BASICS
„A block driver provides access to devices that transfer **randomly accessible data** in fixed-size **blocks** – disk drives, primarily. The Linux kernel sees block devices as being fundamentally **different from char devices**; as a result, block drivers have a **distinct interface** and their own particular challenges.“

*O’Reilly Linux Device Drivers, 3rd Edition*
DATA STRUCTURES
REGISTRATION
BLOCK DRIVER REGISTRATION

<linux/fs.h>

/*
 * major: major number of the device, if 0 is passed into register_blkdev
 * new major number will be allocated
 * name: associated device name (e.g. shown in /proc/devices)
 */

int register_blkdev(unsigned int major, const char *name);
void unregister_blkdev(unsigned int major, const char *name);
THAT’S IT?
<LINUX/GENHD.BLK>
my_blk_dev.mbd = alloc_disk(MINOR);
if (! my_blk_dev.mbd) {
    /*
    * error handling goes here
    */
}
my_blk_dev.mbd->major = major;
my_blk_dev.mbd->first_minor = RB_FIRST_MINOR;
my_blk_dev.mbd->fops = &mbd_fops;
my_blk_dev.mbd->private_data = &mbd_dev;
my_blk_dev.mbd->queue = mbd_dev.mbd_queue;
set_capacity(my_blk_dev.mbd, nsectors*(HARD_SECTOR_SIZE/KERNEL_SECTOR_SIZE));
add_disk(my_blk_dev.mbd);
OPERATIONS
BLOCK_DEVICE_OPERATIONS ≈ FILE_OPERATIONS
struct block_device_operations {
    int (*open) (struct block_device *, fmode_t);
    void (*release) (struct gendisk *, fmode_t);
    //implements ioctl system call
    int (*ioctl) (struct block_device *, fmode_t, unsigned, unsigned long);
    //get geometry of the device
    int (*getgeo) (struct block_device *, struct hd_geometry *);
    struct module *owner;
};
IOCTL
<DRIVERS/BLOCK/IOCTL.C>

/ Today mainly provides the devices geometry
/ Amount of heads, cylinders, sectors, starting point
/ Necessary for e.g. fdisk to work properly
REQUEST PROCESSING
READ/WRITE REQUEST
<LINUX/BLKDEV.H>

Diagram:

- MY_BLK_DEV
  - GENDISC
  - BLOCK_DEVICE_OPERATIONS
  - REQUEST_QUEUE
    - REQUEST
      - BIO
REQUEST QUEUE
SET IT UP

#include <linux/blkdev.h>

/*
 * request: request function
 * lock: spinlock
 */

//request_queue_t is used to manage request queues
request_queue_t *blk_init_queue(request_fn Proc *request,
                                spinlock_t *lock);
void blk_cleanup_queue(request_queue_t *);
struct request *blk_fetch_request(request_queue_t *queue);
/*
 * blk_dequeue_request and blk_requeue_request allow you to remove and add
 * back requests from/to the queue
 */
void blk_dequeue_request(struct request *req);
void blk_requeue_request(request_queue_t *queue, struct request *req);
void blk_stop_queue(request_queue_t *queue);
void blk_start_queue(request_queue_t *queue);
REQUEST QUEUE
SKY ISN’T THE LIMIT

<linux/blkdev.h>

/*
 * DMA: Direct Memory Access bypassing the CPU
 * dma_addr: highest physical address to which a device can perform DMA
 * This information is needed by the kernel to be able to create a
 * bounce buffer if necessary
 * Those operations are way more expensive in terms of performance
 */

void blk_queue_bounce_limit(request_queue_t *queue, u64 dma_addr);
<linux/blkdev.h>
/*
 * return:  -true   all sectors have been transferred and request is completed
 *          -false  still pending
 */
bool __blk_end_request_cur(struct request *req, int error);
REQUEST METHOD
DO IT

<línux/blkdev.h>

```c
void request(request_queue_t *queue);
```
<LINUX/BLKDEV.H>
REQUEST
SET IT UP

<linux/blkdev.h>
/*
 * sector: index of the sector which represents the starting point of our
 * requests operation
 * nr_sectors: number of sectors to be transferred
 * *buffer: pointer to the buffer representing the data source or target
 * rq_data_dir: determines between read and write operations, returns
 *   - zero for read
 *   - non zero for for write
 */

sector_t sector;
unsigned long nr_sectors;
char *buffer;
rq_data_dir(struct request *req);
MY_BLK_DEV

GENDISC

BLOCK_DEVICE_OPERATIONS

REQUEST_QUEUE

REQUEST

BIO
BIO = BLOCKING I/O
<LINUX/BIO.H>

/ describes an blocking I/O request
/ several bios are merged into a request
/ contains (multiple) page(s) of data; each represented as bio_vec which contains
  the length in bytes and the offset of the buffer in the page
<LINUX/BIO.H>
SOURCES

/ O'Reilly Linux Device Drivers, 3rd Edition Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman

/ tldp.org
http://www.tldp.org/LDP/tlk/dd/drivers.html

/ sysplay.in (Demo)
https://sysplay.in/blog/linux-device-drivers/2014/04/disk-on-ram-playing-destructively/