

**NAME**

`ioctl_getfsmap` – retrieve the physical layout of the filesystem

**SYNOPSIS**

```
#include <sys/ioctl.h>
#include <linux/fs.h>
#include <linux/fsmap.h>
```

```
int ioctl(int fd, FS_IOC_GETFSMAP, struct fsmap_head * arg);
```

**DESCRIPTION**

This `ioctl(2)` operation retrieves physical extent mappings for a filesystem. This information can be used to discover which files are mapped to a physical block, examine free space, or find known bad blocks, among other things.

The sole argument to this operation should be a pointer to a single `struct fsmap_head`:

```
struct fsmap {
    __u32 fmr_device;        /* Device ID */
    __u32 fmr_flags;         /* Mapping flags */
    __u64 fmr_physical;      /* Device offset of segment */
    __u64 fmr_owner;         /* Owner ID */
    __u64 fmr_offset;        /* File offset of segment */
    __u64 fmr_length;        /* Length of segment */
    __u64 fmr_reserved[3];   /* Must be zero */
};

struct fsmap_head {
    __u32 fmh_iflags;        /* Control flags */
    __u32 fmh_oflags;        /* Output flags */
    __u32 fmh_count;         /* # of entries in array incl. input */
    __u32 fmh_entries;       /* # of entries filled in (output) */
    __u64 fmh_reserved[6];   /* Must be zero */

    struct fsmap fmh_keys[2]; /* Low and high keys for
                               the mapping search */
    struct fsmap fmh_recs[];  /* Returned records */
};
```

The two `fmh_keys` array elements specify the lowest and highest reverse-mapping key for which the application would like physical mapping information. A reverse mapping key consists of the tuple (device, block, owner, offset). The owner and offset fields are part of the key because some filesystems support sharing physical blocks between multiple files and therefore may return multiple mappings for a given physical block.

Filesystem mappings are copied into the `fmh_recs` array, which immediately follows the header data.

**Fields of struct fsmap\_head**

The `fmh_iflags` field is a bit mask passed to the kernel to alter the output. No flags are currently defined, so the caller must set this value to zero.

The `fmh_oflags` field is a bit mask of flags set by the kernel concerning the returned mappings. If **FMH\_OF\_DEV\_T** is set, then the `fmr_device` field represents a `dev_t` structure containing the major and minor numbers of the block device.

The `fmh_count` field contains the number of elements in the array being passed to the kernel. If this value is 0, `fmh_entries` will be set to the number of records that would have been returned had the array been large enough; no mapping information will be returned.

The `fmh_entries` field contains the number of elements in the `fmh_recs` array that contain useful information.

The *fmr\_reserved* fields must be set to zero.

### Keys

The two key records in *fsmmap\_head.fmh\_keys* specify the lowest and highest extent records in the key space that the caller wants returned. A filesystem that can share blocks between files likely requires the tuple (*device*, *physical*, *owner*, *offset*, *flags*) to uniquely index any filesystem mapping record. Classic non-sharing filesystems might be able to identify any record with only (*device*, *physical*, *flags*). For example, if the low key is set to (8:0, 36864, 0, 0, 0), the filesystem will only return records for extents starting at or above 36 KiB on disk. If the high key is set to (8:0, 1048576, 0, 0, 0), only records below 1 MiB will be returned. The format of *fmr\_device* in the keys must match the format of the same field in the output records, as defined below. By convention, the field *fsmmap\_head.fmh\_keys[0]* must contain the low key and *fsmmap\_head.fmh\_keys[1]* must contain the high key for the request.

For convenience, if *fmr\_length* is set in the low key, it will be added to *fmr\_block* or *fmr\_offset* as appropriate. The caller can take advantage of this subtlety to set up subsequent calls by copying *fsmmap\_head.fmh\_recs[fsmmap\_head.fmh\_entries - 1]* into the low key. The function *fsmmap\_advance* (defined in *linux/fsmmap.h*) provides this functionality.

### Fields of struct fsmap

The *fmr\_device* field uniquely identifies the underlying storage device. If the **FMH\_OF\_DEV\_T** flag is set in the header's *fmr\_oflags* field, this field contains a *dev\_t* from which major and minor numbers can be extracted. If the flag is not set, this field contains a value that must be unique for each unique storage device.

The *fmr\_physical* field contains the disk address of the extent in bytes.

The *fmr\_owner* field contains the owner of the extent. This is an inode number unless **FMR\_OF\_SPECIAL\_OWNER** is set in the *fmr\_flags* field, in which case the value is determined by the filesystem. See the section below about owner values for more details.

The *fmr\_offset* field contains the logical address in the mapping record in bytes. This field has no meaning if the **FMR\_OF\_SPECIAL\_OWNER** or **FMR\_OF\_EXTENT\_MAP** flags are set in *fmr\_flags*.

The *fmr\_length* field contains the length of the extent in bytes.

The *fmr\_flags* field is a bit mask of extent state flags. The bits are:

#### **FMR\_OF\_PREALLOC**

The extent is allocated but not yet written.

#### **FMR\_OF\_ATTR\_FORK**

This extent contains extended attribute data.

#### **FMR\_OF\_EXTENT\_MAP**

This extent contains extent map information for the owner.

#### **FMR\_OF\_SHARED**

Parts of this extent may be shared.

#### **FMR\_OF\_SPECIAL\_OWNER**

The *fmr\_owner* field contains a special value instead of an inode number.

#### **FMR\_OF\_LAST**

This is the last record in the data set.

The *fmr\_reserved* field will be set to zero.

### Owner values

Generally, the value of the *fmr\_owner* field for non-metadata extents should be an inode number. However, filesystems are under no obligation to report inode numbers; they may instead report **FMR\_OWN\_UNKNOWN** if the inode number cannot easily be retrieved, if the caller lacks sufficient privilege, if the filesystem does not support stable inode numbers, or for any other reason. If a filesystem wishes to condition the reporting of inode numbers based on process capabilities, it is strongly urged that the **CAP\_SYS\_ADMIN** capability be used for this purpose.

The following special owner values are generic to all filesystems:

**FMR\_OWN\_FREE**

Free space.

**FMR\_OWN\_UNKNOWN**

This extent is in use but its owner is not known or not easily retrieved.

**FMR\_OWN\_METADATA**

This extent is filesystem metadata.

XFS can return the following special owner values:

**XFS\_FMR\_OWN\_FREE**

Free space.

**XFS\_FMR\_OWN\_UNKNOWN**

This extent is in use but its owner is not known or not easily retrieved.

**XFS\_FMR\_OWN\_FS**

Static filesystem metadata which exists at a fixed address. These are the AG superblock, the AGF, the AGFL, and the AGI headers.

**XFS\_FMR\_OWN\_LOG**

The filesystem journal.

**XFS\_FMR\_OWN\_AG**

Allocation group metadata, such as the free space btrees and the reverse mapping btrees.

**XFS\_FMR\_OWN\_INOBT**

The inode and free inode btrees.

**XFS\_FMR\_OWN\_INODES**

Inode records.

**XFS\_FMR\_OWN\_REFC**

Reference count information.

**XFS\_FMR\_OWN\_COW**

This extent is being used to stage a copy-on-write.

**XFS\_FMR\_OWN\_DEFECTIVE:**

This extent has been marked defective either by the filesystem or the underlying device.

ext4 can return the following special owner values:

**EXT4\_FMR\_OWN\_FREE**

Free space.

**EXT4\_FMR\_OWN\_UNKNOWN**

This extent is in use but its owner is not known or not easily retrieved.

**EXT4\_FMR\_OWN\_FS**

Static filesystem metadata which exists at a fixed address. This is the superblock and the group descriptors.

**EXT4\_FMR\_OWN\_LOG**

The filesystem journal.

**EXT4\_FMR\_OWN\_INODES**

Inode records.

**EXT4\_FMR\_OWN\_BLKBM**

Block bit map.

**EXT4\_FMR\_OWN\_INOBT**

Inode bit map.

**RETURN VALUE**

On error, `-1` is returned, and *errno* is set to indicate the error.

**ERRORS**

The error placed in *errno* can be one of, but is not limited to, the following:

**EBADF**

*fd* is not open for reading.

**EBADMSG**

The filesystem has detected a checksum error in the metadata.

**EFAULT**

The pointer passed in was not mapped to a valid memory address.

**EINVAL**

The array is not long enough, the keys do not point to a valid part of the filesystem, the low key points to a higher point in the filesystem's physical storage address space than the high key, or a nonzero value was passed in one of the fields that must be zero.

**ENOMEM**

Insufficient memory to process the request.

**EOPNOTSUPP**

The filesystem does not support this command.

**EUCLEAN**

The filesystem metadata is corrupt and needs repair.

**VERSIONS**

The **FS\_IOC\_GETFSMAP** operation first appeared in Linux 4.12.

**CONFORMING TO**

This API is Linux-specific. Not all filesystems support it.

**EXAMPLE**

See *io/fsmmap.c* in the *xfsprogs* distribution for a sample program.

**SEE ALSO**

**ioctl(2)**

**COLOPHON**

This page is part of release 5.05 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.