```c
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <asm/uaccess.h>

// define drivername and amount of minors
#define DRIVER_NAME "template"
#define MINORS_COUNT 1

static int driver_open(struct inode *geraetedatei, struct file *instanz);
static int driver_release(struct inode *geraetedatei, struct file *instanz);
static ssize_t driver_read(struct file *instanz, char *user, size_t count, loff_t *
offset);
static ssize_t driver_write(struct file *instanz, const char *user, size_t count, l
off_t *offset);

// map file operations to functions
static struct file_operations fops = {
    .read = driver_read,
    .owner= THIS_MODULE,
    .open = driver_open,
    .release = driver_release,
    .write = driver_write,
};

static struct cdev *driver_object;
static dev_t device_number;
static struct class *template_class;

// starting point (insmod)
static int __init ModInit(void) {
    int major;

     /* alloc_chrdev_region
```

```
        Arguments:  dev:        output parameter for first assigned number
                    baseminor:  first of the requested range of minor numbers
                    count:      the number of minor numbers required
                    name:       the name of the associated device or driver
        Description:    Allocates a range of char device numbers.
                        The major number will be chosen dynamically,
                        and returned (along with the first minor number) in dev.
                        returns zero or a negative error code. */
    if(alloc_chrdev_region(&device_number, 0, MINORS_COUNT, DRIVER_NAME) < 0) {
        printk("Devicenumber 0x%x not available ...\n", device_number );
        return -EIO;
    }

    /* get some memory for cdev driver structure */
    driver_object = cdev_alloc();
    if(driver_object == NULL) {
        printk("cdev_alloc failed ...\n");
        goto free_device_number;
    }

    driver_object->ops = &fops;
    driver_object->owner = THIS_MODULE;

    /* Anmeldung beim Kernel */
    if( cdev_add( driver_object, device_number, MINORS_COUNT )) {
        printk("cdev_add failed ...\n");
        goto free_cdev;
    }

    /* Eintrag im Sysfs*/
    template_class = class_create(THIS_MODULE, DRIVER_NAME);
    /* Erzeugung der Gerätedatei mittels sysfs eintrag */
    device_create(template_class, NULL, device_number, NULL, "%s", DRIVER_NAME);

    major = MAJOR(device_number);
    printk("Major number: %d\n", major);
    return 0;
```

```
free_cdev:
    kobject_put(&driver_object->kobj);
    driver_object = NULL;

free_device_number:
    unregister_chrdev_region( device_number, MINORS_COUNT );
    return -EIO;
}

static int driver_open(struct inode *geraetedatei, struct file *instanz) {
    printk("Open from Minor %d\n", iminor(geraetedatei));
    return 0;
}

static ssize_t driver_read(struct file *instanz, char *user, size_t count, loff_t *
offset) {
    char* string = "hey\n";
    printk("Read from Minor %d\n", iminor(instanz->f_path.dentry->d_inode));
    copy_to_user(user, string, strlen(string)+1);
    return strlen(string)+1;
}

static ssize_t driver_write(struct file *instanz, const char *user, size_t count, l
off_t *offset) {
    printk("Write to Minor %d", iminor(instanz->f_path.dentry->d_inode));
    return 0;
}

static int driver_release(struct inode *geraetedatei, struct file *instanz) {
    printk("Release from Minor %d", iminor(geraetedatei));
    return 0;
}

// ending point (rmmod)
static void __exit ModExit(void) {
    device_destroy(template_class, device_number);
```

```
    class_destroy(template_class);
    printk("trying to unregister 0x%x\n", device_number);
    cdev_del( driver_object );
    unregister_chrdev_region( device_number, 1 );
    printk("exiting\n");
}

// starting and ending points for insmod and rmmod
module_init(ModInit);
module_exit(ModExit);

// meta
MODULE_AUTHOR("Timo Weiß und Michael Knoch");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Just a Modul-Template, without specific functionality.");
MODULE_SUPPORTED_DEVICE("none");
```