

SECTION 3 Manage Hardware

Although most hardware devices can be configured with YaST or are even automatically detected when plugged into the system, it is sometimes helpful to understand how things work in the background.

In this section, you are introduced to the SUSE Linux Enterprise Server 10 hardware management and how device drivers are loaded.

You also learn how to add and replace certain types of hardware.

Objectives

1. Describe the Differences between Devices and Interfaces
2. Describe how Device Drivers Work
3. Describe how Device Drivers Are Loaded
4. Manage Kernel Modules Manually
5. Describe the sysfs File System
6. Describe how udev Works
7. Use the hwup Command

Objective 1 Describe the Differences between Devices and Interfaces

This objective uses the terms “device” and “interface.” These terms are often confused, not only by users and administrators but also by developers of operating systems and related tools.

This course uses the following definitions for device and interface:

- **Device.** A device is a real, physical piece of hardware. This can be a PCI network card, an AGP graphic adapter, a USB printer, or any kind of hardware that you can hold, feel, or break if you want to.
- **Interface.** An interface is a software component associated with a device. To use a physical piece of hardware, it needs to be accessed by a software interface.

A device can have more than one interface.

Interfaces are usually created by a driver. In Linux, a driver is usually a software module that can be loaded into the Linux kernel. Therefore, a driver can be seen as the glue between a device and its interfaces.

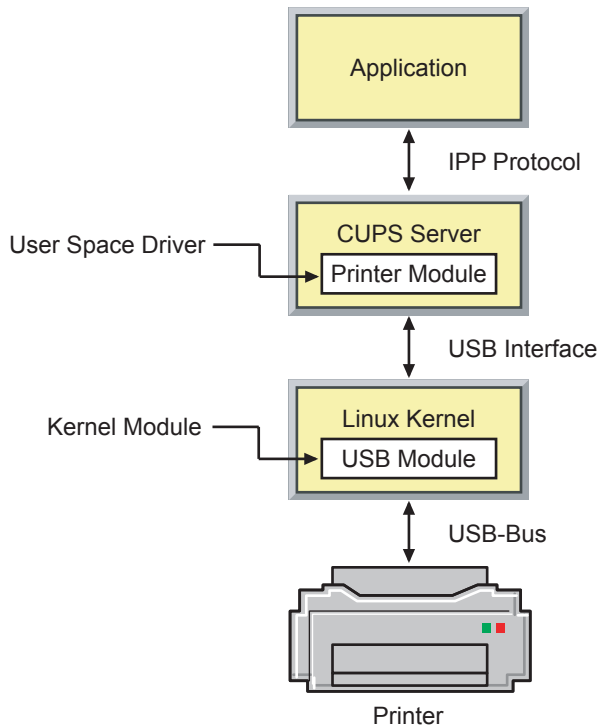
Objective 2 Describe how Device Drivers Work

As described before, device drivers access and use a device. There are 2 basic kinds of device drivers:

- **Kernel modules.** The functionality of the Linux kernel can be extended by kernel modules. These modules can be loaded and removed during runtime. They allow the kernel to provide access to hardware.
- **User space drivers.** Some hardware needs additional drivers that work in user space. Examples of this kind of hardware are printers or scanners.

The following illustrates the roles of kernel and user space drivers:

Figure 3-1



While the handling of user space drivers depends on the framework they are used in, you can manage kernel modules with the following commands:

- **lsmod.** This command lists all loaded kernel modules. For example:

lsmod

- **modprobe.** This command loads kernel modules. Because kernel modules can depend on each other, modprobe automatically resolves these dependencies and loads all required modules. For example:

modprobe usb-storage

In this example, modprobe loads the usb-storage module which is needed to access storage devices connected with the USB bus.

Because this module requires other USB modules, modprobe also loads these modules.

- **rmmod.** This command removes loaded kernel modules. For example:

rmmod usb-storage

Only modules that are not needed can be removed. In this example, the USB device first has to be disconnected before the usb-storage module can be removed.

Kernel modules are files that are stored in the directory */lib/modules/kernel-version/*.

Because modules normally work only with the kernel version they are built for, a new directory is created for every kernel update you install.

Modules are stored in several subdirectories with a filename extension of **.ko** for kernel object. When loading a module with `modprobe`, you can leave out the extension and use just the module name.

Objective 3 Describe how Device Drivers Are Loaded

Because it would be very inconvenient to load all kernel modules manually after every system start, there are several methods to perform this task automatically.

The following is an overview of how device drivers are loaded in SUSE Linux Enterprise Server 10:

- **initrd.** Important device drivers that are necessary to access the root partition are loaded from `initrd`. `initrd` is a special file that is loaded into memory by the boot loader. Examples of such modules are the SCSI host controller and file system drivers.
- **initscripts.** Some `initscripts` are dedicated to loading and setting up hardware devices, such as the ALSA sound script for sound cards.
- **udev.** `udev` also loads kernel modules with the `hwup` command.
- **X Server.** Although the graphics card drivers are not kernel modules, X Server loads special drivers to enable hardware 3D support.
- **manually.** You can always load kernel modules on the command line or in scripts with the `modprobe` command or with the `hwup` or `hwdown` commands.

Objective 4 **Manage Kernel Modules Manually**

Although SUSE Linux Enterprise Server 10 initializes most hardware devices automatically, it can be very helpful to know how kernel modules are managed manually.

To manage kernel modules, you need to understand the following:

- Kernel Module Basics
- Manage Modules from the Command Line
- modprobe Configuration File (*/etc/modprobe.conf*)



For the latest kernel documentation, see `/usr/src/linux/Documentation`.

Kernel Module Basics

The kernel that is installed in the directory `/boot/` is configured for a wide range of hardware. It is not necessary to compile a custom kernel, unless you want to test experimental features and drivers.

Drivers and features of the Linux kernel can either be compiled into the kernel or be loaded as kernel modules. These modules can be loaded later, while the system is running, without having to reboot the computer.

This is especially true of kernel modules that are not required to boot the system. By loading them as components after the system boots, the kernel can be kept relatively small.

The kernel modules are located in the directory `/lib/modules/version/kernel/`.

For example, the modules for the 2.6 kernel can be found in the following directory:

/lib/modules/2.6.16-0.12-default/kernel/

Manage Modules from the Command Line

The following are commands you can use from a command line when working with modules:

- **lsmod.** This command lists the currently loaded modules in the kernel.

The following is an example:

```
DA50:~ # lsmod
Module                Size  Used by
quota_v2              12928  2
edd                  13720  0
joydev               14528  0
sg                   41632  0
st                   44956  0
sr_mod               21028  0
ide_cd               42628  0
cdrom                42780  2 sr_mod,ide_cd
nvram                13448  0
usbserial            35952  0
parport_pc           41024  1
lp                   15364  0
parport              44232  2 parport_pc,lp
ipv6                 276348  44
uhci_hcd             35728  0
intel_agp            22812  1
agpgart              36140  1 intel_agp
evdev                13952  0
usbcore              116572  4 usbserial,uhci_hcd
```

The list includes information about the module name, size of the module, how often the module is used, and by which other modules use it.

- **insmod *module*.** This command loads the indicated *module* into the kernel.

The module must be stored in the directory `/lib/modules/version_number/`. However, it is recommended to use `modprobe` for loading modules.

- **rmmod *module*.** This command removes the indicated *module* from the kernel. However, it can only be removed if no processes are accessing hardware connected to it or corresponding services.

We recommend that you use **modprobe -r** for removing modules.

- **modprobe *module*.** This command loads the indicated *module* into the kernel or removes it (with option -r).

Dependencies of other modules are taken into account when using modprobe. In addition, modprobe reads in the file `/etc/modprobe.conf` for any configuration settings.

This command can only be used if the file `/lib/modules/version/modules.dep` created by the command `depmod` exists. This file is used to add or remove dependencies.

The kernel daemon (Kmod since kernel version 2.2.x) ensures that modules needed in the running operation are automatically loaded using modprobe (such as accessing the CD-ROM drive).



For more detailed information, enter **man modprobe**.

- **depmod.** This command creates the file `/lib/modules/version/modules.dep`. This file contains the dependencies of individual modules on each other.

When a module is loaded (such as with modprobe), `modules.dep` ensures that all modules it depends on are also loaded.

If the file `modules.dep` does not exist, it is created automatically when the system starts by the start script `/etc/init.d/boot`. For this reason, you do not need to create the file manually.

On SUSE Linux Enterprise Server 10, depmode also creates the file `modules.alias`, which is used by `hwup` and `modprobe` to determine which driver needs to be loaded for which device. Learn more about this file in the `hwup` objective in this section.

- **`modinfo option module`**. This command displays information (such as license, author, and description) about the module indicated on the command line.

The following is an example:

```
DA50:~ # modinfo isdn
license:      GPL
author:       Fritz Elfert
description:  ISDN4Linux: link layer
depends:       slhc
supported:    yes
vermagic:     2.6.5-7.21-default 586 REGPARM gcc-3.3
```



For more detailed information, enter **`man modinfo`**.

modprobe Configuration File (/etc/modprobe.conf)

The file `/etc/modprob.conf` is the configuration file for the kernel modules. For example, it contains parameters for the modules that access hardware directly.

The file plays an important role in loading modules. Various command types can be found in the file, such as the following:

- **install.** These instructions let `modprobe` execute commands when loading a specific module into the kernel.

The following is an example:

```
install      eth0      /bin/true
```

- **alias.** These instructions determine which kernel module will be loaded for a specific device file.

The following is an example:

```
alias      eth0      nvnet
```

- **options.** These instructions are options for loading a module.

The following is an example:

```
options      ne      io=0x300 irq=5
```



For more detailed information, enter **man 5 modprobe.conf**.

Exercise 3-1 Manage the Linux Kernel Modules

In this exercise, you load and unload kernel modules.

Do the following:

1. From a terminal window, su to root (**su -**) with a password of **novell**.
2. View the currently loaded kernel modules by entering **lsmod**.
3. Scroll through the modules to see if the joystick module (**joydev**) is loaded.

The 0 in the Used column indicates that the module is not in use.

4. Remove the joystick module from the kernel memory by entering **rmmod joydev**.
5. Verify that the joydev kernel module was removed from memory by entering **lsmod**.

Notice that the module joydev is no longer listed.

6. Load the joystick kernel module by entering **modprobe joydev**.
7. Verify that the joydev kernel module is loaded in memory by entering **lsmod**.
8. View the kernel modules configuration by entering the following:

modprobe -c | less

9. Scroll through the configuration information by pressing the **Spacebar**.
10. When you finish, return to the command line by typing **q**.
11. Create a list of kernel modules dependencies by entering **depmod -v | less**.

It takes a few moments for the information to be generated.

12. Scroll through the dependency information by pressing the **Spacebar**.
13. When you finish, return to the command line by typing **q**.
14. Close the terminal window by entering **exit** twice.

(End of Exercise)

Objective 5 Describe the sysfs File System

sysfs is a virtual file system that is mounted under `/sys`. In a virtual file system, there is no physical device that holds the information. Instead, the file system is generated virtually by the kernel.

The directory and file structure under `/sys`, provides information about the hardware which is currently connected with a system. Under `/sys`, there are 4 main directories:

- **`/sys/bus` and `/sys/devices`.** These directories contain different representations of system hardware. Devices are represented here.

For example, the following represents a digital camera connected to the USB bus:

```
/sys/bus/usb/devices/1-1/
```

This directory contains several files that provide information about the device. The following is a listing of the files in this directory:

```
1-1:1.0          bMaxPower      manufacturer
bcdDevice        bNumConfigurations  maxchild
bConfigurationValue  bNumInterfaces   power
bDeviceClass      detach_state    product
bDeviceProtocol    devnum          serial
bDeviceSubClass    idProduct       speed
bmAttributes       idVendor        version
```

For example, by reading the content from the `manufacturer` file, you can determine the manufacturer of the device:

```
cat manufacturer
OLYMPUS
```

In this case, an Olympus digital camera is connected with the system.

- **/sys/class** and **/sys/block**. The interfaces of the devices are represented under these 2 directories.

For example, the interface belonging to the Olympus digital camera is represented by the following directory:

```
/sys/block/sda/
```

The directory named `/sda` is the digital camera accessed like a SCSI hard disk.

The following is the content of the `/sda` directory:

```
dev      queue  removable  size
device  range  sda1       stat
```

The subdirectory `/sda1` represents the interface to the first partition on the cameras memory card. For example, by reading the content of `/sda1/size`, you can determine the size of the partition:

```
cat sda1/size
31959
```

The partition has a size of 31959 512-byte blocks, which is about 16 MB.

To connect an interface with a device, file system links are used. In the Olympus digital camera example, a link exists from the file `/sys/block/sda/device` to the corresponding device:

```
ll device
lrwxrwxrwx  1 root root 0 Aug 17 14:03 device ->
../../../../devices/pci0000:00/0000:00:1d.0/usb1/1-1/1-1:1.0/hos
t0/0:0:0:0
```

In this way, all interfaces of the system are linked with their corresponding devices.

Beside the representation in **sysfs**, there are also the device files in the **/dev** directory.

These files are needed for applications to access the interfaces of a device. The name “device file” is a bit misleading, the name “interface file” would be more suitable.

Objective 6 Describe how udev Works

Before you can use a hardware device, you need to load the appropriate driver module and set up the corresponding interface. For most devices in SUSE Linux Enterprise Server 10, this is done by udev.

To get an overview about udev, do the following:

- Understand the Purpose of udev
- Understand how udev Works
- Understand Persistent Interface Names

Understand the Purpose of udev

udev has three main purposes:

- **Create device files.** The most obvious task of udev is to create device files under **/dev** automatically when a device is connected to the system. Before, the **/dev** directory was populated with every device, that might appear in the system. This led to a very complex and confusing **/dev** directory, as most of the device files were actually not used.
- **Persistent device names.** Another advantage of udev is that it provides a mechanism for persistent device names.
- **Hotplug replacement.** In SUSE Linux Enterprise Server 10, udev also replaces the hotplug system, which was responsible for the initialization of hardware devices in previous versions. udev is now the central point for hardware initialization in SUSE Linux Enterprise Server 10.

Understand how udev Works

udev is implemented as a daemon (udev), which is started at boot time through the script `/etc/init.d/boot.d/S03boot.udev`. udev communicates with the Linux kernel through the **uevent** interface. When the kernel sends out a uevent message that a device has been added or removed, udevd does the following, based on the udev rules:

- Initializes devices by calling `hwup`.
- Creates device files in `/dev`.
- Sets up network interfaces with `ifup`.
- Renames network interfaces, if necessary.
- Mounts storage devices which are identified as hotplug in `/etc/fstab`.
- Informs other applications through HAL (Hardware Abstraction Layer) about the new device.

To handle uevent messages, which have been issued before udevd was started, the udev start script triggers these missed events by parsing the `sysfs` filesystem. In previous SUSE Linux Enterprise Server versions, this part of the system initialisation was done by the `coldplug` script.

Everything udev does, depends on rules, defined in configuration files under `/etc/udev/rules.d/`, which are used to process a uevent.

A detailed description of udev rules is beyond the scope of this course. However, in the following you find the most important facts:

- udev rules are spread over several files, which are processed in alphabetical order. Each line in these files is a rule. Comments can be added with the `#` character.
- Each rule consists of multiple key value pairs. Example for a key value pair: `kernel=="hda"`

- There are two different key types:
 - **match keys.** These keys are used to determine, if rule should be used to process an event.
 - **assignment keys.** These keys determine what to do if a rule is processed.

There always has to be at least one match and one assignment key in rule.

- For every uevent, all rules are processed. Processing does not stop when a matching rule is found.

You can monitor the activity of udev with the tool **udevmonitor**. When you start the tool and change the hardware configuration (e.g. plug or unplug an USB device) the udev activities are displayed on the screen. For more details, you can start the tool the option **--env**.

Understand Persistent Interface Names

The interface files in the directory **/dev** are created and assigned to the corresponding hardware device when the device is recognized and initialized by a driver. Therefore the assignment between device and interface file depends on:

- The order in which device drivers are loaded.
- The order in which devices are connected to a computer.

This can lead to situations, where it's not clear which device file is assigned to which device. Consider the following example:

You have two USB devices, a digital camera and a flash card reader. Both devices are accessed as storage devices through the device files **/dev/sda**, **/dev/sdb** ...

Which device is assigned to which device file, usually depends on the order in which they are plugged in. The first devices gets sda and so on.

udev can help to make this more predictable. With the help of sysfs, udev can find out, which device is connected to which interface file. The easiest solution for persistent device names would be to rename the interface files, for example from `/dev/sda1` to `/dev/camera`.

Unfortunately, interface files can not be renamed under Linux. The only exception to this rule are network interfaces, which traditionally have no interface files under `/dev`.

Therefore udev uses a different approach. Instead of renaming an interface file, a link with a unique and persistent name is created to the assigned interface file.

By default udev is for example configured to create these links for all storage devices. For each device, a link is created in each of the following subdirectories under `/dev/disk/`:

- **by-id**. The name of the link is based on the vendor and on the name of a device.
- **by-path**. The name of the link is based on the bus position of a device.
- **by-uuid**. The name of the link is based on the serial number of a device.

The udev rules which create these links are located in the file `/etc/udev/rules.d/60-persistent-storage.rules`.

This means, that the association between devices and interface files still depends on the order in which the drivers are loaded or in which order devices are connected with the system.

With udev however, persistent links are created and adjusted everytime the device configuration changes.

As mentioned before, network interfaces are treated differently. They don't have interface files and they can be directly renamed by udev.

Persistent network interfaces are configured as udev rule in the file **/etc/udev/rules.d/30-net_persistent_names.rules**

The following is an example rule:

```
SUBSYSTEM=="net", ACTION=="add", SYSFS{address}=="00:30:05:4b:98:85",  
IMPORT="/sbin/rename_netiface %k eth0"
```

The matching key `SYSFS{address}` is used to identify a network device by its MAC address. At the end of the rule, the name of the interface is given. In this example `eth0`.



In SUSE Linux Enterprise Server 9, it was possible to configure persistent network interface names in the interface configuration files under `/etc/sysconfig/network`. This is not supported anymore in SUSE Linux Enterprise Server 10. Interface names now have to be configured in the udev rule.

Exercise 3-2 Add a device symlink with udev

In this exercise, you create a udev rule, that creates a symlink in `/dev` when a device is plugged in. To perform this exercise, you need a USB mouse.

Do the following:

1. Open a terminal window and **su-** to the root user.
2. **cd** to the `/dev` directory.
3. Make sure that there is no symlink or device **geekomouse** in the `/dev` directory. This can be done with the command:
ls geekomouse.
4. Open the file `/etc/udev/rules.d/60-persistent-input.rules` with a text editor.
5. Identify the two rules, which are introduced with the **#by-id links** comment.

The rule with the matching key **KERNEL=="mouse**"** creates symlinks of each mouse device under `/dev/input/by-id/`

The names of these symlinks are generated from hardware parameters like the serial number of the mouse. This way a persistent and unique device name is created.

6. Duplicate the mouse line to create a new rule.
7. In the new rule, change the value of the SYMLINK key to `geekomouse` (**SYMLINK+="geekomouse"**).
8. Save and close the file.
9. Unplug your USB mouse, wait a few seconds and plug it in again.
10. Check again, if the symlink `/dev/geekomouse` exists.
11. Unplug the mouse again, and see how the symlink is automatically removed by udev.

(End of Exercise)

Objective 7 Use the hwup Command

The command `hwup` is used by `udev` to load driver modules and to initialize devices. In this objective, you learn how to use this command and how to interpret the corresponding configuration files.

To start a device, `hwup` is called with a hardware description as argument. The hardware description is a unique identifier for a specific device in the system. The following is an example `hwup` command line:

`hwup bus-pci-0000:02:08.0`

The device description consists of the following components:

- **bus.** This determines that the device is identified by the bus it is connected to.
- **pci.** This indicates that the device is connected to the PCI bus.
- **0000:02:08.0.** This is the address of the device in the PCI bus.

You can display the PCI address of a device with the `lspci` command, as in the following example:

```
...
0000:02:08.0 Ethernet controller: Intel Corp. 82801BD
PRO/100 VE (LOM) Ethernet Controller (rev 81)
...
```

As you can see, with the command **`hwup bus-pci-0000:02:08.0`**, the ethernet controller with the PCI address `0000:02:08.0` would be started.

The command **`hwdown <hardware_description>`** can be used to stop a device. `hwdown` is a link to `hwup` and not a separate command.

hwdown unbinds a device from its driver, but does not unload the driver module automatically.

There are two different ways how hwup gets information about devices and about the driver that needs to be loaded for a device:

- From Configuration Files
- From sysfs

From Configuration Files

When a device needs to be initialized, hwup first tries to read a device configuration from files in the directory ***/etc/sysconfig/hardware/***.

In order to determine the correct configuration file, the configuration filenames follow a specific naming scheme.

The following is the filename for a PCI network adapter:

hwcfg-bus-pci-0000:02:08.0

The filename consists of the keyword **hwcfg** and the hardware description of a device.

The following lists the possible variables in a device configuration file:

Table 3-1	Variable	Description
	STARTMODE	<p>This determines when and how a device will be started:</p> <ul style="list-style-type: none"> ■ auto. The device is automatically started at boot time or by udev when the device is connected to the system. ■ manual. The device <i>should not</i> be started automatically, but it <i>can</i> be started manually. ■ off. The device should never be started.
	MODULE	<p>The value of this variable determines the name of the kernel module that should be loaded for the device.</p> <p>If multiple modules have to be loaded, you can use this variable multiple times with any suffix appended.</p> <p>You must then use the same suffixes for multiple MODULE_OPTIONS variables.</p> <p>Example:</p> <pre>MODULE_A="foo" MODULE_B="bar" MODULE_OPTIONS_A="foo-opt" MODULE_OPTIONS_B="bar-opt1=xyz"</pre>
	MODULE_OPTIONS	<p>With this variable, options can be passed to the kernel module.</p>

(continued) **Table 3-1**

Variable	Description
<code>SCRIPT{UP,DOWN}_[type]</code>	This specifies the script to be called for initialization and deconfiguration of a specific device type. This script is called if the type of the device to be initialized matches the type given in this parameter.
<code>SCRIPT{UP,DOWN}</code>	This specifies the script to be called for initialization and deconfiguration of the device. It will be called only if no matching type-specific scripts are configured.

The following is an example of a configuration file for a network adapter:

```
MODULE='e100'  
MODULE_OPTIONS=''  
STARTMODE='auto'
```

The module `e100` is loaded, there are no options for this modul and the device is started automatically at boot time.

The `hwup` command is usually called by `udev`, but you can also use it manually. For example, the following command starts the network card shown in the previous:

`hwup bus-pci-0000:02:08.0`

The last 3 elements of the configuration filename specify the device.

You can use the command `hwdown` to deconfigure devices, as in the following:

`hwdown bus-pci-0000:02:08.0`

From sysfs

When there is no configuration file for a device under `/etc/hardware`, `hwup` uses `sysfs` to find out more about the required driver.

In the file `/sys/bus/devices/<pci_device_id>/modalias` a unique id for a device can be found. The following is the content of the file `modalias` for a network adapter:

```
pci:v00008086d00001039sv00001734sd00001001bc02sc00i00
```

Now `udev` calls the command `modprobe` and uses the `modalias` as parameter.

The file `/lib/modules/<kernel_version>/modules.alias` contains a list with `modalias` IDs and their corresponding kernel driver modules. The information about which driver module handles which `modalias` ID is included in the source files of the modules. The command `depmod` is used to extract the information from the modules and to write them into the file `modules.alias`.

Therefore you can consider the file `modules.alias` as a kind of driver database, which is used to find out, which device is handled by which driver.

When called from `udev` with the `modalias` parameter, `modprobe` looks for the driver in `modules.alias` and load the associated driver.

To prevent `udev/hwup/modprobe` from loading a driver automatically in the described way, a driver can be entered in the file `/etc/modprobe.d/blacklist`. By default the ALSA sound drivers are for example entered here, because these drivers are loaded by the ALSA init script (`alsasound`).

Exercise 3-3 Explore Hardware Initialization

In this exercise, you learn how to shutdown a device with `hwdown` and how to start it again manually. The exercise consists of the following parts:

- Part I: Stop the Ethernet Adapter with `hwdown`
- Part II: Unload the Driver Module
- Part III: Load the Driver Module with a `modalias`

Part I: Stop the Ethernet Adapter with `hwdown`

Do the following:

1. Open a terminal window and `su-` to the root user.
2. Enter the command `ip a`. You should see a list with your network interfaces. (At least `eth0`).
3. Enter the command `lspci`.
4. Look for a device with the description **Ethernet controller** and note the PCI ID of this device below this step. If you have more than one ethernet adapter, choose the first one

-
5. Enter the command:
`hwdown bus-pci-0000:<network_adapter_pci_id>`
 6. Enter the command `ip a` again. The interface of the device which has been shutdown with `hwdown` should not be visible anymore.

Part II: Unload the Driver Module

1. Change into the directory `/etc/sysconfig/hardware/`.

2. Open the configuration file of the ethernet adapter with a text editor:
hwcfg-bus-pci-0000:<network_adapter_pci_id>
3. From the **MODULE** option in the configuration file, note the name of the kernel driver module below this step .

-
4. Enter the command **lsmod | grep <module_name>**. As you can see, is the module still loaded.
 5. Unload the module with the command
rmmod <module_name>
 6. Verify with **lsmod | grep <module_name>**, that the module is not loaded anymore.

Part III: Load the Driver Module with a modalias

1. Change into the directory **/sys/bus/pci/devices/**.
2. Change into the directory **0000:<network_adapter_pci_id>**
3. Enter **cat modalias** to display the modalias of the ethernet adapter.
4. Enter the command **modprobe <modalias>**. Use copy and paste from the cat output to enter the modalias.
5. Verify with **lsmod | grep <module_name>**, that modprobe has detected and loaded the driver for the network adapter.
6. Enter the command **ip a**, and verify that the network interface is available again.
7. Reboot your system, to restore all network settings.

(End of Exercise)

Summary

Objective	Summary
1. Describe the Differences between Devices and Interfaces	<p>The terms device and interface are often confused. This section uses the following definitions:</p> <ul style="list-style-type: none">■ Device. A device is a physical piece of hardware.■ Interface. An interface is a software component that is used to access a device. <p>One device can have more than one interface.</p> <p>An interface is created by a device driver.</p>
2. Describe how Device Drivers Work	<p>There are 2 basic kinds of device drivers:</p> <ul style="list-style-type: none">■ Kernel modules. Kernel modules are loaded into the Linux kernel and extend its functionality.■ User space drivers. These drivers run within user space applications. <p>Some devices require both: kernel modules and user space drivers.</p> <p>You can use the following commands to manage kernel modules:</p> <ul style="list-style-type: none">■ lsmod. Use lsmod to list loaded drivers.■ modprobe. Use modprobe load kernel modules.

Objective	Summary
2. Describe how Device Drivers Work (continued)	<ul style="list-style-type: none">■ rmmod. Use <code>rmmod</code> to remove loaded kernel modules. <p>The kernel modules are files that are stored in the directory <code>/lib/modules/kernel-version/</code>.</p>
3. Describe how Device Drivers Are Loaded	<p>In a SUSE Linux Enterprise Server 10 system, kernel modules are loaded in the following ways:</p> <ul style="list-style-type: none">■ From <code>initrd</code>■ By <code>initscripts</code>■ By <code>udev</code>■ By the X Server■ Manually by the user <code>root</code>
4. Manage Kernel Modules Manually	<p>The <code>hwup</code> command is used to start preconfigured devices.</p> <p>The device configuration files are stored in the directory <code>/etc/sysconfig/hardware/</code>.</p> <p>The filename of the configuration file contains a unique identifier for the corresponding device.</p> <p>In the configuration file, the following variables can be used:</p> <ul style="list-style-type: none">■ <code>STARTMODE</code>■ <code>MODULE</code>■ <code>MODULE_OPTIONS</code>■ <code>SCRIPT{UP,DOWN}_[type]</code>■ <code>SCRIPT{UP,DOWN}</code>

Objective	Summary
5. Describe the sysfs File System	<p>The sysfs file system provides a representation of all devices and interfaces of a system.</p> <p>Devices are represented in the directories: /sys/bus and /sys/devices.</p> <p>Interfaces are represented by the directories /sys/class and /sys/block.</p> <p>A device and its interfaces are connected with file system links.</p>
6. Describe how udev Works	<p>In SUSE Linux Enterprise Server 10, udev also replaces the hotplug system, which was responsible for the initialization of hardware devices in previous versions. udev is the central point for hardware initialization in SUSE Linux Enterprise Server 10.</p>
7. Use the hwup Command	<p>hwup is used to load device drivers. Information about the required drivers for a device is either taken from a configuration file under /etc/sysconfig/hardware/ or from the file /lib/modules/kernel_version/modules.alias.</p>