# Part 29 – EXPORT\_SYMBOL in Linux Device Driver

[ <https://embetronicx.com/tutorials/linux/device-drivers/export_symbol-in-linux-device-driver/> ]

When you are writing multiple drivers (modules) in the same device, you may wanted to use some of the functions form one module to another module. How will we do that? If we use only **extern**then it won’t help you. We must have use some advance thing. So, We have to tell the kernel, that I want to share this function to other modules.

For example, take **printk()** function. This function will be defined in **source/kernel/printk/printk.c**. Then how can we able to access that **printk()** in our driver?

In this article we will see how to do it.

# EXPORT\_SYMBOL in Linux Device Driver

## Introduction

In programming language, a symbol is either a variable or a function. Or more generally, we can say, a symbol is a name representing an space in the memory, which stores data (variable, for reading and writing) or instructions (function, for executing).

When you look at some kernel codes, you may find **EXPORT\_SYMBOL()** very often. Have you wondered any time what the heck is that?

In the Linux Kernel 2.4, all the non-static symbols are exported to the kernel space automatically. But later, in Linux Kernel 2.6 instead of exporting all non-static symbols, they wanted to export the only symbols which is marked by **EXPORT\_SYMBOL()** macro.

## EXPORT\_SYMBOL’s role

When some symbols (variables or functions) are using **EXPORT\_SYMBOL**macro (ex. **EXPORT\_SYMBOL(func\_name)**), those symbols are exposed to all the loadable kernel driver. You can call them directly in your kernel module without modifying the kernel code. In other words, It tells the **kbuild**mechanism that the symbol referred to should be part of the global list of kernel symbols. That allows the kernel modules to access them.

Only the symbols that have been explicitly exported can by used by other modules.

Another macro is also available to export the symbols like **EXPORT\_SYMBOL**. That is **EXPORT\_SYMBOL\_GPL()**.

**EXPORT\_SYMBOL** exports the symbol to any loadable module.  
**EXPORT\_SYMBOL\_GPL**exports the symbol only to GPL-licensed modules.

## How to use EXPORT\_SYMBOL?

* Declare and define the symbol (functions or variables) which you want to make it visible to other kernel modules. Then below the definition, use **EXPORT\_SYMBOL(symbol name)**. Now it is visible to all loadable modules.
* Now take the kernel driver who is gonna use the above exported symbol. Declare the symbol using **extern**. Then use the symbol directly.
* Finally, load the module first, who has the definition of the export symbol. Then load the caller module using **insmod**“.

## Limitation

* That symbol should not be **static** or **inline.**
* Order of loading the driver matters. ie. We should load the module which has the definition of the symbol, then only we can load the module who is using that symbol.

# Driver Source Code – EXPORT\_SYMBOL in Linux

First I will explain you the concept of driver code attached below.

In this tutorial we have two drivers.

***Driver 1*** has one function called **etx\_shared\_func**and one global variable called **etx\_count.**This function and variable has been shared among with all the loadable modules using **EXPORT\_SYMBOL.**

***Driver 2*** will be using that variable and function which are shared by Driver1. When we read this Driver 2, then it will call the shared function and we can read the variable too.

## Driver1.c

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kdev\_t.h>

#include <linux/fs.h>

#include <linux/cdev.h>

#include <linux/device.h>

dev\_t dev = 0;

static struct class \*dev\_class;

static struct cdev etx\_cdev;

static int \_\_init etx\_driver\_init(void);

static void \_\_exit etx\_driver\_exit(void);

static int etx\_open(struct inode \*inode, struct file \*file);

static int etx\_release(struct inode \*inode, struct file \*file);

static ssize\_t etx\_read(struct file \*filp, char \_\_user \*buf, size\_t len,loff\_t \* off);

static ssize\_t etx\_write(struct file \*filp, const char \*buf, size\_t len, loff\_t \* off);

int etx\_count = 0;

void etx\_shared\_func(void)

{

printk(KERN\_INFO "Shared function been called!!!\n");

etx\_count++;

}

//EXPORT\_SYMBOL\_GPL(etx\_shared\_func);

EXPORT\_SYMBOL(etx\_shared\_func);

EXPORT\_SYMBOL(etx\_count);

static struct file\_operations fops =

{

.owner = THIS\_MODULE,

.read = etx\_read,

.write = etx\_write,

.open = etx\_open,

.release = etx\_release,

};

static int etx\_open(struct inode \*inode, struct file \*file)

{

printk(KERN\_INFO "Device File Opened...!!!\n");

return 0;

}

static int etx\_release(struct inode \*inode, struct file \*file)

{

printk(KERN\_INFO "Device File Closed...!!!\n");

return 0;

}

static ssize\_t etx\_read(struct file \*filp, char \_\_user \*buf, size\_t len, loff\_t \*off)

{

printk(KERN\_INFO "Data Read : Done!\n");

return 1;

}

static ssize\_t etx\_write(struct file \*filp, const char \_\_user \*buf, size\_t len, loff\_t \*off)

{

printk(KERN\_INFO "Data Write : Done!\n");

return len;

}

static int \_\_init etx\_driver\_init(void)

{

/\*Allocating Major number\*/

if((alloc\_chrdev\_region(&dev, 0, 1, "etx\_Dev1")) <0){

printk(KERN\_INFO "Cannot allocate major number\n");

return -1;

}

printk(KERN\_INFO "Major = %d Minor = %d \n",MAJOR(dev), MINOR(dev));

/\*Creating cdev structure\*/

cdev\_init(&etx\_cdev,&fops);

/\*Adding character device to the system\*/

if((cdev\_add(&etx\_cdev,dev,1)) < 0){

printk(KERN\_INFO "Cannot add the device to the system\n");

goto r\_class;

}

/\*Creating struct class\*/

if((dev\_class = class\_create(THIS\_MODULE,"etx\_class1")) == NULL){

printk(KERN\_INFO "Cannot create the struct class\n");

goto r\_class;

}

/\*Creating device\*/

if((device\_create(dev\_class,NULL,dev,NULL,"etx\_device1")) == NULL){

printk(KERN\_INFO "Cannot create the Device 1\n");

goto r\_device;

}

printk(KERN\_INFO "Device Driver 1 Insert...Done!!!\n");

return 0;

r\_device:

class\_destroy(dev\_class);

r\_class:

unregister\_chrdev\_region(dev,1);

return -1;

}

void \_\_exit etx\_driver\_exit(void)

{

device\_destroy(dev\_class,dev);

class\_destroy(dev\_class);

cdev\_del(&etx\_cdev);

unregister\_chrdev\_region(dev, 1);

printk(KERN\_INFO "Device Driver 1 Remove...Done!!!\n");

}

module\_init(etx\_driver\_init);

module\_exit(etx\_driver\_exit);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("EmbeTronicX <embetronicx@gmail.com>");

MODULE\_DESCRIPTION("EXPORT\_SYMBOL Driver - 1");

MODULE\_VERSION("1.25");

## driver2.c

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kdev\_t.h>

#include <linux/fs.h>

#include <linux/cdev.h>

#include <linux/device.h>

dev\_t dev = 0;

static struct class \*dev\_class;

static struct cdev etx\_cdev;

static int \_\_init etx\_driver\_init(void);

static void \_\_exit etx\_driver\_exit(void);

static int etx\_open(struct inode \*inode, struct file \*file);

static int etx\_release(struct inode \*inode, struct file \*file);

static ssize\_t etx\_read(struct file \*filp, char \_\_user \*buf, size\_t len,loff\_t \* off);

static ssize\_t etx\_write(struct file \*filp, const char \*buf, size\_t len, loff\_t \* off);

extern int etx\_count;

void etx\_shared\_func(void); //Function declaration is by default extern

static struct file\_operations fops =

{

.owner = THIS\_MODULE,

.read = etx\_read,

.write = etx\_write,

.open = etx\_open,

.release = etx\_release,

};

static int etx\_open(struct inode \*inode, struct file \*file)

{

printk(KERN\_INFO "Device File Opened...!!!\n");

return 0;

}

static int etx\_release(struct inode \*inode, struct file \*file)

{

printk(KERN\_INFO "Device File Closed...!!!\n");

return 0;

}

static ssize\_t etx\_read(struct file \*filp, char \_\_user \*buf, size\_t len, loff\_t \*off)

{

etx\_shared\_func();

printk(KERN\_INFO "%d time(s) shared function called!\n", etx\_count);

printk(KERN\_INFO "Data Read : Done!\n");

return 0;

}

static ssize\_t etx\_write(struct file \*filp, const char \_\_user \*buf, size\_t len, loff\_t \*off)

{

printk(KERN\_INFO "Data Write : Done!\n");

return len;

}

static int \_\_init etx\_driver\_init(void)

{

/\*Allocating Major number\*/

if((alloc\_chrdev\_region(&dev, 0, 1, "etx\_Dev2")) <0){

printk(KERN\_INFO "Cannot allocate major number\n");

return -1;

}

printk(KERN\_INFO "Major = %d Minor = %d \n",MAJOR(dev), MINOR(dev));

/\*Creating cdev structure\*/

cdev\_init(&etx\_cdev,&fops);

/\*Adding character device to the system\*/

if((cdev\_add(&etx\_cdev,dev,1)) < 0){

printk(KERN\_INFO "Cannot add the device to the system\n");

goto r\_class;

}

/\*Creating struct class\*/

if((dev\_class = class\_create(THIS\_MODULE,"etx\_class2")) == NULL){

printk(KERN\_INFO "Cannot create the struct class\n");

goto r\_class;

}

/\*Creating device\*/

if((device\_create(dev\_class,NULL,dev,NULL,"etx\_device2")) == NULL){

printk(KERN\_INFO "Cannot create the Device 1\n");

goto r\_device;

}

printk(KERN\_INFO "Device Driver 2 Insert...Done!!!\n");

return 0;

r\_device:

class\_destroy(dev\_class);

r\_class:

unregister\_chrdev\_region(dev,1);

return -1;

}

void \_\_exit etx\_driver\_exit(void)

{

device\_destroy(dev\_class,dev);

class\_destroy(dev\_class);

cdev\_del(&etx\_cdev);

unregister\_chrdev\_region(dev, 1);

printk(KERN\_INFO "Device Driver 2 Remove...Done!!!\n");

}

module\_init(etx\_driver\_init);

module\_exit(etx\_driver\_exit);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("EmbeTronicX <embetronicx@gmail.com>");

MODULE\_DESCRIPTION("EXPORT\_SYMBOL Driver - 2");

MODULE\_VERSION("1.26");

## MakeFile

obj-m += driver1.o

obj-m += driver2.o

KDIR = /lib/modules/$(shell uname -r)/build

all:

make -C $(KDIR) M=$(shell pwd) modules

clean:

make -C $(KDIR) M=$(shell pwd) clean

# Compiling and Testing Driver

* Build the driver by using Makefile (**sudo make**)
* After compiling, you can able to see the file named as “***Module.symvers***“. If you open that file, then our shared function and variable will be mentioned there.

***0x1db7034a       etx\_shared\_func         /home/embetronicx/driver/driver1            EXPORT\_SYMBOL******0x6dcb135c       etx\_count                    /home/embetronicx/driver/driver1             EXPORT\_SYMBOL***

* Load the driver 1 using **sudo insmod driver1.ko**(Driver 1 should be loaded first. If you try to load the Driver 2 first, then you will get an error like “***insmod: ERROR: could not insert module driver2.ko: Unknown symbol in module***“).
* Load the driver 1 using **sudo insmod driver2.ko**
* Now check the **dmesg**

*[ 393.814900] Major = 246 Minor = 0  
[ 393.818413] Device Driver 1 Insert…Done!!!  
[ 397.620296] Major = 245 Minor = 0  
[ 397.629002] Device Driver 2 Insert…Done!!!*

* Then do  **cat /proc/kallsyms | grep etx\_shared\_func**or**cat /proc/kallsyms | grep etx\_count** to check whether our shared function and variable become the part of kernel’s symbol table or not.
* Now we can read the driver by using **sudo cat /dev/etx\_device2**
* Now check the **dmesg**

*[ 403.739998] Device File Opened…!!!  
[ 403.740018] Shared function been called!!!  
[ 403.740021] 1 time(s) shared function called!  
[ 403.740023] Data Read : Done!  
[ 403.740028] Device File Closed…!!!*

* Now we can see the print from shared function and variable count also.
* Unload the module 2 using **sudo rmmod driver2**(Driver 2 should be unloaded first. If you unload the Driver 1 first, then you will get error like “***rmmod: ERROR: Module driver1 is in use by: driver2***“).
* Unload the module 1 using **sudo rmmod driver1**