a `pid_t`
n_bytes	`sizeof(pid_t)`
dev_info_ptr	NULL

Description:

This command sets the process group ID of the session leader for a controlling terminal.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcsetsid()* cover function.

Input:

The process group ID.

Output:

None.

See also:

DCMD_CHR_TCGETSID

devctl(), *tcsetsid()* in the QNX Neutrino *C Library Reference*

DCMD_CHR_TTYINFO

Get information about a terminal device

Synopsis:

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

#define DCMD_CHR_TTYINFO   __DIOF(_DCMD_CHR, 10, struct _ttyinfo)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_TTYINFO
dev_data_ptr	A pointer to a `struct _ttyinfo` (see below)
n_bytes	`sizeof(struct _ttyinfo)`
dev_info_ptr	NULL

Description:

This command obtains information about a terminal device. It returns the name of the device associated with its file descriptor, and the number of clients that have an open file descriptor to this terminal device.

Input:

None.

Output:

A filled-in `_ttyinfo` structure:

```
struct _ttyinfo {
   int    opencount;
   char   ttyname[32];
};
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_CHR_WAITINFO

Get information about blocked processes

Synopsis:

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

#define DCMD_CHR_WAITINFO   __DIOTF(_DCMD_CHR, 11, struct _ttywaitinfo)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_CHR_WAITINFO
dev_data_ptr	A pointer to a `struct _ttywaitinfo`
n_bytes	`sizeof(struct _ttywaitinfo)` (see below)
dev_info_ptr	NULL

Description:

This command gets information about blocked processes. The client passes in the queue to query and the number of blocked processes that it can handle seeing.

The associated data types are:

```
typedef enum _tty_queue {
        TTY_NULL_Q,
        TTY_DEVCTL_Q,
        TTY_DRAIN_Q,
        TTY_WRITE_Q,
        TTY_READ_Q,
        TTY_OPEN_Q     /* By definition we won't see see anything here,
                          since we have to open the device to query it */
} _ttyqueue;

struct _pidtid {
        pid_t pid;
        int   tid;
        int   offset;
        int   nbytes;
};

struct _ttywaitinfo {
        _ttyqueue      queue;
        unsigned int   num;
```

```
        struct _pidtid blocked[0];
};
```

Input:

A `_ttywaitinfo` structure, with:

- *queue* set to the desired `_tty_queue` value
- *num* set to the number of entries in the *blocked* array
- enough space to hold the *blocked* array

Output:

A `_ttywaitinfo` structure, with the *blocked* array filled in, and *num* set to the number of entries that were filled in.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

ENOTTY

You specified an invalid queue.

See also:

devctl() in the QNX Neutrino *C Library Reference*

Chapter 6
DCMD_DUMPER_*

This chapter describes the *devctl()* commands that apply to postmortem dumps and the `dumper` utility.

DCMD_DUMPER_GETPATH

Get the path of the dump file associated with a process ID

Synopsis:

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

#define DCMD_DUMPER_GETPATH  __DIOTF(_DCMD_MISC, DUMPER_GETPATH, dump_info_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor for the dumper that you obtained by opening **/proc/dumper**.
dcmd	DCMD_DUMPER_GETPATH
dev_data_ptr	A pointer to a `dump_info_t` structure
n_bytes	`sizeof(dump_info_t)`
dev_info_ptr	NULL

Description:

This command gets the path of the dump file associated with a given process ID and lets you determine when the dump is finished. To use this command, first use the DCMD_DUMPER_NOTIFYEVENT *devctl()* command to register for a pulse when the dump begins. When you receive the pulse, use DCMD_DUMPER_GETPATH, passing it the process ID included in the pulse. The command blocks until the dump is finished (because `dumper` is single threaded) and returns the matching pathname if one exists.

DCMD_DUMPER_GETPATH uses a `dump_info_t` structure for both input and output:

```
typedef union {
        struct {
                int pid;
        } i;
        struct {
                char dump_pathname[PATH_MAX + 1];
        } o;
} dump_info_t;
```

Input:

The *i* (input) member includes:

pid

The process ID.

Output:

The *o* (output) member includes:

dump_pathname

The path of the associated dump file.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

ESRCH

A dump file couldn't be found for the process ID. This could mean that `dumper`'s list of dump file names isn't big enough; you can use the `-E` option to specify the number of entries in it.

See also:

DCMD_DUMPER_NOTIFYEVENT, *DCMD_DUMPER_REMOVEALL DCMD_DUMPER_REMOVEEVENT*

devctl() in the QNX Neutrino *C Library Reference*

`dumper` in the *Utilities Reference*

DCMD_DUMPER_NOTIFYEVENT

Register for dump notifications

Synopsis:

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

#define DCMD_DUMPER_NOTIFYEVENT __DIOT(_DCMD_MISC, DUMPER_NOTIFYEVENT, struct sigevent)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor for the dumper that you obtained by opening **/proc/dumper**
dcmd	DCMD_DUMPER_NOTIFYEVENT
dev_data_ptr	A pointer to a `struct sigevent`
n_bytes	`sizeof(struct sigevent)`
dev_info_ptr	NULL

Description:

This command registers a program for dump notifications.

Input:

A `struct sigevent` that's filled in to indicate what type of notification you want.

Output:

None.

Example:

```
#include <stdio.h>
#include <stdint.h>
#include <inttypes.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/dcmd_dumper.h>
#include <fcntl.h>
#include <unistd.h>
#include <devctl.h>
#include <sys/neutrino.h>
```

```
int dumper_notify_attach(struct sigevent *devent)
{
    int dumper_fd;
    dumper_fd = open("/proc/dumper",O_RDONLY);
    if (dumper_fd >= 0) {
        devctl(dumper_fd, DCMD_DUMPER_NOTIFYEVENT, devent, sizeof(*devent), NULL);
        fcntl(dumper_fd, F_SETFD, FD_CLOEXEC);
    } else {
        dumper_fd = -1;
    }
    return dumper_fd;
}

#define DUMP_PULSE_CODE 0x50

int main(int argc, const char *argv[], const char*envp[])
{
    int dp_chid=-1;
    int dp_coid=-1;
    struct sigevent devent;
    struct _pulse gpulse;
    int dumper_fd=-1;
    int rcvid;
    pid_t pid;

    // create death pulses channel
    dp_chid = ChannelCreate(_NTO_CHF_FIXED_PRIORITY);
    if(dp_chid==-1){
        perror("ERROR: ChannelCreate");
        exit( -1 );
    }
    dp_coid = ConnectAttach(0, 0, dp_chid, _NTO_SIDE_CHANNEL, _NTO_COF_CLOEXEC);
    if(dp_coid==-1){
        perror("ERROR: ConnectAttach");
        exit( -1 );
    }
    SIGEV_PULSE_INIT(&devent, dp_coid, sched_get_priority_max(SCHED_RR),
                     DUMP_PULSE_CODE, -1);
    dumper_fd=dumper_notify_attach(&devent);
    if(dumper_fd==-1){
        perror("ERROR: opening /proc/dumper");
        exit( -1 );
    }
    for (;;) {
        // Blocks waiting for a pulse
        rcvid = MsgReceivePulse(dp_chid, &gpulse, sizeof(gpulse),NULL);
        switch (gpulse.code) {
            case DUMP_PULSE_CODE: // something died
                pid = gpulse.value.sival_int;
                fprintf(stderr, "Received Death Pulse code %"PRId8"\n", gpulse.code);
                fprintf(stderr, "Process Pid %d died abnormally\n" , pid);
                break;
            default:
                fprintf(stderr, "Unknown pulse code: %"PRId8"\n" , gpulse.code);
```

```
                break;
            }
        }
        if (dumper_fd >=0)
        {
            devctl(dumper_fd, DCMD_DUMPER_REMOVEALL, NULL, 0, NULL);

            /* This would have worked too, because we attached only one event:
            devctl(dumper_fd, DCMD_DUMPER_REMOVEEVENT, NULL, 0, NULL);
            */
            close(dumper_fd);
        }
        if (dp_coid >=0)
            ConnectDetach(dp_coid);
        if (dp_chid >=0)
            ChannelDestroy(dp_chid);
        exit(0);
    }
```

See also:

DCMD_DUMPER_GETPATH, *DCMD_DUMPER_REMOVEALL*, *DCMD_DUMPER_REMOVEEVENT*

devctl() in the QNX Neutrino *C Library Reference*

`dumper` in the *Utilities Reference*

DCMD_DUMPER_REMOVEALL

Remove all notification events for a process

Synopsis:

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

#define DCMD_DUMPER_REMOVEALL   __DIOT(_DCMD_MISC, DUMPER_REMOVEALL, 0)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor for the dumper that you obtained by opening **/proc/dumper**.
dcmd	DCMD_DUMPER_REMOVEALL
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command removes all dumper notifications for the calling process. The *devctl()* function returns EOK whether or not there were any events for the calling process.

Input:

None.

Output:

None.

Example:

See DCMD_DUMPER_NOTIFYEVENT.

See also:

DCMD_DUMPER_GETPATH, *DCMD_DUMPER_NOTIFYEVENT*, *DCMD_DUMPER_REMOVEEVENT*

devctl() in the QNX Neutrino *C Library Reference*

`dumper` in the *Utilities Reference*

DCMD_DUMPER_REMOVEEVENT

Remove a notification event

Synopsis:

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

#define DCMD_DUMPER_REMOVEEVENT  __DIOT(_DCMD_MISC, DUMPER_REMOVEEVENT, 0)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor for the dumper that you obtained by opening **/proc/dumper**.
dcmd	DCMD_DUMPER_REMOVEEVENT
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command removes the first notification event found for the calling process. The *devctl()* function returns EOK if an event was found, or EINVAL if there were no events for the calling process.

Input:

None.

Output:

None.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EINVAL

There were no events for the calling process.

Example:

See DCMD_DUMPER_NOTIFYEVENT.

See also:

DCMD_DUMPER_GETPATH, *DCMD_DUMPER_NOTIFYEVENT*, *DCMD_DUMPER_REMOVEALL*

devctl() in the QNX Neutrino *C Library Reference*

`dumper` in the *Utilities Reference*

Chapter 7
DCMD_ETFS_*

This chapter describes the *devctl()* commands that apply to embedded transaction filesystems. For more information about ETFS, see "Embedded transaction filesystem (ETFS)" in the Filesystems chapter of the *System Architecture* guide, and the entries for `fs-etfs-ram` and `etfsctl` in the *Utilities Reference*.

Some of these commands require a file descriptor that corresponds to the filesystem partition, **/dev/etfs2**, not to the raw partition, **/dev/etfs1**.

DCMD_ETFS_DEFRAG

Defragment an embedded transaction filesystem

Synopsis:

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

#define DCMD_ETFS_DEFRAG    __DION(_DCMD_MEM, 104)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the filesystem partition, **/dev/etfs2**
dcmd	DCMD_ETFS_DEFRAG
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command defragments the filesystem. You must be **root** or have the appropriate permissions to use this command.

Input:

None.

Output:

None.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EINVAL

The file descriptor doesn't correspond to the filesystem partition, **/dev/etfs2**.

EPERM

You don't have the required permissions.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`etfsctl`, `fs-etfs-ram` in the *Utilities Reference*

DCMD_ETFS_ERASE

Erase an embedded transaction filesystem

Synopsis:

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

#define DCMD_ETFS_ERASE     __DION(_DCMD_MEM, 102)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the filesystem partition, **/dev/etfs2**
dcmd	DCMD_ETFS_ERASE
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command erases the entire device. You must be **root** or have the appropriate permissions to use this command.

Input:

None.

Output:

None.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EBUSY

There are open files in the filesystem.

EINVAL

The file descriptor doesn't correspond to the filesystem partition, **/dev/etfs2**.

EPERM

You don't have the required permissions.

See also:

DCMD_ETFS_ERASE_RANGE

devctl() in the QNX Neutrino *C Library Reference*

`etfsctl`, `fs-etfs-ram` in the *Utilities Reference*

DCMD_ETFS_ERASE_RANGE

Erase a specified range in an embedded transaction filesystem

Synopsis:

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

#define DCMD_ETFS_ERASE_RANGE     __DIOT(_DCMD_MEM, 106, struct etfs_erase_range)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the filesystem partition, **/dev/etfs2**
dcmd	DCMD_ETFS_ERASE_RANGE
dev_data_ptr	A pointer to a `struct etfs_erase_range`
n_bytes	`sizeof(struct etfs_erase_range)`
dev_info_ptr	NULL

Description:

This command erases a specified range of the filesystem. You must be **root** or have the appropriate permissions to use this command.

Input:

A `etfs_erase_range` structure, which is defined as:

```
struct etfs_erase_range {
        int64_t         offset;
        int64_t         length;
};
```

The members include:

offset

The offset to start erasing at, in bytes.

length

The number of bytes to erase, or -1 to erase to the end of the partition.

Output:

None.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EBUSY

There are open files in the filesystem.

EINVAL

The file descriptor doesn't correspond to the filesystem partition, **/dev/etfs2**.

EPERM

You don't have the required permissions.

See also:

DCMD_ETFS_ERASE

devctl() in the QNX Neutrino *C Library Reference*

`etfsctl`, `fs-etfs-ram` in the *Utilities Reference*

DCMD_ETFS_FLUSH_COUNT

*Flush the **.badblks** and **.counts** files for an embedded transaction filesystem*

Synopsis:

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

#define DCMD_ETFS_FLUSH_COUNT     __DION(_DCMD_MEM, 107)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_ETFS_FLUSH_COUNT
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command causes these files to be written on the device:

.badblks

A list of the bad blocks.

.counts

An array of the blocks showing the read and erase counts, used in wear leveling.

Input:

None.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`etfsctl`, `fs-etfs-ram` in the *Utilities Reference*

DCMD_ETFS_FORMAT

Format an embedded transaction filesystem

Synopsis:

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

#define DCMD_ETFS_FORMAT    __DION(_DCMD_MEM, 103)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the filesystem partition, **/dev/etfs2**
dcmd	DCMD_ETFS_FORMAT
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command erases the device and then formats an empty filesystem. You must be **root** or have the appropriate permissions to use this command.

Input:

None.

Output:

None.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EBUSY

There are open files in the filesystem.

EINVAL

The file descriptor doesn't correspond to the filesystem partition, **/dev/etfs2**.

EPERM

You don't have the required permissions.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`etfsctl`, `fs-etfs-ram` in the *Utilities Reference*

DCMD_ETFS_INFO

Get information about an embedded transaction filesystem

Synopsis:

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

#define DCMD_ETFS_INFO      __DIOF(_DCMD_MEM, 105, struct etfs_info)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_ETFS_INFO
dev_data_ptr	A pointer to a `struct etfs_info`
n_bytes	`sizeof(struct etfs_info)`
dev_info_ptr	NULL

Description:

This command gets information about the filesystem.

Input:

None.

Output:

A filled-in `etfs_info` structure, which is defined as:

```
struct etfs_info {
        char    name[16];
        int             numblks;
        int             clusters2blk;
        int             clustersize;
        int             erase_cnt;
        int             clean_cnt;
        int             spare_cnt;
        int             filthy_cnt;
        int             inactive_cnt;
        int             xtnts_cnt;
        int             cache_cnt;
        int             devread_cnt;
        int             devwrite_cnt;
        int             cacheread_cnt;
```

```
        int             mine_cnt;
        int             copy_cnt;
        int             defrag_cnt;
        int             eccerr_cnt;
        int             chkerr_cnt;
        int             deverr_cnt;
        int             files_cnt;
        int             open_cnt;
        int             badblks_cnt;
        int             blksize;
        int             expansion[15];          /* For future expansion */
} ;
```

The members include:

name

The name of the device, which usually encodes a part number or size.

numblks

The number of blocks on the device.

clusters2blk

The number of clusters to a block on the device.

clustersize

The size of a cluster. Typically 1 KB or 2 KB.

erase_cnt

The number of erases on the part (while running).

clean_cnt

The number of erased blocks immediately ready for writing.

spare_cnt

The number of spare blocks.

filthy_cnt

The number of free blocks that are waiting to be erased and made clean.

inactive_cnt

The number of clusters not being used but trapped.

xtnts_cnt

The number of cache buffers.

cache_cnt

The number of cluster cache buffers.

devread_cnt

The number of cluster reads from the device.

devwrite_cnt

The number of cluster writes to the device.

cacheread_cnt

The number of cluster reads from cache.

mine_cnt

The number of mining operations to recover dead space in a block. This is how inactive clusters create filthy blocks, which become clean after being erased.

copy_cnt

The number of block-copy operations. Copies occur two ways: the first way is a read in a block that has a soft ECC error, which is an indication that the block is getting weak. The block is copied to a new fresh block and the block with the ECC error is erased. In the second way, a block with a low erase count is forced into service by copying its data to a new block and erasing and putting this block into service.

defrag_cnt

The number of files defragmented.

eccerr_cnt

The number of CRC data errors that are corrected by ECC.

chkerr_cnt

The number of CRC data errors.

deverr_cnt

The number of hard device errors. This is bad and usually indicates a hardware problem.

files_cnt

The number of files.

open_cnt

The number of open files.

badblks_cnt

The number of blocks marked as bad and taken out of service.

blksize

The block size, in bytes.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`etfsctl`, `fs-etfs-ram` in the *Utilities Reference*

DCMD_ETFS_START

Start an embedded transaction filesystem

Synopsis:

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

#define DCMD_ETFS_START     __DION(_DCMD_MEM, 101)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the filesystem partition, **/dev/etfs2**
dcmd	DCMD_ETFS_START
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command makes the filesystem on the device continue or resume operations. You must be **root** or have the appropriate permissions to use this command.

Input:

None.

Output:

None.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EBADFSYS

The filesystem couldn't be started.

EBUSY

There are open files in the filesystem.

EINVAL

The file descriptor doesn't correspond to the filesystem partition, **/dev/etfs2**.

ENODEV

The filesystem hasn't been mounted.

EPERM

You don't have the required permissions.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`etfsctl`, `fs-etfs-ram` in the *Utilities Reference*

DCMD_ETFS_STOP

Stop an embedded transaction filesystem

Synopsis:

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

#define DCMD_ETFS_STOP      __DION(_DCMD_MEM, 100)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the filesystem partition, **/dev/etfs2**
dcmd	DCMD_ETFS_STOP
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command stops the filesystem on the device. You must be **root** or have the appropriate permissions to use this command.

Input:

None.

Output:

None.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EBUSY

There are open files in the filesystem.

EINVAL

The file descriptor doesn't correspond to the filesystem partition, **/dev/etfs2**.

ENODEV

The filesystem hasn't been mounted.

EPERM

You don't have the required permissions.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`etfsctl`, `fs-etfs-ram` in the *Utilities Reference*

Chapter 8
DCMD_F3S_*

This chapter describes the *devctl()* commands that apply to the F3S resource manager.

- We've deprecated the DCMD_F3S_LOCKDOWN and DCMD_F3S_ULOCKDOWN commands. Use *DCMD_F3S_LOCKSSR*, *DCMD_F3S_READSSR*, *DCMD_F3S_STATSSR*, and *DCMD_F3S_WRITESSR* instead.
- The DCMD_F3S_BREAK and DCMD_F3S_DEFRAG commands aren't currently implemented.

DCMD_F3S_ARRAYINFO

Get the total amount of contiguous flash

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_ARRAYINFO __DIOTF(_DCMD_F3S, F3S_ARRAYINFO, f3s_arrayinfo_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_ARRAYINFO
dev_data_ptr	A pointer to a `f3s_arrayinfo_t` structure (see below)
n_bytes	`sizeof(f3s_arrayinfo_t)`
dev_info_ptr	NULL

Description:

This command gets the total amount of contiguous flash on the entire chip or contiguous chips.

Input:

None.

Output:

A pointer to a `f3s_arrayinfo_t` structure, filled in by the driver:

```
typedef struct f3s_arrayinfo_s
{
  uint32_t status;      /* info status */
  uint32_t total_size;  /* total size of array expressed in bytes */
  uint32_t unit_size;   /* size of a chip expressed in bytes */
  uint32_t chip_size;   /* size of a unit expressed in bytes */
}
f3s_arrayinfo_t;
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_CLRCMP

Set compresssion off for a flash filesystem

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_CLRCMP    __DIOT(_DCMD_F3S, F3S_CLRCMP, f3s_cmdcmp_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_CLRCMP
dev_data_ptr	A pointer to a `f3s_cmdcmp_t` structure (see below)
n_bytes	`sizeof(f3s_cmdcmp_t)`
dev_info_ptr	NULL

Description:

This command sets compression off for a flash filesystem.

Input:

A `f3s_cmdcmp_t` structure:

```
typedef struct f3s_cmdcmp_s
{
  uint32_t status;    /* cmdcmp status */
  uint32_t cmp_flag;  /* compression flag */
}
f3s_cmdcmp_t;
```

Set the fields to 0.

Output:

None.

See also:

DCMD_F3S_GETCMP, *DCMD_F3S_SETCMP*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_ERASE

Erase sectors in a raw flash filesystem partition

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_ERASE     __DIOT(_DCMD_F3S, F3S_ERASE, f3s_erase_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_ERASE
dev_data_ptr	A pointer to a `f3s_erase_t` structure (see below)
n_bytes	`sizeof(f3s_erase_t)`
dev_info_ptr	NULL

Description:

This command erases sectore in a raw flash filesystem partition.

Input:

A `f3s_erase_t` structure:

```
typedef struct f3s_erase_s
{
  uint32_t status;  /* erase status */
  uint32_t offset;  /* offset of first unit expressed in bytes */
  uint32_t limit;   /* limit of last unit expressed in bytes */
}
f3s_erase_t;
```

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_EXIT

Exit the driver

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_EXIT      __DIOT(_DCMD_F3S, F3S_EXIT, f3s_name_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_EXIT
dev_data_ptr	A pointer to a `f3s_name_t` structure
n_bytes	`sizeof(f3s_name_t)`
dev_info_ptr	NULL

Description:

This command exits the driver.

Input:

A `f3s_name_t` structure:

```
typedef struct f3s_name_s
{
  uint32_t status;       /* mount status */
  uint32_t mount_flags;  /* flags for mount */
  uint32_t name_length;  /* length of name */
/*unit8_t name;             root name */
}
f3s_name_t;
```

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_FORMAT

Format a flash filesystem

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_FORMAT    __DIOT(_DCMD_F3S, F3S_FORMAT, f3s_format_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_FORMAT
dev_data_ptr	A pointer to a `f3s_format_t` structure (see below)
n_bytes	`sizeof(f3s_format_t)`
dev_info_ptr	NULL

Description:

This command formats a flash filesystem.

Input:

A `f3s_format_t` structure:

```
typedef struct f3s_format_s
{
  uint32_t status;       /* format status */
  uint32_t offset;       /* offset of first unit expressed in bytes */
  uint32_t limit;        /* limit of last unit expressed in bytes */
  uint32_t unit_spare;   /* number of spare units */
  uint32_t align_pow2;   /* alignment power of two */
  uint32_t xip_pow2;     /* xip power of two */
  uint32_t name_length;  /* length of name */
/*unit8_t name;             root name */
}
f3s_format_t;
```

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_GEOINFO

Get the flash geometry

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_GEOINFO    __DIOTF(_DCMD_F3S, F3S_GEOINFO, f3s_geoinfo_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_GEOINFO
dev_data_ptr	A pointer to a `f3s_geoinfo_t` structure (see below)
n_bytes	`sizeof(f3s_geoinfo_t)`
dev_info_ptr	NULL

Description:

This command gets the flash geometry.

This command returns a maximum of 16 geometries. In the future when flash parts have more than 16, there will be an extended form of this command.

Input:

A `f3s_geoinfo_t` structure:

```
struct geo_list
{
  uint16_t unit_num;       /* Number of sectors in run */
  uint16_t unit_pow2;      /* Sector size for this run */
};

typedef struct f3s_geoinfo_s
{
  uint32_t status;
  _Paddr64t base;            /* Phys base address */
  uint32_t size;             /* Size of flash */
  uint16_t chipwidth;        /* Width of a single chip, in bytes */
  uint16_t interleave;       /* Number of chips in parallel on data bus */
  uint16_t num_geo;          /* Number of entries in the geo array */
```

```
  struct geo_list geo[16];   /* Maximum of 16 distinct geos */
}f3s_geoinfo_t;
```

Output:

A filled-in `f3s_geoinfo_t` structure.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_GETCMP

Get the compresssion setting for a flash filesystem

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_GETCMP    __DIOTF(_DCMD_F3S, F3S_GETCMP, f3s_cmdcmp_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_GETCMP
dev_data_ptr	A pointer to a `f3s_cmdcmp_t` structure (see below)
n_bytes	`sizeof(f3s_cmdcmp_t)`
dev_info_ptr	NULL

Description:

This command gets the compression setting for a flash filesystem.

Input:

A `f3s_cmdcmp_t` structure:

```
typedef struct f3s_cmdcmp_s
{
  uint32_t status;    /* cmdcmp status */
  uint32_t cmp_flag;  /* compression flag */
}
f3s_cmdcmp_t;
```

Output:

If *cmd_flag* is nonzero, the filesystem is compressed.

See also:

DCMD_F3S_CLRCMP, *DCMD_F3S_SETCMP*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_LOCKSSR

Lock a secure silicon region

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_LOCKSSR    __DIOTF(_DCMD_F3S, F3S_LOCKSSR, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/fs0**
dcmd	DCMD_F3S_LOCKSSR
dev_data_ptr	NULL, or a pointer to a `uint32_t`
n_bytes	0 or `sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command locks a secure silicon region. You can:

- Lock the entire SSR. For this case, the data pointer should be NULL, and *n_bytes* should be 0.
- Lock a specific region. For this case, the data pointer should not be NULL, and *n_bytes* should be 1. Currently, we use 1 byte to store the number of the region.

Input:

The number of the region to lock, or NULL to lock the entire OTP area.

Output:

None.

Example:

The Spansion S25FL QSPI flash supports multiple regions in the OTP (SSR) area. Each region can be locked separately:

- Case 1: Lock the entire OTP area. In the call to *devctl()*, don't specify the data pointer or size parameter:

  ```
  devctl(fd, DCMD_F3S_LOCKSSR, NULL, 0, NULL);
  ```

- Case 2: Lock a specific region. Specify the data pointer and size parameters in the call to *devctl()*. For example, to lock region 17:

```
lock = 17;
devctl(fd, DCMD_F3S_LOCKSSR, &lock, 1, NULL);
```

See also:

DCMD_F3S_READSSR, *DCMD_F3S_STATSSR*, *DCMD_F3S_WRITESSR*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_LOCK

Lock a partition

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_LOCK      __DIOT(_DCMD_F3S, F3S_LOCK, f3s_unitlock_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_LOCK
dev_data_ptr	A pointer to a `f3s_unitlock_t` structure (see below)
n_bytes	`sizeof(f3s_unitlock_t)`
dev_info_ptr	NULL

Description:

This command locks a partition.

Input:

A `f3s_unitlock_t` structure:

```
typedef struct f3s_unitlock_s
{
  uint32_t status;  /* lock status */
  uint32_t offset;  /* offset of first unit expressed in bytes */
  uint32_t limit;   /* limit of last unit expressed in bytes */
}
f3s_unitlock_t;
```

Output:

None.

See also:

DCMD_F3S_UNLOCK, *DCMD_F3S_UNLOCKALL*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_MOUNT

Mount a flash filesystem partition

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_MOUNT     __DIOT(_DCMD_F3S, F3S_MOUNT, f3s_name_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_MOUNT
dev_data_ptr	A pointer to a `f3s_name_t` structure (see below)
n_bytes	`sizeof(f3s_name_t)`
dev_info_ptr	NULL

Description:

This command mounts a flash filesystem partition.

Input:

A `f3s_name_t` structure:

```
typedef struct f3s_name_s
{
  uint32_t status;       /* mount status */
  uint32_t mount_flags;  /* flags for mount */
  uint32_t name_length;  /* length of name */
/*unit8_t name;             root name */
}
f3s_name_t;
```

Output:

None.

See also:

DCMD_F3S_UMOUNT

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_PARTINFO

Get information about a partition

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_PARTINFO   __DIOTF(_DCMD_F3S, F3S_PARTINFO, f3s_partinfo_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_PARTINFO
dev_data_ptr	A pointer to a `f3s_partinfo_t` structure (see below)
n_bytes	`sizeof(f3s_partinfo_t)`
dev_info_ptr	NULL

Description:

This command gets information about a partition.

Input:

None.

Output:

A `f3s_partinfo_t` structure, filled in by the driver:

```
typedef struct f3s_partinfo_s
{
  uint32_t status;       /* info status */
  uint32_t total_size;   /* total size of partition expressed in bytes */
  uint32_t unit_size;    /* size of a unit expressed in bytes */
  uint32_t spare_size;   /* spare size within partition expressed in bytes */
  uint32_t retire_size;  /* retired size within partition expressed in bytes */
  uint32_t head_size;    /* overhead size expressed in bytes */
  uint32_t pad_size;     /* alignment padding size expressed in bytes */
  uint32_t free_size;    /* free size in partition expressed in bytes */
  uint32_t stale_size;   /* stale size in partition expressed in bytes */
  uint32_t resv_size;    /* reserved size in partition expressed in bytes */
}
f3s_partinfo_t;
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_READSSR

Read data from the secure silicon region

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_READSSR    __DIOTF(_DCMD_F3S, F3S_READSSR, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_READSSR
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command reads data from the secure silicon region. You can open **/dev/fs0**, move the file position to the appropriate offset, and then read data there.

Input:

The number of bytes to read.

Output:

The number of bytes that were read.

See also:

DCMD_F3S_LOCKSSR, *DCMD_F3S_STATSSR*, *DCMD_F3S_WRITESSR*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_RECLAIMCTL

Provide runtime control of reclaims

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_RECLAIMCTL __DIOTF(_DCMD_F3S, F3S_RECLAIMCTL, f3s_reclaimctl_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_RECLAIMCTL
dev_data_ptr	A pointer to a `f3s_reclaimctl_t` structure (see below)
n_bytes	`sizeof(f3s_reclaimctl_t)`
dev_info_ptr	NULL

Description:

This command provides runtime control of reclaim operations.

Input:

A `f3s_reclaimctl_t` structure:

```
typedef struct f3s_reclaimctl_s
{
  int16_t super_count;
  int16_t stale_percent;
  int16_t reclaim_enable;
  int16_t reserved;        /* To pad the structure */
}f3s_reclaimctl_t;
```

The members include:

super_count

The number of overwrite hops. If the value isn't -1, it becomes the new effective reclaim threshold.

stale_percent

The percentage of stale blocks. If the value isn't -1, it becomes the new effective reclaim trigger.

reclaim_enable

1 to enable reclaims, 0 to disable them, or -1 to leave the current setting unchanged.

Output:

A `f3s_reclaimctl_t` structure that's filled in by the driver with the old settings.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_RECLAIM

Reclaim deleted blocks

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_RECLAIM   __DIOT(_DCMD_F3S, F3S_RECLAIM, f3s_reclaim_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_RECLAIM
dev_data_ptr	A pointer to a `f3s_reclaim_t` structure (see below)
n_bytes	`sizeof(f3s_reclaim_t)`
dev_info_ptr	NULL

Description:

This command reclaims deleted blocks.

Input:

A `f3s_reclaim_t` structure:

```
typedef struct f3s_reclaim_s
{
  uint32_t status;  /* reclaim status */
  uint32_t limit;   /* lower limit of free size wanted expressed in bytes */
}
f3s_reclaim_t;
```

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_SETCMP

Set compresssion on for a flash filesystem

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_SETCMP    __DIOT(_DCMD_F3S, F3S_SETCMP, f3s_cmdcmp_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_SETCMP
dev_data_ptr	A pointer to a `f3s_cmdcmp_t` structure (see below)
n_bytes	`sizeof(f3s_cmdcmp_t)`
dev_info_ptr	NULL

Description:

This command sets compression on for a flash filesystem.

Input:

A pointer to a `f3s_cmdcmp_t` structure:

```
typedef struct f3s_cmdcmp_s
{
  uint32_t status;    /* cmdcmp status */
  uint32_t cmp_flag;  /* compression flag */
}
f3s_cmdcmp_t;
```

The input isn't actually used.

Output:

None.

See also:

DCMD_F3S_CLRCMP, *DCMD_F3S_GETCMP*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_STATSSR

Get the lock status of a secure silicon region

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_STATSSR    __DIOTF(_DCMD_F3S, F3S_STATSSR, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_STATSSR
dev_data_ptr	An array of type `uint32_t`
n_bytes	The number of entries in the array
dev_info_ptr	NULL

Description:

This command gets the lock status of a secure silicon region.

Input:

- For most flash parts, the OTP area can be locked entirely. So you may pass a 1-byte data buffer to get the lock status of the entire OTP area. A value of 1 means locked, and 0 means unlocked. For example:

  ```
  devctl(fd, DCMD_F3S_STATSSR, &stat, 1, NULL);
  ```

- For some flash parts, the OTP area is divided into multiple small regions (for example 32), and each region can be locked separately. You may pass a 4-byte data buffer to get the lock stat of all 32 regions. Each bit represents the lock status of each region. For example:

  ```
  devctl(fd, DCMD_F3S_STATSSR, &stat, 4, NULL) ;
  ```

Output:

The lock status.

See also:

DCMD_F3S_LOCKSSR, *DCMD_F3S_READSSR*, *DCMD_F3S_WRITESSR*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_UMOUNT

Unmount a flash filesystem partition

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_UMOUNT    __DIOT(_DCMD_F3S, F3S_UMOUNT, f3s_name_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_UMOUNT
dev_data_ptr	A pointer to a `f3s_name_t` structure (see below)
n_bytes	`sizeof(f3s_name_t)`
dev_info_ptr	NULL

Description:

This command unmounts a flash filesystem partition.

Input:

A `f3s_name_t` structure:

```
typedef struct f3s_name_s
{
  uint32_t status;       /* mount status */
  uint32_t mount_flags;  /* flags for mount */
  uint32_t name_length;  /* length of name */
/*unit8_t name;             root name */
}
f3s_name_t;
```

Output:

None.

See also:

DCMD_F3S_MOUNT

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_UNITINFO

Get information about a unit (erase sector)

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_UNITINFO   __DIOTF(_DCMD_F3S, F3S_UNITINFO, f3s_unitinfo_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_UNITINFO
dev_data_ptr	A pointer to a `f3s_unitinfo_t` (see below)
n_bytes	`sizeof(f3s_unitinfo_t)`
dev_info_ptr	NULL

Description:

This command gets information about a unit (erase sector).

Input:

A `f3s_unitinfo_t` structure:

```
typedef struct f3s_unitinfo_s
{
  uint32_t status;       /* info status */
  uint32_t offset;       /* offset of unit in partition */
  uint32_t unit_size;    /* size of a unit expressed in bytes */
  uint32_t erase_count;  /* count of erasures on unit */
}
f3s_unitinfo_t;
```

Fill in the *offset* before calling *devctl()*.

Output:

A `f3s_unitinfo_t` structure, filled in by the driver.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_UNLOCKALL

Unlock all flash filesystem partitions

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_UNLOCKALL __DIOT(_DCMD_F3S, F3S_UNLOCKALL, f3s_arrayunlock_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_UNLOCKALL
dev_data_ptr	A pointer to a `f3s_arrayunlock_t` structure
n_bytes	`sizeof(f3s_arrayunlock_t)`
dev_info_ptr	NULL

Description:

This command unlocks all the partitions in a flash filesystem.

Input:

A `f3s_arrayunlock_t` structure:

```
typedef struct f3s_arrayunlock_s
{
  uint32_t status;  /* unlock status */
}
f3s_arrayunlock_t;
```

Output:

None.

See also:

DCMD_F3S_LOCK, *DCMD_F3S_UNLOCK*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_UNLOCK

Unlock a flash filesystem partition

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_UNLOCK     __DIOT(_DCMD_F3S, F3S_UNLOCK, f3s_unitunlock_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_UNLOCK
dev_data_ptr	A pointer to a `f3s_unitunlock_t`
n_bytes	`sizeof(f3s_unitunlock_t)`
dev_info_ptr	NULL

Description:

This command unlocks a flash filesystem partition.

Input:

A `f3s_unitunlock_t` structure:

```
typedef struct f3s_unitunlock_s
{
  uint32_t status;  /* unlock status */
  uint32_t offset;  /* offset of first unit expressed in bytes */
  uint32_t limit;   /* limit of last unit expressed in bytes */
}
f3s_unitunlock_t;
```

Output:

None.

See also:

DCMD_F3S_LOCK, *DCMD_F3S_UNLOCKALL*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_F3S_WRITESSR

Write data to the secure silicon region

Synopsis:

```
#include <sys/dcmd_f3s.h>

#define DCMD_F3S_WRITESSR    __DIOTF(_DCMD_F3S, F3S_WRITESSR, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_F3S_WRITESSR
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command writes data to the secure silicon region. You can open **/dev/fs0**, move the file position to the appropriate offset, and then write data there.

Input:

The number of bytes to write.

Output:

The number of bytes that were written.

See also:

DCMD_F3S_LOCKSSR, *DCMD_F3S_READSSR*, *DCMD_F3S_STATSSR*

devctl() in the QNX Neutrino *C Library Reference*

Chapter 9
DCMD_FSEVMGR_*

This chapter describes the *devctl()* commands concerning filesystem events.

DCMD_FSEVMGR_AUTHORIZE

Authorize inotify writing for an OCB

Synopsis:

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

#define DCMD_FSEVMGR_AUTHORIZE   __DIOT(_DCMD_FSEVMGR, 7, uint64_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/fsevents**
dcmd	DCMD_FSEVMGR_AUTHORIZE
dev_data_ptr	A pointer to a `uint64_t`
n_bytes	`sizeof(uint64_t)`
dev_info_ptr	NULL

Description:

This commands authorizes inotify writing for an OCB.

Input:

The key obtained from a DCMD_FSEVMGR_WRITER command.

Output:

See also:

DCMD_FSEVMGR_WRITER

devctl() in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_CHECK

Check to see if the filesystem event manager is running

Synopsis:

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

#define DCMD_FSEVMGR_CHECK   __DION(_DCMD_FSEVMGR, 1)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/fsevents**
dcmd	DCMD_FSEVMGR_CHECK
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command checks to see if the filesystem event manager, `fsevmgr`, is running. If it's running, *devctl()* returns EOK.

Input:

None.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_DBGLEVEL

Set the debugging level for the filesystem event manager

Synopsis:

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

#define DCMD_FSEVMGR_DBGLEVEL   __DIOT(_DCMD_FSEVMGR, 12, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/fsevents**
dcmd	DCMD_FSEVMGR_DBGLEVEL
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command sets the debugging level for the filesystem event manager.

Input:

The debugging level.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_FILTER_ADD

Add or update an inotify watch

Synopsis:

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

#define DCMD_FSEVMGR_FILTER_ADD   __DIOTF(_DCMD_FSEVMGR, 3, fsevmgr_watch_add_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by calling *inotify_init()*
dcmd	DCMD_FSEVMGR_FILTER_ADD
dev_data_ptr	A pointer to a `fsevmgr_watch_add_t`
n_bytes	`sizeof(fsevmgr_watch_add_t)`
dev_info_ptr	NULL

Description:

This command adds a watch on a file descriptor. You should call *inotify_add_watch()* instead of using this command directly.

Input:

A filled-in `fsevmgr_watch_add_t` structure:

```
typedef struct fsevmgr_watch_add_s {
        int32_t wd;
        uint32_t mask;
        int fd;
} fsevmgr_watch_add_t;
```

The members include:

wd

The watch descriptor, which you can use to remove the watch. You should initialize this member to 0.

mask

A bitwise OR of the IN_* flags defined in **<inotify.h>**; for more information, see *inotify_add_watch()* in the *C Library Reference*.

fd

A file descriptor for the path that you want to watch.

Output:

On success, the watch descriptor is stored in the *wd* member.

See also:

DCMD_FSEVMGR_FILTER_REM

devctl(), *inotify_add_watch()*, *inotify_init()*, *inotify_rm_watch()* in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_FILTER_REM

Remove an inotify watch

Synopsis:

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

#define DCMD_FSEVMGR_FILTER_REM   __DIOT(_DCMD_FSEVMGR, 4, int32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by calling *inotify_init()*
dcmd	DCMD_FSEVMGR_FILTER_REM
dev_data_ptr	A pointer to an `int32_t`
n_bytes	`sizeof(int32_t)`
dev_info_ptr	NULL

Description:

This command removes an inotify watch. You should call *inotify_rm_watch()* instead of using this command directly.

Input:

A watch descriptor.

Output:

None.

See also:

devctl(), *inotify_add_watch()*, *inotify_init()*, *inotify_rm_watch()* in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_FSEVENTCHID

Get the channel ID for the filesystem event manager's fsevents interface

Synopsis:

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

#define DCMD_FSEVMGR_FSEVENTCHID   __DIOF(_DCMD_FSEVMGR, 10, fsevmgr_eventchid_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/fsevents**
dcmd	DCMD_FSEVMGR_FSEVENTCHID
dev_data_ptr	A pointer to a `fsevmgr_eventchid_t`
n_bytes	`sizeof(fsevmgr_eventchid_t)`
dev_info_ptr	NULL

Description:

This command gets the channel ID for the filesystem event manager's fsevents interface.

Input:

None.

Output:

A filled-in `fsevmgr_eventchid_t` structure:

```
typedef struct fsevmgr_eventchid_s {
        pid_t pid;              /* Process ID */
        int32_t chid;           /* Communication channel ID */
} fsevmgr_eventchid_t;
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_INOTIFYCHID

Get the channel ID for the filesystem event manager's inotify interface

Synopsis:

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

#define DCMD_FSEVMGR_INOTIFYCHID        __DIOF(_DCMD_FSEVMGR, 11, fsevmgr_eventchid_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/inotifys**
dcmd	DCMD_FSEVMGR_INOTIFYCHID
dev_data_ptr	A pointer to a `fsevmgr_eventchid_t`
n_bytes	`sizeof(fsevmgr_eventchid_t)`
dev_info_ptr	NULL

Description:

This command gets the channel ID for the filesystem event manager's inotify interface.

Input:

None.

Output:

A filled-in `fsevmgr_eventchid_t` structure:

```
typedef struct fsevmgr_eventchid_s {
        pid_t pid;              /* Process ID */
        int32_t chid;           /* Communication channel ID */
} fsevmgr_eventchid_t;
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_MB_RESTORE

Restore a connection to an event mailbox

Synopsis:

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

#define DCMD_FSEVMGR_MB_RESTORE   __DIOT(_DCMD_FSEVMGR, 15, fsnotify_restore_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/fsevents**
dcmd	DCMD_FSEVMGR_MB_RESTORE
dev_data_ptr	A pointer to a `fsnotify_restore_t`
n_bytes	`sizeof(fsnotify_restore_t)`
dev_info_ptr	NULL

Description:

This command restores a connection to an event mailbox.

Input:

A filled-in `fsnotify_restore_t` structure:

```
typedef struct fsnotify_restore_s {
        uint64_t key;                   /* Writer key */
        uint32_t wrid;                  /* Writer id */
        pid_t pid;                      /* pid at time of save */
        int32_t newchid;                /* chid after restore */
} fsnotify_restore_t;
```

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_MB_STATE

Get the state of an event mailbox

Synopsis:

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

#define DCMD_FSEVMGR_MB_STATE   __DIOTF(_DCMD_FSEVMGR, 14, fsevmgr_mb_state_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/fsevents**
dcmd	DCMD_FSEVMGR_MB_STATE
dev_data_ptr	A pointer to a `fsevmgr_mb_state_t`
n_bytes	`sizeof(fsevmgr_mb_state_t)` plus the size of the array of `fsevmgr_vwatch_t` structures
dev_info_ptr	NULL

Description:

This command gets the state of an event mailbox.

Input:

None.

Output:

A filled-in `fsevmgr_mb_state_t` structure:

```
/*  Information about a watch as used by fsevmgr_mb_state_t
*/
typedef struct fsevmgr_vwatch_s {
        uint32_t wuid;                  /* Watch unique id */
        uint32_t mpuid;                 /* Mount point unique id for the inode */
        uint64_t inode;                 /* Inode number being watched */
        pid_t pid;                      /* Server process id */
} fsevmgr_vwatch_t;


/*  devctl for mailbox state
*/
typedef struct fsevmgr_mb_state_s {
        uint32_t count;                 /* Reported number of watches */
```

```
        uint32_t actualcount;          /* Actual number of watches */
        uint32_t maxcount;             /* Maximum number of watches */
        uint32_t muid;                 /* Mailbox unique id */
        pid_t pid;                     /* Owner id */
        fsevmgr_vwatch_t watch[0];     /* Watches associated with the mailbox */
} fsevmgr_mb_state_t;
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_QNX_EXT

Enable QNX extensions to inotify

Synopsis:

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

#define DCMD_FSEVMGR_QNX_EXT    __DIOT(_DCMD_FSEVMGR, 8, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by calling *inotify_init()*
dcmd	DCMD_FSEVMGR_QNX_EXT
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command enables QNX extensions to the inotify system.

Input:

A bitwise OR of the INOTIFY_QNX_EXT_* bits (defined in **<sys/inotify_ext.h>**) that you want to enable.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_RFILTER_ADD

Add or update a recursive inotify watch

Synopsis:

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

#define DCMD_FSEVMGR_RFILTER_ADD   __DIOTF(_DCMD_FSEVMGR, 9, fsevmgr_watch_add_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by calling *inotify_init()*
dcmd	DCMD_FSEVMGR_RFILTER_ADD
dev_data_ptr	A pointer to a `fsevmgr_watch_add_t`
n_bytes	`sizeof(fsevmgr_watch_add_t)`
dev_info_ptr	NULL

Description:

This command adds a recursive watch on a file descriptor. You should call *inotify_add_watch()* instead of using this command directly.

Input:

A filled-in `fsevmgr_watch_add_t` structure:

```
typedef struct fsevmgr_watch_add_s {
        int32_t wd;
        uint32_t mask;
        int fd;
} fsevmgr_watch_add_t;
```

The members include:

wd

The watch descriptor, which you can use to remove the watch. You should initialize this member to 0.

mask

A bitwise OR of the IN_* flags defined in **<inotify.h>**; for more information, see *inotify_add_watch()* in the *C Library Reference*.

fd

A file descriptor for the path that you want to watch.

Output:

On success, the watch descriptor is stored in the *wd* member.

See also:

DCMD_FSEVMGR_FILTER_REM

devctl(), *inotify_add_watch()*, *inotify_init()*, *inotify_rm_watch()* in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_STATE

Get the event manager's state

Synopsis:

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

#define DCMD_FSEVMGR_STATE   __DIOTF(_DCMD_FSEVMGR, 13, fsevmgr_state_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/fsevents**
dcmd	DCMD_FSEVMGR_STATE
dev_data_ptr	A pointer to a `fsevmgr_state_t`
n_bytes	`sizeof(fsevmgr_state_t)` plus the size of the array of `fsevmgr_vmb_t` structures
dev_info_ptr	NULL

Description:

This command gets the event manager's state.

Input:

None.

Output:

A filled-in `fsevmgr_state_t` structure:

```
/*  Information about a mailbox as used by fsevmgr_state_t
*/
typedef struct fsevmgr_vmb_s {
        uint32_t muid;                  /* Mailbox unique id */
        pid_t pid;                      /* Owner id */
} fsevmgr_vmb_t;

/*  devctl for event manager state
*/
typedef struct fsevmgr_state_s {
    uint32_t count;                     /* Reported number of mailboxes */
    uint32_t actualcount;               /* Actual number of mailboxes */
    uint32_t maxcount;                  /* Maximum number of mailboxes */
    pid_t pid;                          /* Event manager pid */
```

```
    fsevmgr_vmb_t mailbox[0];           /* Mailboxes associated with the event manager */
} fsevmgr_state_t;
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_STATS

Get statistics about the filesystem event manager

Synopsis:

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

#define DCMD_FSEVMGR_STATS    __DIOTF(_DCMD_FSEVMGR, 5, fsevmgr_stats_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/fsevents**
dcmd	DCMD_FSEVMGR_STATS
dev_data_ptr	A pointer to a `fsevmgr_stats_t`
n_bytes	`sizeof(fsevmgr_stats_t)` plus the size of the array of `fsevmgr_mbstats_t` structures.
dev_info_ptr	NULL

Description:

This commands gets statistics about the filesystem event manager.

Input:

None.

Output:

A filled-in `fsevmgr_stats_t` structure:

```
typedef struct fsevmgr_mbstats_s {
        uint64_t rxbytes;               /* Received total bytes  */
        uint64_t txbytes;               /* Transmitted total bytes  */
        uint64_t rxevents;              /* Received events */
        uint64_t txevents;              /* Transmitted events */
        uint64_t writecalls;            /* Number of write calls */
        uint64_t readcalls;             /* Number of read calls */
        uint64_t lostevents;            /* Number of lost events */
        uint32_t size;                  /* Mailbox size in bytes */
        uint32_t used;                  /* Mailbox bytes used */
        uint32_t maxused;               /* Mailbox maximum bytes used */
        uint32_t deferred;              /* Current deferred read count */
        uint32_t maxdeferred;           /* Maximum deferred read count */
        uint32_t watches;               /* Current number of watch filters */
```

```
        uint32_t maxwatches;            /* Maximum number of watch filters */
        uint32_t inotifymask;           /* Current combination of all inotify masks */
        uint32_t maxinotifymask;        /* Combination of all watch masks ever used */
        pid_t owner;                    /* Owning process */
} fsevmgr_mbstats_t;


typedef struct fsevmgr_stats_s {
        uint32_t mailboxes;             /* Number of mailboxes (elements in array) */
        fsevmgr_mbstats_t mbstats[1];   /* Array of mailbox stats */
} fsevmgr_stats_t;
```

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

E2BIG

The buffer provided isn't big enough to hold the data.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

DCMD_FSEVMGR_WRITER

Enable inotify writing for an OCB

Synopsis:

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

#define DCMD_FSEVMGR_WRITER   __DIOTF(_DCMD_FSEVMGR, 6, fsevmgr_chidkey_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/fsevents**
dcmd	DCMD_FSEVMGR_WRITER
dev_data_ptr	A pointer to a `fsevmgr_chidkey_t`
n_bytes	`sizeof(fsevmgr_chidkey_t)`
dev_info_ptr	NULL

Description:

This command enables inotify writing for an OCB.

Input:

An initialize `fsevmgr_chidkey_t` structure:

```
typedef struct fsevmgr_chidkey_s {
        int32_t chid;           /* Communication channel id */
        uint32_t wrid;          /* Writer unique id */
        uint64_t key;           /* Populated with a communication key */
} fsevmgr_chidkey_t;
```

Set *chid* to the channel ID of the would-be writer.

Output:

The *wrid* and *key* are set.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EINVAL

The buffer isn't the right size, the interface isn't for inotify, or the writer is already registered.

See also:

DCMD_FSEVMGR_AUTHORIZE

devctl() in the QNX Neutrino *C Library Reference*

`fsevmgr` in the *Utilities Reference*

Chapter 10
DCMD_FSYS_*

This chapter describes the *devctl()* commands that apply to filesystems.

- _DCMD_BLK and _DCMD_FSYS are the same value.
- DCMD_FSYS_FORCE_RELEARN is identical to DCMD_BLK_FORCE_RELEARN.

The following are used internally:

- DCMD_FSYS_CTL
- DCMD_FSYS_EMODE_GET
- DCMD_FSYS_EMODE_SET
- DCMD_FSYS_MEDIA
- DCMD_FSYS_SWAPIO

DCMD_FSYS_CRYPTO

Control filesystem encryption

Synopsis:

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

#define DCMD_FSYS_CRYPTO    __DIOTF(_DCMD_FSYS, 24, struct fs_crypto)
```

Description:

This command controls encryption of a filesystem. Instead of using this command directly, use the following cover functions (described in the *C Library Reference*):

- *fs_crypto_check*
- *fs_crypto_domain_add*
- *fs_crypto_domain_key_change*
- *fs_crypto_domain_key_check*
- *fs_crypto_domain_key_size*
- *fs_crypto_domain_lock*
- *fs_crypto_domain_query*
- *fs_crypto_domain_remove*
- *fs_crypto_domain_unlock*
- *fs_crypto_enable_option*
- *fs_crypto_enable*
- *fs_crypto_file_get_domain*
- *fs_crypto_file_set_domain*
- *fs_crypto_key_gen*
- *fs_crypto_migrate_control*
- *fs_crypto_migrate_path*
- *fs_crypto_migrate_status*
- *fs_crypto_migrate_tag*
- *fs_crypto_set_logging*

See also:

devctl() in the QNX Neutrino *C Library Reference*

Power-Safe filesystem in the Filesystems chapter of the QNX Neutrino *System Architecture*

DCMD_FSYS_DIRECT_IO

Perform direct I/O

Synopsis:

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

#define DCMD_FSYS_DIRECT_IO      __DIOT(_DCMD_FSYS, 15, struct fs_directio)
#define DCMD_FSYS_DIRECT_IO_IOV  __DIOT(_DCMD_FSYS, 38, struct fs_directio_iov)
#define DCMD_FSYS_DIRECT_IO_OLD  __DIOT(_DCMD_FSYS, 15, struct fs_directio_old)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_DIRECT_IO
dev_data_ptr	A pointer to a `struct fs_directio` (see below)
n_bytes	`sizeof(struct fs_directio)`
dev_info_ptr	A pointer to a 32-bit signed integer. On success, the *devctl()* call populates this buffer with the number of bytes succesfully transferred. Pass NULL if you do not want this information.

Description:

The DCMD_FSYS_DIRECT_IO command performs a direct I/O operation on a file. DCMD_FSYS_DIRECT_IO_IOV allows DCMD_FSYS_DIRECT_IO to be used when the client's buffers are not contiguous in memory.

Input:

`fs_directio`

The `fs_directio` structure is defined in **<sys/dcmd_blk.h>** as follows:

```
struct fs_directio {
        off64_t         offset;
        uint32_t        nbytes;
        uint32_t        flags;
        paddr64_t       paddr;
        uint64_t        vaddr;
};
```

The members include:

offset

The offset in the file to start the operation at.

nbytes

The number of bytes you want to read or write.

flags

The type of operation; one of:

- FS_DIO_READ — read-only
- FS_DIO_WRITE — write-only

You can OR the above with zero or one or both of the following:

- FS_DIO_SYNC — subsequent operations are complete only when the data has been successfully transferred
- FS_DIO_MAP_PHYS — map physical memory

paddr

The physical address of where to read or write the data.

vaddr

The virtual address of where to read or write the data.

fs_directio_iov

You use the `fs_directio_iov` structure when the client's buffers are not contiguous in memory. To use this form of the structure, use DCMD_FSYS_DIRECT_IO_IOV.

The `fs_directio_iov` structure is defined in **<sys/dcmd_blk.h>** as follows:

```
struct fs_directio_iov {
        off64_t         offset;
        _Uint32t        nbytes;
        _Uint32t        flags;
        _Uint32t        niov;
        __FLEXARY(struct __iovec64, iov);
};
```

The *offset* and *flags* members are as for *fs_directio*.

nbytes

The number of bytes you want to read or write. It's the sum of the lengths of each IOV specified by *iov*.

nbytes

The number of bytes you want to read or write.

niov

The number of elements in the *iov* array.

iov

An array of *struct __iovec64*. The *_iov_base* member of the *iov* holds either the *paddr* or the *vaddr*, depending on whether FS_DIO_MAP_PHYS is set in flags.

fs_directio_old

The `fs_directio_old` structure is similar to `fs_directio`, except that the *paddr* member is of type `paddr32_t`, and the *vaddr* member is of type `uint32_t`. If you want this form of the structure, use DCMD_FSYS_DIRECT_IO_OLD.

Output:

None.

Example:

```
struct fs_directio  dio;

/* Send a zero-byte read to see if direct I/O is available on the fd: */

memset(&dio, 0, sizeof dio);
dio.flags = _IO_FLAG_RD;
if (devctl(fd, DCMD_FSYS_DIRECT_IO, &dio, sizeof dio, 0) == EOK)
{
  /* Direct I/O is supported. */
}
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_FSYS_ERRNOTIFY

Register for error notifications from a filesystem

Synopsis:

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

#define DCMD_FSYS_ERRNOTIFY    __DIOT(_DCMD_FSYS, 32, struct blk_errnotify)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_ERRNOTIFY
dev_data_ptr	A pointer to a `struct blk_errnotify` (see below)
n_bytes	`sizeof(struct blk_errnotify)`
dev_info_ptr	NULL

Description:

This command registers a process for error notifications from a block filesystem. The command must come from the local process running with superuser privileges, and be directed at one of the driver's block devices.

If you successfully register for error notifications, the **io-blk.so** module does the following:

- It immediately sends your process an acknowledgement pulse, including the *ack_data* you provide (see below).
- If any errors have occurred in the past, **io-blk.so** sends a notification pulse to your process, with BLK_ERRNOTIFY_STALE as the pulse data.
- If another client is already registered for error notifications, **io-blk.so** disconnects from it. The client receives a pulse with a code of _PULSE_CODE_DISCONNECT.

If a filesystem error occurs, **io-blk.so** sends your process a pulse with the code and priority that you provided (*pulse_code* and *pulse_prio*, respectively). The pulse's value describes the error; it consists of the following 32 bits:

Bits	Mask	Description
31	0x80000000 (BLK_ERRNOTIFY_STALE)	One or more errors occured before the client registered to receive error notifications
30–9	0x7FFFFE00	Currently undefined

Bits	Mask	Description
8–0	0x000001FF	An error code from **<error.h>**

You can use this macro to extract the error code:

```
#define BLK_ERRNOTIFY_GETERROR(v)  ((v) & 0x000001ff)
```

Input:

A pointer to a `blk_errnotify` structure:

```
typedef struct blk_errnotify {
        uint32_t  signature;
        int       chid;
        int       pulse_prio;
        uint32_t  ack_data;
        uint8_t   pulse_code;
        uint8_t   spare[15];   /* reserved; set to zero */
} blk_errnotify_t;
```

The members include:

signature

This must be BLK_ERRNOTIFY_SIGNATURE.

chid

The channel ID that you want `io-blk` to send the pulse to.

pulse_prio

The priority to send the pulse at. This must not be higher than the caller's priority.

ack_data

Data to send back with the acknowledgement pulse.

pulse_code

The notification pulse's code. This must be within the range allowed for user processes (see the entry for `_pulse` in the *C Library Reference*).

Output:

None.

See also:

devctl(), `_pulse` in the QNX Neutrino *C Library Reference*

io-blk.so in the *Utilities Reference*

DCMD_FSYS_FILE_FLAGS

Get the flags for a file

Synopsis:

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

#define DCMD_FSYS_FILE_FLAGS    __DIOTF(_DCMD_FSYS, 20, struct fs_fileflags)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_FILE_FLAGS
dev_data_ptr	A pointer to a `struct fs_fileflags` (see below)
n_bytes	`sizeof(struct fs_fileflags)`
dev_info_ptr	NULL

Description:

This command gets flags that indicate file attributes. The `chattr` utility is a front end for this command; for information about the flags, see its entry in the *Utilities Reference*.

Not all of the flags are currently defined in public headers.

Input:

None.

Output:

The `fs_fileflags` structure is defined as follows:

```
struct fs_fileflags {
        uint16_t        mask[2];
        uint16_t        bits[2];
        char            basetype[16];
};
```

The members include:

mask

An array of masks, one for the generic flags, and one for filesystem-specific flags.

bits

An array of bit settings, one for the generic flags, and one for filesystem-specific flags.

basetype

The null-terminated name of the filesystem.

Example:

```
struct fs_fileflags     ff;

memset(&ff, 0, sizeof ff);
if((err = devctl(fd, DCMD_FSYS_FILE_FLAGS, &ff, sizeof ff, 0)) != EOK) {
    printf("fsys_file_flags=%s(%d)\n", strerror(err), err);
} else {
    printf("fsys_file_flags=%#x:%#x\n", ff.bits[0], ff.bits[1]);
}
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

`chattr` in the *Utilities Reference*

DCMD_FSYS_FILTER_DETACH

Detach the topmost filter from a filesystem

Synopsis:

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

#define DCMD_FSYS_FILTER_DETACH         __DION(_DCMD_FSYS,  28)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_FILTER_DETACH
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

This command detaches the topmost filter from a filesystem.

Input:

None.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_FSYS_FSEVMGR_CHECK

Notification that a filesystem event manager has been loaded

Synopsis:

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

#define DCMD_FSYS_FSEVMGR_CHECK     __DION(_DCMD_FSYS,  23)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_FSEVMGR_CHECK
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

The filesystem event manager, `fsevmgr`, uses this command to notify all of the **io-blk.so** mountpoints that an event manager has been loaded.

Input:

None.

Output:

None.

See also:

Filesystem events in the QNX Neutrino *System Architecture* guide

`fsevmgr` in the *Utilities Reference*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_FSYS_FSNOTIFY

Notify the filesystem event policy manager

Synopsis:

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

#define DCMD_FSYS_FSNOTIFY     __DIOTF(_DCMD_FSYS, 29, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_FSNOTIFY
dev_data_ptr	Depends on the request; see below
n_bytes	Depends on the request; see below
dev_info_ptr	NULL

Description:

The filesystem event policy manager, `fsevmgr` sends these commands to all of the **io-blk.so** mountpoints to make requests:

FSNOTIFY_REQ_ENABLE

Enable the `fsnotify` module. The *dev_data_ptr* argument must be a pointer to a `uint32_t`.

FSNOTIFY_REQ_INFO

Get information about the mountpoint. The *dev_data_ptr* argument must be a pointer to a `fsnotify_info_t` structure:

```
typedef struct _fsnotify_info_s {
        uint32_t command;    /* Structure type command */
        uint32_t uid;        /* Mount point unique id */
} fsnotify_info_t;
```

FSNOTIFY_REQ_WATCH_ADD

Add a watch. The *dev_data_ptr* argument must be a pointer to a `fsnotify_watch_cmd_t` structure:

```
typedef struct fsnotify_watch_cmd_s {
        uint32_t command;    /* Structure type command */
        uint32_t muid;       /* Receiving mailbox unique id */
        uint32_t wuid;       /* Watch unique id */
```

```
        uint32_t mask;      /* inotify watch mask */
        uint64_t inode;     /* inode being watched */
        uint32_t mpuid;     /* Mount point unique id */
} fsnotify_watch_cmd_t;
```

FSNOTIFY_REQ_RWATCH_ADD

Add a recursive watch. The *dev_data_ptr* argument must be a pointer to a `fsnotify_watch_cmd_t` structure, just as for FSNOTIFY_REQ_WATCH_ADD.

Input:

The input depends on the request, but includes at least the FSNOTIFY_REQ_* command itself:

FSNOTIFY_REQ_ENABLE

Set the `uint32_t` to FSNOTIFY_REQ_ENABLE.

FSNOTIFY_REQ_INFO

Set the *command* member to FSNOTIFY_REQ_INFO.

FSNOTIFY_REQ_WATCH_ADD, FSNOTIFY_REQ_RWATCH_ADD

Set the *command* member to FSNOTIFY_REQ_WATCH_ADD or FSNOTIFY_REQ_RWATCH_ADD. Fill in the *muid*, *wuid*, and *mask* fields.

Output:

The input depends on the request:

FSNOTIFY_REQ_ENABLE

None.

FSNOTIFY_REQ_INFO

The mountpoint's ID.

FSNOTIFY_REQ_WATCH_ADD, FSNOTIFY_REQ_RWATCH_ADD

The *inode* and *mpuid* members.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EINVAL

The size of the data structure isn't correct for the given request.

See also:

DCMD_FSYS_FSNOTIFY_SAVE

Filesystem events in the QNX Neutrino *System Architecture* guide

`fsevmgr` in the *Utilities Reference*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_FSYS_FSNOTIFY_SAVE

Save the notification state for the filesystem event policy manager

Synopsis:

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

#define DCMD_FSYS_FSNOTIFY_SAVE   __DIOT(_DCMD_FSYS, 34, fs_fsnotify_save_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_FSNOTIFY_SAVE
dev_data_ptr	A pointer to a `fs_fsnotify_save_t`
n_bytes	`sizeof(fs_fsnotify_save_t)`
dev_info_ptr	NULL

Description:

This command saves the notification state information for the filesystem event policy manager.

Input:

The `fs_fsnotify_save_t` structure is defined as follows:

```
typedef struct fs_fsnotify_save_s {
        char path[255];
} fs_fsnotify_save_t;
```

Set the *path* to be a zero-terminated path for the file where you want to save the state information.

Output:

None.

See also:

DCMD_FSYS_FSNOTIFY

Filesystem events in the QNX Neutrino *System Architecture* guide

`fsevmgr` in the *Utilities Reference*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_FSYS_HOOK_CTL

Control a filesystem hook

Synopsis:

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

#define DCMD_FSYS_HOOK_CTL   __DIOT(_DCMD_FSYS,  30, struct fs_hookctl_s)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_HOOK_CTL
dev_data_ptr	A pointer to a `struct fs_hookctl_s`
n_bytes	`sizeof(struct fs_hookctl_s)`
dev_info_ptr	NULL

Description:

This command controls a filesystem hook.

In order to use this command, your process needs to have the `vfs/hook-control` (BLK_ABILITY_HOOKCTL) custom ability enabled. For more information, see *procmgr_ability()* and *procmgr_ability_lookup()* in the *C Library Reference*.

Input:

The `fs_hookctl_s` structure is defined as follows:

```
/*  Cookie used to identify the VFS Hook control IOCTL
*/
#define FS_VFS_HOOK_CTL_COOKIE    (0x3A10BA57)

typedef struct fs_hookctl_s {
    uint32_t    cookie;           /* Fixed to identify the packet.              */
    uint16_t    command;          /* Command to be sent to the control.         */
    uint16_t    length;           /* Count of bytes appended to this structure. */
    uint32_t    mask;             /* Mask of hooks command should be applied to.*/
    uint8_t     reserved[8];      /* Reserved for alignment and future use.     */
    uint8_t     data[0];          /* data bytes associated with this structure. */
} fs_hookctl_t;
```

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_FSYS_LABEL, DCMD_FSYS_LABEL_RAW

Get the filesystem label

Synopsis:

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

#define DCMD_FSYS_LABEL       __DIOF(_DCMD_FSYS,  22, char[256])
#define DCMD_FSYS_LABEL_RAW   __DIOF(_DCMD_FSYS,  27, uint8_t[256])
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_LABEL or DCMD_FSYS_LABEL_RAW
dev_data_ptr	A `char[256]` or `uint8_t[256]` buffer
n_bytes	The size of the buffer
dev_info_ptr	NULL

Description:

The DCMD_FSYS_LABEL command gets the filesystem's volume label as a string in UTF-8 Unicode format; DCMD_FSYS_LABEL_RAW gets the label as an array of unsigned characters with no conversions performed.

A number of filesystems store the volume label on the media in formats other than UTF-8 Unicode. For example, the Windows NT filesystem stores the volume label in UTF-16 Unicode format; our NTFS implementation (**fs-nt.so**) converts it to UTF-8 Unicode.

The DOS filesystem stores volume labels in a variety of formats:

- Old versions of DOS can use SBCS (Single-Byte Character Set) or DBCS (Double-Byte Character Set) encoding, also known as *DOS codepages*.
- Windows may use UTF-16 Unicode.
- Linux may use UTF-8 Unicode.

Our DOS filesystem (**fs-dos.so**) tries to convert the label to UTF-8 Unicode, but it can't automatically determine which encoding was used; you can use the `dos codepage=...` option to specify the encoding. If the DOS filesystem fails to convert any characters in the label to UTF-8 Unicode, it replaces them with the "bad" ASCII character. By default, this is the ASCII underscore (_) character (ASCII code 0x5F), but you can change this with the `dos badchar=...` option.

You can use the DCMD_FSYS_LABEL_RAW command to work around the difficulties with character conversions that DCMD_FSYS_LABEL may run into. DCMD_FSYS_LABEL_RAW gets the bytes as they're

stored on the media, without any conversion. You can then convert the string to whatever character set you wish.

Input:

None.

Output:

The filesystem label.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <devctl.h>
#include <sys/dcmd_blk.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>

int main (void)
{
    int fd;
    int ret;
    char vol_label[256];

    fd = open ("/dev/hd0t179", O_RDONLY);
    if (fd == -1)
    {
        perror ("open()");
        return (EXIT_FAILURE);
    }

    memset (vol_label, 0, sizeof(vol_label));
    ret = devctl(fd, DCMD_FSYS_LABEL, vol_label, sizeof(vol_label), NULL);

    if (ret == EOK)
    {
        printf ("Label: %s\n", vol_label);
    } else {
        printf ("DCMD_FSYS_LABEL failed: %s\n", strerror(ret) );
        return (EXIT_FAILURE);
    }

    return (EXIT_SUCCESS);
}
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_FSYS_MAP_OFFSET

Map a logical filesystem offset to a physical filesystem or device offset

Synopsis:

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

#define DCMD_FSYS_MAP_OFFSET    __DIOTF(_DCMD_FSYS, 21, union fs_blkmap)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_MAP_OFFSET
dev_data_ptr	A pointer to a `union fs_blkmap`
n_bytes	`sizeof(union fs_blkmap)`
dev_info_ptr	NULL

Description:

This command maps a logical filesystem offset to a physical filesystem or device offset, depending on the flag specified in the input. It uses a `union fs_blkmap` for both input and output:

```
union fs_blkmap {
        struct {
                off64_t         logical;
                uint32_t        flags;
        } i;
        struct {
                off64_t         physical;
                uint32_t        nbytes;
        } o;
};
```

Input:

The *i* (input) member includes:

logical

The logical offset.

flags

One of the following:

- FS_BMAP_FSYS — map to the filesystem.
- FS_BMAP_DEVICE — map to the physical device.

Output:

The *o* (output) member includes:

physical

The starting offset, in bytes.

nbytes

The length of the extent.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

ENXIO

There are no more extents.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_FSYS_MOUNTED_AT, DCMD_FSYS_MOUNTED_BY, DCMD_FSYS_MOUNTED_ON

Get mount information

Synopsis:

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

#define DCMD_FSYS_MOUNTED_ON    __DIOF(_DCMD_FSYS,  16, char[256])
#define DCMD_FSYS_MOUNTED_AT    __DIOF(_DCMD_FSYS,  17, char[256])
#define DCMD_FSYS_MOUNTED_BY    __DIOF(_DCMD_FSYS,  18, char[256])
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_MOUNTED_AT, DCMD_FSYS_MOUNTED_BY, or DCMD_FSYS_MOUNTED_ON
dev_data_ptr	A `char[256]` buffer
n_bytes	The size of the buffer
dev_info_ptr	NULL

Description:

These commands return 256 bytes of character data, giving information about the relationship of the filesystem associated with the given file descriptor to other filesystems:

This command:	Asks:
DCMD_FSYS_MOUNTED_ON	Who am I on top of?
DCMD_FSYS_MOUNTED_BY	Who is on top of me?
DCMD_FSYS_MOUNTED_AT	Where am I? Who is my owner?

Input:

None.

Output:

The associated path.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

ENODEV

There is no such entity.

Example:

See "Mounting options" in the RAM-disk Filesystem chapter of *The QNX Neutrino Cookbook*.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_FSYS_OPTIONS

Get the options that a filesystem was mounted with

Synopsis:

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

#define DCMD_FSYS_OPTIONS    __DIOF(_DCMD_FSYS,  19, char[256])
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_OPTIONS
dev_data_ptr	A `char[256]` buffer
n_bytes	The size of the buffer
dev_info_ptr	NULL

Description:

This command gets the command-line options that the given filesystem was mounted with. The `df` utility uses this command if you specify the `-g` option.

Input:

None.

Output:

A null-terminated string that contains the command-line arguments.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EMSGSIZE

The provided buffer isn't big enough.

Example:

```
char o[265];
int fd, err;
```

```
if(-1 == (fd = open(argv[1] ? argv[1] : "/", O_RDONLY)))
{
    perror ("Couldn't open the device");
    return EXIT_FAILURE;
}

memset(o, 0, sizeof(o));
if((err = devctl(fd, DCMD_FSYS_OPTIONS, o, sizeof(o), 0)) != EOK)
{
    o[0] = '\0';
    printf("Couldn't get the options: %s (%d)\n", strerror(err), err);
} else {
    printf("Fsys options: \"%s\"\n", o);
}
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_FSYS_PGCACHE_CTL

*Control the contents of the disk cache in **io-blk.so***

Synopsis:

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

#define DCMD_FSYS_PGCACHE_CTL   __DIOTF(_DCMD_FSYS, 36, struct pgcache_ctl)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_FSYS_PGCACHE_CTL
dev_data_ptr	A pointer to a `struct pgcache_ctl`
n_bytes	`sizeof(struct pgcache_ctl)`
dev_info_ptr	NULL

Description:

This command controls the contents of the disk cache in **io-blk.so**. The calling program must run with superuser privileges on the local node in order to use this command. The device can be a physical disk or a partition.

Input:

A `pgcache_ctl_t` structure that specifies the details of the operation. This structure is defined in **<sys/dcmd_blk.h>** as follows:

```
typedef struct pgcache_ctl {
    uint32_t  op;                   /* DCMD_FSYS_PGCACHE_CTL_OP_... command code */
    union {
        uint8_t data[60];           /* overall struct size is 64 bytes */
        struct {
            uint32_t  flags;
            off64_t   start_addr;
            off64_t   end_addr;
        } discard;
        struct {
            uint32_t  flags;        /* PGCACHE_CTL_RESIZE_.. flags */
            off64_t   nbytes;
            uint64_t  curr_size;
        } resize;
```

```
    };
} pgcache_ctl_t;
```

The *op* member indicates the operation, and the other fields are specific to the operation:

- DCMD_FSYS_PGCACHE_CTL_OP_DISCARD — selectively discard the contents of the disk cache.
- DCMD_FSYS_PGCACHE_CTL_OP_RESIZE — dynamically resize the disk cache in block I/O (`devb-*`) drivers.

Discarding the contents of the disk cache

The DCMD_FSYS_PGCACHE_CTL_OP_DISCARD operation walks the page cache, finds all cached device pages that contain disk blocks from the specified disk range, and attempts to discard them. It's careful not to discard "dirty" blocks (which are waiting to be written out to disk), locked blocks (which are being worked on by other threads, and may have in-progress I/O on them), and blocks that are being waited for by other threads (to avoid the need to wake up these waiters, and possible races that may arise from that). All such sectors are termed "busy." This command returns EAGAIN if it encounters any busy cache pages.

The disk range is defined by the starting and ending offsets (given in bytes, and measured from the start of the device) specified in the *start_addr* and *end_addr* members of *discard* in the `pgcache_ctl_t` structure. There are currently no flags defined, so you should set *flags* to zero.

It's safe to call this devctl() on a mounted partition or a physical disk that's being actively used by any type of mounted filesystem(s). The disk cache is locked for the duration of this operation.

Resizing the disk cache

The DCMD_FSYS_PGCACHE_CTL_OP_RESIZE operation dynamically resizes the disk cache in block I/O (`devb-*`) drivers. You can do this even while the disk driver is actively handling I/O requests of all types (file I/O, direct I/O, and so on). There's no need to pause I/O or unmount mounted volumes in order to resize the disk cache.

The size of disk cache can be reduced (deflating the disk cache) or increased (inflating the disk cache) to any value between 1 MB and 512 MB.

The *flags* member controls the operation:

- If PGCACHE_CTL_RESIZE_DIFF is set, *nbytes* is interpreted as the change (positive or negative) in the size of the disk cache. If this flag is clear, *nbytes* is interpreted as absolute size of the disk cache.
- If PGCACHE_CTL_RESIZE_PERMANENT is clear, the limit on the size of disk cache (typically set via the `blk cache=...` command-line option) is honored, and the disk cache doesn't grow past it. If this flag is set, the limit on the size of disk cache is increased to accommodate *nbytes*.

In order to avoid long delays in I/O caused by locking the disk cache for extended periods of time, a single DCMD_FSYS_PGCACHE_CTL operation is limited to deflating disk cache by at most 15 MB, or to inflating it by at most 4 MB. For larger changes, make multiple *devctl()* calls.

By default, a disk driver fully preallocates the disk cache when the driver starts, to a maximum size specified by the `blk cache=`*max-size*. It isn't possible to resize a fully preallocated disk cache; in this case, *devctl()* returns ENOTSUP.

In order to resize the disk cache, you must specify the `blk alloc=demand` option on the disk driver's command line. This option makes the driver dynamically allocate disk cache in incremental fashion as needed by I/O (to a maximum size specified by the `blk cache=`*max-size* option).

Output:

The output depends on the operation:

DCMD_FSYS_PGCACHE_CTL_OP_DISCARD

None.

DCMD_FSYS_PGCACHE_CTL_OP_RESIZE

The *curr_size* member contains the current size of disk cache (in bytes), and *nbytes* contains the change in the disk cache's size.

Since the size of disk cache varies with I/O, and because there could be multiple processes calling this *devctl()* concurrently, you should consider the returned size of the disk cache to be a reasonably accurate estimate rather than an exact value.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EAGAIN

The DCMD_FSYS_PGCACHE_CTL_OP_DISCARD operation encountered some busy cache pages.

ENOTSUP

You tried to do a DCMD_FSYS_PGCACHE_CTL operation on a fully preallocated disk cache.

See also:

io-blk.so in the *Utilities Reference*

devctl() in the QNX Neutrino *C Library Reference*

DCMD_FSYS_PREGROW_FILE

Extend a file, optionally zero-filling it

Synopsis:

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

#define DCMD_FSYS_PREGROW_FILE   __DIOT(_DCMD_FSYS,  14, off64_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_PREGROW_FILE
dev_data_ptr	A pointer to a `off64_t`
n_bytes	`sizeof(off64_t)`
dev_info_ptr	NULL

Description:

This command extends a file, optionally zero-filling it. In contrast, the POSIX *ftruncate()* function extends a file *and* zero-fills the new data space.

In order to use this command, your process needs to have the `vfs/pregrow` (BLK_ABILITY_PREGROW) custom ability enabled. For more information, see *procmgr_ability()* and *procmgr_ability_lookup()* in the *C Library Reference*.

The `pregrow-fill` option to **io-blk.so** specifies whether or not this command zeroes the content when growing files. Zeroing the content is the default but increases the time for pregrowing files; not zeroing is very fast but insecure, as it allows access to the old content of the disk blocks.

Input:

The file size.

Output:

None.

Example:

```
int fd;
off64_t sz;

fd=open(...);
```

```
sz=...;

if (devctl(fd, DCMD_FSYS_PREGROW_FILE, &sz, sizeof(sz), NULL) != EOK)
{
   /* Error */
}
```

See also:

devctl(), *ftruncate()* in the QNX Neutrino *C Library Reference*

io-blk.so in the *Utilities Reference*

DCMD_FSYS_STATISTICS, DCMD_FSYS_STATISTICS_CLR

Get filesystem statistics

Synopsis:

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

#define DCMD_FSYS_STATISTICS       __DIOF(_DCMD_FSYS,  11, struct fs_stats)
#define DCMD_FSYS_STATISTICS_CLR   __DIOF(_DCMD_FSYS,  12, struct fs_stats)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_STATISTICS or DCMD_FSYS_STATISTICS_CLR
dev_data_ptr	A pointer to a `struct fs_stats`
n_bytes	`sizeof(struct fs_stats)`
dev_info_ptr	NULL

Description:

These commands get statistics about the filesystem associated with the given file descriptor. DCMD_FSYS_STATISTICS_CLR sets the counters to zero after returning their values. The `fsysinfo` utility is a front end for DCMD_FSYS_STATISTICS.

In order to use these commands, your process needs to have the `vfs/stats-clear` (BLK_ABILITY_STATSCLEAR) custom ability enabled. For more information, see *procmgr_ability()* and *procmgr_ability_lookup()* in the *C Library Reference*.

Input:

None.

Output:

A filled-in `struct fs_stats`, which is defined in **<sys/fs_stats.h>** as follows:

```
struct fs_stats {
    /* Version and Time statistics */
    uint32_t    s_version;              /* Lib version (BLKIO_VERSION)   */
    uint32_t    s_fsys_flags;           /* Fsys mount flags (_MOUNT_*)   */
    uint32_t    s_time_mount;           /* Time that file system mounted */
    uint32_t    s_time_clr;             /* Time that statistics cleared  */
```

```
/* Buffer/DiskIO statistics    */
uint64_t    s_buf_rphys;            /* Physical reads from disk    */
uint64_t    s_buf_wphys;            /* Physical writes to disk     */
uint64_t    s_buf_readahead;        /* Physical predictive reads   */
uint64_t    s_buf_direct;           /* Physical direct-io accesses */
uint64_t    s_buf_badblks;          /* Physical IO errors          */
uint64_t    s_buf_rcache;           /* Cache reads (read hits)     */
uint64_t    s_buf_wcache;           /* Cache writes (write-behind) */
uint64_t    s_buf_mru;              /* MRU cache kB (GLOBAL)       */
uint64_t    s_buf_mfu;              /* MFU cache kB (GLOBAL)       */

/* Name cache statistics       */
uint64_t    s_name_poshits;         /* Positive hits (usable hit)        */
uint64_t    s_name_neghits;         /* Negative hits (usable hit)        */
uint64_t    s_name_misses;          /* Misses (not in cache)             */
uint64_t    s_name_uncacheable;     /* Names not considered (long/ambig) */
uint64_t    s_name_stale;           /* Stale hits (GLOBAL)               */

/* System call API statistics  */
uint64_t    s_syscall_open;         /* Number of open()s        */
uint64_t    s_syscall_stat;         /* Number of stat()s        */
uint64_t    s_syscall_namei;        /* Number of name lookups   */
uint64_t    s_syscall_read;         /* Number of read() calls   */
uint64_t    s_syscall_write;        /* Number of write() calls  */
uint64_t    s_syscall_devctl;       /* Number of devctl() calls */
uint64_t    s_syscall_create;       /* Number of file creations */
uint64_t    s_syscall_unlink;       /* Number of file deletions */

/* Mapping cache statistics    */
uint64_t    s_map_hits;             /* Usable hits           */
uint64_t    s_map_misses;           /* Misses (not in cache) */

/* Vnode statistics            */
uint64_t    s_vnode_create;         /* Created vnode (unique file) */
uint64_t    s_vnode_hits;           /* Usable vnode hit            */
uint64_t    s_vnode_lock;           /* Vnodes locked               */
uint64_t    s_vnode_recycle;        /* Reused vnode (GLOBAL)       */

/* Slab/memory statistics      */
uint64_t    s_slab_pg_map;          /* Memory pages mapped (GLOBAL)   */
uint64_t    s_slab_pg_unmap;        /* Memory pages unmapped (GLOBAL) */

/* Thread pool statistics      */
uint64_t    s_tid_pool_create;      /* Threads created (GLOBAL)   */
uint64_t    s_tid_pool_destroy;     /* Threads destroyed (GLOBAL) */

/* New statistics added June, 2011 */
uint64_t    s_buf_rcache_bytes;     /* Bytes read from the cache        */
uint64_t    s_buf_wcache_bytes;     /* Bytes written into the cache     */
uint64_t    s_buf_rphys_bytes;      /* Physical bytes read from disk    */
uint64_t    s_buf_wphys_bytes;      /* Physical bytes written to disk   */
uint64_t    s_buf_readahead_bytes;  /* Physical bytes from read ahead   */
uint64_t    s_buf_direct_bytes;     /* Physical bytes R/W direct-io     */
```

```
    uint64_t    s_buf_io_count;         /* Count of IO requests created     */
    uint64_t    s_buf_io_bytes;         /* Total bytes of io requests       */

    uint64_t    s_syscall_read_bytes;   /* Number of bytes read via read()  */
    uint64_t    s_syscall_write_bytes;  /* Number of bytes wrote via write()*/

    uint64_t    s_syscall_trunc;        /* Number of truncate() calls       */
    uint64_t    s_syscall_rename;       /* Number of rename() calls         */
    uint64_t    s_syscall_owner;        /* chown/chgrp calls                */
    uint64_t    s_syscall_modes;        /* chmod calls                      */
    uint64_t    s_syscall_sync;         /* Number of sync() calls           */

    uint64_t    s_vfs_relearn;          /* Count of relearn events          */
    uint64_t    s_vfs_periodic;         /* Periodic call count into the fs  */

    uint64_t    s_msg_resume;           /* Number of message resume ops     */

    /* Spares (new statistics)     */
    uint64_t    s_spare[13];
};
```

The *s_time_mount* and *s_time_clr* members are specified as 32-bit time fields for backwards compatibility and are subject to rollover in 2038.

See also:

devctl() in the QNX Neutrino *C Library Reference*

`fsysinfo` in the QNX Neutrino *Utilities Reference*

DCMD_FSYS_STATVFS

Get filesystem information

Synopsis:

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

#define DCMD_FSYS_STATVFS    __DIOF(_DCMD_FSYS,  13, struct __msg_statvfs)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_FSYS_STATVFS
dev_data_ptr	A pointer to a `struct __msg_statvfs`
n_bytes	`sizeof(struct __msg_statvfs)`
dev_info_ptr	NULL

Description:

This command gets information about the filesystem associated with the file descriptor passed to *devctl()*.

Input:

None.

Output:

A filled-in `__msg_statvfs` structure.

(QNX Neutrino 7.0 or later) For compatibility between 32- and 64-bit programs, the DCMD_FSYS_STATVFS command uses a `__msg_statvfs` structure instead of a `statvfs` structure. The `__msg_statvfs` structure is equivalent to the 32-bit version of `statvfs`. The **<sys/statvfs.h>** header file declares some functions and macros that you can use to convert one structure into the other:

```
/* Conversion methods from/to __msg_statvfs and statvfs */
extern void __msg_statvfs_init64(struct __msg_statvfs * _msg, const struct statvfs64 * _statvfsp);
extern void __msg_statvfs_copy64(struct statvfs64 * _statvfsp, const struct __msg_statvfs * _msg);

/* Alias methods, casting to statvfs64 simplifies handling the fsblkcnt_t high and low bytes */
#define __msg_statvfs_init(_msg, _statvfsp) (__msg_statvfs_init64((_msg), (const struct statvfs64 *)(_statvfsp)))
#define __msg_statvfs_copy(_statvfsp, _msg) (__msg_statvfs_copy64((struct statvfs64 *)(_statvfsp), (_msg)))
```

For a description of the `statvfs` structure, see *statvfs()* in the *C Library Reference*.

Example:

For an example of providing the information for this command, see "Filesystem statistics" in the RAM-disk Filesystem chapter of *The QNX Neutrino Cookbook*.

See also:

devctl(), *fstatvfs() statvfs()* in the QNX Neutrino *C Library Reference*

Chapter 11
DCMD_IP_*

This chapter describes the *devctl()* commands that apply to the Internet Protocol stack.

DCMD_IP_FDINFO

Get information about a socket

Synopsis:

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

#define DCMD_IP_FDINFO          __DIOF(_DCMD_IP, 0x06, char)
```

Arguments to *devctl()*:

Argument	Value
filedes	A descriptor for a socket.
dcmd	DCMD_IP_FDINFO
dev_data_ptr	A pointer to a character buffer
n_bytes	The size of the buffer
dev_info_ptr	NULL

Description:

This command gets some information about a socket. This information depends on the protocol used.

Input:

None.

Output:

The information about the socket.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_IP_GDESTADDR

Get the foreign address associated with a socket

Synopsis:

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

#define DCMD_IP_GDESTADDR       __DIOF(_DCMD_IP, 0x02, struct sockaddr)
```

Arguments to *devctl()*:

Argument	Value
filedes	A descriptor for a socket.
dcmd	DCMD_IP_GDESTADDR
dev_data_ptr	A pointer to a `struct sockaddr`
n_bytes	`sizeof(struct sockaddr)`
dev_info_ptr	NULL

Description:

This command gets the foreign address associated with a socket. Clients usually use *getpeername()* to generate this request.

Input:

None.

Output:

A `struct sockaddr` that contains the address.

See also:

DCMD_IP_GSRCADDR, *DCMD_IP_SDESTADDR*, *DCMD_IP_SSRCADDR*

devctl(), *getpeername()* in the QNX Neutrino *C Library Reference*

DCMD_IP_GSRCADDR

Get the local address associated with a socket

Synopsis:

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

#define DCMD_IP_GSRCADDR        __DIOF(_DCMD_IP, 0x00, struct sockaddr)
```

Arguments to *devctl()*:

Argument	Value
filedes	A descriptor for a socket.
dcmd	DCMD_IP_GSRCADDR
dev_data_ptr	A pointer to a `struct sockaddr`
n_bytes	`sizeof(struct sockaddr)`
dev_info_ptr	NULL

Description:

This command gets the local address associated with a socket. Clients usually use *getsockname()* to generate this request.

Input:

None.

Output:

The socket address.

See also:

DCMD_IP_GDESTADDR, *DCMD_IP_SDESTADDR*, *DCMD_IP_SSRCADDR*

devctl(), *getsockname()* in the QNX Neutrino *C Library Reference*

DCMD_IP_LISTEN

Listen for connections on a socket, specifying a queue limit

Synopsis:

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

#define DCMD_IP_LISTEN          __DIOT(_DCMD_IP, 0x04, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A descriptor for a socket.
dcmd	DCMD_IP_LISTEN
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(unint32_t)`
dev_info_ptr	NULL

Description:

This command indicates a willingness to listen for connections on a socket and specifies a queue limit. Clients usually use *listen()* to generate this request.

Input:

The limit on the queue.

Output:

None.

See also:

devctl(), *listen()* in the QNX Neutrino *C Library Reference*

DCMD_IP_SDESTADDR

Set the foreign address associated with a socket

Synopsis:

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

/*
 * Desc:  Set foreign address to associate with socket.
 * Args:  Pointer to struct sockaddr from which address is taken.
 * Notes: Clients usually use connect() to generate this request.
 */
#define DCMD_IP_SDESTADDR       __DIOT(_DCMD_IP, 0x03, struct sockaddr)
```

Arguments to *devctl()*:

Argument	Value
filedes	A descriptor for a socket.
dcmd	DCMD_IP_SDESTADDR
dev_data_ptr	A pointer to a `struct sockaddr`
n_bytes	`sizeof(struct sockaddr)`
dev_info_ptr	NULL

Description:

This command sets the foreign address associated with a socket. Clients usually use *getpeername()* to generate this request.

Input:

A `struct sockaddr` that contains the address.

Output:

None.

See also:

DCMD_IP_GDESTADDR, *DCMD_IP_GSRCADDR*, *DCMD_IP_SSRCADDR*

devctl(), *getpeername()* in the QNX Neutrino *C Library Reference*

DCMD_IP_SHUTDOWN

Set the method of shutting down a socket

Synopsis:

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

#define DCMD_IP_SHUTDOWN        __DIOT(_DCMD_IP, 0x05, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A descriptor for a socket.
dcmd	DCMD_IP_SHUTDOWN
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command sets the method of shutting down a socket. Clients usually use *shutdown()* to generate this request.

Input:

The method you want to use; one of:

SHUT_RD

Don't allow further receives.

SHUT_WR

Don't allow further sends.

SHUT_RDWR

Don't allow further sends or receives.

Output:

None.

See also:

devctl(), *shutdown()* in the QNX Neutrino *C Library Reference*

DCMD_IP_SSRCADDR

Set the local address to associate with a socket

Synopsis:

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

#define DCMD_IP_SSRCADDR        __DIOT(_DCMD_IP, 0x01, struct sockaddr)
```

Arguments to *devctl()*:

Argument	Value
filedes	A descriptor for a socket.
dcmd	DCMD_IP_SSRCADDR
dev_data_ptr	A pointer to a `struct sockaddr`
n_bytes	`sizeof(struct sockaddr)`
dev_info_ptr	NULL

Description:

This command sets the local address to associate with a socket. Clients usually use *bind()* to generate this request.

Input:

A `struct sockaddr` that specifies the address.

Output:

None.

See also:

DCMD_IP_GDESTADDR, *DCMD_IP_GSRCADDR*, *DCMD_IP_SDESTADDR*

bind(), *devctl()* in the QNX Neutrino *C Library Reference*

Chapter 12
DCMD_MEMMGR_*

The only command that's currently defined is DCMD_MEMMGR_MEMOBJ. It's for internal use and shouldn't be called directly; instead use the *shm_ctl()* cover function.

Chapter 13
DCMD_MISC_*

This chapter describes the miscellaneous *devctl()* commands.

DCMD_MISC_MQGETATTR

Get the attributes of a message queue

Synopsis:

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

#define DCMD_MISC_MQGETATTR             __DIOF(_DCMD_MISC, 1, struct mq_attr)
```

Arguments to *devctl()*:

Argument	Value
filedes	A message-queue descriptor (of type `mqd_t`) that you obtained by opening the queue.
dcmd	DCMD_MISC_MQGETATTR
dev_data_ptr	A pointer to a `struct mq_attr`
n_bytes	`sizeof(struct mq_attr)`
dev_info_ptr	NULL

Description:

This command gets the attributes for a message queue. For more information, see *mq_gettattr()* in the QNX Neutrino *C Library Reference*

Input:

None.

Output:

The attributes of the message queue.

See also:

DCMD_MISC_MQSETATTR

devctl(), *mq_gettattr()* in the QNX Neutrino *C Library Reference*

DCMD_MISC_MQSETATTR

Set the attributes of a message queue

Synopsis:

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

#define DCMD_MISC_MQSETATTR   __DIOT(_DCMD_MISC, 2, struct mq_attr)
```

Arguments to *devctl()*:

Argument	Value
filedes	A message-queue descriptor (of type `mqd_t`) that you obtained by opening the queue.
dcmd	DCMD_MISC_MQSETATTR
dev_data_ptr	A pointer to a `struct mq_attr`
n_bytes	`sizeof(struct mq_attr)`
dev_info_ptr	NULL

Description:

This command sets the attributes for a message queue. For more information, see *mq_settattr()* in the QNX Neutrino *C Library Reference*

Input:

The attributes that you want to set for the message queue.

Output:

None.

See also:

DCMD_MISC_MQGETATTR

devctl(), *mq_settattr()* in the QNX Neutrino *C Library Reference*

DCMD_MISC_MQSETCLOSEMSG

Inject a canned message when a queue descriptor is closed

Synopsis:

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

#define DCMD_MISC_MQSETCLOSEMSG __DIOT(_DCMD_MISC, 4, struct { char __data[64];})
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device.
dcmd	DCMD_MISC_MQSETCLOSEMSG
dev_data_ptr	A pointer to a `struct { char __data[64];}`.
n_bytes	`sizeof(struct { char __data[64];})`
dev_info_ptr	NULL

Description:

This command specifies a canned message that's injected when a message descriptor is closed.

The alternate (`mq`) implementation of message queues doesn't support this command.

Input:

The string that you want to inject.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

Chapter 14
DCMD_MMCSD_*

This chapter describes the *devctl()* commands that apply to MultiMedia Card/Secure Digital. This is the older framework for these cards; see also the DCMD_SDMMC_* commands.

The DCMD_MMCSD_GET_ECCERR_ADDR command isn't implemented.

DCMD_MMCSD_CARD_REGISTER

Read a card register

Synopsis:

```
#include <hw/dcmd_sim_mmcsd.h>

#define DCMD_MMCSD_CARD_REGISTER   __DIOTF(_DCMD_CAM, _SIM_MMCSD + 4, struct _mmcsd_card_registe
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_MMCSD_CARD_REGISTER
dev_data_ptr	A pointer to an `MMCSD_CARD_REGISTER` structure, followed by additional data
n_bytes	`sizeof(MMCSD_CARD_REGISTER)` plus the size of the additional data
dev_info_ptr	NULL

Description:

This command reads a card register.

The `MMCSD_CARD_REGISTER` structure is defined as follows:

```
typedef struct _mmcsd_card_register {
        uint32_t                action;
        uint32_t                type;
        uint32_t                address;
        uint32_t                length;
        uint32_t                rsvd[2];
/*      uint8_t                 data[ length ]; variable length data */
} MMCSD_CARD_REGISTER;
```

The members include:

action

The action to take. The only currently defined action is:

- MMCSD_CR_ACTION_READ — read the register

type

The type of register to read; one of the following:

- MMCSD_REG_TYPE_CID — card ID
- MMCSD_REG_TYPE_CSD — card-specific data
- MMCSD_REG_TYPE_EXT_CSD — extended card-specific data (MMC devices only)
- MMCSD_REG_TYPE_SCR — SD Configuration Register (SD devices only)

address

Not used.

length

Not used.

Input:

Set the *action* and *type* members as necessary.

Output:

The *type* member is set to a CARD_TYPE_* on return.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_MMCSD_ERASE

Erase an area on the device

Synopsis:

```
#include <hw/dcmd_sim_mmcsd.h>

#define DCMD_MMCSD_ERASE    __DIOTF(_DCMD_CAM, _SIM_MMCSD + 3, struct _mmcsd_erase)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_MMCSD_ERASE
dev_data_ptr	A pointer to a `MMCSD_ERASE` structure
n_bytes	`sizeof(MMCSD_ERASE)`
dev_info_ptr	NULL

Description:

This command erases part of a device.

Input:

An `MMCSD_ERASE` structure that specifies the area to erase:

```
typedef struct _mmcsd_erase {
        uint32_t                action;
        uint32_t                rsvd;
        uint64_t                lba;
        uint64_t                nlba;
        uint64_t                rsvd2;
} MMCSD_ERASE;
```

The members include:

action

The action to take; one of:

- MMCSD_ERASE_ACTION_NORMAL
- MMCSD_ERASE_ACTION_SECURE

lba

The logical block address to start erasing at.

nlba

The number of logical blocks to erase.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_MMCSD_ERASED_VAL

Get an erased value

Synopsis:

```
#include <hw/dcmd_sim_mmcsd.h>

#define DCMD_MMCSD_ERASED_VAL   __DIOF(_DCMD_CAM, _SIM_MMCSD + 53, uint8_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_MMCSD_ERASED_VAL
dev_data_ptr	A pointer to a `uint8_t`
n_bytes	`sizeof(uint8_t)`
dev_info_ptr	NULL

Description:

This command gets an erased value from the RPMB.

Input:

None.

Output:

The erased value.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_MMCSD_GET_CID

Get card ID information

Synopsis:

```
#include <hw/dcmd_sim_mmcsd.h>

#define DCMD_MMCSD_GET_CID   __DIOTF(_DCMD_CAM, _SIM_MMCSD + 0, struct _mmcsd_cid)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_MMCSD_GET_CID
dev_data_ptr	A pointer to a `MMCSD_CID` structure (see below)
n_bytes	`sizeof(MMCSD_CID)`
dev_info_ptr	NULL

Description:

This command gets the card ID information from the device. The `MMCSD_CID` structure is defined as follows:

```
#define MMCSD_MAX_SLOTNAME_LEN  32

/* CID reg values of the card */
typedef struct _mmcsd_cid {
        uint32_t                flags;
        uint8_t         rsvd[4];
        union{
                struct{
                        uint32_t            cid[4];
                }full_cid;
                union{
                        struct{
                                uint8_t         mid;            /* Manufacture ID */
                                uint8_t         oid[3];         /* OEM/Application ID */
                                uint8_t         pnm[6];         /* Product name */
                                uint8_t         prv;            /* Product revision */
                                uint32_t        psn;            /* Product serial number */
                                uint16_t        mdt;            /* Manufacture date */
                        }sd_cid;
                        struct{
                                uint32_t        mid;            /* Manufacture ID */
                                uint16_t        oid;            /* OEM ID */
                                uint8_t         pnm[8];         /* Product name */
                                uint8_t         hwr;            /* HW revision */
                                uint8_t         fwr;            /* FW revision */
```

```
                                uint32_t        psn;            /* Product serial number */
                                uint8_t         mcd;            /* Month code */
                                uint16_t        ycd;            /* Year code */
                        }mmc_cid;
                }parsed_cid;
        }cid;
        pid_t           pid;                    /* Store PID of driver process for this device */
        uint32_t        speed;                  /* Card speed currently working at */
        uint32_t        media_change;           /* Media change counter */
        uint8_t         hwspec_version;         /* physical layer spec */
        uint8_t         csd_version;            /* CSD structure version */
        uint8_t         mmcprot_version;        /* MMC proto version */
        uint8_t type;                           /* card type, MMC or SD for now */
        char            slotname[MMCSD_MAX_SLOTNAME_LEN];               /* slot name */
} MMCSD_CID;
```

Some of the special values for these members include:

flags

The bits include:

- MMCSD_FULL_CID — request a raw/full CID instead of a parsed CID
- MMCSD_ECC_INFO — request ECC error information
- MMCSD_CARD_STATUS — request an MMC_SEND_STATUS command to see if the card is alive
- MMCSD_DEV_RDONLY — write protected
- MMCSD_DEV_NO_MEDIA — no media inserted
- MMCSD_DEV_RDY — the media is ready to accept I/O
- MMCSD_DEV_PRELOAD — the device is in the slot before the driver started
- MMCSD_DEV_LOCKED — the device is locked
- MMCSD_DEV_MEDIA_ERROR — the device is inserted but there were error when identifying it
- MMCSD_DEV_ECC — the device has an ECC error

type

MMCSD_CARD_TYPE_UNKNOWN, MMCSD_CARD_TYPE_MMC, or MMCSD_CARD_TYPE_SD. This indicates which member of the *parsed_cid* union to examine.

Input:

Zero the entire structure, and then set the *flags* member to a bitwise OR of MMCSD_FULL_CID, MMCSD_ECC_INFO, and MMCSD_CARD_STATUS, as required.

Output:

A filled-in `MMCSD_CID` structure.

See also:

DCMD_MMCSD_GET_CID_RAW

devctl() in the QNX Neutrino *C Library Reference*

DCMD_MMCSD_GET_CID_RAW

Get raw card ID information

Synopsis:

```
#include <hw/dcmd_sim_mmcsd.h>

#define DCMD_MMCSD_GET_CID_RAW   __DIOF(_DCMD_CAM, _SIM_MMCSD + 52, uint32_t[4])
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_MMCSD_GET_CID_RAW
dev_data_ptr	An array of four `uint32_t` entries
n_bytes	The size of the array
dev_info_ptr	NULL

Description:

This command gets the raw card ID information from the device.

Input:

None.

Output:

The raw card ID information.

See also:

DCMD_MMCSD_GET_CID

devctl() in the QNX Neutrino *C Library Reference*

DCMD_MMCSD_GET_CSD

Get card-specific data

Synopsis:

```
#include <hw/dcmd_sim_mmcsd.h>

#define DCMD_MMCSD_GET_CSD   __DIOTF(_DCMD_CAM, _SIM_MMCSD + 2, struct _mmcsd_csd)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_MMCSD_GET_CSD
dev_data_ptr	A pointer to a `MMCSD_CSD` structure (see below)
n_bytes	`sizeof(MMCSD_CSD)`
dev_info_ptr	NULL

Description:

This command gets card-specific data.

Input:

None.

Output:

A filled-in `MMCSD_CSD` structure:

```
typedef struct _mmc_csd_ext {
        uint32_t        hs_max_dtr;
        uint32_t        sectors;
        uint8_t         erase_grp_def;
        uint8_t         hc_erase_group_size;
        uint8_t         hc_wp_grp_size;
        uint8_t         user_wp;
} MMC_CSD_EXT;

typedef struct _mmcsd_csd {
        uint32_t        flags;
        uint8_t         rsvd[4];
        union {
                struct {
                        uint8_t         csd_structure;  /* CSD structure */
                        uint8_t         taac;
                        uint8_t         nsac;
```

```
        uint8_t         tran_speed;
        uint16_t        ccc;
        uint8_t         read_bl_len;
        uint8_t         read_bl_partial;
        uint8_t         write_blk_misalign;
        uint8_t         read_blk_misalign;
        uint8_t         dsr_imp;
        union {
                struct {
                        uint16_t        c_size;
                        uint8_t         vdd_r_curr_min;
                        uint8_t         vdd_r_curr_max;
                        uint8_t         vdd_w_curr_min;
                        uint8_t         vdd_w_curr_max;
                        uint8_t         c_size_mult;
                } csd_ver1;
                struct {
                        uint32_t        c_size;
                } csd_ver2;
        }csd;
        uint8_t         erase_blk_en;
        uint8_t         sector_size;
        uint8_t         wp_grp_size;
        uint8_t         wp_grp_enable;
        uint8_t         r2w_factor;
        uint8_t         write_bl_len;
        uint8_t         write_bl_partial;
        uint8_t         file_format_grp;
        uint8_t         copy;
        uint8_t         perm_write_protect;
        uint8_t         tmp_write_protect;
        uint8_t         file_format;
} sd_csd;
struct {
        uint8_t         csd_structure;  /* CSD structure */
        uint8_t         mmc_prot;
        uint8_t         taac;
        uint8_t         nsac;
        uint8_t         tran_speed;
        uint16_t        ccc;
        uint8_t         read_bl_len;
        uint8_t         read_bl_partial;
        uint8_t         write_blk_misalign;
        uint8_t         read_blk_misalign;
        uint8_t         dsr_imp;
        uint16_t        c_size;
        uint8_t         vdd_r_curr_min;
        uint8_t         vdd_r_curr_max;
        uint8_t         vdd_w_curr_min;
        uint8_t         vdd_w_curr_max;
        uint8_t         c_size_mult;
        union {
                struct {        /* MMC system specification version 3.1 */
                        uint8_t erase_grp_size;
                        uint8_t erase_grp_mult;
                } mmc_v31;
                struct {        /* MMC system specification version 2.2 */
                        uint8_t sector_size;
                        uint8_t erase_grp_size;
                } mmc_v22;
        } erase;
```

```
                        MMC_CSD_EXT     ext_csd;
                        uint8_t         wp_grp_size;
                        uint8_t         wp_grp_enable;
                        uint8_t         r2w_factor;
                        uint8_t         write_bl_len;
                        uint8_t         write_bl_partial;
                /*      uint8_t         file_format_grp; */
                        uint8_t         copy;
                        uint8_t         perm_write_protect;
                        uint8_t         tmp_write_protect;
                        uint8_t         ecc;
                } mmc_csd;
        } csd;
        pid_t           pid;                    /* Store PID of driver process for this device */
        uint8_t         csd_version;            /* CSD structure version */
        uint8_t         mmcprot_version;        /* MMC proto version */
        uint8_t         type;                   /* card type, MMC or SD for now */
} MMCSD_CSD;
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_MMCSD_RPMB_RW_FRAME

Read or write an RPMB frame

Synopsis:

```
#include <hw/dcmd_sim_mmcsd.h>

#define DCMD_MMCSD_RPMB_RW_FRAME    __DIOTF(_DCMD_CAM, _SIM_MMCSD + 50, struct _mmcsd_rpmb_req)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_MMCSD_RPMB_RW_FRAME
dev_data_ptr	A pointer to a `MMCSD_RPMB_REQ` structure
n_bytes	`sizeof(MMCSD_RPMB_REQ)`
dev_info_ptr	NULL

Description:

This command reads or writes an RPMB frame. The `MMCSD_RPMB_REQ` structure is defined as follows:

```
#define RPMB_KEY_MAC_FIELD_LEN   32
#define RPMB_DATA_FIELD_LEN      256
#define RPMB_HASH_DATA_LEN       32
#define RPMB_USABLE_DATA_LEN     (RPMB_DATA_FIELD_LEN - RPMB_HASH_DATA_LEN)
#define RPMB_NONCE_FIELD_LEN     16
#define RPMB_ONE_BLK_MAC_LEN     (RPMB_DATA_FIELD_LEN + RPMB_NONCE_FIELD_LEN + 4 + 2 + 2 + 2 + 2)
                                 /*(write_cntr +  address + block_cnt + result + req_resp) */

typedef struct _mmcsd_rpmb_frame {
    uint8_t         stuff[196];
    uint8_t         key_mac[RPMB_KEY_MAC_FIELD_LEN];
    uint8_t         data[RPMB_DATA_FIELD_LEN];
    uint8_t         nonce[RPMB_NONCE_FIELD_LEN];
    uint32_t        write_cntr;
    uint16_t        address;
    uint16_t        block_cnt;
    uint16_t        result;
    uint16_t        req_resp;
} MMCSD_RPMB_FRAME;

typedef struct _mmcsd_rpmb_req {
    uint16_t        req_type;
    paddr64_t       rpmb_frame_paddr; /* Pointer to an MMCSD_RPMB_FRAME */
} MMCSD_RPMB_REQ;
```

Input:

Set the *req_type* and *req_resp* members to one of the following:

- KEY_PROG_REQUEST — not used.
- READ_CNTR_REQUEST — get the number of writes made to the partition and store it in the *write_cntr* member.
- AUTH_WRITE_REQUEST — write an RMPB block. Set *address* to the address of the data that you want to write, *block_cnt* to the number of blocks to write, and *write_cntr* to the value obtained from a READ_CNTR_REQUEST made just before you make the write request.
- AUTH_READ_REQUEST — read an RPMB block. Set *address* to the address of a buffer where you want the data to be placed. If *address* is NULL, this request simply validates the key.
- STATUS_READ_REQUEST — not used.
- KEY_PROG_RESPONSE — not used.
- READ_CNTR_RESPONSE — not used.
- AUTH_WRITE_RESPONSE — not used.
- AUTH_READ_RESPONSE — not used.

Get a one-time key from the device before calling *devctl()*, and then store the key in the *nonce* member.

The *stuff* member is used for additional data, such as an initialization vector.

Output:

The *result* member is 0 for success; any other value indicates an error. The rest of the output depends on the type of request.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_MMCSD_RPMB_SIZE

Get the size of the RPMB partition

Synopsis:

```
#include <hw/dcmd_sim_mmcsd.h>

#define DCMD_MMCSD_RPMB_SIZE   __DIOF(_DCMD_CAM, _SIM_MMCSD + 54, uint8_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_MMCSD_RPMB_SIZE
dev_data_ptr	A pointer to a `uint8_t`
n_bytes	`sizeof(uint8_t)`
dev_info_ptr	NULL

Description:

This command gets the size of the Replay Protected Memory Block, a partition on eMMC devices that conform to the JEDEC 4.4 standard. This block is used for secure data.

Input:

None.

Output:

The size of the RPMB partition.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_MMCSD_VUC_CMD

Execute a vendor-unique command

Synopsis:

```
#include <hw/dcmd_sim_mmcsd.h>

#define DCMD_MMCSD_VUC_CMD   __DIOTF(_DCMD_CAM, _SIM_MMCSD + 6, struct _mmcsd_vuc_cmd)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_MMCSD_VUC_CMD
dev_data_ptr	An array of `MMCSD_VUC_CMD` structures
n_bytes	The size of the array
dev_info_ptr	NULL

Description:

This command executes a vendor-unique command. The `MMCSD_VUC_CMD` structure is defined as follows:

```
typedef struct _mmcsd_vuc_cmd {
        int             result;
        uint16_t        opcode;
        uint16_t        rsvd2;
        uint32_t        flags;
        uint32_t        arg;
        uint32_t        resp[4];
        uint32_t        blk_sz;
        paddr_t         data_ptr;
        uint32_t        buf_off;
        uint32_t        data_len;
        uint32_t        timeout;
        uint32_t        postdelay_us;
        uint32_t        rsvd[2];
} MMCSD_VUC_CMD;
```

The members include:

result

The return code from the VUC; one of:

- MMC_VUC_SUCCESS

- MMC_VUC_FAILED
- MMC_VUC_NOTISSUED
- MMC_VUC_NODEV

opcode

flags

A bitwise OR of the following:

- MMCSD_VUC_END — this is the last command in the array
- MMCSD_VUC_DATA_NONE —
- MMCSD_VUC_DATA_IN —
- MMCSD_VUC_DATA_OUT —
- MMCSD_VUC_DATA_PHYS — the *data_ptr* member holds the physical address of the data; if this bit isn't set, the device is using PIO mode.
- MMCSD_VUC_RCA — is the relative card address valid in the *rca* field
- MMCSD_VUC_ACMD — an application-specific command is needed
- MMCSD_VUC_NOAC12 — by default, auto CMD12 is enabled
- MMCSD_VUC_RESP_OFF — bit offset
- MMCSD_VUC_RESP_NONE —
- MMCSD_VUC_RESP_R1 —
- MMCSD_VUC_RESP_R1B —
- MMCSD_VUC_RESP_R2 —
- MMCSD_VUC_RESP_R3 —
- MMCSD_VUC_RESP_R6 —
- MMCSD_VUC_RESP_R7 —

MMCSD_VUC_DATA_MSK is a mask for the bits that indicate whether or not data is present and its direction. MMCSD_VUC_RESP_MSK is a mask for the response type.

```
/*
 * for performance,
 * we assume that the RESP bits are the same as internal driver defines.
 * If any of the internal defines change (which I doubt), we will need to
 * add parsing code to do the translation.
#define MMC_RSP_PRESENT (1 << 0)
#define MMC_RSP_136             (1 << 1)        // 136 bit response
#define MMC_RSP_CRC             (1 << 2)        // expect valid crc
#define MMC_RSP_BUSY    (1 << 3)        // card may send busy
#define MMC_RSP_OPCODE  (1 << 4)        // response contains opcode
 */
```

arg

resp[4]

blk_sz

data_ptr

A physical address of a buffer that's provided by client. It's assumed to be noncacheable dma-able (contiguous).

buf_off

For PIO mode (i.e., MMCSD_VUC_DATA_PHYS isn't set), this is the offset of the buffer for this command in the array, starting from the first command structure.

data_len

timeout

The timeout value for the command, in milliseconds. The default is 5 seconds if this member is set to 0.

postdelay_us

The number of microseconds to sleep for after each VUC.

Input:

Output:

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_MMCSD_WRITE_PROTECT

Clear or set write protection

Synopsis:

```
#include <hw/dcmd_sim_mmcsd.h>

#define DCMD_MMCSD_WRITE_PROTECT    __DIOTF(_DCMD_CAM, _SIM_MMCSD + 1, struct _mmcsd_write_protect
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_MMCSD_WRITE_PROTECT
dev_data_ptr	A pointer to a `MMCSD_WRITE_PROTECT` structure (see below)
n_bytes	`sizeof(MMCSD_WRITE_PROTECT)`
dev_info_ptr	NULL

Description:

This command clears or sets write protection.

Input:

A filled-in `MMCSD_WRITE_PROTECT` structure:

```
typedef struct _mmcsd_write_protect {
        uint32_t                action;
        uint32_t                mode;
        uint64_t                lba;
        uint64_t                nlba;
        uint64_t                rsvd2;
} MMCSD_WRITE_PROTECT;
```

The members include:

action

The action to take; one of:

- MMCSD_WP_ACTION_CLR — remove write protection
- MMCSD_WP_ACTION_SET — set write protection

mode

The bits include:

- MMCSD_WP_MODE_PWR_WP_EN — apply power-on period protection

lba

The logical block address to start at.

nlba

The number of logical blocks.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

Chapter 15
DCMD_PROC_*

For information about the following commands, see "Controlling processes via the **/proc** filesystem" in the QNX Neutrino *Programmer's Guide*:

- DCMD_PROC_BREAK
- DCMD_PROC_CHANNELS
- DCMD_PROC_CLEAR_FLAG
- DCMD_PROC_CURTHREAD
- DCMD_PROC_EVENT
- DCMD_PROC_GETALTREG
- DCMD_PROC_GET_BREAKLIST
- DCMD_PROC_GETFPREG
- DCMD_PROC_GETGREG
- DCMD_PROC_GETREGSET
- DCMD_PROC_INFO
- DCMD_PROC_IRQS
- DCMD_PROC_MAPDEBUG_BASE
- DCMD_PROC_MAPDEBUG
- DCMD_PROC_MAPINFO
- DCMD_PROC_PAGEDATA
- DCMD_PROC_PTINFO
- DCMD_PROC_RUN
- DCMD_PROC_SETALTREG
- DCMD_PROC_SET_FLAG
- DCMD_PROC_SETFPREG
- DCMD_PROC_SETGREG
- DCMD_PROC_SETREGSET
- DCMD_PROC_SIGNAL
- DCMD_PROC_STATUS
- DCMD_PROC_STOP
- DCMD_PROC_SYSINFO
- DCMD_PROC_THREADCTL
- DCMD_PROC_TIDSTATUS
- DCMD_PROC_TIMERS
- DCMD_PROC_WAITSTOP

The following commands are for memory partitioning, which has been discontinued:

- DCMD_PROC_ADD_MEMPARTID

- DCMD_PROC_CHG_MEMPARTID
- DCMD_PROC_DEL_MEMPARTID
- DCMD_PROC_GET_MEMPART_LIST

Chapter 16
DCMD_PROF_*

This chapter describes the *devctl()* commands that apply to the profiler.

DCMD_PROF_ATTACH

Attach a process to the profiler

Synopsis:

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

#define DCMD_PROF_ATTACH          __DIOT(_DCMD_MISC, PROF_ATTACH, struct __prof_clientinfo)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor for the profiler that you obtained by opening **/dev/profiler**.
dcmd	DCMD_PROF_ATTACH
dev_data_ptr	A pointer to a `struct __prof_clientinfo` (see below)
n_bytes	`sizeof(struct __prof_clientinfo)`
dev_info_ptr	NULL

Description:

This command attaches a process to the profiler.

The `__prof_clientinfo` structure is defined in **<sys/profiler.h>** as follows:

```
struct __prof_clientinfo {
    unsigned     cmd;          /* Command for this request */
    unsigned     cap_flags;    /* capabilities */
    unsigned     version;      /* Version of the profile code */
    int          reserved;
    uintptr_t    lowpc;        /* Low address for this mapping */
    uintptr_t    highpc;       /* High address for this mapping */
    void         *mcounts;     /* Address in process for count array if present */
    void         *arcdata;     /* Address in process for arc data if present */
    void         *bb_head;     /* Address in process for basic block info if present */
    int          from_off;     /* Offset in shared memory for the from structures */
    int          from_size;    /* Size of froms structures in bytes */
    int          tos_off;      /* Offset in shared memory for the tos structure */
    int          tos_size;     /* Size of tos structures in bytes */
    int          hash_frac;    /* Hash fraction for tos structures */
    int          shmem_key;    /* Key used to identify shared memory */
    int          map_name_len; /* name of mapping identifier if present */
    __FLEXARY(char, map_name); /* char map_name[] */
};
```

The bits for the *cmd* member include:

- PROF_CMD_ADD_MAPPING — add a mapping to a running process.
- PROF_CMD_ARCS — use the old style of arc, which linked a “from IP” to a “to”
- PROF_CMD_ARCS_2 — use the current style of arc, which links a “to or self IP” to a “from” structure
- PROF_CMD_INIT — not implemented
- PROF_CMD_QUERY_THREAD — keep track of thread-level profiling information.
- PROF_CMD_QUERY_SHLIB — try to profile shared libraries.
- PROF_CMD_REMOVE_MAPPING — remove a mapping to a running process.

The bits for the *cap_flags* member include:

- PROF_CAP_ARCCNTS — not used
- PROF_CAP_BBINFO — read basic block information.
- PROF_CAP_SAMPLER — not used
- PROF_CAP_SHLIB — not used
- PROF_CAP_THREAD — do thread-level function call accounting.

Normally, the executable puts the arc information in a shared memory object that has the form **/dev/shmem/prof-pid-***%d*, where *%d* is the key. The profiler agent uses the information in shared memory to get call pairs, total call numbers, and so on.

The name of the mapping is passed in for information purposes. This is typically the basename of the executable for the main text segment, and the `soname` of libraries when passing in shared library information.

Input:

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_PROF_DETACH

Detach a process from the profiler

Synopsis:

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

#define DCMD_PROF_DETACH          __DIOT(_DCMD_MISC, PROF_DETACH, NULL)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor for the profiler that you obtained by opening **/dev/profiler**.
dcmd	DCMD_PROF_DETACH
dev_data_ptr	A pointer to a `struct __prof_clientinfo` (see below)
n_bytes	`sizeof(struct __prof_clientinfo)`
dev_info_ptr	NULL

Description:

This command detaches the calling process from the profiler.

Although DCMD_PROF_DETACH is declared with a NULL:

```
#define DCMD_PROF_DETACH          __DIOT(_DCMD_MISC, PROF_DETACH, NULL)
```

you should use it like this:

```
struct __prof_clientinfo clocal;

memset (clocal, 0, sizeof (clocal));
devctl(prof_fd, DCMD_PROF_DETACH, &clocal, sizeof clocal, 0);
```

or else the command fails (even though it doesn't actually use the data).

Input:

None.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_PROF_MAPPING_ADD

Add a mapping from a running process

Synopsis:

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

#define DCMD_PROF_MAPPING_ADD     __DIOT(_DCMD_MISC, PROF_MAPPING_ADD, struct __prof_clientinfo)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor for the profiler that you obtained by opening **/dev/profiler**.
dcmd	DCMD_PROF_MAPPING_ADD
dev_data_ptr	A pointer to a `struct __prof_clientinfo` (see *DCMD_PROF_ATTACH*), optionally followed by the name of the mapping identifier.
n_bytes	`sizeof(struct __prof_clientinfo)` plus the length of the name
dev_info_ptr	NULL

Description:

This command adds a mapping from a running process.

Input:

Initialize the structure to 0, and then:

- Set the *cmd* member to PROF_CMD_ADD_MAPPING, ORing in PROF_CMD_ARCS or PROF_CMD_ARCS_2, depending on the type of arcs (see *DCMD_PROF_ATTACH*).
- Set *lowpc* and *highpc* to the low and high addresses for the mapping.
- Specify the offsets and sizes of the “froms” and “tos” structures.
- Specify the *hash_frac* member. This represents the fraction of text space to allocate for the “from” hash buckets. The value is based on the minimum number of bytes of separation between two subroutine call points in the object code. A value of 2 saves a reasonable amount of space for profiling data structures without (in practice) sacrificing any granularity.
- Optionally specify an integer to use as the *shmem_key* member.
- Optionally provide a name of the mapping identifier.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_PROF_MAPPING_REM

Remove a mapping from a running process

Synopsis:

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

#define DCMD_PROF_MAPPING_REM     __DIOT(_DCMD_MISC, PROF_MAPPING_REM, struct __prof_clientinfo)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor for the profiler that you obtained by opening **/dev/profiler**.
dcmd	DCMD_PROF_MAPPING_REM
dev_data_ptr	A pointer to a `struct __prof_clientinfo` (see *DCMD_PROF_ATTACH*)
n_bytes	`sizeof(struct __prof_clientinfo)`
dev_info_ptr	NULL

Description:

This command removes a mapping from a running process.

Input:

Initialize the structure to 0, and then:

- Set the *cmd* member to PROF_CMD_REM_MAPPING, ORing in PROF_CMD_ARCS or PROF_CMD_ARCS_2, depending on the type of arcs (see *DCMD_PROF_ATTACH*).
- Set *lowpc* and *highpc* to the low and high addresses for the mapping.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_PROF_QUERY

Query if a capability is needed

Synopsis:

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

#define DCMD_PROF_QUERY          __DIOT(_DCMD_MISC, PROF_QUERY, struct __prof_clientinfo)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor for the profiler that you obtained by opening **/dev/profiler**.
dcmd	DCMD_PROF_QUERY
dev_data_ptr	A pointer to a `struct __prof_clientinfo` (see *DCMD_PROF_ATTACH*)
n_bytes	`sizeof(struct __prof_clientinfo)`
dev_info_ptr	NULL

Description:

This command queries if a capability is needed, returning either EOK or EACCES.

Input:

Set the *cmd* member of the `struct __prof_clientinfo` to one of the following:

- PROF_CMD_QUERY_THREAD — keep track of thread-level profiling information.
- PROF_CMD_QUERY_SHLIB — try to profile shared libraries.

Output:

None.

Errors:

The *devctl()* function can return the following, in addition to the error codes listed in its entry in the *C Library Reference*:

EACCES

The capability isn't needed.

See also:

devctl() in the QNX Neutrino *C Library Reference*

Chapter 17
DCMD_PTPD_*

This chapter describes the *devctl()* commands that apply to the Precision Time Protocol. If you specify the `-k` option, `ptpd` and `ptpd-avb` register **/dev/ptpd** (DEFAULT_PTPD_PATH, defined in **<dcmd_ptpd.h>**) in the path namespace; use these commands with a file descriptor from opening this path.

Some of these commands are supported only by `ptpd-avb`; see the individual entries.

DCMD_PTPD_DELAYMS

Get the master-to-slave delay

Synopsis:

```
#include <dcmd_ptpd.h>

#define DCMD_PTPD_DELAYMS    __DIOF(_DCMD_NET, 10, MyTimeInternal )
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/ptpd**
dcmd	DCMD_PTPD_DELAYMS
dev_data_ptr	A pointer to a `MyTimeInternal` structure
n_bytes	`sizeof(MyTimeInternal)`
dev_info_ptr	NULL

Description:

The DCMD_PTPD_DELAYMS command gets the network latency in the direction from the master to the slave for the Precision Time Protocol.

In order to use this command, you must specify the `-K` option when you start `ptpd` or `ptpd-avb`.

Input:

None

Output:

A filled-in `MyTimeInternal` structure, which is defined in **<dcmd_ptpd.h>** as follows:

```
typedef struct {
        uint32_t seconds;
        uint32_t nanoseconds;
} MyTimeInternal;
```

Example:

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <fcntl.h>
#include <devctl.h>
#include <dcmd_ptpd.h>

int main(int argc, char **argv) {
        int                                     fd, ret;
        MyTimeInternal          delayMS, delaySM, currTime;
        MyStatusInternal        status = PTPD_STATUS_NO_ERROR;
        MyPTPDINFOInternal      info;
        MyTAIUTCInternal        utcOffset;

        if ((fd = open( DEFAULT_PTPD_PATH, O_RDONLY)) == -1) {
                        fprintf( stderr,"open failed\n");
                        return(1);
        }


        /* Find out what the value is set to initially */
        if ( (ret = devctl(fd, DCMD_PTPD_DELAYMS, &delayMS, sizeof(delayMS), NULL)) != 0 ) {
                fprintf( stderr,"devctl returned error %x", ret );
                exit( -1 );
        }
        printf("ptpd delayMS is %d sec, %d nanosec\n",  delayMS.seconds, delayMS.nanoseconds);

        if ( (ret = devctl(fd, DCMD_PTPD_DELAYSM, &delaySM, sizeof(delaySM), NULL)) != 0 ) {
                fprintf( stderr,"devctl returned error %x", ret );
                exit( -1 );
        }
        printf("ptpd delaySM is %d sec, %d nanosec\n",  delaySM.seconds, delaySM.nanoseconds);

#if defined(__QNXNTO__) && defined(PTP_8021_AS)
        if ( (ret = devctl(fd, DCMD_PTPD_STATUS, &status, sizeof(status), NULL)) != 0 ) {
                fprintf( stderr,"devctl returned error %x", ret );
                exit( -1 );
        }
        printf("ptpd status is %d \n", status);

        if ( (ret = devctl(fd, DCMD_PTPD_INFO, &info, sizeof(info), NULL)) != 0 ) {
                fprintf( stderr,"devctl returned error %x", ret );
                exit( -1 );
        }
        printf("ptpd peerdelay is %llu \n", info.peerdelay);
        if (info.neighborrateratio_valid) {
                printf("ptpd neighborrateratio %d\n", info.neighborrateratio);
        } else
                printf("ptpd neighborrateratio invalid\n");
#endif

        if ( (ret = devctl(fd, DCMD_PTPD_GET_TAI_UTC_OFFSET, &utcOffset, sizeof(utcOffset), NULL)) != 0 ) {
                fprintf( stderr, "devctl returned error %x", ret );
                exit ( -1 );
        }
        if (utcOffset.currentUtcOffsetValid) {
                printf("current UTC Offset is %d\n", utcOffset.currentUtcOffset);
```

```
        } else {
                printf("current UTC offset is invalid\n");
        }

        return(0);
}
```

See also:

DCMD_PTPD_DELAYSM, DCMD_PTPD_GET_PDELAY_INTERVAL, DCMD_PTPD_GET_TAI_UTC_OFFSET, DCMD_PTPD_GET_TIME, DCMD_PTPD_INFO, DCMD_PTPD_SEND_SIGNALING_MSG, DCMD_PTPD_SET_PDELAY_INTERVAL, DCMD_PTPD_STATUS

devctl() in the QNX Neutrino *C Library Reference*

`ptpd`, `ptpd-avb` in the *Utilities Reference*

http://ptpd.sourceforge.net/

DCMD_PTPD_DELAYSM

Get the slave-to-master delay

Synopsis:

```
#include <dcmd_ptpd.h>

#define DCMD_PTPD_DELAYSM    __DIOF(_DCMD_NET, 11, MyTimeInternal )
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/ptpd**
dcmd	DCMD_PTPD_DELAYSM
dev_data_ptr	A pointer to a `MyTimeInternal` structure
n_bytes	`sizeof(MyTimeInternal)`
dev_info_ptr	NULL

Description:

The DCMD_PTPD_DELAYSM command gets the network latency in the direction from the slave to the master for the Precision Time Protocol.

In order to use this command, you must specify the `-K` option when you start `ptpd` or `ptpd-avb`.

Input:

None

Output:

A filled-in `MyTimeInternal` structure, which is defined in **<dcmd_ptpd.h>** as follows:

```
typedef struct {
        uint32_t seconds;
        uint32_t nanoseconds;
} MyTimeInternal;
```

Example:

See DCMD_PTPD_DELAYMS

See also:

DCMD_PTPD_DELAYMS, DCMD_PTPD_GET_PDELAY_INTERVAL, DCMD_PTPD_GET_TAI_UTC_OFFSET, DCMD_PTPD_GET_TIME, DCMD_PTPD_INFO, DCMD_PTPD_SEND_SIGNALING_MSG, DCMD_PTPD_SET_PDELAY_INTERVAL, DCMD_PTPD_STATUS

devctl() in the QNX Neutrino *C Library Reference*

`ptpd`, `ptpd-avb` in the *Utilities Reference*

http://ptpd.sourceforge.net/

DCMD_PTPD_GET_PDELAY_INTERVAL

Get the pdelay interval

Synopsis:

```
#include <dcmd_ptpd.h>

#define DCMD_PTPD_GET_PDELAY_INTERVAL   __DIOF(_DCMD_NET, 17, MyPDelayIntervalInternal )
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/ptpd**
dcmd	DCMD_PTPD_GET_PDELAY_INTERVAL
dev_data_ptr	A pointer to a `MyPDelayIntervalInternal`
n_bytes	`sizeof(MyPDelayIntervalInternal)`
dev_info_ptr	NULL

Description:

(QNX Neutrino 7.0 or later) This command gets the mean time interval between successive PDELAY_REQ messages in the Precision Time Protocol.

In order to use this command, you must specify the `-K` option when you start `ptpd-avb`. Note that `ptpd` doesn't support this command.

Input:

None.

Output:

A `MyPDelayIntervalInternal` that `ptpd` sets to the base-2 logarithm of the current interval, in seconds:

```
typedef uint32_t MyPDelayIntervalInternal;
```

Example:

```
MyPDelayIntervalInternal my_pdelay_interval;

if(devctl(fd, DCMD_PTPD_GET_PDELAY_INTERVAL, &my_pdelay_interval,
          sizeof my_pdelay_interval, NULL) != EOK)
{
```

```
    /* Error */
}
```

See also:

DCMD_PTPD_DELAYMS, DCMD_PTPD_DELAYSM, DCMD_PTPD_GET_TAI_UTC_OFFSET, DCMD_PTPD_GET_TIME, DCMD_PTPD_INFO, DCMD_PTPD_SEND_SIGNALING_MSG, DCMD_PTPD_SET_PDELAY_INTERVAL, DCMD_PTPD_STATUS

devctl() in the QNX Neutrino *C Library Reference*

`ptpd-avb` in the *Utilities Reference*

http://ptpd.sourceforge.net/

DCMD_PTPD_GET_TAI_UTC_OFFSET

Get the offset from TAI to UTC

Synopsis:

```
#include <dcmd_ptpd.h>

#define DCMD_PTPD_GET_TAI_UTC_OFFSET   __DIOF(_DCMD_NET, 18, MyTAIUTCInternal )
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/ptpd**
dcmd	DCMD_PTPD_GET_TAI_UTC_OFFSET
dev_data_ptr	A pointer to a `MyTAIUTCInternal` structure
n_bytes	`sizeof(MyTAIUTCInternal)`
dev_info_ptr	NULL

Description:

(QNX Neutrino 7.0.4 or later) This command gets the offset from Internation Atomic Time (TAI) to Universal Coordinated Time (UTC). TAI doesn't account for leap seconds, so the offset is the number of leap seconds inserted into UTC, as reported by the master clock in the Precision Time Protocol.

In order to use this command, you must specify the `-K` option when you start `ptpd` or `ptpd-avb`.

Input:

None.

Output:

A filled-in `MyTAIUTCInternal` structure, which is defined in **<dcmd_ptpd.h>** as follows:

```
typedef struct {
    uint16_t currentUtcOffset;
    uint8_t currentUtcOffsetValid;
} MyTAIUTCInternal;
```

The members include:

currentUtcOffset

The number of leap seconds inserted into UTC.

currentUtcOffsetValid

Nonzero if the *currentUtcOffset* member is valid.

Example:

See DCMD_PTPD_DELAYMS

See also:

DCMD_PTPD_DELAYMS, DCMD_PTPD_DELAYSM, DCMD_PTPD_GET_PDELAY_INTERVAL, DCMD_PTPD_GET_TIME, DCMD_PTPD_INFO, DCMD_PTPD_SEND_SIGNALING_MSG, DCMD_PTPD_SET_PDELAY_INTERVAL, DCMD_PTPD_STATUS

devctl() in the QNX Neutrino *C Library Reference*

`ptpd`, `ptpd-avb` in the *Utilities Reference*

http://ptpd.sourceforge.net/

DCMD_PTPD_GET_TIME

Get the current time

Synopsis:

```
#include <dcmd_ptpd.h>

#define DCMD_PTPD_GET_TIME   __DIOF(_DCMD_NET, 15, MyTimeInternal )
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/ptpd**
dcmd	DCMD_PTPD_GET_TIME
dev_data_ptr	A pointer to a `MyTimeInternal`
n_bytes	`sizeof(MyTimeInternal)`
dev_info_ptr	NULL

Description:

(QNX Neutrino 7.0 or later) This command gets the current time that's maintained by the Precision Time Protocol.

In order to use this command, you must specify the `-K` option when you start `ptpd` or `ptpd-avb`.

Input:

None.

Output:

A filled-in `MyTimeInternal` structure, which is defined in **<dcmd_ptpd.h>** as follows:

```
typedef struct {
        uint32_t seconds;
        uint32_t nanoseconds;
} MyTimeInternal;
```

Example:

```
MyTimeInternal curr_time;
```

```
if(devctl(fd, DCMD_PTPD_GET_TIME, &curr_time, sizeof(curr_time), NULL) != EOK)
{
   /* Error */
}
```

See also:

DCMD_PTPD_DELAYMS, DCMD_PTPD_DELAYSM, DCMD_PTPD_GET_PDELAY_INTERVAL, DCMD_PTPD_GET_TAI_UTC_OFFSET, DCMD_PTPD_INFO, DCMD_PTPD_SEND_SIGNALING_MSG, DCMD_PTPD_SET_PDELAY_INTERVAL, DCMD_PTPD_STATUS

devctl() in the QNX Neutrino *C Library Reference*

`ptpd`, `ptpd-avb` in the *Utilities Reference*

http://ptpd.sourceforge.net/

DCMD_PTPD_INFO

Get the peer delay and neighbor rate ratio

Synopsis:

```
#include <dcmd_ptpd.h>

#define DCMD_PTPD_INFO   __DIOF(_DCMD_NET, 13, MyPTPDINFOInternal )
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/ptpd**
dcmd	DCMD_PTPD_INFO
dev_data_ptr	A pointer to a `MyPTPDINFOInternal` structure
n_bytes	`sizeof(MyPTPDINFOInternal)`
dev_info_ptr	NULL

Description:

(QNX Neutrino 7.0 or later) This command gets information about the Precision Time Protocol.

In order to use this command, you must specify the `-K` option when you start `ptpd-avb`. Note that `ptpd` doesn't support this command.

Input:

None.

Output:

A filled-in `MyPTPDINFOInternal` structure, which is defined in **<dcmd_ptpd.h>** as follows:

```
typedef struct {
 int64_t peerdelay;
 int32_t neighborrateratio;
 uint8_t neighborrateratio_valid;
} MyPTPDINFOInternal;
```

The members include:

peerdelay

The delay calculated from the timestamps of messages sent between two peers.

neighborrateratio

The neighbor rate ratio, which is used to adjust for the differences in frequencies between the clocks in adjacent devices on the network.

neighborrateratio_valid

Nonzero if the *neighborrateratio* member is valid.

Example:

See DCMD_PTPD_DELAYMS

See also:

DCMD_PTPD_DELAYMS, DCMD_PTPD_DELAYSM, DCMD_PTPD_GET_PDELAY_INTERVAL, DCMD_PTPD_GET_TAI_UTC_OFFSET, DCMD_PTPD_GET_TIME, DCMD_PTPD_SEND_SIGNALING_MSG, DCMD_PTPD_SET_PDELAY_INTERVAL, DCMD_PTPD_STATUS

devctl() in the QNX Neutrino *C Library Reference*

`ptpd-avb` in the *Utilities Reference*

http://ptpd.sourceforge.net/

DCMD_PTPD_SEND_SIGNALING_MSG

Send a signaling message

Synopsis:

```
#include <dcmd_ptpd.h>

#define DCMD_PTPD_SEND_SIGNALING_MSG   __DION(_DCMD_NET, 16 )
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/ptpd**
dcmd	DCMD_PTPD_SEND_SIGNALING_MSG
dev_data_ptr	NULL
n_bytes	0
dev_info_ptr	NULL

Description:

(QNX Neutrino 7.0 or later) This command sends a signaling message for the Precision Time Protocol.

In order to use this command, you must specify the `-K` option when you start `ptpd-avb`. Note that `ptpd` doesn't support this command.

Input:

None.

Output:

None.

Example:

```
if(devctl(fd, DCMD_PTPD_SEND_SIGNALING_MSG, NULL, 0, NULL) != EOK)
{
   /* Error */
}
```

See also:

DCMD_PTPD_DELAYMS, DCMD_PTPD_DELAYSM, DCMD_PTPD_GET_PDELAY_INTERVAL, DCMD_PTPD_GET_TAI_UTC_OFFSET, DCMD_PTPD_GET_TIME, DCMD_PTPD_INFO, DCMD_PTPD_SET_PDELAY_INTERVAL, DCMD_PTPD_STATUS

devctl() in the QNX Neutrino *C Library Reference*

`ptpd-avb` in the *Utilities Reference*

http://ptpd.sourceforge.net/

DCMD_PTPD_SET_PDELAY_INTERVAL

Set the pdelay interval

Synopsis:

```
#include <dcmd_ptpd.h>

#define DCMD_PTPD_SET_PDELAY_INTERVAL __DIOT(_DCMD_NET, 14, MyPDelayIntervalInternal )
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/ptpd**
dcmd	DCMD_PTPD_SET_PDELAY_INTERVAL
dev_data_ptr	A pointer to a `MyPDelayIntervalInternal`
n_bytes	`sizeof(MyPDelayIntervalInternal)`
dev_info_ptr	NULL

Description:

(QNX Neutrino 7.0 or later) This command sets the mean time interval between successive PDELAY_REQ messages in the Precision Time Protocol.

In order to use this command, you must specify the `-K` option when you start `ptpd-avb`. Note that `ptpd` doesn't support this command.

Input:

A `MyPDelayIntervalInternal` that's set to the base-2 logarithm of the interval, in seconds, that you want to use:

```
typedef uint32_t MyPDelayIntervalInternal;
```

Output:

None.

Example:

```
MyPDelayIntervalInternal my_pdelay_interval = 3;

if(devctl(fd, DCMD_PTPD_SET_PDELAY_INTERVAL, &my_pdelay_interval,
          sizeof my_pdelay_interval, NULL) != EOK)
{
```

```
    /* Error */
}
```

See also:

DCMD_PTPD_DELAYMS, DCMD_PTPD_DELAYSM, DCMD_PTPD_GET_PDELAY_INTERVAL, DCMD_PTPD_GET_TAI_UTC_OFFSET, DCMD_PTPD_GET_TIME, DCMD_PTPD_INFO, DCMD_PTPD_SEND_SIGNALING_MSG, DCMD_PTPD_STATUS

devctl() in the QNX Neutrino *C Library Reference*

`ptpd-avb` in the *Utilities Reference*

http://ptpd.sourceforge.net/

DCMD_PTPD_STATUS

Get the error status for the Precision Time Protocol

Synopsis:

```
#include <dcmd_ptpd.h>

#define DCMD_PTPD_STATUS    __DIOF(_DCMD_NET, 12, MyStatusInternal )
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening **/dev/ptpd**
dcmd	DCMD_PTPD_STATUS
dev_data_ptr	A pointer to a `MyStatusInternal`
n_bytes	`sizeof(MyStatusInternal)`
dev_info_ptr	NULL

Description:

(QNX Neutrino 7.0 or later) This command gets the status of the Precision Time Protocol.

In order to use this command, you must specify the `-K` option when you start `ptpd-avb`. Note that `ptpd` doesn't support this command.

Input:

None.

Output:

A filled-in `MyStatusInternal`, which is defined in **<dcmd_ptpd.h>** as follows:

```
typedef uint32_t   MyStatusInternal;
```

The status is one of the following:

PTPD_STATUS_NO_ERROR

No error.

PTPD_STATUS_LINK_DOWN

The link is down.

PTPD_STATUS_GRANDMASTER_FAILURE

The root timing reference (the clock that all other clocks sychronize themselves to) failed.

Example:

See DCMD_PTPD_DELAYMS

See also:

DCMD_PTPD_DELAYMS, DCMD_PTPD_DELAYSM, DCMD_PTPD_GET_PDELAY_INTERVAL, DCMD_PTPD_GET_TAI_UTC_OFFSET, DCMD_PTPD_GET_TIME, DCMD_PTPD_INFO, DCMD_PTPD_SEND_SIGNALING_MSG, DCMD_PTPD_SET_PDELAY_INTERVAL

devctl() in the QNX Neutrino *C Library Reference*

`ptpd-avb` in the *Utilities Reference*

http://ptpd.sourceforge.net/

Chapter 18
DCMD_SDIO_*

This chapter describes the *devctl()* commands that apply to Secure Digital Input/Output cards.

DCMD_SDIO_ATTACH_DEVICE

Attach a device

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_ATTACH_DEVICE   __DIOTF(_DCMD_MISC, 0x12, sdio_dev_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_ATTACH_DEVICE
dev_data_ptr	A pointer to a `sdio_dev_t`
n_bytes	`sizeof(sdio_dev_t)`
dev_info_ptr	NULL

Description:

This command attaches a device. The `sdio_dev_t` structure is defined as follows:

```
typedef struct _sdio_dev_t {
        uint16_t        vid;    /* Vendor ID */
        uint16_t        did;    /* Device ID */
        uint16_t        ccd;    /* Class code */
        int16_t         fun;    /* Function number */
} sdio_dev_t;
```

Input:

Fill in the members as required. You must specify the vendor and device IDs. If you specify SDIO_FUNC_ANY for the function number, you can't specify SDIO_CLASS_ANY for the class code.

Output:

Whichever of the function or class code that you didn't specify is filled in.

See also:

DCMD_SDIO_DETACH_DEVICE

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_CFG_IOMEM

Configure I/O memory

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_CFG_IOMEM             __DIOT (_DCMD_MISC, 0x05, sdio_iomem_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_CFG_IOMEM
dev_data_ptr	A pointer to a `sdio_iomem_t`
n_bytes	`sizeof(sdio_iomem_t)`
dev_info_ptr	NULL

Description:

This command configures I/O memory.

Input:

A filled-in `sdio_iomem_t` structure:

```
typedef struct _sdio_iomem_t {
        uint8_t         func;
        uint8_t         blkmode;
        uint8_t         opcode;
        uint8_t         rsvd;
        uint32_t        address;
        uint32_t        blksz;
} sdio_iomem_t;
```

The members include:

func

The I/O function number.

blkmode

The block mode.

opcode

The command code.

address

The address of the I/O port.

blksize

The block size.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_CLR_IOREG

Clear some bits in an I/O register

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_CLR_IOREG             __DIOT (_DCMD_MISC, 0x04, sdio_ioreg_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_CLR_IOREG
dev_data_ptr	A pointer to a `sdio_ioreg_t`
n_bytes	`sizeof(sdio_ioreg_t)`
dev_info_ptr	NULL

Description:

This command clears bits in an I/O register. The `sdio_ioreg_t` structure is defined as follows:

```
typedef struct _sdio_ioreg_t {
        uint8_t         func;
        uint8_t         val;
        uint8_t         rsvd[2];
        uint32_t        reg;
} sdio_ioreg_t;
```

The members include:

func

The function number

val

The bits that you want to clear in the register.

reg

The register to set.

Input:

Initialize all members of the structure.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_DETACH_DEVICE

Detach a device

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_DETACH_DEVICE __DIOT (_DCMD_MISC, 0x13, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_DETACH_DEVICE
dev_data_ptr	A pointer to an `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command detaches a device.

Input:

The function number for the device that you want to detach.

Output:

None.

See also:

DCMD_SDIO_ATTACH_DEVICE

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_DEV_START

Start a device handler

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_DEV_START    __DIOT (_DCMD_MISC, 0x16, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_DEV_START
dev_data_ptr	A pointer to an `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command starts a device handler.

Input:

The I/O function whose device handler you want to start.

Output:

None.

See also:

DCMD_SDIO_DEV_STOP

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_DEV_STOP

Stop a device handler

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_DEV_STOP              __DIOT (_DCMD_MISC, 0x17, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_DEV_STOP
dev_data_ptr	A pointer to an `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command stops a device handler.

Input:

The I/O function whose device handler you want to stop.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_GET_HCCAP

Get the host controller capabilities

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_GET_HCCAP    __DIOF (_DCMD_MISC, 0x06, uint32_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_GET_HCCAP
dev_data_ptr	A pointer to a `uint32_t`
n_bytes	`sizeof(uint32_t)`
dev_info_ptr	NULL

Description:

This command gets the device's host controller capabilities.

Input:

None.

Output:

The host controller capabilities; a bitwise OR of the following:

- SDIO_HCAP_DMA — the controller supports DMA
- SDIO_HCAP_SG — DMA supports scatter/gather
- SDIO_HCAP_PIO — the controller supports PIO
- SDIO_HCAP_18V — 1.8v is supported
- SDIO_HCAP_30V — 3.0v is supported
- SDIO_HCAP_33V — 3.3v is supported
- SDIO_HCAP_BW1 — 1-bit bus is supported
- SDIO_HCAP_BW4 — 4-bit bus is supported
- SDIO_HCAP_BW8 — 8-bit bus is supported
- SDIO_HCAP_CD_INTR — the host supports card detect interrupt
- SDIO_HCAP_ACMD12 — auto stop command(ACMD12) is supported

- SDIO_HCAP_HS — high-speed devices are supported

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_INTR_DISABLE

Disable a function interrupt

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_INTR_DISABLE  __DIOT (_DCMD_MISC, 0x15, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_INTR_DISABLE
dev_data_ptr	A pointer to an `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command disables a function interrupt.

Input:

The index of the function whose interrupt you want to disable.

Output:

None.

See also:

DCMD_SDIO_INTR_ENABLE

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_INTR_ENABLE

Enable a function interrupt

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_INTR_ENABLE   __DIOT (_DCMD_MISC, 0x14, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_INTR_ENABLE
dev_data_ptr	A pointer to an `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command enables a function interrupt.

Input:

The index of the function whose interrupt you want to enable.

Output:

None.

See also:

DCMD_SDIO_INTR_DISABLE

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_READ_IOREG

Read an I/O register

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_READ_IOREG    __DIOTF(_DCMD_MISC, 0x01, sdio_ioreg_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_READ_IOREG
dev_data_ptr	A pointer to a `sdio_ioreg_t`
n_bytes	`sizeof(sdio_ioreg_t)`
dev_info_ptr	NULL

Description:

This command reads an I/O register. The `sdio_ioreg_t` structure is defined as follows:

```
typedef struct _sdio_ioreg_t {
        uint8_t         func;
        uint8_t         val;
        uint8_t         rsvd[2];
        uint32_t        reg;
} sdio_ioreg_t;
```

The members include:

func

The function number

val

reg

The register to be read.

Input:

Set the *func* and *reg* members.

Output:

The value read from the register.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_SET_IOREG

Set some bits in an I/O register

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_SET_IOREG             __DIOT (_DCMD_MISC, 0x03, sdio_ioreg_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_SET_IOREG
dev_data_ptr	A pointer to a `sdio_ioreg_t`
n_bytes	`sizeof(sdio_ioreg_t)`
dev_info_ptr	NULL

Description:

This command sets an I/O register. The `sdio_ioreg_t` structure is defined as follows:

```
typedef struct _sdio_ioreg_t {
        uint8_t         func;
        uint8_t         val;
        uint8_t         rsvd[2];
        uint32_t        reg;
} sdio_ioreg_t;
```

The members include:

func

The function number

val

The bits that you want to set in the register.

reg

The register to set.

Input:

Initialize all members of the structure.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_SHMEM_FINI

Unmap the device's shared memory object

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_SHMEM_FINI    __DIOT (_DCMD_MISC, 0x11, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_SHMEM_FINI
dev_data_ptr	A pointer to an `int`
n_bytes	`sizeof(int)`
dev_info_ptr	NULL

Description:

This command unmaps the device's shared memory object.

Input:

The I/O function whose shared memory you want to unmap, or -1 for the default function.

Output:

None.

See also:

DCMD_SDIO_SHMEM_INIT

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_SHMEM_INIT

Initialize the device's shared memory object

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_SHMEM_INIT    __DIOT (_DCMD_MISC, 0x10, sdio_shmcfg_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_SHMEM_INIT
dev_data_ptr	A pointer to a `sdio_shmcfg_t`
n_bytes	`sizeof(sdio_shmcfg_t)`
dev_info_ptr	NULL

Description:

This command initializes a shared memory object for the device to use.

Input:

A filled-in `sdio_shmcfg_t` structure:

```
typedef struct _sdio_shmcfg_t {
        int32_t         nsize;          /* total size in bytes */
        int32_t         extra;          /* given by caller: extra size in bytes for
                                           private usage */
        int32_t         ntbd;           /* given by caller: Tx descriptor number */
        int32_t         nrbd;           /* given by caller: Rx descriptor number */
        uint32_t        flag;           /* given by caller: flags */
#define SDIO_SHMEM_TX_MUTEX             (1 << 0)
#define SDIO_SHMEM_TX_SEMAPHORE (1 << 1)
#define SDIO_SHMEM_RX_MUTEX             (1 << 2)
#define SDIO_SHMEM_RX_SEMAPHORE (1 << 3)
        char            sname[16];      /* given by caller: shared memory name */

        int32_t         off_tbd;        /* Tx descriptor offset */
        int32_t         off_rbd;        /* Rx descriptor offset */

        /* private to process */
        int             shmfd;
```

```
        void            *shmvptr;
} sdio_shmcfg_t;
```

Output:

None.

See also:

DCMD_SDIO_SHMEM_FINI

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_VENDOR

Execute a vendor-specific command

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_VENDOR   __DIOTF(_DCMD_MISC, 0x80, int)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_VENDOR
dev_data_ptr	A pointer to an `int`, optionally followed by a character buffer
n_bytes	`sizeof(int)` plus the size of the buffer
dev_info_ptr	NULL, or a pointer to a buffer where the device can store any data to be returned

Description:

This command executes a vendor-specific command.

Input:

The command you want to run.

Output:

The results of the command.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDIO_WRITE_IOREG

Write to an I/O register

Synopsis:

```
#include <mmc/sdio.h>

#define DCMD_SDIO_WRITE_IOREG   __DIOT (_DCMD_MISC, 0x02, sdio_ioreg_t)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDIO_WRITE_IOREG
dev_data_ptr	A pointer to a `sdio_ioreg_t`
n_bytes	`sizeof(sdio_ioreg_t)`
dev_info_ptr	NULL

Description:

This command writes to an I/O register. The `sdio_ioreg_t` structure is defined as follows:

```
typedef struct _sdio_ioreg_t {
        uint8_t         func;
        uint8_t         val;
        uint8_t         rsvd[2];
        uint32_t        reg;
} sdio_ioreg_t;
```

The members include:

func

The function number

val

The value to write.

reg

The register to write the value in.

Input:

Set all members of the structure.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

Chapter 19
DCMD_SDMMC_*

This chapter describes the *devctl()* commands that apply to Secure Digital/MultiMedia Card. This is the newer framework for these cards; see also the DCMD_MMCSD_* commands.

DCMD_SDMMC_ASSD_APDU

Transfer an APDU packet to the device

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_ASSD_APDU    __DIOTF(_DCMD_CAM, _SIM_SDMMC + 8, struct _sdmmc_assd_apdu)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_ASSD_APDU
dev_data_ptr	A pointer to a `SDMMC_ASSD_APDU` that's followed by a buffer
n_bytes	`sizeof(SDMMC_ASSD_APDU)` plus the size of the buffer
dev_info_ptr	NULL

Description:

This command transfers Application Protocol Data Unit packets to an Advanced Security SD.

The `SDMMC_ASSD_APDU` structure is defined as follows:

```
typedef struct _sdmmc_assd_apdu {
        uint32_t        length;
        uint32_t        rsvd[7];
/*      uint8_t         data[ length ];         variable length data */
} SDMMC_ASSD_APDU;
```

The *length* member is the length of the data that follows the structure.

Input:

Fill in the *length* member.

Output:

The buffer that *data* points to is filled with the response, and *length* is set to be its length.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDMMC_ASSD_CONTROL

Control an Advanced Security SD card operation

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_ASSD_CONTROL   __DIOT(_DCMD_CAM, _SIM_SDMMC + 7, struct _sdmmc_assd_control)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_ASSD_CONTROL
dev_data_ptr	A pointer to a `SDMMC_ASSD_CONTROL` structure
n_bytes	`sizeof(SDMMC_ASSD_CONTROL)`
dev_info_ptr	NULL

Description:

This command controls an ASSD operation.

Input:

A filled-in `SDMMC_ASSD_CONTROL` structure:

```
typedef struct _sdmmc_assd_control {
#define SDMMC_AC_OP_START_SUSPEND       0x3
#define SDMMC_AC_OP_CLEAR_SUSPEND       0x2
#define SDMMC_AC_OP_SELECT_RESET        0x1
        uint8_t         operation;

#define SDMMC_AC_SSI_MAX                        0xf
        uint8_t         sec_sys_idx;
        uint16_t        rsvd[11];
} SDMMC_ASSD_CONTROL;
```

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDMMC_ASSD_PROPERTIES

Get Advanced Security SD card properties

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_ASSD_PROPERTIES    __DIOF(_DCMD_CAM, _SIM_SDMMC + 6, struct _sdmmc_assd_proper
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_ASSD_PROPERTIES
dev_data_ptr	A pointer to a `SDMMC_ASSD_PROPERTIES` structure
n_bytes	`sizeof(SDMMC_ASSD_PROPERTIES)`
dev_info_ptr	NULL

Description:

This command gets ASSD properties.

Input:

None.

Output:

A filled-in `SDMMC_ASSD_PROPERTIES` structure:

```
typedef struct _sdmmc_assd_properties {
        uint8_t         assd_version;
        uint8_t         assd_sec_sys_vendor_id;
        uint16_t        assd_sec_sys;

        uint16_t        suspendible_sec_sys;
        uint16_t        sup_auth_alg;
        uint16_t        sup_enc_alg;
        uint16_t        cl_support;

        uint8_t         sec_read_latency;           /* 250ms units */
        uint8_t         sec_write_latency;          /* 250ms units */
        uint8_t         wr_sec_bus_busy;            /* 250ms units */
        uint8_t         ctrl_sys_bus_busy;          /* 250ms units */
```

```
        uint8_t         pmem_support;
        uint8_t         pmem_rd_time;                   /* 100ms units */
        uint8_t         pmem_wr_time;                   /* 250ms units */

        uint8_t         rsvd[17];
} SDMMC_ASSD_PROPERTIES;
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDMMC_ASSD_STATUS

Get the status of an Advanced Security SD card operation

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_ASSD_STATUS   __DIOF(_DCMD_CAM, _SIM_SDMMC + 5, struct _sdmmc_assd_status)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_ASSD_STATUS
dev_data_ptr	A pointer to a `SDMMC_ASSD_STATUS` structure
n_bytes	`sizeof(SDMMC_ASSD_STATUS)`
dev_info_ptr	NULL

Description:

This command gets the status of an ASSD operation.

Input:

None.

Output:

A filled-in `SDMMC_ASSD_STATUS` structure:

```
typedef struct _sdmmc_assd_status {
        uint8_t         assd_state;
        uint8_t         assd_err_state;
        uint8_t         assd_sec_sys_err;
        uint8_t         pmem_state;
        uint8_t         auth_alg;
        uint8_t         enc_alg;
        uint8_t         active_sec_system;
        uint8_t         sec_token_prot;
        uint16_t        read_block_count;
        uint16_t        suspended_sec_sys;
        uint32_t        rsvd[6];
} SDMMC_ASSD_STATUS;
```

The members include:

assd_state

The state of the command:

- ASSD_STATE_SCP — secure command in progress
- ASSD_STATE_SCC — secure command complete
- ASSD_STATE_SCA — secure command aborted

assd_err_state

The error state:

- ASSD_ERR_STATE_NE — no error
- ASSD_ERR_STATE_AE — authorization error
- ASSD_ERR_STATE_ANF — area not found
- ASSD_ERR_STATE_RO — range over
- ASSD_ERR_STATE_CE — condition error

assd_sec_sys_err

pmem_state

auth_alg

enc_alg

active_sec_system

sec_token_prot

read_block_count

suspended_sec_sys

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDMMC_CARD_REGISTER

Read or write a card register

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_CARD_REGISTER   __DIOTF(_DCMD_CAM, _SIM_SDMMC + 4, struct _sdmmc_card_registe
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_CARD_REGISTER
dev_data_ptr	A pointer to a `SDMMC_CARD_REGISTER` structure that's followed by a data buffer
n_bytes	`sizeof(SDMMC_CARD_REGISTER)` plus the size of the data buffer
dev_info_ptr	NULL

Description:

This command reads or writes a card register. It uses two data structures to input and output information:

- A `SDMMC_CARD_REGISTER` structure is used to specify the action, the register type, and the register address (when doing a write).
- A register data buffer is used to store the read data or to specify the write data.

`SDMMC_CARD_REGISTER`:

The `SDMMC_CARD_REGISTER` structure is defined as follows:

```
typedef struct _sdmmc_card_register {
        uint32_t                action;
        uint32_t                type;
        uint32_t                address;
        uint32_t                length;
        uint32_t                rsvd[2];
/*      uint8_t                 data[ length ]; variable length data */
} SDMMC_CARD_REGISTER;
```

The members include:

action

The action to take; currently SDMMC_CR_ACTION_READ and SDMMC_CR_ACTION_WRITE are supported, although the SDMMC_CR_ACTION_WRITE action is only supported for the register type SDMMC_REG_TYPE_EXT_CSD and can only write one byte per *devctl()* call.

type

The register you want to read or write; one of:

- SDMMC_REG_TYPE_CID — card ID
- SDMMC_REG_TYPE_CSD — card-specific data
- SDMMC_REG_TYPE_EXT_CSD — extended card-specific data (MMC devices only)
- SDMMC_REG_TYPE_SCR — SD configuration register (SD devices only)

It's set to one of the following on return:

- SDMMC_CARD_TYPE_UNKNOWN
- SDMMC_CARD_TYPE_MMC
- SDMMC_CARD_TYPE_SD

address

The byte offset of the register to write to.

This field is only supported for the SDMMC_CR_ACTION_WRITE action. Currently only registers of type SDMMC_REG_TYPE_EXT_CSD are supported. ECSD registers with a byte offset greater than or equal to 192 are marked as read-only in the JEDEC eMMC specification, so the value of this field must be less than 192.

length

Not used.

Register data buffer:

The register data buffer is used to store the register data in a read or write operation. If the action field of the SDMMC_CARD_REGISTER structure is set to SDMMC_CR_ACTION_READ, the read data will be returned in the data buffer that follows this structure. If the action field is set to SDMMC_CR_ACTION_WRITE, the write data should be stored in the data buffer that follows the structure.

Currently the SDMMC_CR_ACTION_WRITE action is only supported for the SDMMC_REG_TYPE_EXT_CSD register type. Only one byte can be written to ECSD registers per *devctl()* call.

Data buffer size:

Each register type requires a different data buffer size to store the read or write data without data cropping. You must ensure that your application uses the correct literal value to set the data buffer

size. If the data buffer is too small to fit the full register, the driver will not overflow the buffer but will only return a subset of the read data.

Register Type	Data Buffer Size
SDMMC_REG_TYPE_CID	16
SDMMC_REG_TYPE_CSD	16
SDMMC_REG_TYPE_EXT_CSD (Read)	512
SDMMC_REG_TYPE_EXT_CSD (Write)	1
SDMMC_REG_TYPE_SCR	8

Input:

Set the *action* and *type* members. If using SDMMC_CR_ACTION_WRITE, fill the register data buffer with the value you would like to write to the register.

Output:

The filled-in structure. If using SDMMC_CR_ACTION_READ, the register data buffer will be filled with the register read data.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDMMC_DEVICE_HEALTH

Get information about the device's health

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_DEVICE_HEALTH   __DIOF(_DCMD_CAM, _SIM_SDMMC + 1, union _sdmmc_device_health)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_DEVICE_HEALTH
dev_data_ptr	A pointer to a `SDMMC_DEVICE_HEALTH` structure
n_bytes	`sizeof(SDMMC_DEVICE_HEALTH)`
dev_info_ptr	NULL

Description:

This command gets information about the device's health.

Input:

None.

Output:

A filled-in `SDMMC_DEVICE_HEALTH` structure:

```
typedef struct _sandisk_health {
        uint32_t        mid;                    /* Manufacture ID */
        uint8_t         lifetime;
        uint8_t         nv_avg_pe;              /* NV Cache avg P/E cycle */
        uint8_t         eua_avg_pe;             /* Enhanced User Area avg P/E cycle */
        uint8_t         mlc_avg_pe;             /* MLC avg P/E cycle */
} SANDISK_HEALTH;

typedef struct _samsung_health {
        uint32_t        mid;                    /* Manufacture ID */
        uint32_t        bank0_rsvd_blocks;
        uint32_t        bank1_rsvd_blocks;
        uint32_t        bank2_rsvd_blocks;
        uint32_t        bank3_rsvd_blocks;
        uint32_t        init_bad_blocks;
```

```
        uint32_t        runtime_bad_blocks;
        uint32_t        slc_max_ec;             /* SLC Maximum Erase Count */
        uint32_t        slc_min_ec;             /* SLC Minimum Erase Count */
        uint32_t        slc_avg_ec;             /* SLC Average Erase Count */
        uint32_t        mlc_max_ec;             /* MLC Maximum Erase Count */
        uint32_t        mlc_min_ec;             /* MLC Minimum Erase Count */
        uint32_t        mlc_avg_ec;             /* MLC Average Erase Count */
        uint32_t        max_ec;                 /* Overall Maximum Erase Count */
        uint32_t        min_ec;                 /* Overall Minimum Erase Count */
        uint32_t        avg_ec;                 /* Overall Average Erase Count */
        uint32_t        read_reclaim;
} SAMSUNG_HEALTH;

typedef struct _toshiba_health {                /* supported from v4.41 onwards */
        uint32_t        mid;                    /* Manufacture ID */
        uint32_t        lifetime_total;
        uint32_t        lifetime_rsvd_blk;
        uint32_t        lifetime_avg_pe;
        uint32_t        mlc_max_pe;             /* MLC Maximum P/E cycle range */
        uint32_t        mlc_avg_pe;             /* MLC Average P/E cycle range */
        uint32_t        slc_max_pe;             /* SLC Maximum P/E cycle range */
        uint32_t        slc_avg_pe;             /* SLC Average P/E cycle range */
} TOSHIBA_HEALTH;

typedef union _sdmmc_device_health {
        uint32_t        mid;                    /* Manufacture ID */
        SANDISK_HEALTH  sandisk;
        SAMSUNG_HEALTH  samsung;
        TOSHIBA_HEALTH  toshiba;
        uint8_t                 bytes[512];
} SDMMC_DEVICE_HEALTH;
```

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDMMC_DEVICE_INFO

Get information about the device

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_DEVICE_INFO   __DIOF(_DCMD_CAM, _SIM_SDMMC + 0, struct _sdmmc_device_info)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_DEVICE_INFO
dev_data_ptr	A pointer to a `SDMMC_DEVICE_INFO` structure (see below)
n_bytes	`sizeof(SDMMC_DEVICE_INFO)`
dev_info_ptr	NULL

Description:

This command gets information about the device.

Input:

None.

Output:

A filled-in `SDMMC_DEVICE_INFO` structure:

```
typedef struct _sdmmc_device_info {
        uint32_t                        dtype;
        uint32_t                        flags;
        uint32_t                        mid;
        uint32_t                        oid;
        uint8_t                         pnm[8];
        uint32_t                        prv;
        uint32_t                        psn;
        uint32_t                        month;
        uint32_t                        year;
        uint8_t                         vu[8];
        uint32_t                        rca;
        uint32_t                        spec_vers;
        uint32_t                        spec_rev;
        uint32_t                        security;
```

```
        uint64_t                        caps;
        uint32_t                        dtr;
        uint32_t                        timing;
        uint32_t                        bus_width;
        uint32_t                        sectors;
        uint32_t                        sector_size;
        uint32_t                        super_page_size;
        uint32_t                        native_sector_size;
        uint32_t                        wp_size;
        uint32_t                        erase_size;
        uint32_t                        optimal_trim_size;
        uint32_t                        optimal_read_size;
        uint32_t                        optimal_write_size;
        uint32_t                        speed_class;
        uint32_t                        start_sector;
        uint32_t                        rsvd[34];
} SDMMC_DEVICE_INFO;
```

The members include:

dtype

The device type; either DEV_TYPE_MMC or DEV_TYPE_SD.

flags

The bits include:

- DEV_FLAG_CARD_LOCKED — the card is locked
- DEV_FLAG_WP — the card is write-protected

mid

The manufacturer ID:

- MID_MMC_SANDISK — 0x02
- MID_MMC_SANDISK_2 — 0x45
- MID_MMC_TOSHIBA — 0x11
- MID_MMC_MICRON — 0x13
- MID_MMC_SAMSUNG — 0x15
- MID_MMC_HYNIX — 0x90
- MID_MMC_NUMONYX — 0xFE

oid

The OEM ID.

pnm[8]

The product name.

prv

The product revision number.

psn

The product serial number.

month

The month of manufacture.

year

The year of manufacture.

vu[8]

Vendor-unique information (e.g., the SanDisk firmware revision).

rca

The relative card address.

spec_vers

spec_rev

security

caps

The device's capabilities; a bitwise OR of the following:

- DEV_CAP_HC — high capacity
- DEV_CAP_HS — high speed
- DEV_CAP_HS200 — high speed 200
- DEV_CAP_DDR50 — DDR
- DEV_CAP_UHS — UHS
- DEV_CAP_TRIM — TRIM is supported
- DEV_CAP_SECURE — Secure Purge is supported
- DEV_CAP_SANITIZE — Sanitize is supported
- DEV_CAP_BKOPS — Background operations are supported
- DEV_CAP_CMD23 — CMD23 is supported
- DEV_CAP_SLEEP — Sleep/awake supported
- DEV_CAP_ASSD — ASSD
- DEV_CAP_HPI_CMD12 —
- DEV_CAP_HPI_CMD13 —
- DEV_CAP_DISCARD — Discard supported

dtr

The current data transfer rate.

timing

The current timing; one of:

- TIMING_HS200
- TIMING_SDR104
- TIMING_SDR50
- TIMING_SDR25
- TIMING_SDR12
- TIMING_DDR50
- TIMING_HS
- TIMING_LS

bus_width

The current bus width.

sectors

sector_size

super_page_size

native_sector_size

wp_size

erase_size

optimal_trim_size

optimal_read_size

optimal_write_size

speed_class

The speed class; one of:

- SPEED_CLASS_0 — legacy/non-compliant
- SPEED_CLASS_2 — approximately 2 MB/sec
- SPEED_CLASS_4 — approximately 4 MB/sec
- SPEED_CLASS_6 — approximately 6 MB/sec
- SPEED_CLASS_10 — approximately 10 MB/sec

start_sector

The physical start sector.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDMMC_ERASE

Erase the device

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_ERASE   __DIOTF(_DCMD_CAM, _SIM_SDMMC + 2, struct _sdmmc_erase)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_ERASE
dev_data_ptr	A pointer to a `SDMMC_ERASE` structure (see below)
n_bytes	`sizeof(SDMMC_ERASE)`
dev_info_ptr	NULL

Description:

This command erases the device.

```
typedef struct _sdmmc_erase {
        uint32_t                action;
        uint32_t                rsvd;
        uint64_t                lba;
        uint64_t                nlba;
        uint64_t                rsvd2;
} SDMMC_ERASE;
```

The members include:

action

The action to take; one of:

- SDMMC_ERASE_ACTION_NORMAL
- SDMMC_ERASE_ACTION_SECURE
- SDMMC_ERASE_ACTION_TRIM
- SDMMC_ERASE_ACTION_SECURE_TRIM
- SDMMC_ERASE_ACTION_SECURE_PURGE
- SDMMC_ERASE_ACTION_DISCARD
- SDMMC_ERASE_ACTION_SANITIZE

lba

The logical block address.

nlba

The number of logical blocks.

Input:

Fill in the action and the *lba* and *nlba* members.

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDMMC_LOCK_UNLOCK

Lock or unlock the device

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_LOCK_UNLOCK   __DIOT(_DCMD_CAM, _SIM_SDMMC + 9, struct _sdmmc_lock_unlock)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_LOCK_UNLOCK
dev_data_ptr	A pointer to a `SDMMC_LOCK_UNLOCK` structure
n_bytes	`sizeof(SDMMC_LOCK_UNLOCK)`
dev_info_ptr	NULL

Description:

This command locks or unlocks the device.

Input:

A filled-in `SDMMC_LOCK_UNLOCK` structure:

```
#define SDMMC_LU_PWD_SIZE                       16

typedef struct _sdmmc_lock_unlock {
        uint32_t        action;
        uint32_t        pwd_len;
        uint8_t         pwd[SDMMC_LU_PWD_SIZE];
        uint32_t        rsvd[8];
} SDMMC_LOCK_UNLOCK;
```

The *action* can be one of the following:

- SDMMC_LU_ACTION_ERASE
- SDMMC_LU_ACTION_LOCK
- SDMMC_LU_ACTION_CLR
- SDMMC_LU_ACTION_SET
- SDMMC_LU_ACTION_UNLOCK

Output:

None.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDMMC_PART_INFO

Get or clear information about a partition

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_PART_INFO    __DIOTF(_DCMD_CAM, _SIM_SDMMC + 10, struct _sdmmc_partition_info)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_PART_INFO
dev_data_ptr	A pointer to a `SDMMC_PARTITION_INFO` structure
n_bytes	`sizeof(SDMMC_PARTITION_INFO)`
dev_info_ptr	NULL

Description:

This command gets or clears information about a partition. The `SDMMC_PARTITION_INFO` structure is defined as:

```
typedef struct _sdmmc_partition_info {
        uint32_t                action;
        uint32_t                rsvd;
        uint32_t                ptype;
        uint32_t                pflags;
        uint64_t                start_lba;
        uint64_t                num_lba;
        uint64_t                rc;
        uint64_t                wc;
        uint64_t                tc;
        uint64_t                ec;
        uint64_t                dc;
        uint32_t                rsvd1[64];
} SDMMC_PARTITION_INFO;
```

The members include:

action

The action to take; one of:

- SDMMC_PI_ACTION_GET — get the current values
- SDMMC_PI_ACTION_CLR — get the current values, and then reset the counters

ptype

The partition type; one of the following:

- SDMMC_PTYPE_USER
- SDMMC_PTYPE_BOOT1
- SDMMC_PTYPE_BOOT2
- SDMMC_PTYPE_RPMB
- SDMMC_PTYPE_GP1
- SDMMC_PTYPE_GP2
- SDMMC_PTYPE_GP3
- SDMMC_PTYPE_GP4

pflags

The partition flags; a bitwise OR of the following:

- SDMMC_PFLAG_WP
- SDMMC_PFLAG_ENH
- SDMMC_PFLAG_VIRTUAL

start_lba

The starting logical block address.

num_lba

The number of logical blocks.

rc

The read count (sectors).

wc

The written count (sectors).

tc

The TRIM count (sectors).

ec

The erase count (sectors).

dc

The discard count (sectors).

Input:

Fill in the action.

Output:

The current counts are filled in.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDMMC_PWR_MGNT

Control power management for the device

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_PWR_MGNT    __DIOTF(_DCMD_CAM, _SIM_SDMMC + 11, struct _sdmmc_pwr_mgnt)
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_PWR_MGNT
dev_data_ptr	A pointer to a `SDMMC_PWR_MGNT` structure
n_bytes	`sizeof(SDMMC_PWR_MGNT)`
dev_info_ptr	NULL

Description:

This command controls power management for the device. The `SDMMC_PWR_MGNT` structure is defined as:

```
typedef struct _sdmmc_pwr_mgnt {
#define SDMMC_PM_ACTION_GET             0x00
#define SDMMC_PM_ACTION_SET             0x01
        uint32_t                action;
        uint32_t                rsvd;

        uint64_t                idle_time;   /* time in ms til device enters idle */
        uint64_t                sleep_time;  /* time in ms til device enters sleep */
        uint32_t                rsvd1[16];
} SDMMC_PWR_MGNT;
```

Input:

Set the *action* member to one of the following:

- SDMMC_PM_ACTION_GET — get the current times; you don't need to fill the other members of the structure. The chipset's default values are used as the minimums.
- SDMMC_PM_ACTION_SET — set the times to the given values

The other members of the structure include:

idle_time

The time in milliseconds until the device enters the idle state.

sleep_time

The time in milliseconds until the device goes to sleep.

Output:

For a SDMMC_PM_ACTION_GET action, the *idle_time* and *sleep_time* members are filled with the current times.

See also:

devctl() in the QNX Neutrino *C Library Reference*

DCMD_SDMMC_WRITE_PROTECT

Clear or set write protection

Synopsis:

```
#include <hw/dcmd_sim_sdmmc.h>

#define DCMD_SDMMC_WRITE_PROTECT    __DIOTF(_DCMD_CAM, _SIM_SDMMC + 3, struct _sdmmc_write_protect
```

Arguments to *devctl()*:

Argument	Value
filedes	A file descriptor that you obtained by opening the device
dcmd	DCMD_SDMMC_WRITE_PROTECT
dev_data_ptr	A pointer to a `SDMMC_WRITE_PROTECT` structure
n_bytes	`sizeof(SDMMC_WRITE_PROTECT)`
dev_info_ptr	NULL

Description:

This command sets or clears write protection. The `SDMMC_WRITE_PROTECT` structure is defined as follows:

```
typedef struct _sdmmc_write_protect {
        uint32_t                action;
        uint32_t                mode;
        uint64_t                lba;
        uint64_t                nlba;
        uint64_t                prot;
        uint32_t                rsvd[4];
} SDMMC_WRITE_PROTECT;
```

The members include:

action

The action to take; one of:

- SDMMC_WP_ACTION_CLR — remove write protection
- SDMMC_WP_ACTION_SET — set write protection
- SDMMC_WP_ACTION_PROT — 32 write protection bits (representing 32 write protect groups)

- SDMMC_WP_ACTION_PROT_TYPE — 64 write protection bits (representing 32 write protect groups) where:
  - 0x0 WPG not protected
  - 0x1 WPG temporary WP
  - 0x2 WPG power-on WP
  - 0x3 WPG permanent WP

mode

The mode:

- SDMMC_WP_MODE_PWR_WP_EN — apply power-on period protection

lba

Logical block address

nlba

The number of logical blocks to protect.

prot

Filled in on return with the resulting protection.

Input:

Set the members as appropriate.

Output:

A filled-in structure.

See also:

devctl() in the QNX Neutrino *C Library Reference*

Chapter 20
DCMD_SPI_*

The *devctl()* commands that apply to the Serial Perphiral Interface framework have cover functions:

Command	Cover function
DCMD_SPI_GET_DEVINFO	*spi_getdevinfo()*
DCMD_SPI_GET_DRVINFO	*spi_getdrvinfo()*
DCMD_SPI_SET_CONFIG	*spi_setcfg()*

For more information, see the *SPI (Serial Peripheral Interface) Framework* technote.

Chapter 21
FIO*

This chapter describes the *ioctl()* commands that apply to file I/O.

FIOASYNC

Set or clear asynchronous I/O

Synopsis:

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

#define FIOASYNC        _IOW('f', 125, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	FIOASYNC
Additional argument	A pointer to an `int`

Description:

This command sets or clears asynchronous I/O, depending on the value given. It's implemented as a call to *fcntl()* with the F_GETFL command (which becomes a DCMD_ALL_GETFLAGS *devctl()* command), followed by a call to *fcntl()* with the F_SETFL command (which becomes a DCMD_ALL_SETFLAGS *devctl()* command), setting or clearing O_ASYNC.

The O_ASYNC flag isn't currently supported.

Input:

Zero to clear O_ASYNC, or nonzero to set it.

Output:

See also:

DCMD_ALL_GETFLAGS, DCMD_ALL_SETFLAGS

fcntl(), *ioctl()* in the QNX Neutrino *C Library Reference*

FIOCLEX

Set "close on exec" on a file descriptor

Synopsis:

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

#define FIOCLEX         _IO('f', 1)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	FIOCLEX

Description:

This command sets the "close on exec" flag on a file descriptor. It's implemented as a call to *fcntl()* with a command of F_SETFD and an additional argument of FD_CLOEXEC.

Input:

None.

Output:

None.

See also:

FIONCLEX

fcntl(), *ioctl()* in the QNX Neutrino *C Library Reference*

FIOGETOWN

Get the owner of a file

Synopsis:

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

#define FIOGETOWN       _IOR('f', 123, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	FIOGETOWN
Additional argument	A pointer to an `int`

Description:

This command gets the owner of the file. It's implemented as a call to *fcntl()* with a command of F_GETOWN.

Input:

None.

Output:

If the file descriptor refers to a socket, the process or process group ID that's specified to receive SIGURG signals when out-of-band data is available. Positive values indicate a process ID; negative values—other than -1, which indicates an error—indicate a process group ID.

See also:

FIOSETOWN

fcntl(), *ioctl()* in the QNX Neutrino *C Library Reference*

FIONBIO

Set or clear nonblocking I/O

Synopsis:

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

#define FIONBIO         _IOW('f', 126, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	FIONBIO
Additional argument	A pointer to an `int`

Description:

This command sets or clears non-blocking I/O. It's implemented as a call to *fcntl()* with a command of F_GETFL (which becomes a DCMD_ALL_GETFLAGS *devctl()* command), followed by a call to *fcntl()* with a command of F_SETFL (which becomes a DCMD_ALL_SETFLAGS *devctl()* command), setting or clearing O_NONBLOCK.

Input:

Zero to clear O_NONBLOCK, or nonzero to set it.

Output:

None.

See also:

DCMD_ALL_GETFLAGS, DCMD_ALL_SETFLAGS

fcntl(), *ioctl()* in the QNX Neutrino *C Library Reference*

FIONCLEX

Clear "close on exec" on a file descriptor

Synopsis:

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

#define FIONCLEX        _IO('f', 2)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	FIONCLEX

Description:

This command clears the "close on exec" flag on a file descriptor. It's implemented as a call to *fcntl()* with a command of F_SETFD and an additional argument of ~FD_CLOEXEC.

Input:

None.

Output:

None.

See also:

FIOCLEX

fcntl(), *ioctl()* in the QNX Neutrino *C Library Reference*

FIONREAD

Determine the number of characters waiting to be read

Synopsis:

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

#define FIONREAD        _IOR('f', 127, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	FIONREAD
Additional argument	A pointer to an `int`

Description:

This command determine the number of characters waiting to be read. It's also implemented as the DCMD_CHR_ISCHARS *devctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcischars()* cover function.

Input:

None.

Output:

The number of characters.

See also:

DCMD_CHR_ISCHARS, TIOCOUTQ

ioctl() in the QNX Neutrino *C Library Reference*

FIONSPACE

Get the amount of space in the send queue

Synopsis:

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

#define FIONSPACE       _IOR('f', 120, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	FIONSPACE
Additional argument	A pointer to an `int`

Description:

This command gets the amount of space in the send queue.

Input:

None.

Output:

The amount of space.

See also:

ioctl() in the QNX Neutrino *C Library Reference*

FIONWRITE

Get the number of bytes outstanding in the send queue.

Synopsis:

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

#define FIONWRITE       _IOR('f', 121, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	FIONWRITE
Additional argument	A pointer to an `int`

Description:

This command gets the number of bytes outstanding in the send queue.

Input:

None.

Output:

The number of bytes.

See also:

ioctl() in the QNX Neutrino *C Library Reference*

FIOSETOWN

Set the owner of a file

Synopsis:

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

#define FIOSETOWN       _IOW('f', 124, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	FIOSETOWN
Additional argument	A pointer to an `int`

Description:

This command sets the owner of the file. It's implemented as a call to *fcntl()* with a command of F_SETOWN.

Input:

If the file descriptor refers to a socket, the process or process group ID that you want to receive SIGURG signals when out-of-band data is available. Positive values indicate a process ID; negative values, other than -1, indicate a process group ID.

Output:

None.

See also:

FIOGETOWN

fcntl(), *ioctl()* in the QNX Neutrino *C Library Reference*

Chapter 22
TC*

This chapter describes the *ioctl()* commands that apply to terminals.

TCFLSH

Flush buffers

Synopsis:

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

#define TCFLSH          _IOW('T',  7, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TCFLSH
Additional argument	A pointer to an `integer`

Description:

This command flushes the input and/or output stream associated with the file descriptor.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcflush()* cover function.

Input:

The queue selector; one of:

- 0 — discard all data that's received, but not yet read, on the device associated with *fildes*.
- 1 — discard all data that's written, but not yet transmitted, on the device associated with *fildes*.
- 2 — discard all data that's written, but not yet transmitted, as well as all data that's received, but not yet read, on the device associated with *fildes*.

Output:

None.

See also:

TIOCFLUSH

ioctl() in the QNX Neutrino *C Library Reference*

TCGETA

Get the terminal properties in a `termio` structure

Synopsis:

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

#define TCGETA          _IOR('T', 1, struct termio)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TCGETA
Additional argument	A pointer to a `struct termio`

Description:

This command gets the current terminal control settings of a device.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcgetattr()* cover function.

The TCGETS *ioctl()* command is similar, but it uses a `struct termios` instead of a `struct termio`.

Input:

None.

Output:

A filled-in `termio` structure.

See also:

TIOCGETA, TCGETS, TCSETA, TCSETAF, TCSETAW

ioctl(), in the QNX Neutrino *C Library Reference*

TCGETS

Get the terminal properties in a `termios` structure

Synopsis:

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

#define TCGETS          _IOR('T', 13, struct termios)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TCGETS
Additional argument	A pointer to a `struct termios`

Description:

This command gets the current terminal control settings of a device. It's similar to the TIOCGETA *ioctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcgetattr()* cover function.

The TCGETA *ioctl()* command is similar, but it uses a `struct termio` instead of a `struct termios`.

Input:

None.

Output:

A filled-in `termios` structure.

See also:

TCGETA, TCSETS, TCSETSF, TCSETSW, TIOCGETA

ioctl(), `termios` in the QNX Neutrino *C Library Reference*

TCSBRK

Assert a break condition over a communications line

Synopsis:

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

#define TCSBRK          _IOW('T',  5, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TCSBRK
Additional argument	A pointer to an `int`

Description:

This command asserts a break condition over a communications line for the given number of milliseconds. You should use the *tcsendbreak()* cover function instead of this command.

Input:

The number of milliseconds.

Output:

None.

See also:

ioctl(), *tcsendbreak()* in the QNX Neutrino *C Library Reference*

TCSETA, TCSETAF, TCSETAW

Set terminal properties from a `termio` structure

Synopsis:

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

#define TCSETA          _IOW('T', 2, struct termio)
#define TCSETAW         _IOW('T', 3, struct termio)
#define TCSETAF         _IOW('T', 4, struct termio)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TCSETA, TCSETAF, or TCSETAW
Additional argument	A pointer to a `struct termio`

Description:

These commands change the current terminal control settings of a device:

- TCSETA — make the change immediately
- TCSETAF — Don't make the change until all currently written data has been transmitted, at which point any received but unread data is also discarded.
- TCSETAW — don't make the change until all currently written data has been transmitted.

These commands are for internal use, and you shouldn't use them directly. Instead use the *tcsetattr()* cover function.

The TCSETS, TCSETSF, *and* TCSETSW *ioctl()* commands are similar, but they use a `struct termios` instead of a `struct termio`.

Input:

A `termio` structure.

Output:

None.

See also:

TCGETA, TCSETS, TCSETSF, TCSETSW, TIOCSETA, TIOCSETAF, TIOCSETAW

ioctl(), in the QNX Neutrino *C Library Reference*

TCSETS, TCSETSF, TCSETSW

Set terminal properties from a `termios` structure

Synopsis:

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

#define TCSETS          _IOW('T', 14, struct termios)
#define TCSETSW         _IOW('T', 15, struct termios)
#define TCSETSF         _IOW('T', 16, struct termios)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TCSETS, TCSETSF, or TCSETSW
Additional argument	A pointer to a `struct termios`

Description:

These commands change the current terminal control settings of a device:

- TCSETS — make the change immediately
- TCSETSF — Don't make the change until all currently written data has been transmitted, at which point any received but unread data is also discarded.
- TCSETSW — don't make the change until all currently written data has been transmitted.

They're similar to the following *ioctl()* commands:

TC* command	TIOC* command
TCSETS	TIOCSETA
TCSETSF	TIOCSETAF
TCSETSW	TIOCSETAW

These commands are for internal use, and you shouldn't use them directly. Instead use the *tcsetattr()* cover function.

The TCSETA, TCSETAF, *and* TCSETAW *ioctl()* commands are similar, but they use a `struct termio` instead of a `struct termios`.

Input:

A `termios` structure.

Output:

None.

See also:

TCGETS, TCSETA, TCSETAF, TCSETAW, TIOCSETA, TIOCSETAF, TIOCSETAW

ioctl(), `termios` in the QNX Neutrino *C Library Reference*

TCXONC

Perform a flow-control operation on a data stream

Synopsis:

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

#define TCXONC          _IOW('T',  6, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TCXONC
Additional argument	A pointer to an `int`

Description:

This command performs a flow-control operation on the data stream associated with the file descriptor. It's also implemented as the DCMD_CHR_TCFLOW *devctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcflow()* cover function.

Input:

Output:

See also:

DCMD_CHR_TCFLOW

ioctl(), *tcflow()* in the QNX Neutrino *C Library Reference*

Chapter 23
TIOC*

This chapter describes the *ioctl()* commands that apply to terminals.

TIOCCBRK

Clear the break bit

Synopsis:

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

#define TIOCCBRK        _IO('t', 122)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCCBRK

Description:

This command clears the break bit. It's implemented as a call to *tcgetattr()* to get the current settings, followed by a call to *tcsetattr()* to clear BRKINT and set IGNBRK.

The TIOCSBRK command, which sets the break bit, isn't currently implemented.

Input:

None.

Output:

None.

See also:

ioctl() in the QNX Neutrino *C Library Reference*

TIOCCDTR

Clear "data terminal ready"

Synopsis:

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

#define TIOCCDTR        _IO('t', 120)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCCDTR

Description:

This command clears "data terminal ready."

Input:

None.

Output:

None.

See also:

DCMD_CHR_LINESTATUS, DCMD_CHR_SERCTL, TIOCSDTR

ioctl() in the QNX Neutrino *C Library Reference*

TIOCDRAIN

Wait until output has drained

Synopsis:

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

#define TIOCDRAIN       _IO('t',  94)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCDRAIN

Description:

It's also implemented as the DCMD_CHR_TCDRAIN *devctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcdrain()* cover function.

Input:

None.

Output:

None.

See also:

DCMD_CHR_TCDRAIN

ioctl() in the QNX Neutrino *C Library Reference*

TIOCEXCL, TIOCSINUSE

Set exclusive use of a terminal

Synopsis:

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

#define TIOCSINUSE      TIOCEXCL

#define TIOCEXCL        _IO('t', 13)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCEXCL or TIOCSINUSE

Description:

These commands set exclusive use of a terminal.

Input:

None.

Output:

None.

See also:

TIOCNXCL

ioctl() in the QNX Neutrino *C Library Reference*

TIOCFLUSH

Flush buffers

Synopsis:

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

#define TIOCFLUSH       _IOW('t', 16, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCFLUSH
Additional argument	A pointer to an `integer`

Description:

This command flushes the input and/or output stream associated with the file descriptor. It's also implemented as the DCMD_CHR_TCFLUSH *devctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcflush()* cover function.

Input:

The queue selector; one of:

- FREAD — discard all data that's received, but not yet read, on the device associated with *fildes*.
- FWRITE — discard all data that's written, but not yet transmitted, on the device associated with *fildes*.
- 0 — discard all data that's written, but not yet transmitted, as well as all data that's received, but not yet read, on the device associated with *fildes*.

Output:

None.

See also:

DCMD_CHR_TCFLUSH, TCFLSH

ioctl() in the QNX Neutrino *C Library Reference*

TIOCGETA

Get the terminal properties in a `termios` structure

Synopsis:

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

#define TIOCGETA        _IOR('t', 19, struct termios)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCGETA
Additional argument	A pointer to a `struct termios`

Description:

This command gets the current terminal control settings of a device. It's also implemented as the DCMD_CHR_TCGETATTR *devctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcgetattr()* cover function.

Input:

None.

Output:

A filled-in `termios` structure.

See also:

DCMD_CHR_TCGETATTR, TCGETA, TCGETS, TIOCSETA, TIOCSETAF, TIOCSETAW

ioctl(), `termios` in the QNX Neutrino *C Library Reference*

TIOCGETC

Get a terminal's special characters

Synopsis:

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

#define TIOCGETC        _IOR('t', 18, struct tchars)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCGETC
Additional argument	A pointer to a `struct tchars`

Description:

This command gets a terminal's special characters.

Input:

None.

Output:

A filled-in `struct tchars` (defined in **<sgtty.h>**):

```
struct tchars {
    char t_intrc;   /* interrupt */
    char t_quitc;   /* quit */
    char t_startc;  /* start output */
    char t_stopc;   /* stop output */
    char t_eofc;    /* end-of-file */
    char t_brkc;    /* input delimiter (like nl) */
};
```

See also:

TIOCSETC

ioctl() in the QNX Neutrino *C Library Reference*

TIOCGETP

Get a terminal's parameters

Synopsis:

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

#define TIOCGETP        _IOR('t', 8, struct sgttyb)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCGETP
Additional argument	A pointer to a `struct sgttyb`

Description:

This command gets a terminal's parameters.

Input:

None.

Output:

A filled-in `struct sgttyb` (defined in **<sgtty.h>**):

```
struct sgttyb {
    char sg_ispeed; /* input speed */
    char sg_ospeed; /* output speed */
    char sg_erase;  /* erase character */
    char sg_kill;   /* kill character */
    int  sg_flags;  /* mode flags */
};
```

The mode flags, which are defined in **<sys/termio.h>** include the following:

- TANDEM — send stopc when the output queue is full.
- CBREAK — half-cooked mode.
- LCASE — simulate lower case.
- CRMOD — map `\r` to `\r\n` on output (ONLCR & ICRNL.
- RAW — no I/O processing.
- ODDP — get/send odd parity.
- EVENP — get/send even parity.

- ANYP — get any parity/send none.

as well as the following, which is defined in **<termios.h>**:

- ECHO — enable echo.

See also:

TIOCSETN, TIOCSETP

ioctl() in the QNX Neutrino *C Library Reference*

TIOCGLTC

Get the local special characters

Synopsis:

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

#define TIOCGLTC        _IOR('t', 116, struct ltchars)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCGLTC
Additional argument	A pointer to a `struct ltchars`

Description:

This command gets the local special characters.

Input:

None.

Output:

A filled-in `struct ltchars` (defined in **<sgtty.h>**):

```
struct ltchars {
    char    t_suspc;    /* stop process signal */
    char    t_dsuspc;   /* delayed stop process signal */
    char    t_rprntc;   /* reprint line */
    char    t_flushc;   /* flush output (toggles) */
    char    t_werasc;   /* word erase */
    char    t_lnextc;   /* literal next character */
};
```

See also:

TIOCSLTC

ioctl() in the QNX Neutrino *C Library Reference*

TIOCGETPGRP, TIOCGPGRP

Get the process group ID associated with a device

Synopsis:

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

#define TIOCGPGRP       _IOR('t', 119, int)
#define TIOCGETPGRP     _IOR('t', 131, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCGETPGRP or TIOCGPGRP
Additional argument	A pointer to an `int`

Description:

These commands get the process group ID associated with a device. They're also implemented as the DCMD_CHR_TCGETPGRP *devctl()* command.

These commands are for internal use, and you shouldn't use them directly. Instead use the *tcgetpgrp()* cover function.

Input:

None.

Output:

The process group ID.

See also:

DCMD_CHR_TCGETPGRP, TIOCSETPGRP, TIOCSPGRP

ioctl() in the QNX Neutrino *C Library Reference*

TIOCGSIZE, TIOCGWINSZ

Get the size of a character device

Synopsis:

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

#define TIOCGSIZE       TIOCGWINSZ
#define TIOCGWINSZ      _IOR('t', 104, struct winsize)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCGWINSZ
Additional argument	A pointer to a `struct winsize`

Description:

This command gets the size of a character device (also known as the window size). It's also implemented as the DCMD_CHR_GETSIZE *devctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcgetsize()* cover function.

Input:

None.

Output:

The `winsize` structure is defined in **<sys/ioctl.h>** as follows:

```
struct winsize {
        unsigned short  ws_row;
        unsigned short  ws_col;
        unsigned short  ws_xpixel;
        unsigned short  ws_ypixel;
};
```

See also:

DCMD_CHR_GETSIZE, TIOCSSIZE, TIOCSWINSZ

ioctl() in the QNX Neutrino *C Library Reference*

TIOCHPCL

Hang up on last close

Synopsis:

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

#define TIOCHPCL        _IO('t',  2)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCHPCL

Description:

This command sets the “hang up on last close” flag. It's implemented as a call to *tcgetattr()* to get the current settings, followed by a call to *tcsetattr()* to set HUPCL.

Input:

None.

Output:

None.

See also:

ioctl() in the QNX Neutrino *C Library Reference*

TIOCLGET

Get the local modes

Synopsis:

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

#define TIOCLGET        _IOR('t', 124, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCLGET
Additional argument	A pointer to an `int`

Description:

This command gets the current local modes. It's implemented as a call to *tcgetattr()* to get the current settings.

Input:

None.

Output:

The current bit settings; a combination of zero or more of the following:

ECHO

Enable echo.

ECHOE

Echo ERASE as destructive backspace.

ECHOK

Echo KILL as a line erase.

ECHONL

Echo `\n`, even if ECHO is off.

ICANON

Canonical input mode (line editing enabled).

IEXTEN

QNX Neutrino extensions to POSIX are enabled.

ISIG

Enable signals.

NOFLSH

Disable flush after interrupt, quit, or suspend.

TOSTOP

Send SIGTTOU for background output.

These are the same bits that appear in the *c_lflag* member of the `termios` structure.

See also:

TIOCLSET

ioctl(), `termios` in the QNX Neutrino *C Library Reference*

TIOCLSET

Set the entire local mode word

Synopsis:

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

#define TIOCLSET        _IOW('t', 125, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCLSET
Additional argument	A pointer to an `int`

Description:

This command sets the entire local mode word. It's implemented as a call to *tcgetattr()* to get the current settings, followed by a call to *tcsetattr()* to replace the local modes with the bits given.

Input:

The new bit settings; a combination of zero or more of the following:

ECHO

Enable echo.

ECHOE

Echo ERASE as destructive backspace.

ECHOK

Echo KILL as a line erase.

ECHONL

Echo `\n`, even if ECHO is off.

ICANON

Canonical input mode (line editing enabled).

IEXTEN

QNX Neutrino extensions to POSIX are enabled.

ISIG

Enable signals.

NOFLSH

Disable flush after interrupt, quit, or suspend.

TOSTOP

Send SIGTTOU for background output.

These are the same bits that appear in the *c_lflag* member of the `termios` structure.

Output:

None.

See also:

TIOCLGET

ioctl(), `termios` in the QNX Neutrino *C Library Reference*

TIOCMBIC

Clear specific modem bits

Synopsis:

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

#define TIOCMBIC        _IOW('t', 107, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCMBIC
Additional argument	A pointer to an `int`

Description:

This command clears specific bits in the modem control registers on a tty device.

Input:

The bits that you want to clear; zero of more of the following:

- TIOCM_DTR — data terminal ready.
- TIOCM_RTS — request to send.

Output:

None.

See also:

TIOCMBIS, TIOCMGET, TIOCMSET

ioctl() in the QNX Neutrino *C Library Reference*

TIOCMBIS

Set specific modem bits

Synopsis:

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

#define TIOCMBIS        _IOW('t', 108, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCMBIS
Additional argument	A pointer to an `int`

Description:

This command sets specific bits in the modem control registers on a tty device.

Input:

The bits that you want to set; zero of more of the following:

- TIOCM_DTR — data terminal ready.
- TIOCM_RTS — request to send.

Output:

None.

See also:

TIOCMBIC, TIOCMGET, TIOCMSET

ioctl() in the QNX Neutrino *C Library Reference*

TIOCMGET

Get all modem bits

Synopsis:

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

#define TIOCMGET        _IOR('t', 106, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCMGET
Additional argument	A pointer to an `int`

Description:

This command gets all modem status bits for the terminal device. It's also implemented as the DCMD_CHR_LINESTATUS *devctl()* command.

Input:

None.

Output:

The modem bits, a combination of the following:

- TIOCM_DTR — data terminal ready.
- TIOCM_RTS — request to send.
- TIOCM_CTS — clear to send.
- TIOCM_DSR — data set ready.
- TIOCM_RI or TIOCM_RNG — ring.
- TIOCM_CAR or TIOCM_CD — carrier detect.

See also:

DCMD_CHR_LINESTATUS, TIOCMBIC, TIOCMBIS, TIOCMSET

ioctl() in the QNX Neutrino *C Library Reference*

TIOCMSET

Set all modem bits

Synopsis:

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

#define TIOCMSET        _IOW('t', 109, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCMSET
Additional argument	A pointer to an `int`

Description:

This command sets all modem status bits for the terminal device.

Input:

The modem bits, a combination of the following:

- TIOCM_DTR — data terminal ready.
- TIOCM_RTS — request to send.

Output:

None.

See also:

TIOCMBIC, TIOCMBIS, TIOCMGET

ioctl() in the QNX Neutrino *C Library Reference*

TIOCNOTTY

Make a terminal not be the controlling terminal

Synopsis:

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

#define TIOCNOTTY       _IO('t', 113)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCNOTTY

Description:

This command calls *tcsetsid()* to make the terminal not be the controlling terminal for the process.

Input:

None.

Output:

None.

See also:

TIOCSCTTY

ioctl(), *tcsetsid()* in the QNX Neutrino *C Library Reference*

TIOCNXCL

Reset exclusive use of a terminal

Synopsis:

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

#define TIOCNXCL        _IO('t', 14)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCNXCL

Description:

This command resets the exclusive use of a terminal.

Input:

None.

Output:

None.

See also:

TIOCEXCL, TIOCSINUSE

ioctl() in the QNX Neutrino *C Library Reference*

TIOCOUTQ

Get the output queue size

Synopsis:

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

#define TIOCOUTQ        _IOR('t', 115, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCOUTQ
Additional argument	A pointer to an `int`

Description:

This command gets the output queue size. It's also implemented as the DCMD_CHR_OSCHARS *devctl()* command.

Input:

None.

Output:

The number of characters.

See also:

DCMD_CHR_OSCHARS, FIONREAD

ioctl() in the QNX Neutrino *C Library Reference*

TIOCPKT

Set or clear packet mode

Synopsis:

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

#define TIOCPKT         _IOW('t', 112, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCPKT
Additional argument	A pointer to an `int`

Description:

This command sets or clears packet mode.

Input:

A combination of the following bits:

```
#define         TIOCPKT_DATA            0x00            /* data packet */
#define         TIOCPKT_FLUSHREAD       0x01            /* flush packet */
#define         TIOCPKT_FLUSHWRITE      0x02            /* flush packet */
#define         TIOCPKT_STOP            0x04            /* stop output */
#define         TIOCPKT_START           0x08            /* start output */
#define         TIOCPKT_NOSTOP          0x10            /* no more ^S, ^Q */
#define         TIOCPKT_DOSTOP          0x20            /* now do ^S ^Q */
#define         TIOCPKT_IOCTL           0x40            /* state change of pty driver */
```

Output:

None.

See also:

ioctl() in the QNX Neutrino *C Library Reference*

TIOCSCTTY

Make a terminal the controlling terminal

Synopsis:

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

#define TIOCSCTTY       _IO('t',  97)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCSCTTY

Description:

This command calls *tcsetsid()* to make the terminal the controlling terminal for the process.

Input:

None.

Output:

None.

See also:

TIOCNOTTY

ioctl(), *tcsetsid()* in the QNX Neutrino *C Library Reference*

TIOCSDTR

Set "data terminal ready"

Synopsis:

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

#define TIOCSDTR        _IO('t', 121)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCSDTR

Description:

This command sets "data terminal ready."

Input:

None.

Output:

None.

See also:

DCMD_CHR_LINESTATUS, DCMD_CHR_SERCTL, TIOCCDTR

ioctl() in the QNX Neutrino *C Library Reference*

TIOCSETA, TIOCSETAF, TIOCSETAW

Set terminal properties from a `termios` structure

Synopsis:

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

#define TIOCSETA        _IOW('t', 20, struct termios)   /* set termios struct */
#define TIOCSETAW       _IOW('t', 21, struct termios)   /* drain output, set */
#define TIOCSETAF       _IOW('t', 22, struct termios)   /* drn out, fls in, set */
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCSETA, TIOCSETAF, or TIOCSETAW
Additional argument	A pointer to a `struct termios`

Description:

These commands change the current terminal control settings of a device:

- TIOCSETA — make the change immediately
- TIOCSETAF — Don't make the change until all currently written data has been transmitted, at which point any received but unread data is also discarded.
- TIOCSETAW — don't make the change until all currently written data has been transmitted.

They're also implemented as *devctl()* commands:

ioctl() command	*devctl()* command
TIOCSETA	DCMD_CHR_TCSETATTR
TIOCSETAF	DCMD_CHR_TCSETATTRF
TIOCSETAW	DCMD_CHR_TCSETATTRD

These commands are for internal use, and you shouldn't use them directly. Instead use the *tcsetattr()* cover function.

Input:

A `termios` structure.

Output:

None.

See also:

DCMD_CHR_TCSETATTR, DCMD_CHR_TCSETATTRD, DCMD_CHR_TCSETATTRF, TIOCGETA

ioctl(), `termios` in the QNX Neutrino *C Library Reference*

TIOCSETC

Set a terminal's special characters

Synopsis:

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

#define TIOCSETC        _IOW('t', 17, struct tchars)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCSETC
Additional argument	A pointer to a `struct tchars`

Description:

This command sets a terminal's special characters.

Input:

A filled-in `struct tchars` (defined in **<sgtty.h>**):

```
struct tchars {
    char t_intrc;   /* interrupt */
    char t_quitc;   /* quit */
    char t_startc;  /* start output */
    char t_stopc;   /* stop output */
    char t_eofc;    /* end-of-file */
    char t_brkc;    /* input delimiter (like nl) */
};
```

Output:

None.

See also:

TIOCGETC

ioctl() in the QNX Neutrino *C Library Reference*

TIOCSETN, TIOCSETP

Set a terminal's parameters

Synopsis:

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

#define TIOCSETP        _IOW('t', 9, struct sgttyb)
#define TIOCSETN        _IOW('t',10, struct sgttyb)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCSETN or TIOCSETP
Additional argument	A pointer to a `struct sgttyb`

Description:

These commands set a terminal's parameters. TIOCSETP flushes the tty; TIOCSETN doesn't.

Input:

A filled-in `struct sgttyb` (defined in **<sgtty.h>**):

```
struct sgttyb {
    char sg_ispeed; /* input speed */
    char sg_ospeed; /* output speed */
    char sg_erase;  /* erase character */
    char sg_kill;   /* kill character */
    int  sg_flags;  /* mode flags */
};
```

The mode flags, which are defined in **<sys/termio.h>** include the following:

- TANDEM — send stopc when the output queue is full.
- CBREAK — half-cooked mode.
- LCASE — simulate lower case.
- CRMOD — map `\r` to `\r\n` on output (ONLCR & ICRNL.
- RAW — no I/O processing.
- ODDP — get/send odd parity.
- EVENP — get/send even parity.
- ANYP — get any parity/send none.

as well as the following, which is defined in **<termios.h>**:

- ECHO — enable echo.

Output:

None.

See also:

TIOCGETP

ioctl() in the QNX Neutrino *C Library Reference*

TIOCSLTC

Set the local special characters

Synopsis:

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

#define TIOCSLTC        _IOW('t', 117, struct ltchars)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCSLTC
Additional argument	A pointer to a `struct ltchars`

Description:

This command sets the local special characters.

Input:

A filled-in `struct ltchars` (defined in **<sgtty.h>**):

```
struct ltchars {
    char    t_suspc;    /* stop process signal */
    char    t_dsuspc;   /* delayed stop process signal */
    char    t_rprntc;   /* reprint line */
    char    t_flushc;   /* flush output (toggles) */
    char    t_werasc;   /* word erase */
    char    t_lnextc;   /* literal next character */
};
```

Output:

None.

See also:

TIOCGLTC

ioctl() in the QNX Neutrino *C Library Reference*

TIOCSETPGRP, TIOCSPGRP

Set the process group ID associated with a device

Synopsis:

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

#define TIOCSPGRP       _IOW('t', 118, int)
#define TIOCSETPGRP     _IOW('t', 130, int)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCSETPGRP, TIOCSPGRP
Additional argument	A pointer to an `int`

Description:

These commands set the process group ID associated with a device. They're also implemented as the DCMD_CHR_TCSETPGRP *devctl()* command.

These commands are for internal use, and you shouldn't use them directly. Instead use the *tcsetpgrp()* cover function.

Input:

The process group ID.

Output:

None.

See also:

DCMD_CHR_TCSETPGRP, TIOCGETPGRP, TIOCGPGRP

ioctl() in the QNX Neutrino *C Library Reference*

TIOCSTART

Start output

Synopsis:

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

#define TIOCSTART       _IO('t',  110)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCSTART

Description:

This command starts output. It's like pressing **Ctrl–Q**.

Input:

None.

Output:

None.

See also:

TIOCSTOP

ioctl() in the QNX Neutrino *C Library Reference*

TIOCSTI

Simulate terminal input

Synopsis:

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

#define TIOCSTI         _IOW('t', 114, char)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCSTI
Additional argument	A pointer to a `char`

Description:

This command simulates terminal input by calling *tcinject()* to inject a single character into the stream.

Input:

The character you want to inject.

Output:

None.

See also:

ioctl(), *tcinject()* in the QNX Neutrino *C Library Reference*

TIOCSTOP

Stop output

Synopsis:

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

#define TIOCSTOP        _IO('t',  111)
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCSTOP

Description:

This command stops output. It's like pressing **Ctrl–S**.

Input:

None.

Output:

None.

See also:

TIOCSTART

ioctl() in the QNX Neutrino *C Library Reference*

TIOCSSIZE, TIOCSWINSZ

Set the size of a character device

Synopsis:

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

#define TIOCSSIZE       TIOCSWINSZ
#define TIOCSWINSZ      _IOW('t', 103, struct winsize)  /* set window size */
```

Arguments to *ioctl()*:

Argument	Value
fd	A file descriptor that you obtained by opening the device
request	TIOCSWINSZ
Additional argument	A pointer to a `struct winsize`

Description:

This command sets the size of a character device (also known as the window size). It's also implemented as the DCMD_CHR_SETSIZE *devctl()* command.

This command is for internal use, and you shouldn't use it directly. Instead use the *tcsetsize()* cover function.

Input:

None.

Output:

The `winsize` structure is defined in **<sys/ioctl.h>** as follows:

```
struct winsize {
        unsigned short  ws_row;
        unsigned short  ws_col;
        unsigned short  ws_xpixel;
        unsigned short  ws_ypixel;
};
```

See also:

DCMD_CHR_SETSIZE, TIOCGSIZE, TIOCGWINSZ

ioctl() in the QNX Neutrino *C Library Reference*

Index

A

B

C

D

E

F

O

P

R

S

T

U

V

W