- for all commands, use a single 1-byte descriptor for the *ack* field

See 2.7.4.

5.2 Block Device

The virtio block device is a simple virtual block device (ie. disk). Read and write requests (and other exotic requests) are placed in one of its queues, and serviced (probably out of order) by the device except where noted.

5.2.1 Device ID

2

5.2.2 Virtqueues

0 requestq1

. . .

N-1 requestqN

N=1 if VIRTIO_BLK_F_MQ is not negotiated, otherwise N is set by *num_queues*.

5.2.3 Feature bits

VIRTIO_BLK_F_SIZE_MAX (1) Maximum size of any single segment is in *size_max*.

VIRTIO_BLK_F_SEG_MAX (2) Maximum number of segments in a request is in *seg_max*.

VIRTIO_BLK_F_GEOMETRY (4) Disk-style geometry specified in *geometry*.

VIRTIO_BLK_F_RO (5) Device is read-only.

VIRTIO_BLK_F_BLK_SIZE (6) Block size of disk is in *blk_size*.

VIRTIO_BLK_F_FLUSH (9) Cache flush command support.

VIRTIO_BLK_F_TOPOLOGY (10) Device exports information on optimal I/O alignment.

VIRTIO_BLK_F_CONFIG_WCE (11) Device can toggle its cache between writeback and writethrough modes.

VIRTIO_BLK_F_MQ (12) Device supports multiqueue.

VIRTIO_BLK_F_DISCARD (13) Device can support discard command, maximum discard sectors size in *max_discard_sectors* and maximum discard segment number in *max_discard_seg*.

VIRTIO_BLK_F_WRITE_ZEROES (14) Device can support write zeroes command, maximum write zeroes sectors size in *max_write_zeroes_sectors* and maximum write zeroes segment number in *max_write_-zeroes_seg*.

VIRTIO_BLK_F_LIFETIME (15) Device supports providing storage lifetime information.

VIRTIO_BLK_F_SECURE_ERASE (16) Device supports secure erase command, maximum erase sectors count in *max_secure_erase_sectors* and maximum erase segment number in *max_secure_erase_seg*.

VIRTIO_BLK_F_ZONED(17) Device is a Zoned Block Device, that is, a device that follows the zoned storage device behavior that is also supported by industry standards such as the T10 Zoned Block Command standard (ZBC r05) or the NVMe(TM) NVM Express Zoned Namespace Command Set Specification 1.1b (ZNS). For brevity, these standard documents are referred as ”ZBD standards” from this point on in the text.

5.2.3.1 Legacy Interface: Feature bits

VIRTIO_BLK_F_BARRIER (0) Device supports request barriers.

VIRTIO_BLK_F_SCSI (7) Device supports scsi packet commands.

Note: In the legacy interface, VIRTIO_BLK_F_FLUSH was also called VIRTIO_BLK_F_WCE.

5.2.4 Device configuration layout

The block device has the following device configuration layout.

```
struct virtio_blk_config {
        le64 capacity;
        le32 size_max;
        le32 seg_max;
        struct virtio_blk_geometry {
                le16 cylinders;
                u8 heads;
                u8 sectors;
        } geometry;
        le32 blk_size;
        struct virtio_blk_topology {
                // # of logical blocks per physical block (log2)
                u8 physical_block_exp;
                // offset of first aligned logical block
                u8 alignment_offset;
                // suggested minimum I/O size in blocks
                le16 min_io_size;
                // optimal (suggested maximum) I/O size in blocks
                le32 opt_io_size;
        } topology;
        u8 writeback;
        u8 unused0;
        u16 num_queues;
        le32 max_discard_sectors;
        le32 max_discard_seg;
        le32 discard_sector_alignment;
        le32 max_write_zeroes_sectors;
        le32 max_write_zeroes_seg;
        u8 write_zeroes_may_unmap;
        u8 unused1[3];
        le32 max_secure_erase_sectors;
        le32 max_secure_erase_seg;
        le32 secure_erase_sector_alignment;
        struct virtio_blk_zoned_characteristics {
                le32 zone_sectors;
                le32 max_open_zones;
                le32 max_active_zones;
                le32 max_append_sectors;
                le32 write_granularity;
                u8 model;
                u8 unused2[3];
        } zoned;
};
```

The *capacity* of the device (expressed in 512-byte sectors) is always present. The availability of the others all depend on various feature bits as indicated above.

The field *num_queues* only exists if VIRTIO_BLK_F_MQ is set. This field specifies the number of queues.

The parameters in the configuration space of the device *max_discard_sectors discard_sector_alignment* are expressed in 512-byte units if the VIRTIO_BLK_F_DISCARD feature bit is negotiated. The *max_write_-zeroes_sectors* is expressed in 512-byte units if the VIRTIO_BLK_F_WRITE_ZEROES feature bit is negotiated. The parameters in the configuration space of the device *max_secure_erase_sectors secure_erase_-sector_alignment* are expressed in 512-byte units if the VIRTIO_BLK_F_SECURE_ERASE feature bit is negotiated.

If the VIRTIO_BLK_F_ZONED feature is negotiated, then in *virtio_blk_zoned_characteristics*,

- *zone_sectors* value is expressed in 512-byte sectors.
- *max_append_sectors* value is expressed in 512-byte sectors.
- *write_granularity* value is expressed in bytes.

The *model* field in *zoned* may have the following values:

```
#define VIRTIO_BLK_Z_NONE      0
#define VIRTIO_BLK_Z_HM        1
#define VIRTIO_BLK_Z_HA        2
```

Depending on their design, zoned block devices may follow several possible models of operation. The three models that are standardized for ZBDs are drive-managed, host-managed and host-aware.

While being zoned internally, drive-managed ZBDs behave exactly like regular, non-zoned block devices. For the purposes of virtio standardization, drive-managed ZBDs can always be treated as non-zoned devices. These devices have the VIRTIO_BLK_Z_NONE model value set in the *model* field in *zoned*.

Devices that offer the VIRTIO_BLK_F_ZONED feature while reporting the VIRTIO_BLK_Z_NONE zoned model are drive-managed zoned block devices. In this case, the driver treats the device as a regular non-zoned block device.

Host-managed zoned block devices have their LBA range divided into Sequential Write Required (SWR) zones that require some additional handling by the host for correct operation. All write requests to SWR zones are required be sequential and zones containing some written data need to be reset before that data can be rewritten. Host-managed devices support a set of ZBD-specific I/O requests that can be used by the host to manage device zones. Host-managed devices report VIRTIO_BLK_Z_HM in the *model* field in *zoned*.

Host-aware zoned block devices have their LBA range divided to Sequential Write Preferred (SWP) zones that support random write access, similar to regular non-zoned devices. However, the device I/O performance might not be optimal if SWP zones are used in a random I/O pattern. SWP zones also support the same set of ZBD-specific I/O requests as host-managed devices that allow host-aware devices to be managed by any host that supports zoned block devices to achieve its optimum performance. Host-aware devices report VIRTIO_BLK_Z_HA in the *model* field in *zoned*.

Both SWR zones and SWP zones are sometimes referred as sequential zones.

During device operation, sequential zones can be in one of the following states: empty, implicitly-open, explicitly-open, closed and full. The state machine that governs the transitions between these states is described later in this document.

SWR and SWP zones consume volatile device resources while being in certain states and the device may set limits on the number of zones that can be in these states simultaneously.

Zoned block devices use two internal counters to account for the device resources in use, the number of currently open zones and the number of currently active zones.

Any zone state transition from a state that doesn't consume a zone resource to a state that consumes the same resource increments the internal device counter for that resource. Any zone transition out of a state that consumes a zone resource to a state that doesn't consume the same resource decrements the counter. Any request that causes the device to exceed the reported zone resource limits is terminated by the device with a "zone resources exceeded" error as defined for specific commands later.

5.2.4.1 Legacy Interface: Device configuration layout

When using the legacy interface, transitional devices and drivers MUST format the fields in struct virtio_-blk_config according to the native endian of the guest rather than (necessarily when not using the legacy interface) little-endian.

5.2.5 Device Initialization

1. The device size can be read from *capacity*.

2. If the VIRTIO_BLK_F_BLK_SIZE feature is negotiated, *blk_size* can be read to determine the optimal sector size for the driver to use. This does not affect the units used in the protocol (always 512 bytes), but awareness of the correct value can affect performance.
3. If the VIRTIO_BLK_F_RO feature is set by the device, any write requests will fail.
4. If the VIRTIO_BLK_F_TOPOLOGY feature is negotiated, the fields in the *topology* struct can be read to determine the physical block size and optimal I/O lengths for the driver to use. This also does not affect the units in the protocol, only performance.
5. If the VIRTIO_BLK_F_CONFIG_WCE feature is negotiated, the cache mode can be read or set through the *writeback* field. 0 corresponds to a writethrough cache, 1 to a writeback cache[4]. The cache mode after reset can be either writeback or writethrough. The actual mode can be determined by reading *writeback* after feature negotiation.
6. If the VIRTIO_BLK_F_DISCARD feature is negotiated, *max_discard_sectors* and *max_discard_seg* can be read to determine the maximum discard sectors and maximum number of discard segments for the block driver to use. *discard_sector_alignment* can be used by OS when splitting a request based on alignment.
7. If the VIRTIO_BLK_F_WRITE_ZEROES feature is negotiated, *max_write_zeroes_sectors* and *max_-write_zeroes_seg* can be read to determine the maximum write zeroes sectors and maximum number of write zeroes segments for the block driver to use.
8. If the VIRTIO_BLK_F_MQ feature is negotiated, *num_queues* field can be read to determine the number of queues.
9. If the VIRTIO_BLK_F_SECURE_ERASE feature is negotiated, *max_secure_erase_sectors* and *max_-secure_erase_seg* can be read to determine the maximum secure erase sectors and maximum number of secure erase segments for the block driver to use. *secure_erase_sector_alignment* can be used by OS when splitting a request based on alignment.
10. If the VIRTIO_BLK_F_ZONED feature is negotiated, the fields in *zoned* can be read by the driver to determine the zone characteristics of the device. All *zoned* fields are read-only.

5.2.5.1 Driver Requirements: Device Initialization

Drivers SHOULD NOT negotiate VIRTIO_BLK_F_FLUSH if they are incapable of sending VIRTIO_BLK_-T_FLUSH commands.

If neither VIRTIO_BLK_F_CONFIG_WCE nor VIRTIO_BLK_F_FLUSH are negotiated, the driver MAY deduce the presence of a writethrough cache. If VIRTIO_BLK_F_CONFIG_WCE was not negotiated but VIRTIO_BLK_F_FLUSH was, the driver SHOULD assume presence of a writeback cache.

The driver MUST NOT read *writeback* before setting the FEATURES_OK *device status* bit.

Drivers MUST NOT negotiate the VIRTIO_BLK_F_ZONED feature if they are incapable of supporting devices with the VIRTIO_BLK_Z_HM, VIRTIO_BLK_Z_HA or VIRTIO_BLK_Z_NONE zoned model.

If the VIRTIO_BLK_F_ZONED feature is offered by the device with the VIRTIO_BLK_Z_HM zone model, then the VIRTIO_BLK_F_DISCARD feature MUST NOT be offered by the driver.

If the VIRTIO_BLK_F_ZONED feature and VIRTIO_BLK_F_DISCARD feature are both offered by the device with the VIRTIO_BLK_Z_HA or VIRTIO_BLK_Z_NONE zone model, then the driver MAY negotiate these two bits independently.

If the VIRTIO_BLK_F_ZONED feature is negotiated, then

- if the driver that can not support host-managed zoned devices reads VIRTIO_BLK_Z_HM from the *model* field of *zoned*, the driver MUST NOT set FEATURES_OK flag and instead set the FAILED bit.

[4]Consistent with 5.2.6.2, a writethrough cache can be defined broadly as a cache that commits writes to persistent device backend storage before reporting their completion. For example, a battery-backed writeback cache actually counts as writethrough according to this definition.

- if the driver that can not support zoned devices reads VIRTIO_BLK_Z_HA from the *model* field of *zoned*, the driver MAY handle the device as a non-zoned device. In this case, the driver SHOULD ignore all other fields in *zoned*.

5.2.5.2 Device Requirements: Device Initialization

Devices SHOULD always offer VIRTIO_BLK_F_FLUSH, and MUST offer it if they offer VIRTIO_BLK_F_-CONFIG_WCE.

If VIRTIO_BLK_F_CONFIG_WCE is negotiated but VIRTIO_BLK_F_FLUSH is not, the device MUST initialize *writeback* to 0.

The device MUST initialize padding bytes *unused0* and *unused1* to 0.

If the device that is being initialized is a not a zoned device, the device SHOULD NOT offer the VIRTIO_-BLK_F_ZONED feature.

The VIRTIO_BLK_F_ZONED feature cannot be properly negotiated without FEATURES_OK bit. Legacy devices MUST NOT offer VIRTIO_BLK_F_ZONED feature bit.

If the VIRTIO_BLK_F_ZONED feature is not accepted by the driver,

- the device with the VIRTIO_BLK_Z_HA or VIRTIO_BLK_Z_NONE zone model SHOULD proceed with the initialization while setting all zoned characteristics fields to zero.
- the device with the VIRTIO_BLK_Z_HM zone model MUST fail to set the FEATURES_OK device status bit when the driver writes the Device Status field.

If the VIRTIO_BLK_F_ZONED feature is negotiated, then the *model* field in *zoned* struct in the configuration space MUST be set by the device

- to the value of VIRTIO_BLK_Z_NONE if it operates as a drive-managed zoned block device or a non-zoned block device.
- to the value of VIRTIO_BLK_Z_HM if it operates as a host-managed zoned block device.
- to the value of VIRTIO_BLK_Z_HA if it operates as a host-aware zoned block device.

If the VIRTIO_BLK_F_ZONED feature is negotiated and the device *model* field in *zoned* struct is VIRTIO_-BLK_Z_HM or VIRTIO_BLK_Z_HA,

- the *zone_sectors* field of *zoned* MUST be set by the device to the size of a single zone on the device. All zones of the device have the same size indicated by *zone_sectors* except for the last zone that MAY be smaller than all other zones. The driver can calculate the number of zones on the device as

  ```
          nr_zones = (capacity + zone_sectors - 1) / zone_sectors;
  ```

  and the size of the last zone as

  ```
          zs_last = capacity - (nr_zones - 1) * zone_sectors;
  ```

- The *max_open_zones* field of the *zoned* structure MUST be set by the device to the maximum number of zones that can be open on the device (zones in the implicit open or explicit open state). A value of zero indicates that the device does not have any limit on the number of open zones.
- The *max_active_zones* field of the *zoned* structure MUST be set by the device to the maximum number zones that can be active on the device (zones in the implicit open, explicit open or closed state). A value of zero indicates that the device does not have any limit on the number of active zones.
- the *max_append_sectors* field of *zoned* MUST be set by the device to the maximum data size of a VIRTIO_BLK_T_ZONE_APPEND request that can be successfully issued to the device. The value of this field MUST NOT exceed the *seg_max* * *size_max* value. A device MAY set the *max_append_-sectors* to zero if it doesn't support VIRTIO_BLK_T_ZONE_APPEND requests.

- the *write_granularity* field of *zoned* MUST be set by the device to the offset and size alignment constraint for VIRTIO_BLK_T_OUT and VIRTIO_BLK_T_ZONE_APPEND requests issued to a sequential zone of the device.
- the device MUST initialize padding bytes *unused2* to 0.

5.2.5.3 Legacy Interface: Device Initialization

Because legacy devices do not have FEATURES_OK, transitional devices MUST implement slightly different behavior around feature negotiation when used through the legacy interface. In particular, when using the legacy interface:

- the driver MAY read or write *writeback* before setting the DRIVER or DRIVER_OK *device status* bit
- the device MUST NOT modify the cache mode (and *writeback*) as a result of a driver setting a status bit, unless the DRIVER_OK bit is being set and the driver has not set the VIRTIO_BLK_F_CONFIG_WCE driver feature bit.
- the device MUST NOT modify the cache mode (and *writeback*) as a result of a driver modifying the driver feature bits, for example if the driver sets the VIRTIO_BLK_F_CONFIG_WCE driver feature bit but does not set the VIRTIO_BLK_F_FLUSH bit.

5.2.6 Device Operation

The driver enqueues requests to the virtqueues, and they are used by the device (not necessarily in order). Each request except VIRTIO_BLK_T_ZONE_APPEND is of form:

```
struct virtio_blk_req {
        le32 type;
        le32 reserved;
        le64 sector;
        u8 data[];
        u8 status;
};
```

The type of the request is either a read (VIRTIO_BLK_T_IN), a write (VIRTIO_BLK_T_OUT), a discard (VIRTIO_BLK_T_DISCARD), a write zeroes (VIRTIO_BLK_T_WRITE_ZEROES), a flush (VIRTIO_BLK_-T_FLUSH), a get device ID string command (VIRTIO_BLK_T_GET_ID), a secure erase (VIRTIO_BLK_T_-SECURE_ERASE), or a get device lifetime command (VIRTIO_BLK_T_GET_LIFETIME).

```
#define VIRTIO_BLK_T_IN           0
#define VIRTIO_BLK_T_OUT          1
#define VIRTIO_BLK_T_FLUSH        4
#define VIRTIO_BLK_T_GET_ID       8
#define VIRTIO_BLK_T_GET_LIFETIME 10
#define VIRTIO_BLK_T_DISCARD      11
#define VIRTIO_BLK_T_WRITE_ZEROES 13
#define VIRTIO_BLK_T_SECURE_ERASE   14
```

The *sector* number indicates the offset (multiplied by 512) where the read or write is to occur. This field is unused and set to 0 for commands other than read, write and some zone operations.

VIRTIO_BLK_T_IN requests populate *data* with the contents of sectors read from the block device (in multiples of 512 bytes). VIRTIO_BLK_T_OUT requests write the contents of *data* to the block device (in multiples of 512 bytes).

The *data* used for discard, secure erase or write zeroes commands consists of one or more segments. The maximum number of segments is *max_discard_seg* for discard commands, *max_secure_erase_seg* for secure erase commands and *max_write_zeroes_seg* for write zeroes commands. Each segment is of form:

```
struct virtio_blk_discard_write_zeroes {
       le64 sector;
       le32 num_sectors;
       struct {
```

```
                le32 unmap:1;
                le32 reserved:31;
        } flags;
};
```

sector indicates the starting offset (in 512-byte units) of the segment, while *num_sectors* indicates the number of sectors in each discarded range. *unmap* is only used in write zeroes commands and allows the device to discard the specified range, provided that following reads return zeroes.

VIRTIO_BLK_T_GET_ID requests fetch the device ID string from the device into *data*. The device ID string is a NUL-padded ASCII string up to 20 bytes long. If the string is 20 bytes long then there is no NUL terminator.

The *data* used for VIRTIO_BLK_T_GET_LIFETIME requests is populated by the device, and is of the form

```
struct virtio_blk_lifetime {
  le16 pre_eol_info;
  le16 device_lifetime_est_typ_a;
  le16 device_lifetime_est_typ_b;
};
```

The *pre_eol_info* specifies the percentage of reserved blocks that are consumed and will have one of these values:

```
/* Value not available */
#define VIRTIO_BLK_PRE_EOL_INFO_UNDEFINED    0
/* < 80% of reserved blocks are consumed */
#define VIRTIO_BLK_PRE_EOL_INFO_NORMAL       1
/* 80% of reserved blocks are consumed */
#define VIRTIO_BLK_PRE_EOL_INFO_WARNING      2
/* 90% of reserved blocks are consumed */
#define VIRTIO_BLK_PRE_EOL_INFO_URGENT       3
/* All others values are reserved */
```

The *device_lifetime_est_typ_a* refers to wear of SLC cells and is provided in increments of 10used, and so on, thru to 11 meaning estimated lifetime exceeded. All values above 11 are reserved.

The *device_lifetime_est_typ_b* refers to wear of MLC cells and is provided with the same semantics as *device_lifetime_est_typ_a*.

The final *status* byte is written by the device: either VIRTIO_BLK_S_OK for success, VIRTIO_BLK_S_-IOERR for device or driver error or VIRTIO_BLK_S_UNSUPP for a request unsupported by device:

```
#define VIRTIO_BLK_S_OK        0
#define VIRTIO_BLK_S_IOERR     1
#define VIRTIO_BLK_S_UNSUPP    2
```

The status of individual segments is indeterminate when a discard or write zero command produces VIRTIO_BLK_S_IOERR. A segment may have completed successfully, failed, or not been processed by the device.

The following requirements only apply if the VIRTIO_BLK_F_ZONED feature is negotiated.

In addition to the request types defined for non-zoned devices, the type of the request can be a zone report (VIRTIO_BLK_T_ZONE_REPORT), an explicit zone open (VIRTIO_BLK_T_ZONE_OPEN), a zone close (VIRTIO_BLK_T_ZONE_CLOSE), a zone finish (VIRTIO_BLK_T_ZONE_FINISH), a zone_append (VIRTIO_BLK_T_ZONE_APPEND), a zone reset (VIRTIO_BLK_T_ZONE_RESET) or a zone reset all (VIRTIO_BLK_T_ZONE_RESET_ALL).

```
#define VIRTIO_BLK_T_ZONE_APPEND    15
#define VIRTIO_BLK_T_ZONE_REPORT    16
#define VIRTIO_BLK_T_ZONE_OPEN      18
#define VIRTIO_BLK_T_ZONE_CLOSE     20
#define VIRTIO_BLK_T_ZONE_FINISH    22
#define VIRTIO_BLK_T_ZONE_RESET     24
#define VIRTIO_BLK_T_ZONE_RESET_ALL 26
```

Requests of type VIRTIO_BLK_T_OUT, VIRTIO_BLK_T_ZONE_OPEN, VIRTIO_BLK_T_ZONE_CLOSE, VIRTIO_BLK_T_ZONE_FINISH, VIRTIO_BLK_T_ZONE_APPEND, VIRTIO_BLK_T_ZONE_RESET or VIRTIO_BLK_T_ZONE_RESET_ALL may be completed by the device with VIRTIO_BLK_S_OK, VIRTIO_BLK_S_IOERR or VIRTIO_BLK_S_UNSUPP *status*, or, additionally, with VIRTIO_BLK_S_ZONE_INVALID_CMD, VIRTIO_BLK_S_ZONE_UNALIGNED_WP, VIRTIO_BLK_S_ZONE_OPEN_RESOURCE or VIRTIO_BLK_S_ZONE_ACTIVE_RESOURCE ZBD-specific status codes.

Besides the request status, VIRTIO_BLK_T_ZONE_APPEND requests return the starting sector of the appended data back to the driver. For this reason, the VIRTIO_BLK_T_ZONE_APPEND request has the layout that is extended to have the *append_sector* field to carry this value:

```
struct virtio_blk_req_za {
        le32 type;
        le32 reserved;
        le64 sector;
        u8 data[];
        le64 append_sector;
        u8 status;
};
```

```
#define VIRTIO_BLK_S_ZONE_INVALID_CMD     3
#define VIRTIO_BLK_S_ZONE_UNALIGNED_WP    4
#define VIRTIO_BLK_S_ZONE_OPEN_RESOURCE   5
#define VIRTIO_BLK_S_ZONE_ACTIVE_RESOURCE 6
```

Requests of the type VIRTIO_BLK_T_ZONE_REPORT are reads and requests of the type VIRTIO_BLK_T_ZONE_APPEND are writes. VIRTIO_BLK_T_ZONE_OPEN, VIRTIO_BLK_T_ZONE_CLOSE, VIRTIO_BLK_T_ZONE_FINISH, VIRTIO_BLK_T_ZONE_RESET and VIRTIO_BLK_T_ZONE_RESET_ALL are non-data requests.

Zone sector address is a 64-bit address of the first 512-byte sector of the zone.

VIRTIO_BLK_T_ZONE_OPEN, VIRTIO_BLK_T_ZONE_CLOSE, VIRTIO_BLK_T_ZONE_FINISH and VIRTIO_BLK_T_ZONE_RESET requests make the zone operation to act on a particular zone specified by the zone sector address in the *sector* of the request.

VIRTIO_BLK_T_ZONE_RESET_ALL request acts upon all applicable zones of the device. The *sector* value is not used for this request.

In ZBD standards, the VIRTIO_BLK_T_ZONE_REPORT request belongs to ”Zone Management Receive” command category and VIRTIO_BLK_T_ZONE_OPEN, VIRTIO_BLK_T_ZONE_CLOSE, VIRTIO_BLK_T_ZONE_FINISH and VIRTIO_BLK_T_ZONE_RESET/VIRTIO_BLK_T_ZONE_RESET_ALL requests are categorized as ”Zone Management Send” commands. VIRTIO_BLK_T_ZONE_APPEND is categorized separately from zone management commands and is the only request that uses the *append_secctor* field *virtio_blk_req_za* to return to the driver the sector at which the data has been appended to the zone.

VIRTIO_BLK_T_ZONE_REPORT is a read request that returns the information about the current state of zones on the device starting from the zone containing the *sector* of the request. The report consists of a header followed by zero or more zone descriptors.

A zone report reply has the following structure:

```
struct virtio_blk_zone_report {
        le64   nr_zones;
        u8     reserved[56];
        struct virtio_blk_zone_descriptor zones[];
};
```

The device sets the *nr_zones* field in the report header to the number of fully transferred zone descriptors in the data buffer.

A zone descriptor has the following structure:

```
struct virtio_blk_zone_descriptor {
        le64   z_cap;
        le64   z_start;
        le64   z_wp;
        u8     z_type;
        u8     z_state;
        u8     reserved[38];
};
```

The zone descriptor field *z_type virtio_blk_zone_descriptor* indicates the type of the zone.

The following zone types are available:

```
#define VIRTIO_BLK_ZT_CONV     1
#define VIRTIO_BLK_ZT_SWR      2
#define VIRTIO_BLK_ZT_SWP      3
```

Read and write operations into zones with the VIRTIO_BLK_ZT_CONV (Conventional) type have the same behavior as read and write operations on a regular block device. Any block in a conventional zone can be read or written at any time and in any order.

Zones with VIRTIO_BLK_ZT_SWR can be read randomly, but must be written sequentially at a certain point in the zone called the Write Pointer (WP). With every write, the Write Pointer is incremented by the number of sectors written.

Zones with VIRTIO_BLK_ZT_SWP can be read randomly and should be written sequentially, similarly to SWR zones. However, SWP zones can accept random write operations, that is, VIRTIO_BLK_T_OUT requests with a start sector different from the zone write pointer position.

The field *z_state* of *virtio_blk_zone_descriptor* indicates the state of the device zone.

The following zone states are available:

```
#define VIRTIO_BLK_ZS_NOT_WP   0
#define VIRTIO_BLK_ZS_EMPTY    1
#define VIRTIO_BLK_ZS_IOPEN    2
#define VIRTIO_BLK_ZS_EOPEN    3
#define VIRTIO_BLK_ZS_CLOSED   4
#define VIRTIO_BLK_ZS_RDONLY   13
#define VIRTIO_BLK_ZS_FULL     14
#define VIRTIO_BLK_ZS_OFFLINE  15
```

Zones of the type VIRTIO_BLK_ZT_CONV are always reported by the device to be in the VIRTIO_BLK_ZS_-NOT_WP state. Zones of the types VIRTIO_BLK_ZT_SWR and VIRTIO_BLK_ZT_SWP can not transition to the VIRTIO_BLK_ZS_NOT_WP state.

Zones in VIRTIO_BLK_ZS_EMPTY (Empty), VIRTIO_BLK_ZS_IOPEN (Implicitly Open), VIRTIO_BLK_-ZS_EOPEN (Explicitly Open) and VIRTIO_BLK_ZS_CLOSED (Closed) state are writable, but zones in VIRTIO_BLK_ZS_RDONLY (Read-Only), VIRTIO_BLK_ZS_FULL (Full) and VIRTIO_BLK_ZS_OFFLINE (Offline) state are not. The write pointer value (*z_wp*) is not valid for Read-Only, Full and Offline zones.

The zone descriptor field *z_cap* contains the maximum number of 512-byte sectors that are available to be written with user data when the zone is in the Empty state. This value shall be less than or equal to the *zone_sectors* value in *virtio_blk_zoned_characteristics* structure in the device configuration space.

The zone descriptor field *z_start* contains the zone sector address.

The zone descriptor field *z_wp* contains the sector address where the next write operation for this zone should be issued. This value is undefined for conventional zones and for zones in VIRTIO_BLK_ZS_-RDONLY, VIRTIO_BLK_ZS_FULL and VIRTIO_BLK_ZS_OFFLINE state.

Depending on their state, zones consume resources as follows:

- a zone in VIRTIO_BLK_ZS_IOPEN and VIRTIO_BLK_ZS_EOPEN state consumes one open zone resource and, additionally,
- a zone in VIRTIO_BLK_ZS_IOPEN, VIRTIO_BLK_ZS_EOPEN and VIRTIO_BLK_ZS_CLOSED state consumes one active resource.

Attempts for zone transitions that violate zone resource limits must fail with VIRTIO_BLK_S_ZONE_OPEN_-RESOURCE or VIRTIO_BLK_S_ZONE_ACTIVE_RESOURCE *status*.

Zones in the VIRTIO_BLK_ZS_EMPTY (Empty) state have the write pointer value equal to the sector address of the zone. In this state, the entire capacity of the zone is available for writing. A zone can transition from this state to

- VIRTIO_BLK_ZS_IOPEN when a successful VIRTIO_BLK_T_OUT request or VIRTIO_BLK_T_ZONE_-APPEND with a non-zero data size is received for the zone.
- VIRTIO_BLK_ZS_EOPEN when a successful VIRTIO_BLK_T_ZONE_OPEN request is received for the zone

When a VIRTIO_BLK_T_ZONE_RESET request is issued to an Empty zone, the request is completed successfully and the zone stays in the VIRTIO_BLK_ZS_EMPTY state.

Zones in the VIRTIO_BLK_ZS_IOPEN (Implicitly Open) state transition from this state to

- VIRTIO_BLK_ZS_EMPTY when a successful VIRTIO_BLK_T_ZONE_RESET request is received for the zone,
- VIRTIO_BLK_ZS_EMPTY when a successful VIRTIO_BLK_T_ZONE_RESET_ALL request is received by the device,
- VIRTIO_BLK_ZS_EOPEN when a successful VIRTIO_BLK_T_ZONE_OPEN request is received for the zone,
- VIRTIO_BLK_ZS_CLOSED when a successful VIRTIO_BLK_T_ZONE_CLOSE request is received for the zone,
- VIRTIO_BLK_ZS_CLOSED implicitly by the device when another zone is entering the VIRTIO_BLK_-ZS_IOPEN or VIRTIO_BLK_ZS_EOPEN state and the number of currently open zones is at *max_-open_zones* limit,
- VIRTIO_BLK_ZS_FULL when a successful VIRTIO_BLK_T_ZONE_FINISH request is received for the zone.
- VIRTIO_BLK_ZS_FULL when a successful VIRTIO_BLK_T_OUT or VIRTIO_BLK_T_ZONE_APPEND request that causes the zone to reach its writable capacity is received for the zone.

Zones in the VIRTIO_BLK_ZS_EOPEN (Explicitly Open) state transition from this state to

- VIRTIO_BLK_ZS_EMPTY when a successful VIRTIO_BLK_T_ZONE_RESET request is received for the zone,
- VIRTIO_BLK_ZS_EMPTY when a successful VIRTIO_BLK_T_ZONE_RESET_ALL request is received by the device,
- VIRTIO_BLK_ZS_EMPTY when a successful VIRTIO_BLK_T_ZONE_CLOSE request is received for the zone and the write pointer of the zone has the value equal to the start sector of the zone,
- VIRTIO_BLK_ZS_CLOSED when a successful VIRTIO_BLK_T_ZONE_CLOSE request is received for the zone and the zone write pointer is larger then the start sector of the zone,
- VIRTIO_BLK_ZS_FULL when a successful VIRTIO_BLK_T_ZONE_FINISH request is received for the zone,
- VIRTIO_BLK_ZS_FULL when a successful VIRTIO_BLK_T_OUT or VIRTIO_BLK_T_ZONE_APPEND request that causes the zone to reach its writable capacity is received for the zone.

When a VIRTIO_BLK_T_ZONE_EOPEN request is issued to an Explicitly Open zone, the request is completed successfully and the zone stays in the VIRTIO_BLK_ZS_EOPEN state.

Zones in the VIRTIO_BLK_ZS_CLOSED (Closed) state transition from this state to

- VIRTIO_BLK_ZS_EMPTY when a successful VIRTIO_BLK_T_ZONE_RESET request is received for the zone,

- VIRTIO_BLK_ZS_EMPTY when a successful VIRTIO_BLK_T_ZONE_RESET_ALL request is received by the device,
- VIRTIO_BLK_ZS_IOPEN when a successful VIRTIO_BLK_T_OUT request or VIRTIO_BLK_T_ZONE_-APPEND with a non-zero data size is received for the zone.
- VIRTIO_BLK_ZS_EOPEN when a successful VIRTIO_BLK_T_ZONE_OPEN request is received for the zone,

When a VIRTIO_BLK_T_ZONE_CLOSE request is issued to a Closed zone, the request is completed successfully and the zone stays in the VIRTIO_BLK_ZS_CLOSED state.

Zones in the VIRTIO_BLK_ZS_FULL (Full) state transition from this state to VIRTIO_BLK_ZS_EMPTY when a successful VIRTIO_BLK_T_ZONE_RESET request is received for the zone or a successful VIRTIO_BLK_-T_ZONE_RESET_ALL request is received by the device.

When a VIRTIO_BLK_T_ZONE_FINISH request is issued to a Full zone, the request is completed successfully and the zone stays in the VIRTIO_BLK_ZS_FULL state.

The device may automatically transition zones to VIRTIO_BLK_ZS_RDONLY (Read-Only) or VIRTIO_-BLK_ZS_OFFLINE (Offline) state from any other state. The device may also automatically transition zones in the Read-Only state to the Offline state. Zones in the Offline state may not transition to any other state. Such automatic transitions usually indicate hardware failures. The previously written data may only be read from zones in the Read-Only state. Zones in the Offline state can not be read or written.

VIRTIO_BLK_S_ZONE_UNALIGNED_WP is set by the device when the request received from the driver attempts to perform a write to an SWR zone and at least one of the following conditions is met:

- the starting sector of the request is not equal to the current value of the zone write pointer.
- the ending sector of the request data multiplied by 512 is not a multiple of the value reported by the device in the field *write_granularity* in the device configuration space.

VIRTIO_BLK_S_ZONE_OPEN_RESOURCE is set by the device when a zone operation or write request received from the driver can not be handled without exceeding the *max_open_zones* limit value reported by the device in the configuration space.

VIRTIO_BLK_S_ZONE_ACTIVE_RESOURCE is set by the device when a zone operation or write request received from the driver can not be handled without exceeding the *max_active_zones* limit value reported by the device in the configuration space.

A zone transition request that leads to both the *max_open_zones* and the *max_active_zones* limits to be exceeded is terminated by the device with VIRTIO_BLK_S_ZONE_ACTIVE_RESOURCE *status* value.

The device reports all other error conditions related to zoned block model operation by setting the VIRTIO_-BLK_S_ZONE_INVALID_CMD value in *status* of *virtio_blk_req* structure.

5.2.6.1 Driver Requirements: Device Operation

The driver SHOULD check if the content of the *capacity* field has changed upon receiving a configuration change notification.

A driver MUST NOT submit a request which would cause a read or write beyond *capacity*.

A driver SHOULD accept the VIRTIO_BLK_F_RO feature if offered.

A driver MUST set *sector* to 0 for a VIRTIO_BLK_T_FLUSH request. A driver SHOULD NOT include any data in a VIRTIO_BLK_T_FLUSH request.

The length of *data* MUST be a multiple of 512 bytes for VIRTIO_BLK_T_IN and VIRTIO_BLK_T_OUT requests.

The length of *data* MUST be a multiple of the size of struct virtio_blk_discard_write_zeroes for VIRTIO_-BLK_T_DISCARD, VIRTIO_BLK_T_SECURE_ERASE and VIRTIO_BLK_T_WRITE_ZEROES requests.

The length of *data* MUST be 20 bytes for VIRTIO_BLK_T_GET_ID requests.

VIRTIO_BLK_T_DISCARD requests MUST NOT contain more than *max_discard_seg* struct virtio_blk_-discard_write_zeroes segments in *data*.

VIRTIO_BLK_T_SECURE_ERASE requests MUST NOT contain more than *max_secure_erase_seg* struct virtio_blk_discard_write_zeroes segments in *data*.

VIRTIO_BLK_T_WRITE_ZEROES requests MUST NOT contain more than *max_write_zeroes_seg* struct virtio_blk_discard_write_zeroes segments in *data*.

If the VIRTIO_BLK_F_CONFIG_WCE feature is negotiated, the driver MAY switch to writethrough or write-back mode by writing respectively 0 and 1 to the *writeback* field. After writing a 0 to *writeback*, the driver MUST NOT assume that any volatile writes have been committed to persistent device backend storage.

The *unmap* bit MUST be zero for discard commands. The driver MUST NOT assume anything about the data returned by read requests after a range of sectors has been discarded.

A driver MUST NOT assume that individual segments in a multi-segment VIRTIO_BLK_T_DISCARD or VIRTIO_BLK_T_WRITE_ZEROES request completed successfully, failed, or were processed by the device at all if the request failed with VIRTIO_BLK_S_IOERR.

The following requirements only apply if the VIRTIO_BLK_F_ZONED feature is negotiated.

A zone sector address provided by the driver MUST be a multiple of 512 bytes.

When forming a VIRTIO_BLK_T_ZONE_REPORT request, the driver MUST set a sector within the sector range of the starting zone to report to *sector* field. It MAY be a sector that is different from the zone sector address.

In VIRTIO_BLK_T_ZONE_OPEN, VIRTIO_BLK_T_ZONE_CLOSE, VIRTIO_BLK_T_ZONE_FINISH and VIR-TIO_BLK_T_ZONE_RESET requests, the driver MUST set *sector* field to point at the first sector in the target zone.

In VIRTIO_BLK_T_ZONE_RESET_ALL request, the driver MUST set the field *sector* to zero value.

The *sector* field of the VIRTIO_BLK_T_ZONE_APPEND request MUST specify the zone sector address of the zone to which data is to be appended at the position of the write pointer. The size of the data that is appended MUST be a multiple of *write_granularity* bytes and MUST NOT exceed the *max_append_sectors* value provided by the device in *virtio_blk_zoned_characteristics* configuration space structure.

Upon a successful completion of a VIRTIO_BLK_T_ZONE_APPEND request, the driver MAY read the starting sector location of the written data from the request field *append_sector*.

All VIRTIO_BLK_T_OUT requests issued by the driver to sequential zones and VIRTIO_BLK_T_ZONE_-APPEND requests MUST have:

1. the data size that is a multiple of the number of bytes reported by the device in the field *write_granularity* in the *virtio_blk_zoned_characteristics* configuration space structure.
2. the value of the field *sector* that is a multiple of the number of bytes reported by the device in the field *write_granularity* in the *virtio_blk_zoned_characteristics* configuration space structure.
3. the data size that will not exceed the writable zone capacity when its value is added to the current value of the write pointer of the zone.

5.2.6.2 Device Requirements: Device Operation

The device MAY change the content of the *capacity* field during operation of the device. When this happens, the device SHOULD trigger a configuration change notification.

A device MUST set the *status* byte to VIRTIO_BLK_S_IOERR for a write request if the VIRTIO_BLK_F_RO feature if offered, and MUST NOT write any data.

The device MUST set the *status* byte to VIRTIO_BLK_S_UNSUPP for discard, secure erase and write zeroes commands if any unknown flag is set. Furthermore, the device MUST set the *status* byte to VIRTIO_-BLK_S_UNSUPP for discard commands if the *unmap* flag is set.

For discard commands, the device MAY deallocate the specified range of sectors in the device backend storage.

For write zeroes commands, if the *unmap* is set, the device MAY deallocate the specified range of sectors in the device backend storage, as if the discard command had been sent. After a write zeroes command is completed, reads of the specified ranges of sectors MUST return zeroes. This is true independent of whether *unmap* was set or clear.

The device SHOULD clear the *write_zeroes_may_unmap* field of the virtio configuration space if and only if a write zeroes request cannot result in deallocating one or more sectors. The device MAY change the content of the field during operation of the device; when this happens, the device SHOULD trigger a configuration change notification.

A write is considered volatile when it is submitted; the contents of sectors covered by a volatile write are undefined in persistent device backend storage until the write becomes stable. A write becomes stable once it is completed and one or more of the following conditions is true:

1. neither VIRTIO_BLK_F_CONFIG_WCE nor VIRTIO_BLK_F_FLUSH feature were negotiated, but VIRTIO_BLK_F_FLUSH was offered by the device;
2. the VIRTIO_BLK_F_CONFIG_WCE feature was negotiated and the *writeback* field in configuration space was 0 **all the time between the submission of the write and its completion**;
3. a VIRTIO_BLK_T_FLUSH request is sent **after the write is completed** and is completed itself.

If the device is backed by persistent storage, the device MUST ensure that stable writes are committed to it, before reporting completion of the write (cases 1 and 2) or the flush (case 3). Failure to do so can cause data loss in case of a crash.

If the driver changes *writeback* between the submission of the write and its completion, the write could be either volatile or stable when its completion is reported; in other words, the exact behavior is undefined.

If VIRTIO_BLK_F_FLUSH was not offered by the device[5], the device MAY also commit writes to persistent device backend storage before reporting their completion. Unlike case 1, however, this is not an absolute requirement of the specification.

Note: An implementation that does not offer VIRTIO_BLK_F_FLUSH and does not commit completed writes will not be resilient to data loss in case of crashes. Not offering VIRTIO_BLK_F_FLUSH is an absolute requirement for implementations that do not wish to be safe against such data losses.

If the device is backed by storage providing lifetime metrics (such as eMMC or UFS persistent storage), the device SHOULD offer the VIRTIO_BLK_F_LIFETIME flag. The flag MUST NOT be offered if the device is backed by storage for which the lifetime metrics described in this document cannot be obtained or for which such metrics have no useful meaning. If the metrics are offered, the device MUST NOT send any reserved values, as defined in this specification.

Note: The device lifetime metrics *pre_eol_info*, *device_lifetime_est_a* and *device_lifetime_est_b* are discussed in the JESD84-B50 specification.

The complete JESD84-B50 is available at the JEDEC website (https://www.jedec.org) pursuant to JEDEC's licensing terms and conditions. This information is provided to simplfy passthrough implementations from eMMC devices.

If the VIRTIO_BLK_F_ZONED feature is not negotiated, the device MUST reject VIRTIO_BLK_T_ZONE_REPORT, VIRTIO_BLK_T_ZONE_OPEN, VIRTIO_BLK_T_ZONE_CLOSE, VIRTIO_BLK_T_ZONE_FINISH, VIRTIO_BLK_T_ZONE_APPEND, VIRTIO_BLK_T_ZONE_RESET and VIRTIO_BLK_T_ZONE_RESET_ALL requests with VIRTIO_BLK_S_UNSUPP status.

The following device requirements only apply if the VIRTIO_BLK_F_ZONED feature is negotiated.

If a request of type VIRTIO_BLK_T_ZONE_OPEN, VIRTIO_BLK_T_ZONE_CLOSE, VIRTIO_BLK_T_ZONE_FINISH or VIRTIO_BLK_T_ZONE_RESET is issued for a Conventional zone (type VIRTIO_BLK_ZT_CONV), the device MUST complete the request with VIRTIO_BLK_S_ZONE_INVALID_CMD *status*.

[5] Note that in this case, according to 5.2.5.2, the device will not have offered VIRTIO_BLK_F_CONFIG_WCE either.

If the zone specified by the VIRTIO_BLK_T_ZONE_APPEND request is not a SWR zone, then the request SHALL be completed with VIRTIO_BLK_S_ZONE_INVALID_CMD *status*.

The device handles a VIRTIO_BLK_T_ZONE_OPEN request by attempting to change the state of the zone with the *sector* address to VIRTIO_BLK_ZS_EOPEN. If the transition to this state can not be performed, the request MUST be completed with VIRTIO_BLK_S_ZONE_INVALID_CMD *status*. If, while processing this request, the available zone resources are insufficient, then the zone state does not change and the request MUST be completed with VIRTIO_BLK_S_ZONE_OPEN_RESOURCE or VIRTIO_BLK_S_ZONE_ACTIVE_RESOURCE value in the field *status*.

The device handles a VIRTIO_BLK_T_ZONE_CLOSE request by attempting to change the state of the zone with the *sector* address to VIRTIO_BLK_ZS_CLOSED. If the transition to this state can not be performed, the request MUST be completed with VIRTIO_BLK_S_ZONE_INVALID_CMD value in the field *status*.

The device handles a VIRTIO_BLK_T_ZONE_FINISH request by attempting to change the state of the zone with the *sector* address to VIRTIO_BLK_ZS_FULL. If the transition to this state can not be performed, the zone state does not change and the request MUST be completed with VIRTIO_BLK_S_ZONE_INVALID_CMD value in the field *status*.

The device handles a VIRTIO_BLK_T_ZONE_RESET request by attempting to change the state of the zone with the *sector* address to VIRTIO_BLK_ZS_EMPTY state. If the transition to this state can not be performed, the zone state does not change and the request MUST be completed with VIRTIO_BLK_S_ZONE_INVALID_CMD value in the field *status*.

The device handles a VIRTIO_BLK_T_ZONE_RESET_ALL request by transitioning all sequential device zones in VIRTIO_BLK_ZS_IOPEN, VIRTIO_BLK_ZS_EOPEN, VIRTIO_BLK_ZS_CLOSED and VIRTIO_BLK_ZS_FULL state to VIRTIO_BLK_ZS_EMPTY state.

Upon receiving a VIRTIO_BLK_T_ZONE_APPEND request or a VIRTIO_BLK_T_OUT request issued to a SWR zone in VIRTIO_BLK_ZS_EMPTY or VIRTIO_BLK_ZS_CLOSED state, the device attempts to perform the transition of the zone to VIRTIO_BLK_ZS_IOPEN state before writing data. This transition may fail due to insufficient open and/or active zone resources available on the device. In this case, the request MUST be completed with VIRTIO_BLK_S_ZONE_OPEN_RESOURCE or VIRTIO_BLK_S_ZONE_ACTIVE_RESOURCE value in the *status*.

If the *sector* field in the VIRTIO_BLK_T_ZONE_APPEND request does not specify the lowest sector for a zone, then the request SHALL be completed with VIRTIO_BLK_S_ZONE_INVALID_CMD value in *status*.

A VIRTIO_BLK_T_ZONE_APPEND request or a VIRTIO_BLK_T_OUT request that has the data range that exceeds the remaining writable capacity for the zone, then the request SHALL be completed with VIRTIO_BLK_S_ZONE_INVALID_CMD value in *status*.

If a request of the type VIRTIO_BLK_T_ZONE_APPEND is completed with VIRTIO_BLK_S_OK status, the field *append_sector* in *virtio_blk_req_za* MUST be set by the device to contain the first sector of the data written to the zone.

If a request of the type VIRTIO_BLK_T_ZONE_APPEND is completed with a status other than VIRTIO_BLK_S_OK, the value of *append_sector* field in *virtio_blk_req_za* is undefined.

A VIRTIO_BLK_T_ZONE_APPEND request that has the data size that exceeds *max_append_sectors* configuration space value, then,

- if *max_append_sectors* configuration space value is reported as zero by the device, the request SHALL be completed with VIRTIO_BLK_S_UNSUPP *status*.
- if *max_append_sectors* configuration space value is reported as a non-zero value by the device, the request SHALL be completed with VIRTIO_BLK_S_ZONE_INVALID_CMD *status*.

If a VIRTIO_BLK_T_ZONE_APPEND request, a VIRTIO_BLK_T_IN request or a VIRTIO_BLK_T_OUT request issued to a SWR zone has the range that has sectors in more than one zone, then the request SHALL be completed with VIRTIO_BLK_S_ZONE_INVALID_CMD value in the field *status*.

A VIRTIO_BLK_T_OUT request that has the *sector* value that is not aligned with the write pointer for the zone, then the request SHALL be completed with VIRTIO_BLK_S_ZONE_UNALIGNED_WP value in the field *status*.

In order to avoid resource-related errors while opening zones implicitly, the device MAY automatically transition zones in VIRTIO_BLK_ZS_IOPEN state to VIRTIO_BLK_ZS_CLOSED state.

All VIRTIO_BLK_T_OUT requests or VIRTIO_BLK_T_ZONE_APPEND requests issued to a zone in the VIRTIO_BLK_ZS_RDONLY state SHALL be completed with VIRTIO_BLK_S_ZONE_INVALID_CMD *status*.

All requests issued to a zone in the VIRTIO_BLK_ZS_OFFLINE state SHALL be completed with VIRTIO_-BLK_S_ZONE_INVALID_CMD value in the field *status*.

The device MUST consider the sectors that are read between the write pointer position of a zone and the end of the last sector of the zone as unwritten data. The sectors between the write pointer position and the end of the last sector within the zone capacity during VIRTIO_BLK_T_ZONE_FINISH request processing are also considered unwritten data.

When unwritten data is present in the sector range of a read request, the device MUST process this data in one of the following ways -

1. Fill the unwritten data with a device-specific byte pattern. The configuration, control and reporting of this byte pattern is beyond the scope of this standard. This is the preferred approach.
2. Fail the request. Depending on the driver implementation, this may prevent the device from becoming operational.

If both the VIRTIO_BLK_F_ZONED and VIRTIO_BLK_F_SECURE_ERASE features are negotiated, then

1. the field *secure_erase_sector_alignment* in the configuration space of the device MUST be a multiple of *zone_sectors* value reported in the device configuration space.
2. the data size in VIRTIO_BLK_T_SECURE_ERASE requests MUST be a multiple of *zone_sectors* value in the device configuration space.

The device MUST handle a VIRTIO_BLK_T_SECURE_ERASE request in the same way it handles VIRTIO_BLK_T_ZONE_RESET request for the zone range specified in the VIRTIO_BLK_T_SECURE_ERASE request.

5.2.6.3 Legacy Interface: Device Operation

When using the legacy interface, transitional devices and drivers MUST format the fields in struct virtio_blk_-req according to the native endian of the guest rather than (necessarily when not using the legacy interface) little-endian.

When using the legacy interface, transitional drivers SHOULD ignore the used length values.

Note: Historically, some devices put the total descriptor length, or the total length of device-writable buffers there, even when only the status byte was actually written.

The *reserved* field was previously called *ioprio*. *ioprio* is a hint about the relative priorities of requests to the device: higher numbers indicate more important requests.

```
#define VIRTIO_BLK_T_FLUSH_OUT    5
```

The command VIRTIO_BLK_T_FLUSH_OUT was a synonym for VIRTIO_BLK_T_FLUSH; a driver MUST treat it as a VIRTIO_BLK_T_FLUSH command.

```
#define VIRTIO_BLK_T_BARRIER     0x80000000
```

If the device has VIRTIO_BLK_F_BARRIER feature the high bit (VIRTIO_BLK_T_BARRIER) indicates that this request acts as a barrier and that all preceding requests SHOULD be complete before this one, and all following requests SHOULD NOT be started until this is complete.

Note: A barrier does not flush caches in the underlying backend device in host, and thus does not serve as data consistency guarantee. Only a VIRTIO_BLK_T_FLUSH request does that.

Some older legacy devices did not commit completed writes to persistent device backend storage when VIRTIO_BLK_F_FLUSH was offered but not negotiated. In order to work around this, the driver MAY set the *writeback* to 0 (if available) or it MAY send an explicit flush request after every completed write.

If the device has VIRTIO_BLK_F_SCSI feature, it can also support scsi packet command requests, each of these requests is of form:

```
/* All fields are in guest's native endian. */
struct virtio_scsi_pc_req {
        u32 type;
        u32 ioprio;
        u64 sector;
        u8 cmd[];
        u8 data[][512];
#define SCSI_SENSE_BUFFERSIZE   96
        u8 sense[SCSI_SENSE_BUFFERSIZE];
        u32 errors;
        u32 data_len;
        u32 sense_len;
        u32 residual;
        u8 status;
};
```

A request type can also be a scsi packet command (VIRTIO_BLK_T_SCSI_CMD or VIRTIO_BLK_T_SCSI_-CMD_OUT). The two types are equivalent, the device does not distinguish between them:

```
#define VIRTIO_BLK_T_SCSI_CMD     2
#define VIRTIO_BLK_T_SCSI_CMD_OUT 3
```

The *cmd* field is only present for scsi packet command requests, and indicates the command to perform. This field MUST reside in a single, separate device-readable buffer; command length can be derived from the length of this buffer.

Note that these first three (four for scsi packet commands) fields are always device-readable: *data* is either device-readable or device-writable, depending on the request. The size of the read or write can be derived from the total size of the request buffers.

sense is only present for scsi packet command requests, and indicates the buffer for scsi sense data.

data_len is only present for scsi packet command requests, this field is deprecated, and SHOULD be ignored by the driver. Historically, devices copied data length there.

sense_len is only present for scsi packet command requests and indicates the number of bytes actually written to the *sense* buffer.

residual field is only present for scsi packet command requests and indicates the residual size, calculated as data length - number of bytes actually transferred.

5.2.6.4 Legacy Interface: Framing Requirements

When using legacy interfaces, transitional drivers which have not negotiated VIRTIO_F_ANY_LAYOUT:

- MUST use a single 8-byte descriptor containing *type*, *reserved* and *sector*, followed by descriptors for *data*, then finally a separate 1-byte descriptor for *status*.
- For SCSI commands there are additional constraints. *sense* MUST reside in a single separate device-writable descriptor of size 96 bytes, and *errors*, *data_len*, *sense_len* and *residual* MUST reside a single separate device-writable descriptor.

See 2.7.4.

5.3 Console Device

The virtio console device is a simple device for data input and output. A device MAY have one or more ports. Each port has a pair of input and output virtqueues. Moreover, a device has a pair of control IO