NAME

st − SCSI tape device

SYNOPSIS

#include <sys/mtio.h>

int ioctl(int *fd***, int** *request* **[, (void *)***arg3***]);**
int ioctl(int *fd***, MTIOCTOP, (struct mtop *)***mt_cmd***);**
int ioctl(int *fd***, MTIOCGET, (struct mtget *)***mt_status***);**
int ioctl(int *fd***, MTIOCPOS, (struct mtpos *)***mt_pos***);**

DESCRIPTION

The **st** driver provides the interface to a variety of SCSI tape devices. Currently, the driver takes control of all detected devices of type "sequential-access". The **st** driver uses major device number 9.

Each device uses eight minor device numbers. The lowermost five bits in the minor numbers are assigned sequentially in the order of detection. In the 2.6 kernel, the bits above the eight lowermost bits are concatenated to the five lowermost bits to form the tape number. The minor numbers can be grouped into two sets of four numbers: the principal (auto-rewind) minor device numbers, *n*, and the "no-rewind" device numbers, (*n* + 128). Devices opened using the principal device number will be sent a **REWIND** command when they are closed. Devices opened using the "no-rewind" device number will not. (Note that using an auto-rewind device for positioning the tape with, for instance, mt does not lead to the desired result: the tape is rewound after the mt command and the next command starts from the beginning of the tape).

Within each group, four minor numbers are available to define devices with different characteristics (block size, compression, density, etc.) When the system starts up, only the first device is available. The other three are activated when the default characteristics are defined (see below). (By changing compile-time constants, it is possible to change the balance between the maximum number of tape drives and the number of minor numbers for each drive. The default allocation allows control of 32 tape drives. For instance, it is possible to control up to 64 tape drives with two minor numbers for different options.)

Devices are typically created by:

```
mknod −m 666 /dev/st0 c 9 0
mknod −m 666 /dev/st0l c 9 32
mknod −m 666 /dev/st0m c 9 64
mknod −m 666 /dev/st0a c 9 96
mknod −m 666 /dev/nst0 c 9 128
mknod −m 666 /dev/nst0l c 9 160
mknod −m 666 /dev/nst0m c 9 192
mknod −m 666 /dev/nst0a c 9 224
```

There is no corresponding block device.

The driver uses an internal buffer that has to be large enough to hold at least one tape block. Before Linux 2.1.121, the buffer is allocated as one contiguous block. This limits the block size to the largest contiguous block of memory the kernel allocator can provide. The limit is currently 128 kB for 32-bit architectures and 256 kB for 64-bit architectures. In newer kernels the driver allocates the buffer in several parts if necessary. By default, the maximum number of parts is 16. This means that the maximum block size is very large (2 MB if allocation of 16 blocks of 128 kB succeeds).

The driver's internal buffer size is determined by a compile-time constant which can be overridden with a kernel startup option. In addition to this, the driver tries to allocate a larger temporary buffer at run time if necessary. However, run-time allocation of large contiguous blocks of memory may fail and it is advisable not to rely too much on dynamic buffer allocation before Linux 2.1.121 (this applies also to demand-loading the driver with kerneld or kmod).

The driver does not specifically support any tape drive brand or model. After system start-up the tape device options are defined by the drive firmware. For example, if the drive firmware selects fixed-block mode, the tape device uses fixed-block mode. The options can be changed with explicit **ioctl**(2) calls and remain in effect when the device is closed and reopened. Setting the options affects both the auto-rewind

and the nonrewind device.

Different options can be specified for the different devices within the subgroup of four. The options take effect when the device is opened. For example, the system administrator can define one device that writes in fixed-block mode with a certain block size, and one which writes in variable-block mode (if the drive supports both modes).

The driver supports **tape partitions** if they are supported by the drive. (Note that the tape partitions have nothing to do with disk partitions. A partitioned tape can be seen as several logical tapes within one medium.) Partition support has to be enabled with an **ioctl**(2). The tape location is preserved within each partition across partition changes. The partition used for subsequent tape operations is selected with an **ioctl**(2). The partition switch is executed together with the next tape operation in order to avoid unnecessary tape movement. The maximum number of partitions on a tape is defined by a compile-time constant (originally four). The driver contains an **ioctl**(2) that can format a tape with either one or two partitions.

Device */dev/tape* is usually created as a hard or soft link to the default tape device on the system.

Starting from Linux 2.6.2, the driver exports in the sysfs directory */sys/class/scsi_tape* the attached devices and some parameters assigned to the devices.

Data transfer

The driver supports operation in both fixed-block mode and variable-block mode (if supported by the drive). In fixed-block mode the drive writes blocks of the specified size and the block size is not dependent on the byte counts of the write system calls. In variable-block mode one tape block is written for each write call and the byte count determines the size of the corresponding tape block. Note that the blocks on the tape don't contain any information about the writing mode: when reading, the only important thing is to use commands that accept the block sizes on the tape.

In variable-block mode the read byte count does not have to match the tape block size exactly. If the byte count is larger than the next block on tape, the driver returns the data and the function returns the actual block size. If the block size is larger than the byte count, an error is returned.

In fixed-block mode the read byte counts can be arbitrary if buffering is enabled, or a multiple of the tape block size if buffering is disabled. Before Linux 2.1.121 allow writes with arbitrary byte count if buffering is enabled. In all other cases (before Linux 2.1.121 with buffering disabled or newer kernel) the write byte count must be a multiple of the tape block size.

In Linux 2.6, the driver tries to use direct transfers between the user buffer and the device. If this is not possible, the driver's internal buffer is used. The reasons for not using direct transfers include improper alignment of the user buffer (default is 512 bytes but this can be changed by the HBA driver), one or more pages of the user buffer not reachable by the SCSI adapter, and so on.

A filemark is automatically written to tape if the last tape operation before close was a write.

When a filemark is encountered while reading, the following happens. If there are data remaining in the buffer when the filemark is found, the buffered data is returned. The next read returns zero bytes. The following read returns data from the next file. The end of recorded data is signaled by returning zero bytes for two consecutive read calls. The third read returns an error.

Ioctls

The driver supports three **ioctl**(2) requests. Requests not recognized by the **st** driver are passed to the **SCSI** driver. The definitions below are from */usr/include/linux/mtio.h*:

MTIOCTOP — perform a tape operation

This request takes an argument of type *(struct mtop *)*. Not all drives support all operations. The driver returns an **EIO** error if the drive rejects an operation.

```
/* Structure for MTIOCTOP - mag tape op command: */
struct mtop {
    short   mt_op;       /* operations defined below */
    int     mt_count;    /* how many of them */
};
```

Magnetic tape operations for normal tape use:

MTBSF
Backward space over *mt_count* filemarks.

MTBSFM
Backward space over *mt_count* filemarks. Reposition the tape to the EOT side of the last filemark.

MTBSR
Backward space over *mt_count* records (tape blocks).

MTBSS
Backward space over *mt_count* setmarks.

MTCOMPRESSION
Enable compression of tape data within the drive if *mt_count* is nonzero and disable compression if *mt_count* is zero. This command uses the MODE page 15 supported by most DATs.

MTEOM
Go to the end of the recorded media (for appending files).

MTERASE
Erase tape. With Linux 2.6, short erase (mark tape empty) is performed if the argument is zero. Otherwise, long erase (erase all) is done.

MTFSF
Forward space over *mt_count* filemarks.

MTFSFM
Forward space over *mt_count* filemarks. Reposition the tape to the BOT side of the last filemark.

MTFSR
Forward space over *mt_count* records (tape blocks).

MTFSS
Forward space over *mt_count* setmarks.

MTLOAD
Execute the SCSI load command. A special case is available for some HP autoloaders. If *mt_count* is the constant **MT_ST_HPLOADER_OFFSET** plus a number, the number is sent to the drive to control the autoloader.

MTLOCK
Lock the tape drive door.

MTMKPART
Format the tape into one or two partitions. If *mt_count* is positive, it gives the size of partition 1 and partition 0 contains the rest of the tape. If *mt_count* is zero, the tape is formatted into one partition. From Linux 4.6, a negative *mt_count* specifies the size of partition 0 and the rest of the tape contains partition 1. The physical ordering of partitions depends on the drive. This command is not allowed for a drive unless the partition support is enabled for the drive (see **MT_ST_CAN_PARTITIONS** below).

MTNOP
No op—flushes the driver's buffer as a side effect. Should be used before reading status with **MTIOCGET**.

MTOFFL
Rewind and put the drive off line.

MTRESET
Reset drive.

MTRETEN
Re-tension tape.

MTREW
Rewind.

MTSEEK
Seek to the tape block number specified in *mt_count*. This operation requires either a SCSI-2 drive that supports the **LOCATE** command (device-specific address) or a Tandberg-compatible SCSI-1 drive (Tandberg, Archive Viper, Wangtek, ...). The block number should be one that was previously returned by **MTIOCPOS** if device-specific addresses are used.

MTSETBLK
Set the drive's block length to the value specified in *mt_count*. A block length of zero sets the drive to variable block size mode.

MTSETDENSITY
Set the tape density to the code in *mt_count*. The density codes supported by a drive can be found from the drive documentation.

MTSETPART
The active partition is switched to *mt_count*. The partitions are numbered from zero. This command is not allowed for a drive unless the partition support is enabled for the drive (see **MT_ST_CAN_PARTITIONS** below).

MTUNLOAD
Execute the SCSI unload command (does not eject the tape).

MTUNLOCK
Unlock the tape drive door.

MTWEOF
Write *mt_count* filemarks.

MTWSM
Write *mt_count* setmarks.

Magnetic tape operations for setting of device options (by the superuser):

MTSETDRVBUFFER
Set various drive and driver options according to bits encoded in *mt_count*. These consist of the drive's buffering mode, a set of Boolean driver options, the buffer write threshold, defaults for the block size and density, and timeouts (only since Linux 2.1). A single operation can affect only one item in the list below (the Booleans counted as one item.)

A value having zeros in the high-order 4 bits will be used to set the drive's buffering mode. The buffering modes are:

0 The drive will not report **GOOD** status on write commands until the data blocks are actually written to the medium.

1 The drive may report **GOOD** status on write commands as soon as all the data has been transferred to the drive's internal buffer.

2 The drive may report **GOOD** status on write commands as soon as (a) all the data has been transferred to the drive's internal buffer, and (b) all buffered data from different initiators has been successfully written to the medium.

To control the write threshold the value in *mt_count* must include the constant **MT_ST_WRITE_THRESHOLD** bitwise ORed with a block count in the low 28 bits. The block count refers to 1024-byte blocks, not the physical block size on the tape. The threshold cannot exceed the driver's internal buffer size (see DESCRIPTION, above).

To set and clear the Boolean options the value in *mt_count* must include one of the constants **MT_ST_BOOLEANS**, **MT_ST_SETBOOLEANS**, **MT_ST_CLEARBOOLEANS**, or

MT_ST_DEFBOOLEANS bitwise ORed with whatever combination of the following options is desired. Using**MT_ST_BOOLEANS** the options can be set to the v alues defined in the corresponding bits. With **MT_ST_SETBOOLEANS** the options can be selectively set and with **MT_ST_DEFBOOLEANS** selectively cleared.

The default options for a tape device are set with **MT_ST_DEFBOOLEANS**. A nonactive tape device (e.g., device with minor 32 or 160) is activated when the default options for it are defined the first time. An activated device inherits from the device activated at start-up the options not set explicitly.

The Boolean options are:

MT_ST_BUFFER_WRITES (Default: true)
Buffer all write operations in fixed-block mode. If this option is false and the drive uses a fixed block size, then all write operations must be for a multiple of the block size. This option must be set false to write reliable multivolume archives.

MT_ST_ASYNC_WRITES (Default: true)
When this option is true, write operations return immediately without waiting for the data to be transferred to the drive if the data fits into the driver's buffer. The write threshold determines how full the buffer must be before a new SCSI write command is issued. Any errors reported by the drive will be held until the next operation. This option must be set false to write reliable multivolume archives.

MT_ST_READ_AHEAD (Default: true)
This option causes the driver to provide read buffering and read-ahead in fixed-block mode. If this option is false and the drive uses a fixed block size, then all read operations must be for a multiple of the block size.

MT_ST_TWO_FM (Default: false)
This option modifies the driver behavior when a file is closed. The normal action is to write a single filemark. If the option is true, the driver will write two filemarks and backspace over the second one.

Note: This option should not be set true for QIC tape drives since they are unable to overwrite a filemark. These drives detect the end of recorded data by testing for blank tape rather than two consecutive filemarks. Most other current drives also detect the end of recorded data and using two filemarks is usually necessary only when interchanging tapes with some other systems.

MT_ST_DEBUGGING (Default: false)
This option turns on various debugging messages from the driver (effective only if the driver was compiled with **DEBUG** defined nonzero).

MT_ST_FAST_EOM (Default: false)
This option causes the **MTEOM** operation to be sent directly to the drive, potentially speeding up the operation but causing the driver to lose track of the current file number normally returned by the **MTIOCGET** request. If **MT_ST_FAST_EOM** is false, the driver will respond to an **MTEOM** request by forward spacing over files.

MT_ST_AUTO_LOCK (Default: false)
When this option is true, the drive door is locked when the device file is opened and unlocked when it is closed.

MT_ST_DEF_WRITES (Default: false)
The tape options (block size, mode, compression, etc.) may change when changing from one device linked to a drive to another device linked to the same drive depending on how the devices are defined. This option defines when the changes are enforced by the driver using SCSI-commands and when the drives auto-detection capabilities are relied upon. If this option is false, the driver sends the SCSI-commands immediately when the device is changed. If the option is true, the SCSI-commands are not sent until a write is requested.

In this case, the drive firmware is allowed to detect the tape structure when reading and the SCSI-commands are used only to make sure that a tape is written according to the correct specification.

MT_ST_CAN_BSR (Default: false)
When read-ahead is used, the tape must sometimes be spaced backward to the correct position when the device is closed and the SCSI command to space backward over records is used for this purpose. Some older drives can't process this command reliably and this option can be used to instruct the driver not to use the command. The end result is that, with read-ahead and fixed-block mode, the tape may not be correctly positioned within a file when the device is closed. With Linux 2.6, the default is true for drives supporting SCSI-3.

MT_ST_NO_BLKLIMS (Default: false)
Some drives don't accept the **READ BLOCK LIMITS** SCSI command. If this is used, the driver does not use the command. The drawback is that the driver can't check before sending commands if the selected block size is acceptable to the drive.

MT_ST_CAN_PARTITIONS (Default: false)
This option enables support for several partitions within a tape. The option applies to all devices linked to a drive.

MT_ST_SCSI2LOGICAL (Default: false)
This option instructs the driver to use the logical block addresses defined in the SCSI-2 standard when performing the seek and tell operations (both with **MTSEEK** and **MTIOCPOS** commands and when changing tape partition). Otherwise, the device-specific addresses are used. It is highly advisable to set this option if the drive supports the logical addresses because they count also filemarks. There are some drives that support only the logical block addresses.

MT_ST_SYSV (Default: false)
When this option is enabled, the tape devices use the System V semantics. Otherwise, the BSD semantics are used. The most important difference between the semantics is what happens when a device used for reading is closed: in System V semantics the tape is spaced forward past the next filemark if this has not happened while using the device. In BSD semantics the tape position is not changed.

MT_NO_WAIT (Default: false)
Enables immediate mode (i.e., don't wait for the command to finish) for some commands (e.g., rewind).

An example:

```
struct mtop mt_cmd;
mt_cmd.mt_op = MTSETDRVBUFFER;
mt_cmd.mt_count = MT_ST_BOOLEANS |
        MT_ST_BUFFER_WRITES | MT_ST_ASYNC_WRITES;
ioctl(fd, MTIOCTOP, mt_cmd);
```

The default block size for a device can be set with **MT_ST_DEF_BLKSIZE** and the default density code can be set with **MT_ST_DEFDENSITY**. The values for the parameters are or'ed with the operation code.

With Linux 2.1.x and later, the timeout values can be set with the subcommand **MT_ST_SET_TIMEOUT** ORed with the timeout in seconds. The long timeout (used for rewinds and other commands that may take a long time) can be set with **MT_ST_SET_LONG_TIMEOUT**. The kernel defaults are very long to make sure that a successful command is not timed out with any drive. Because of this, the driver may seem stuck even if it is only waiting for the timeout. These commands can be used to set more practical values for a specific drive. The timeouts set for one device apply for all devices linked to the same drive.

Starting from Linux 2.4.19 and Linux 2.5.43, the driver supports a status bit which indicates whether the drive requests cleaning. The method used by the drive to return cleaning information is set using the **MT_ST_SEL_CLN** subcommand. If the value is zero, the cleaning bit is always zero. If the value is one, the TapeAlert data defined in the SCSI-3 standard is used (not yet implemented). Values 2–17 are reserved. If the lowest eight bits are >= 18, bits from the extended sense data are used. The bits 9–16 specify a mask to select the bits to look at and the bits 17–23 specify the bit pattern to look for. If the bit pattern is zero, one or more bits under the mask indicate the cleaning request. If the pattern is nonzero, the pattern must match the masked sense data byte.

MTIOCGET — get status

This request takes an argument of type *(struct mtget *)*.

```
/* structure for MTIOCGET - mag tape get status command */
struct mtget {
    long     mt_type;
    long     mt_resid;
    /* the following registers are device dependent */
    long     mt_dsreg;
    long     mt_gstat;
    long     mt_erreg;
    /* The next two fields are not always used */
    daddr_t  mt_fileno;
    daddr_t  mt_blkno;
};
```

mt_type

The header file defines many values for *mt_type*, but the current driver reports only the generic types **MT_ISSCSI1** (Generic SCSI-1 tape) and **MT_ISSCSI2** (Generic SCSI-2 tape).

mt_resid

contains the current tape partition number.

mt_dsreg

reports the drive's current settings for block size (in the low 24 bits) and density (in the high 8 bits). These fields are defined by**MT_ST_BLKSIZE_SHIFT**, **MT_ST_BLKSIZE_MASK**, **MT_ST_DENSITY_SHIFT**, and **MT_ST_DENSITY_MASK**.

mt_gstat

reports generic (device independent) status information. The header file defines macros for testing these status bits:

GMT_EOF(*x*)

The tape is positioned just after a filemark (always false after an **MTSEEK** operation).

GMT_BOT(*x*)

The tape is positioned at the beginning of the first file (always false after an **MTSEEK** operation).

GMT_EOT(*x*)

A tape operation has reached the physical End Of Tape.

GMT_SM(*x*)

The tape is currently positioned at a setmark (always false after an **MTSEEK** operation).

GMT_EOD(*x*)

The tape is positioned at the end of recorded data.

GMT_WR_PROT(*x*)

The drive is write-protected. For some drives this can also mean that the drive does not support writing on the current medium type.

GMT_ONLINE(*x*)
The last **open**(2) found the drive with a tape in place and ready for operation.

GMT_D_6250(*x*)
GMT_D_1600(*x*)
GMT_D_800(*x*)
This "generic" status information reports the current density setting for 9-track ½" tape drives only.

GMT_DR_OPEN(*x*)
The drive does not have a tape in place.

GMT_IM_REP_EN(*x*)
Immediate report mode. This bit is set if there are no guarantees that the data has been physically written to the tape when the write call returns. It is set zero only when the driver does not buffer data and the drive is set not to buffer data.

GMT_CLN(*x*)
The drive has requested cleaning. Implemented since Linux 2.4.19 and Linux 2.5.43.

mt_erreg
The only field defined in *mt_erreg* is the recovered error count in the low 16 bits (as defined by **MT_ST_SOFTERR_SHIFT** and **MT_ST_SOFTERR_MASK**). Due to inconsistencies in the way drives report recovered errors, this count is often not maintained (most drives do not by default report soft errors but this can be changed with a SCSI MODE SELECT command).

mt_fileno
reports the current file number (zero-based). This value is set to −1 when the file number is unknown (e.g., after **MTBSS** or **MTSEEK**).

mt_blkno
reports the block number (zero-based) within the current file. This value is set to −1 when the block number is unknown (e.g., after **MTBSF**, **MTBSS**, or **MTSEEK**).

MTIOCPOS — get tape position
This request takes an argument of type *(struct mtpos *)* and reports the drive's notion of the current tape block number, which is not the same as *mt_blkno* returned by **MTIOCGET**. This drive must be a SCSI-2 drive that supports the **READ POSITION** command (device-specific address) or a Tandberg-compatible SCSI-1 drive (Tandberg, Archive Viper, Wangtek, ...).

```
/* structure for MTIOCPOS − mag tape get position command */
struct mtpos {
    long mt_blkno;    /* current block number */
};
```

RETURN VALUE

EACCES
An attempt was made to write or erase a write-protected tape. (This error is not detected during **open**(2).)

EBUSY
The device is already in use or the driver was unable to allocate a buffer.

EFAULT
The command parameters point to memory not belonging to the calling process.

EINVAL
An **ioctl**(2) had an invalid argument, or a requested block size was invalid.

EIO The requested operation could not be completed.

ENOMEM
The byte count in **read**(2) is smaller than the next physical block on the tape. (Before Linux 2.2.18 and Linux 2.4.0 the extra bytes have been silently ignored.)

ENOSPC
A write operation could not be completed because the tape reached end-of-medium.

ENOSYS
Unknown **ioctl**(2).

ENXIO
During opening, the tape device does not exist.

EOVERFLOW
An attempt was made to read or write a variable-length block that is larger than the driver's internal buffer.

EROFS
Open is attempted with **O_WRONLY** or **O_RDWR** when the tape in the drive is write-protected.

FILES

*/dev/st**
the auto-rewind SCSI tape devices

*/dev/nst**
the nonrewind SCSI tape devices

NOTES

- When exchanging data between systems, both systems have to agree on the physical tape block size. The parameters of a drive after startup are often not the ones most operating systems use with these devices. Most systems use drives in variable-block mode if the drive supports that mode. This applies to most modern drives, including DATs, 8mm helical scan drives, DLTs, etc. It may be advisable to use these drives in variable-block mode also in Linux (i.e., use **MTSETBLK** or **MTSETDEFBLK** at system startup to set the mode), at least when exchanging data with a foreign system. The drawback of this is that a fairly large tape block size has to be used to get acceptable data transfer rates on the SCSI bus.
- Many programs (e.g., **tar**(1)) allow the user to specify the blocking factor on the command line. Note that this determines the physical block size on tape only in variable-block mode.
- In order to use SCSI tape drives, the basic SCSI driver, a SCSI-adapter driver and the SCSI tape driver must be either configured into the kernel or loaded as modules. If the SCSI-tape driver is not present, the drive is recognized but the tape support described in this page is not available.
- The driver writes error messages to the console/log. The SENSE codes written into some messages are automatically translated to text if verbose SCSI messages are enabled in kernel configuration.
- The driver's internal buffering allows good throughput in fixed-block mode also with small **read**(2) and **write**(2) byte counts. With direct transfers this is not possible and may cause a surprise when moving to the 2.6 kernel. The solution is to tell the software to use larger transfers (often telling it to use larger blocks). If this is not possible, direct transfers can be disabled.

SEE ALSO

mt(1)

The file *drivers/scsi/README.st* or *Documentation/scsi/st.txt* (kernel >= 2.6) in the Linux kernel source tree contains the most recent information about the driver and its configuration possibilities