The Linux USB sub-system

Brad Hards
Sigma Bravo Pty Ltd.

The Linux USB sub-system

by Brad Hards

This document is an early draft of a guide to using the USB sub-system under Linux.

This document is matched to Linux kernel revision 2.3.99pre6. Some of this stuff is outright guesswork, especially the lesser known and more expensive devices.

Table of Contents

List of Figures

I. Using the Linux USB sub-system

Chapter 1. USB Introduction

This chapter provides a brief introduction into the Universal Serial Bus (USB) in general, and may be skipped if you are already familiar with the bus, and its software and hardware architecture.

1.1. The Universal Serial Bus

In 1994 an alliance of four industrial partners (Compaq, Intel, Microsoft and NEC) started to specify the Universal Serial Bus (USB). The bus was originally designed with these intentions:

- Connection of the PC to the telephone
- Ease-of-use
- Port expansion

The specification (version 1.0) was first released in January 1996 and the latest official version 1.1 was released in September 1998 The document is still under development and a version 2.0 was announced in 1999. The USB is strictly hierarchical and it is controlled by one host. The host uses a master / slave protocol to communicate with attached USB devices. This means that every kind of communication is initiated by the host and devices cannot establish any direct connection to other devices. This seems to be a drawback in comparison to other bus architectures but it is not because the USB was designed as a compromise of costs and performance. The master / slave protocol solves implicitly problems like collision avoidance or distributed bus arbitration. The current implementation of the USB allows 127 devices to be connected at the same time and the total communication bandwidth is limited to 12Mbit/s. However use of low speed devices, management of USB "interrupts" and other overheads mean that actual throughput cannot exceed about 8.5Mbit/s under near ideal conditions, and typical performance may be around 2Mbit/s.

1.2. Host Controllers

Most modern motherboard chipsets provide a USB host controller. Older machines which are not equipped with a USB host controller can be upgraded using a PCI cards with a host controller built in.

USB host controllers are compatible with either the Open Host Controller Interface (OHCI, by Compaq) or the Universal Host Controller Interface (UHCI, by Intel) standard. Both types have the same capabilities and USB devices work with both host controller types. Basically the hardware of UHCI is simpler and therefore it is cheaper, but needs a more complex device driver, which could cause slightly more CPU load.

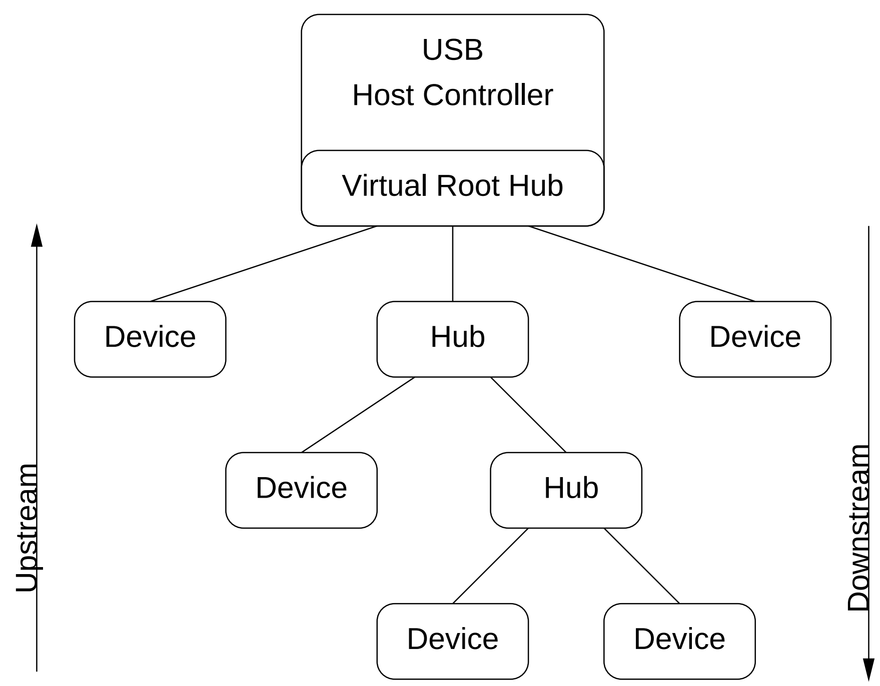
1.3. USB Devices and Transfer Characteristics

There are a wide range of USB devices intended for a wide range of purposes, and this means that implementation details can vary widely.

A device can be self powered, bus powered or both. The USB can provide a power supply up to 500mA for its devices. If there are only bus powered devices on the bus the maximum power dissipation could be exceeded and therefore self powered devices exist. They need to have their own power supply. Devices that support both power types can switch to self powered mode when attaching an external power supply.

Even the maximum communication speed can differ for particular USB devices. The USB specification differentiates between low speed and full speed devices. Low speed devices (such as mice, keyboards, joysticks etc.) communicate at 1.5MBit/s and have only limited capabilities. Full speed devices (such as audio and video systems) can use up to 90% of the 12Mbit/s which is about 10Mbit/s including the protocol overhead.

Figure 1-1. USB Topology



Version 2.0 of the USB specification is being developed, and is expected to deliver up to 480Mbit/s raw throughput.

1.3.1. Hubs

Physically there exist one, two or four USB ports at the rear panel of a computer. These ports can be used to attach normal devices or a hub. A hub is a USB device which extends the number of ports to connect other USB devices. The maximum number of user devices is reduced by the number of hubs on the bus (i.e. if you attach 50 hubs, then at most 77 (=127-50) additional devices can be attached. Hubs are always full speed devices. If the hub is self powered, then any device can be attached to it. However if the hub is bus powered, then only low power (100mA max) devices can be attached to it. A bus powered hub should not be connected to another bus powered hub - you should alternate between bus powered and self powered hubs.

Normally the physical ports of the host controller are handled by a virtual root hub. This hub is simulated by the host controllers device driver and helps to unify the bus topology. So every port can be handled in the same way by the USB subsystem's hub driver (see Figure 1-1).

1.3.2. Data Flow Types

The communication on the USB is done in two directions and uses four different transfer types. Data directed from the host to a device is called downstream or OUT transfer. The other direction is called upstream or IN transfer. Depending on the device type different transfer variants are used:

- *Control transfers* are used to request and send reliable short data packets. It is used to configure devices and every one is required to support a minimum set of control commands. The standard commands are:

  GET_STATUS
  CLEAR_FEATURE
  SET_FEATURE
  SET_ADDRESS
  GET_DESCRIPTOR
  SET_DESCRIPTOR
  GET_CONFIGURATION
  SET_CONFIGURATION
  GET_INTERFACE
  SET_INTERFACE
  SYNCH_FRAME

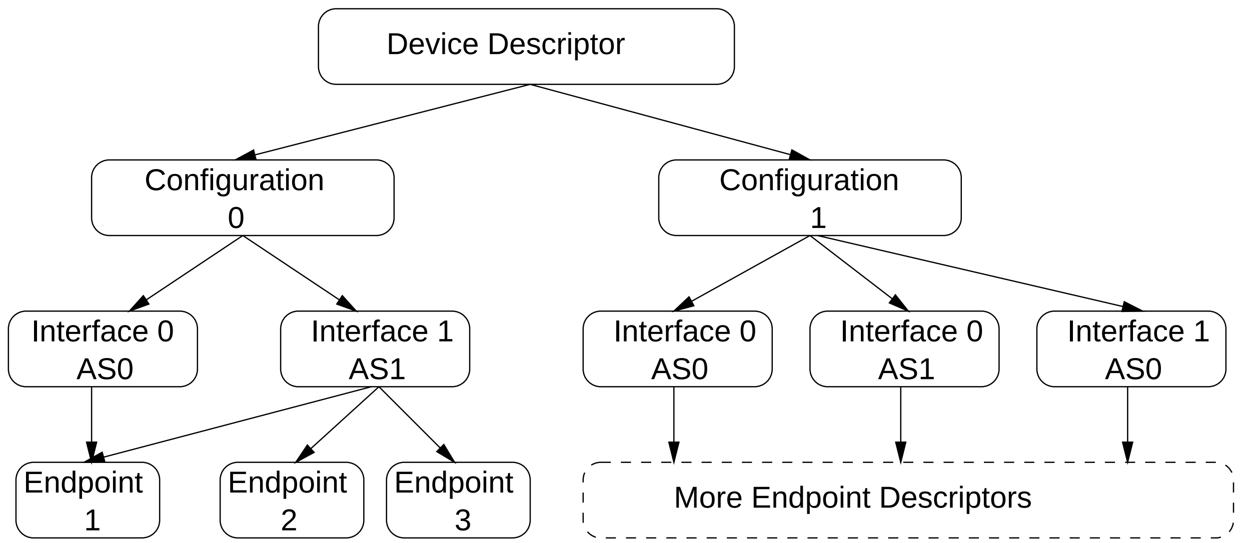
Further control commands can be used to transfer vendor specific data.

- *Bulk transfers* are used to request or send reliable data packets up to the full bus bandwidth. Devices like scanners or scsi adapters use this transfer type.
- *Interrupt transfers* are similar to bulk transfers which are polled periodically. If an interrupt transfer was submitted the host controller driver will automatically repeat this request in a specified interval (1ms - 127ms).
- *Isochronous transfers* send or receive data streams in realtime with guaranteed bus bandwidth but without any reliability. In general these transfer types are used for audio and video devices.

1.4. Enumeration and Device Descriptors

Whenever a USB device is attached to the bus it will be enumerated by the USB subsystem - i.e an unique device number (1-127) is assigned and then the device descriptor is read. The desciptor is a data structure which contains information about the device and its properties. The USB standard defines a hierarchy of descriptors (see Figure 1-2).

Figure 1-2. USB Descriptor



1.4.1. Standard Descriptors

- A *Device Descriptor* describes general information about a USB device. It includes information that applies globally to the device and all of the device s configurations. A USB device has only one device descriptor.
- The *Configuration Descriptor* gives information about a specific device configuration. A USB device has one or more configuration descriptors. Each configuration has one or more interfaces and each interface has zero or more endpoints. An endpoint is not shared among different interfaces within a single configuration, although a single interface can have several *alternate settings* which may use the same endpoint. Endpoints may be shared among interfaces that are part of different configurations without this restriction. Configurations can only be activated by the standard control transfer set_configuration. Different configurations

can be used to change global device settings, such as power consumption.

- An *Interface Descriptor* describes a specific interface within a configuration. A configuration provides one or more interfaces, each with zero or more endpoint descriptors describing a unique set of endpoints within the configuration. An interface may include alternate settings that allow the endpoints and/or their characteristics to be varied after the device has been configured. The default setting for an interface is always alternate setting zero. Alternate settings can be selected exclusively by the standard control transfer set_interface. For example a multifunctional device like a video camera with internal microphone could have three alternate settings to change the bandwidth allocation on the bus.

  Camera activated
  Microphone activated
  Camera and microphone activated

- An *Endpoint Descriptor* contains information required by the host to determine the bandwidth requirements of each endpoint. An endpoint represents a logical data source or sink of a USB device. The endpoint zero is used for all control transfers and there is never a descriptor for this endpoint. The USB specification uses the terms pipe and endpoint interchangably.
- *String Descriptors* are optional and provide additional information in human readable unicode format. They can be used for vendor and device names or serial numbers.

1.4.2. Device Classes

The standard device and interface descriptors contain fields that are related to classification: class, sub-class and protocol. These fields may be used by a host system to associate a device or interface to a driver, depending on how they are specified by the class specification. Valid values for the class fields of the device and interface descriptors are defined by the USB Device Working Group.

Grouping devices or interfaces together in classes and then specifying the characteristics in a Class Specification allows the development of host software which can manage multiple implementations based on that class. Such host software adapts its operation to a specific device or interface using descriptive information presented by the device. A class specification serves as a framework defining the minimum operation of all devices or interfaces which identify themselves as members of the class.

1.4.3. Human Interface Devices (HID)

The HID class consists primarily of devices that are used by humans to control the operation of computer systems. Typical examples of HID class devices include:

Keyboards and pointing devices for example, standard mouse devices, trackballs, and joysticks.
Front-panel controls for example: knobs, switches, buttons, and sliders.
Controls that might be found on devices such as telephones, VCR remote controls, games or simulation devices for example: data gloves, throttles, steering wheels, and rudder pedals.

1.5. USB Device Drivers

Finding device drivers for USB devices presents some interesting situations. In some cases the whole USB device is handled by a single device driver. In other cases, each interface of the device has a separate device driver.

Chapter 2. How to get USB devices working under Linux

2.1. Basic USB Configuration

You need a late version kernel. Kernel versions 2.2.7 and later contain the USB code. You should, in an ideal world, be running the current 2.3.x kernel, ideally with any pre-patches for the next kernel. The current 2.2.x kernels do contain some USB code, but it is badly out of date, and will not match up with this document. It is possible to use the 2.3.x USB code with a 2.2.x kernel - see later in this document for details. USB code is in fairly early development, so the changes between each version (and the bugs) tend to change fairly fast. Support on the mailing list for anything except the lastest version is scant at best.

The mailing list is `<linux-usb@suse.com>`. To subscribe, send a mail message to `<majordomo@suse.com>` with content of **`subscribe linux-usb`**. If you want to stop getting mailing list content, send a mail message to `<majordomo@suse.com>` with content of **`unsubscribe linux-usb`**.

You need to configure USB into your kernel. Use of **`make menuconfig`** is recommended. Under **`USB Support`**, you need to select **`Support for USB`**. You also need to select either **`UHCI (Intel PIIX4, VIA, ...) support`**, **`UHCI Alternate Driver (JE) support`** or **`OHCI-HCD (Compaq, iMacs, OPTi, SiS, ALi, ...) support`**. Which one you select is dependent on what kind of motherboard or adapter you have. Intel and Via motherboards, and Via-based adapters are UHCI, and you can use either of the two UHCI drivers - there seems to be little user visible difference between them. Ali and SiS chipsets, Compaq and NEC motherboards, iMacs and any adapter using Opti chips (just about all of them) are OHCI, and you should use OHCI-HCD. If you do not know what kind of controller to choose, check your motherboard documentation. You can also look at `/proc/pci` for a hint - if the USB entry is of the form 0xHHHH, where HHHH are hex digits (e.g. something like **`I/O at 0xe400`**), then it is UHCI. If it is of the form **`32 bit`**

memory at 0xHH000000, where HH are hex digits (e.g. something like **32 bit memory at 0xee000000**), then it is OHCI. Failing that, just try both.

Always build in the Preliminary USB device filesystem. Leaving this out will make resolving problems almost impossible, and is essential if you need to check that your kernel is configured correctly, and your USB devices are being recognised correctly.

You also need to select whichever devices you want to use, for example **USB Human Interface Device (HID) support** (with the appropriate subordinate options) for a USB keyboard, mouse, joystick, tablet or gamepad, **USB Scanner support** for certain scanners, **USB Audio support** for USB speakers, **USB Modem (CDC ACM) support** for a POTS or ISDN modem, **USB Printer support** for a USB printer, **USB Serial Converter support** (with the appropriate subordinate options) for some serial port type devices, **USB CPiA Camera support** for cameras based on the Vision CPiA chipset, **USB IBM (Xirlink) C-it Camera support** for camera based on the IBM camera chipset, **USB OV511 Camera support** for cameras based on OmniVision's OV511 chipset, **USB Kodak DC-2xx Camera support** for downloading images from Kodak's DC-200 series cameras, **USB Mass Storage support** for mass storage devices, **USS720 parport driver** for certain parallel port adapters, **DABUSB driver** for an experimental Digital Audio Broadcast receiver and **PLUSB Prolific USB-Network driver** for certain USB to USB type connections. You should be able to use modules, kernel only, or split modules and kernel code.

USB hubs are automatically supported. Some devices may stop and start working between kernel versions. Remember that you are using experimental code. Devices not listed in this document are not working at the time of writing, although developers are always welcome to contribute to the current codebase.

If you want to use a stable kernel (**2.2.x**), you can get a patch which adds the latest USB code to certain stable kernels. This patch, often referred to as a *backport patch*, can be obtained from http://www.suse.cz/development/usb-backport/. You apply the patch to the applicable stable kernel, and recompile. Note that the USS720 driver does not work in the backport, since the parallel port code on which it depends is substantially different in earlier kernels. USB storage is also significantly different between 2.2 and 2.3 series kernels.

Rebuild the kernel and the modules (if you configured to build as modules), and install the new kernel and the new modules. Reboot the system. If you need instruction on how to do this, refer to the Linux Kernel HOWTO.

If you are using modules, you need to load the following modules:

- `usbcore.o`
- `usb-uhci.o`, `uhci.o` or `usb-ohci.o`

and any driver modules, such as `scanner.o` or `printer.o`.

Inspect the kernel logs. If there isn't anything that could be USB related, likely causes are use of the wrong driver (UHCI when you needed OHCI or OHCI when you needed UHCI), not physically installing the hardware, a BIOS configuration that disables USB or stuffing up the configuration or installation of the kernel.

2.2. USB Device Filesystem

The USB device filesystem is a dynamically generated filesystem, similar to the `/proc` filesystem. This filesystem can be mounted just about anywhere, however it is customarily mounted on `/proc/bus/usb`, which is an entry node created by the USB code, intended to be used as a mount point for this system. Mounting in other locations may break utilities.

You need to select "Preliminary USB Device Filesystem" to make this work. You also need to enable general `/proc` support, and to have it mounted (normally automatic).

To mount the filesystem, you need to be root. Use the mount command: **`mount -t usbdevfs none /proc/bus/usb`**. Note that the **`none`** keyword is arbitrary - you can use anything, and some people prefer to use **`usbdevfs`**, as it makes the mount output look better.

If you do not want to have to mount the filesystem everytime you reboot the system, you can add the following to `/etc/fstab` after the `/proc` entry:

```
none            /proc/bus/usb            usbdevfs        de-
faults   0   0
```

This has the same effect as the mount command.

After you have mounted the filesystem, the contents of `/proc/bus/usb` should look something like:

```
dr-xr-xr-x   1 root     root            0 Jan 26 10:40 001
-r-r-r-   1 root     root            0 Jan 26 10:40 devices
-r-r-r-   1 root     root            0 Jan 26 10:40 drivers
```

. You may have more than one numbered entry if your machine has more than one universal serial bus controller.

Full interpretation of the `devices` and `drivers` files is provided later in this guide, in the user's section. Use of the numbered entries is provided later in this guide, in the programmer's guide.

2.3. USB Human Interface Device (HID) Configuration

2.3.1. General HID Configuration

There are two options for using a USB mouse or a USB keyboard - the standalone Boot Protocol way and the full featured HID driver way. The Boot Protocol way is generally inferior, and this document describes the full featured way. The Boot Protocol way may be appropriate for embedded systems and other systems with resource constraints and no real need for the full keyboard and mouse capabilities.

It is important to remember that the HID driver handles those devices (or actually those interfaces on each device) that claim to comply with the Human Interface Device (HID) specification. However the HID specification doesn't say anything about what the HID driver should do with information received from a HID device, or where the information that is sent to a device comes from, since this is obviously dependent on what the device is supposed to be doing, and what the operating system is. Linux (at the operating system kernel level) supports four interfaces to a HID device - keyboard, mouse, joystick and a generic interface, known as the event interface.

2.3.2. HID Mouse Configuration

In the kernel configuration stage, you need to turn on **`USB Human Interface Device (HID) support`** and **`Mouse Support`** Do *not* turn on **`USB HIDBP Mouse support`**. Perform the normal kernel rebuild and installation steps. If you are installing as modules, you need to load the `input.o`, `hid.o` and `mousedev.o` modules.

Plug in a USB mouse and check that your mouse has been correctly sensed by the kernel. If you don't have a kernel message, look for the changes to `/proc/bus/usb/devices`.

Since USB supports multiple identical devices, you can have multiple mice plugged in. You can get each mouse seperately, or you can get them all mixed together. You almost always want the mixed version, and that is what will be used together. You need to set up a device node entry for the mixed mice. It is customary to create the entries for this device in the `/dev/input/` directory. Use the following commands:

```
mkdir /dev/input
mknod /dev/input/mice c 13 63
```

If you **`cat /dev/input/mice`** you should see some bizarre looking characters as you move the mouse or click any of the buttons.

If you want to use the mouse under X, you have various options. Which one you select is dependent on what version of XFree86 you are using and whether you are using only USB for your mouse (or mice), or whether you want to use a USB mouse and some other kind of pointer device.

- You need to edit the `XF86Config` file (usually `/usr/X11R6/lib/X11/XF86Config` or `/etc/X11/XF86Config`).
- If you are using XFree86 version 4.0 or later, add a InputDevice section that looks like the following:

```
Section "InputDevice"
     Identifier  "USB Mice"
     Driver      "mouse"
     Option      "Protocol"   "IMPS/2"
     Option      "Device"     "/dev/input/mice"
EndSection
```

or, if you want to use a wheel mouse, something like:

```
Section "InputDevice"
     Identifier  "USB Mice"
     Driver      "mouse"
     Option      "Protocol"   "IMPS/2"
     Option      "Device"     "/dev/input/mice"
     Option      "ZAxisMapping"   "4 5"
     Option      "Buttons"        "5"
EndSection
```

may be more useful. Consult the XFree86 documentation for a detailed explaination and more examples.

You also need to add an entry to each applicable ServerLayout Section. These are normally at the end of the configuration file. If you only have a USB mouse (or USB mice), then replace the line with the "CorePointer" entry with the following line:

```
InputDevice "USB Mice" "CorePointer"
```

If you want to use both a USB mouse (or USB mice) and some other kind of pointer device, then add (do not replace) the following line to the applicable ServerLayout sections:

```
InputDevice "USB Mice" "SendCoreEvents"
```

- If you are using only a USB mouse (or USB mice) with XFree86 3.3, edit the Pointer section so that it looks like the following:

```
Section "Pointer"
    Protocol    "IMPS/2"
    Device      "/dev/input/mice"
EndSection
```

- If you are trying to use a USB mouse (or USB mice) in addition to another pointer type device with XFree86 3.3, then you need to use the XInput extensions. Keep the existing Pointer (or modify it as required for the other device if you are doing an initial installation), and add the following entry (anywhere sensible, ideally in the Input devices area):

```
Section "Xinput"
   SubSection "Mouse"
        DeviceName   "USB Mice"
        Protocol     "IMPS/2"
        Port         "/dev/input/mice"
        AlwaysCore
   EndSubSection
EndSection
```

- Restart the X server. If you don't have any mouse support at this point, remember that Ctrl-Alt-F1 will get you a virtual terminal that you can use to kill the xserver and start debugging from the error messages.

If you want to use the mouse under gpm, run (or kill and restart if it is already running) gpm with the following options. **`gpm -m /dev/input/mice -t imps2`** (as superuser remember). You can make this the default if you edit the initialisation files. These are typically named something like `rc.d` and are in `/etc/rc.d/` on RedHat distributions.

If you have both a USB mouse (or USB mice) and some other kind of pointer device, you may wish to use gpm in repeater mode. If you have a PS/2 mouse on `/dev/psaux` and a USB mouse (or USB mice) on `/dev/input/mice`, then the following gpm command would probably be appropriate: **`gpm -m /dev/input/mice -t imps2 -M -m /dev/psaux -t ps2 -R imps2`**. Note that this will make the output appear on `/dev/gpmdata`, which is a FIFO and does not need to be created in advance. You can use this as the mouse "device" to non-X programs, and both mice will work together.

2.3.3. Keyboard Configuration

You may not need any operating system support at all to use a USB keyboard if you have a PC architecture. There are several BIOS available where the BIOS can provide USB support from a keyboard plugged into the root hub on the motherboard. This may or may not work through other hubs and does not normally work with add-in boards, so you might want to add in support anyway. You definately want to add keyboard support if you add any operating system support, as the Linux USB support will disable the BIOS support. You also need to use Linux USB keyboard support if you want to use any of the "multimedia" types keys that are provided with some USB keybords.

In the kernel configuration stage, you need to turn on **`USB Human Interface Device (HID) support`** and **`Keyboard support`**. Do *not* turn on **`USB HIDBP Keyboard support`**. Perform the normal kernel rebuild and installation steps. If you are installing as modules, you need to load the `hid.o`, `input.o` and `keybdev.o` modules.

Check the kernel logs to ensure that your keyboard is being correctly sensed by the kernel.

At this point, you should be able to use your USB keyboard as a normal keyboard. Be aware that LILO is not USB aware, and that unless your BIOS supports a USB keyboard, you may not be able to select a non-default boot image using the USB keyboard. I have personally used only a USB keyboard (and USB mouse) and have experienced no problems.

2.3.4. USB Joystick and Gamepad support

In the kernel configuration stage, you need to turn on **`USB Human Interface Device (HID) support`** and **`Joystick support`**. Perform the normal kernel rebuild and installation steps. If you are installing as modules, you need to load the `hid.o`, `input.o` and `joydev.o` modules.

You need to set up a device node entry for the joystick. It is customary to create the entries for USB device in the `/dev/usb/` directory. You can use the following commands to create four device nodes, although there is no reason why you can not use more:

```
mknod /dev/input/js0 c 13 0
mknod /dev/input/js1 c 13 1
mknod /dev/input/js2 c 13 2
mknod /dev/input/js3 c 13 3
```

If you plug in a gamepad or joystick and **`cat /dev/input/js0`** you should see some bizarre looking characters as you move the stick or click any of the buttons.

You should now be able to use the USB joystick or gamepad with any of the normal games or other joystick compatible applications.

2.4. USB Scanner Support

USB scanners do not have a common class specification, hence it is not possible to make a device driver that works for all scanners. Not withstanding this, Linux USB does have a generic scanner driver that provides the communication link between the device and user space.

The USB Scanner driver is capable of supporting just about any scanner. However meaningful output depends on the availability of a suitable user space tool, such as SANE, see http://www.mostang.com/sane.

If your scanner is not automatically associated with the scanner driver, you have two options. The first option is to edit the source code for the scanner driver - I assume that if you can do this, you won’t need any instructions. The second option is to use modules, and to load the module with the vendor and product IDs specified as module options: **`insmod scanner.o vendor=0xXXXX product=0xYYYY`**, where XXXX is the hexadecimal vendor ID code, and YYYY is the hexadecimal product ID code, which you can extract from the contents of `/proc/bus/usb/devices` when you have the scanner plugged in.

You need to create an appropriate device file entry, using commands (as root) like the following:

```
mknod /dev/usb/scanner0 c 180 48
chmod a+rw /dev/usb/scanner0
```

To use the scanner device, you will need a suitable tool. SANE is strongly recommended. If you don't already have it installed or your installation is old, you will need to retrieve the package from the closest of the sites referenced at http://www.mostang.com/sane/source.html.

For instructions on building and installing SANE, refer to the various README files within the SANE distribution, and more detailed instructions for specific scanners that are provided later in this document.

2.5. USB Audio Support

USB audio systems such as speakers normally comply with the USB audio class specification. If your audio device does not comply with the specification, it will not work with Linux at this time.

You need to turn on the **`Sound card support`** kernel option, which is in the **`Sound`** section. You will then be able to turn on the **`USB Audio support`** option in the normal USB section.

If you elected to use modules, you need to load the module called `audio.o`.

You can test your installation by dumping some random data out to the audio interface, using a command like **`cat /bin/bash > /dev/dsp`**. USB audio is integrated into the normal Linux sound architecture, and most tools should work fine with USB speakers. See the Sound HOWTO for more details and suggestions.

2.6. USB Modem Support

USB modems (for plain old telephone service, ISDN or cable) are supposed to conform to the Communication Device Class (CDC) specification, and usually to the Abstract Control Model (ACM) sub-class. If your modem doesn't comply with the specification, it will not work with this driver.

You need to select the **USB Modem (CDC ACM) support** kernel option. If you build as modules, you need to install the `acm.o` option.

You need to set up the device node entries for the various modems. You can use up to 32 modems with this driver. Use the following commands to set up the first four:

```
mknod /dev/usb/ttyACM0 c 166 0
mknod /dev/usb/ttyACM1 c 166 1
mknod /dev/usb/ttyACM2 c 166 2
mknod /dev/usb/ttyACM3 c 166 3
```

You should now be able to use a terminal emulator program to attach to this device and connect to your modem or other terminal device. Apparently this driver is working quite well, but I have not personally tested it.

2.7. Printer Support

You need to set up a device node entry for the printer. Use the following command:

```
mknod /dev/usb/lp0 c 180 0
```

Different printer tools are installed with various different rights, and this may affect the access priviledges that you need with the device entry. It is normally correct to make `/dev/usb/lp0` rights the same as `/dev/lp0`, using the chmod, chown and chgrp commands.

You should now be able to use this device in a normal `/etc/printcap` entry. I recommend use of automated tools to generate such files, such as RedHat's control panel print-tool.I find this driver to be pretty good, although perhaps a bit slow.

If this does not appear to work, check that you have actually loaded the module, and double-check the `/etc/printcap` entry - especially that the device file matches the one you just created.

2.8. USB Serial Converter support

This serial driver supports a wide range of USB to serial adapters. It also allows communication with the Handspring Visor. You need to turn on **`USB Serial Converter support`**, and then select whichever of the various particular drivers you need. The **`Generic Serial Driver`** can be used with a wide range of converters, by specifying the vendor and product ID codes when you load the USB serial converter module: **`insmod usb-serial.o vendor=0xVVVV product-0xPPPP`**, where you need to change the **`VVVV`** and **`PPPP`** to match your device.

The serial port driver uses a major number of 188. Up to sixteen serial ports are supported. To create the appropriate device entries, use the following commands:

```
mknod /dev/usb/ttyUSB0 c 188 0
mknod /dev/usb/ttyUSB1 c 188 1
mknod /dev/usb/ttyUSB2 c 188 2
mknod /dev/usb/ttyUSB3 c 188 3
mknod /dev/usb/ttyUSB4 c 188 4
mknod /dev/usb/ttyUSB5 c 188 5
mknod /dev/usb/ttyUSB6 c 188 6
mknod /dev/usb/ttyUSB7 c 188 7
mknod /dev/usb/ttyUSB8 c 188 8
mknod /dev/usb/ttyUSB9 c 188 9
mknod /dev/usb/ttyUSB10 c 188 10
mknod /dev/usb/ttyUSB11 c 188 11
mknod /dev/usb/ttyUSB12 c 188 12
mknod /dev/usb/ttyUSB13 c 188 13
mknod /dev/usb/ttyUSB14 c 188 14
```

```
mknod /dev/usb/ttyUSB15 c 188 15
```

You should now be able to plug in a serial device into the adapter, and use the `/dev/usb/ttyUSB0` just as if it were a normal serial port.

To communicate with the Handspring Visor, you need an additional application package known as Pilot-Xfer. Installation and use of this package is covered later in this guide.

2.9. USB Cameras

2.9.1. Generic camera configuration

Under Linux, USB cameras use the Video4Linux You need to set up a device node entry for the CPiA camera. Use the following command, if you have no other Video4Linux devices:

```
mknod /dev/video0 c 81 0
ln -s /dev/video0 /dev/video
```

To use the device, you need some video tools. There are a fairly wide range of tools available. http://millennium.diads.com/bdirks/ has a package that is a generally named something like `apps20000217.tgz`, depending on the date of release. It has both X and text-mode tools. Using the text mode tools will allow you to do things like

```
./vctrl 320x240x24
./vcat | rawtoppm -bgr 320 240 | xv -
```

These tools default to using `/dev/video` if not otherwise specified, hence the symbolic link made previously.

There are other suitable tools available. Links to some of these tools are provided in the Linux USB web site at http://www.linux-usb.org.

2.9.2. USB CPiA Camera support

This driver supports a certain chipset made by Vision, and used in a range of USB cameras (notably the Creative WebCamII). You can see the range of products at http://www.vvl.co.uk/products/oems/videoconferencing.htm. To make the CPiA camera driver work, you need to select **`Video For Linux`** support (under **`Character Devices`** if using **`menuconfig`**), and then select **`CPiA Video For Linux`** and **`CPiA USB Lowlevel Support`** options within the **`Video For Linux`**.

2.9.3. USB OV511 Camera support

This driver supports a certain chipset made by OmniVision, and used in a range of USB cameras (notably the Creative WebCam III). To make the OV511 driver work, you also need to select **`Video For Linux`** support (under **`Character Devices`** if using **`menuconfig`**).

2.9.4. USB IBM (Xirlink) C-it Camera support

This driver supports the Xirlink C-it cameras, also sold by IBM. To make the OV511 driver work, you also need to select **`Video For Linux`** support (under **`Character Devices`** if using **`menuconfig`**).

2.10. USB and Digital Still Cameras

While most digital still cameras support a serial line at up to 115 Kbps, the fastest ways to get pictures out of your camera onto Linux involve USB:

USB Connection for Camera

Some cameras can connect directly to USB, normally using a custom jack on the camera, which is smaller than the standard type B jacks you see on the upstream side of hubs and other devices.

USB Connection for Memory

A number of cameras don’t have a direct USB hookup, but may come with an adapter that lets you connect their memory cards instead. Those cards are accessed using the mass storage driver, as if they were disk drives. (USB mass storage support is discussed elsewhere in this document.) You can buy such adapters separately; be sure that Linux supports the adapter model you want to use.

PCMCIA Connection for Memory

Inexpensive PCMCIA adapters ($10US) let you treat memory cards like IDE disks. This won’t be discussed any further here, but you should know the option exists. The transfer rate is higher than for USB, but the downside is that PCMCIA support is generally only available on laptops.

Only the first of these approaches is discussed in this section, where the camera has a direct connection to USB. The table below summarizes the most important information:

Still Camera	Drivers	Applications	Notes
Kodak DC-240/280	dc2xx	gPhoto	The first still cameras supported under Linux. USB lets you download pictures, take new ones, change camera settings, and more.

Still Camera	Drivers	Applications	Notes
Kodak DC-220/260/265/290, HP PhotoSmart C-500	dc2xx	ODS (no GUI)	DigitaOS runs these cameras, letting you run scripts on the camera as well as the sort of functionality available on the other DC-2xx series USB cameras.
Mustek MDC-800	mdc800	gPhoto	Only one camera at a time supported.
Sony DSC-505, Toshiba PDR-M4, and many USB card readers	storage, fat filesystem	ls, cp, mc, &c.	readonly for now

Your kernel may already have the *dc2xx*, *mdc800* and *storage* compiled as modules, or as part of the kernel. If it doesn't, rebuild so that you have them available.

2.10.1. Filesystem Device Nodes

If you use the dc2xx or mdc800 driver, you have all the security hooks you need. The instructions shown below assume that PAM will be set up to change ownership of these nodes to the logged-in user using the `/etc/console.perms` file. If that is not done, think carefully about the ownership and permissions you use in the instructions below. (Mode 0666 works for trusted systems, such as single users workstations with no rlogin.)

The dc2xx driver supports up to sixteen devices, but most users won't need to use more than one or two cameras. If these device nodes aren't part of your distribution, make at least the first few of them as follows:

```
# mknod -m 0660 /dev/usb/dc2xx0 c 180 80
# mknod -m 0660 /dev/usb/dc2xx1 c 180 81
```

```
# mknod -m 0660 /dev/usb/dc2xx2 c 180 82
# mknod -m 0660 /dev/usb/dc2xx3 c 180 83
...
# mknod -m 0660 /dev/usb/dc2xx15 c 180 95
```

The mdc800 driver supports one device. If its device node isn't part of your distribution, make it as follows:

```
# mknod -m 0660 /dev/usb/mdc800 c 180 32
```

If your connectivity uses the storage driver, see the section on that driver for more information.

2.10.2. Sanity Checking

After you think your kernel is set up, connect your camera to your computer with USB (power it up!) and check to see if a kernel message (perhaps in `/var/log/messages`) reports the driver saw the camera. If it didn't, you can do a bit of troubleshooting if you've got "procfs" and the prototype "usbdevfs" code set up.

The first check is whether USB is basically set up. You should see the driver listed in `/proc/bus/usb/drivers`; if it's not there, use "modprobe" to load it. There should be a bus directory such as `/proc/bus/usb/001` also; if there isn't, you may need to "modprobe" the right host controller driver (OHCI or UHCI).

Then make sure that the kernel saw the camera. With it still connected and powered up, check for an entry in `/proc/bus/usb/devices` corresponding to your camera. (The product ID string may name it.) If there is no camera there, check your USB connections. If you find the camera listed there, make sure it's one of the supported cameras. There should be an "I: ..." line showing that driver. For example, at the end of the line it would have "Driver=dc2xx", "Driver=mdc800", or "Driver=storage". If you don't see that, then the troubleshooting gets interesting.

2.10.3. gPhoto

gPhoto is a graphical application that knows how to talk to many different digital cameras. At this writing its current version (0.4.3) knows how to use USB to talk to three cameras: appropriate camera model and use an "other" (non-TTY) device path name as shown above. (Use dc2xx0 unless you have multiple cameras.) Check the "Camera Summary" to see if it's connected properly.

At that point, you should be able to download thumbnails of all the pictures in the camera, then download the pictures you've chosen. (At this writing, it's visibly faster to use OHCI than to use UHCI.)

More information about gPhoto is available at its web site, www.gPhoto.org.

2.10.4. DigitaOS Cameras and ODS

At this writing, gPhoto does not know how to communicate with cameras using the "DigitaOS" environment. However, there is a separate "Open Digita Services" project which can be used to do that. See that project's web site, at ods.sourceforge.net.

(There are also various Perl and Python scripts available to provide some of this functionality, if you prefer to develop in those languages rather than in C.)

2.10.5. Other Approaches (Developers only)

The solutions described above are, at this writing, the most widely used ones. You may be interested to know about some other solutions that might work.

You can configure the *usbserial* module with "generic serial" driver support. This lets you provide kernel module parameters that let the serial driver handle specific bulk-only devices (using USB product and vendor IDs). The Kodak or ODS cameras mentioned above are bulk-only, so this might permit `/dev/usb/ttyN` style paths to work instead of `/dev/usb/dc2xxN` style ones.

Systems which enable the "prototype usbdevfs" support can configure it to allow appropriately intelligent user mode programs to access USB devices. This requires libraries that know how to interact with `/proc/bus/usb/NNN/MMM` files for each USB device, ways to address security issues that arise (programs need read/write access), and ways to have user and kernel mode software interact safely. Such work is under way. See for example the Java USB project at jusb.sourceforge.net and the C "libusb" at libusb.sourceforge.net

2.11. Mass Storage Devices

The mass storage device driver can potentially be used with a wide a wide range of USB devices, not all of which would normally be considered to be mass storage. This is because the driver is really an interface between the USB stack and the SCSI layer. Despite this, the instructions in this section are oriented around devices like USB floppy drives, Zip drives, LS120 drives and USB CDROMs.

Since the mass storage driver presents the USB device as a SCSI device, you need to turn on **`SCSI support`**, which is under **`SCSI support`** in the configuration script.

You may find that you can only build this driver as a module. This is not abnormal, and is caused by the main kernel SCSI layer.

After you have compiled the kernel and rebooted (or added the relevant modules, which includes `usb-storage` in this case), you should check `/proc/scsi/scsi`. Information about your device should be listed. If it isn't, you will need to type the following, as root:

```
echo "scsi add-single-device 1 0 0 0" > /proc/scsi/scsi
```

You need to make an entry in `/etc/fstab`. A suitable entry for a floppy disk would be:

```
/dev/sda    /mnt/usbfd       auto            noauto,user 0   0
```

A suitable entry for a Zip disk would be:

```
/dev/sda4    /mnt/usbzip      vfat            noauto,user 0   0
```

A suitable entry for a CDROM disk would be:

```
/dev/scd0    /mnt/usbcdrom    iso9660         ro,noauto,user 0   0
```

A suitable entry for a hard disk with a single partition would be:

```
/dev/sda1    /mnt/usbhd       ext2            defaults   1   2
```

Note that the above entries assume you have no other SCSI devices. If you do have other devices, then the USB disk may not be `/dev/sda`, but could instead be `/dev/sdb`, `/dev/sdc` or some other device. You would then need to substitute the right device entry for `/dev/sda` or `/dev/scd0` in the entries above.

Having created the entr[y/ies] in `/etc/fstab`, you need to create matching mount point[s] in the actual filesystem. So if you made an entry as shown above for the USB floppy disk, then the mount point would be made (as root) by:

```
mkdir /mnt/usbfd
```

You should now be able to mount your floppy disk with a command like:

```
mount /mnt/usbfd
```

or like:

```
mount /dev/sda
```

2.12. USS720 driver

The USS720 is a USB to Parallel port chip made by Lucent that normally acts like a USB Printer Class device. Indeed you can use a USS720 based bridge and a parallel port printer with the USB Printer driver, (see above). However there is also a mode (known as register mode) which makes the USS720 look like normal parallel port hardware. This driver makes use of that mode.

If you have the opportunity, look at **/proc/sys/dev/parport** before you load the module or reboot with a kernel with USS720 enabled. You should note the number of parallel ports you have - typically one, under **/proc/sys/dev/parport/parport0**. After you put USS720 support into your kernel, you should have another port (perhaps **/proc/sys/dev/parport1**). If you look at the appropriate **hardware** entry, you should see something like the following:

```
[bradh@rachel bradh]$ more /proc/sys/dev/parport/parport1/hardware
base:   0x0
irq:    none
dma:    none
modes:  PCSPP,TRISTATE,COMPAT,EPP,ECP
```

Every distribution should have device node entries for parallel ports - typically **/dev/lpX** or **/dev/parX**, where the **X** is some number. If you need to create them, parallel ports device nodes use character major number 6.

You should now be able to use the USS720 for anything that you would normally need a parallel port for, except that certain timing sensitive applications may not work, since the emulation is rather slow. As for a real USB printer, I suggest use of automated tools to generate a **/etc/printcap** entry if you are connecting up a printer.

If you didn't get to select USS720 support at the configuration stage, you need to turn on **Parallel port support** support (under **General setup** if using **menuconfig**).

2.13. DABUSB driver

This device requires hardware that has not yet been released, and is currently provided mainly to show how to write a driver. For more information, see http://dab.in.tum.de.

2.14. PLUSB Prolific USB-Network driver

Prolific manufacture a range of USB chips, including the PL2301 and PL2302 devices that allow two USB host controllers to be linked, providing a simple point to point link at up to 5Mbps. This driver supports both PL2301 andPL2302 chips.

Aserton, Bencent, Butterfly, Camtel, Longshine, Prolific, StarMount, Swann, Univex and VVMer cables are all known to use the PL2301 or PL2302 chips. Entrega, Xircom and Belkin adapters do not use Prolific chips and are not supported by this driver.

In addition to selecting this option to either be compiled into the kernel or to be build as a module, you need to configure the appropriate networking interfaces. If you have two machines (say zhora and rachel, then on zhora you would enter **`ifconfig plusb0 192.168.0.1 pointopoint 192.168.0.2`**. On rachel you would enter **`ifconfig plusb0 192.168.0.2 pointopoint 192.168.0.1`**. To test that it is working, on zhora, enter **`ping 192.168.0.2`**. If you get packets back, it is working correctly.

For further information on networking, I suggest that you refer to the many excellent Linux networking documents, including the Linux Network Administrators' Guide and the Linux Network HOWTO.

2.15. USB ADMtek Pegasus-based device support

This driver suppport a range of ethernet adapters based on a chipset produce by ADMtek. Just about all adapters that provide 10/100BaseT are based on this chipset.

In addition to selecting this option to either be compiled into the kernel or to be build as a module, you need to configure the appropriate networking interfaces. The device will appear as another ethernet device, and can be checked using entries like **`cat /proc/net/dev`** or tools like **ifconfig**.

For further information on networking, I suggest that you refer to the many excellent Linux networking documents, including the Linux Network Administrators' Guide and the Linux Network HOWTO.

2.16. USB Diamond Rio500 support

To be done. See http://rio500.sourceforge.net

2.17. D-Link USB FM radio support

The D-Link USB FM radio driver uses the Video4Linux interface, similar to the webcams discussed above.

If the appropriate device node entries do not exist, you should create them:

```
mknod /dev/video0 c 81 0
mknod /dev/video1 c 81 1
mknod /dev/video2 c 81 2
mknod /dev/video3 c 81 3
ln -s /dev/video0 /dev/video
```

After plugging the radio in, all you should you need to do is to start up an appropriate application - I normally use kradio and KTuner from KDE, although any Video4Linux radio application should work successfully.

Chapter 3. Using Linux USB

3.1. Reading the Linux USB Device Filesystem output

The USB device filesystem is a dynamically generated filesystem that complements the normal device node system, and can be used to write user space device drivers. Writing of user space device drivers is covered in the programmer's section of this guide. In addition to the device nodes, there are two files that are also generated - the `drivers` and `devices` files. If you followed the instructions in the installation chapter, you should find them as `/proc/bus/usb/drivers` and `/proc/bus/usb/device` respectively. If the `/proc/bus/usb` directory is empty, you have not mounted the filesystem, or you have mounted it in the wrong location.

`/proc/bus/usb/drivers` just lists the currently registered drivers (even if the driver is not being used by any device). This is most useful when testing module installation, and checking for USB support in an unknown kernel. Here is an example of its use:

```
[bradh@rachel bradh]$ more /proc/bus/usb/drivers
hid
ov511
cpia
printer
hub
```

`/proc/bus/usb/devices` lists information about the devices currently attached to the USB bus. This is very useful when trying to figure out if the device is correctly enumerated. Here is an example of its use, showing the root hub, a hub, a mouse and a camera:

```
T:  Bus=00 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12  MxCh= 2
B:  Alloc= 28/900 us ( 3%), #Int=  2, #Iso=  0
D:  Ver= 1.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
```

```
P:  Vendor=0000 ProdID=0000 Rev= 0.00
C:* #Ifs= 1 Cfg#= 1 Atr=40 MxPwr=  0mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   8 Ivl=255ms
T:  Bus=00 Lev=01 Prnt=01 Port=01 Cnt=01 Dev#=  2 Spd=12  MxCh= 4
D:  Ver= 1.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0451 ProdID=1446 Rev= 1.00
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   1 Ivl=255ms
T:  Bus=00 Lev=02 Prnt=02 Port=00 Cnt=01 Dev#=  3 Spd=12  MxCh= 0
D:  Ver= 1.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0553 ProdID=0002 Rev= 1.00
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=400mA
I:  If#= 1 Alt= 0 #EPs= 1 Cls=ff(vend.) Sub=00 Prot=ff Driver=cpia
E:  Ad=81(I) Atr=01(Isoc) MxPS=   0 Ivl=  1ms
I:  If#= 1 Alt= 1 #EPs= 1 Cls=ff(vend.) Sub=00 Prot=ff Driver=cpia
E:  Ad=81(I) Atr=01(Isoc) MxPS= 448 Ivl=  1ms
I:  If#= 1 Alt= 2 #EPs= 1 Cls=ff(vend.) Sub=00 Prot=ff Driver=cpia
E:  Ad=81(I) Atr=01(Isoc) MxPS= 704 Ivl=  1ms
I:  If#= 1 Alt= 3 #EPs= 1 Cls=ff(vend.) Sub=00 Prot=ff Driver=cpia
E:  Ad=81(I) Atr=01(Isoc) MxPS= 960 Ivl=  1ms
T:  Bus=00 Lev=02 Prnt=02 Port=02 Cnt=02 Dev#=  5 Spd=1.5 MxCh= 0
D:  Ver= 1.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=046d ProdID=c001 Rev= 1.10
S:  Manufacturer=Logitech
S:  Product=USB-PS/2 Mouse
C:* #Ifs= 1 Cfg#= 1 Atr=a0 MxPwr= 50mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=02 Driver=hid
E:  Ad=81(I) Atr=03(Int.) MxPS=   8 Ivl= 10ms
```

The information in the `/proc/bus/usb/devices` output is arranged in groups:

- The line that starts with **`T:`** is the topology. **`Bus`** indicates which bus the device is on. **`Lev`** indicates the level of the device, starting at level 00 for the root hub, level 01 for any device attached to the root hub, level 02 for devices attached to hubs at level

01, and so on. **Prnt** is the parent device for this device (always 00 for the root hub, and 01 for the devices attached to the root hub). **Port** is the port on the parent device, starting at 00 for the first port on each device. **Prnt/Port** is unique per bus. **Cnt** indicates what number device this is, at this level, based on the enumeration order within that level of the topology, starting at 01 for the first device. **Dev#** indicates what number device this is, irrespective of level, based on the bus enumeration order. This is unique per bus. **Spd** indicates what speed this device is running at, in Mbps (either 1.5 or 12 with the current version of USB). **MxCh** indicates how many devices can be connected to this device, and is 00 for anything except a hub. **Driver** indicates which device driver is being used for this device - an entry of **(none)** indicates that no driver is being used.

- The line that starts with **D:** is information from the device descriptor. **Ver** indicates which USB specification version the device claims to meet. **Cls** indicates which device class the device is claiming to meet, in both hexadecimal and as a string. A **Cls** entry of **00(>ifc)** indicates that the device class specification compliance is interface dependent, and the interface descriptor should be read for device class information. **Sub** indicates which sub-class (within the **Cls** entry), the device meets. **Prot** indicates which protocol within a class or sub-class the device claims to meet. **MxPS** indicates how big the packets from Endpoint 0 are. **#Cfgs** indicates how many configurations this device has.
- Much like **D:**, the line that starts with **P:** is information from the device descriptor, and is seperated mainly because it wouldn't all fit on one line. **Vendor** indicates the Vendor Identification code for the device, and **ProdID** indicates the Product Identification code for the device. **Rev** indicates the product revision number.
- Refer to the USB specification clause 9.7.1 for further information on device descriptors.
- The lines that start with **S:**, if any, are the vendor and product strings that the device returned.
- The line that starts with **C:** is information from the configuration descriptor - the number of **C:** lines per device is given by **#Cfgs**, and the entry followed by an asterisk is the current configuration. **#If** indicates how many interfaces the device has. **Cfg#** indicates which configuration is being described. **Atr** is a hexadecimal

indication of the device attributes (0x80 for bus-powered, 0x40 for self-powered, 0x20 for remote wake-up capable). **`MxPwr`** is the maximum power draw for this device configuration, in milliamps. Refer to USB specification clause 9.7.2 for further information on configuration descriptors.

- The line that starts with **`I:`** is information from the interface descriptor - the number of **`I:`** lines per **`C:`** line is given by the **`#Ifs`** entry. **`If#`** indicates which interface is being described within a given device configuration. **`Alt`** indicates which alternate setting of this interface is being described. **`#EPs`** indicates how many endpoints there are within the alternate setting for this endpoint. **`Cls`** indicates which class the alternate setting of the interface corresponds to, in both hexadecimal and as a character string. **`Sub`** indicates which sub-class the alternate setting of the interface belongs to. **`Prot`** indicates which interface protocol (within a class and sub-class tuple) the alternate setting of the interface conforms to. **`Driver`** indicates which of the various USB drivers has claimed this interface. See USB specification clause 9.7.3 for further information.
- The line that starts with **`E:`** is information from the interface descriptor - the number of **`E:`** lines per **`I:`** line is given by the **`#EPs`** entry. Endpoint 0 is not displayed. **`Ad`** indicates the endpoint address, with a letter to indicate whether the endpoint is an In or Out endpoint. **`Atr`** indicate the attribute (transfer type) associated with the endpoint, followed by a string translating the transfer type. **`MxPS`** indicates the maximum packet size this endpoint is capable of sending or receiving, as appropriate. **`Ivl`** indicates the interval, in milliseconds, between polling of interrupt endpoints. **`Ivl`** is ignored for bulk and control transfers, and is set to 1 for isochronous transfers. See USB specification clause 9.7.4 for further information on endpoint descriptors.

Refer to `linux/Documentation/usb/proc_usb_info.txt` for more information on using the USB device filesystem information.

3.2. Configuring extra keys on USB keyboards

To understand how to configure the additional keys that are provided on some USB keyboards, it helps to understand they way in which key processing works under Linux and X.

The most fundamental level of keyboard processing is a scancode. The USB keyboard driver pipes scancodes into the normal kernel keyboard driver. Depending on the mode in which the keyboard is operated, these can either be sent to the application directly (known as scancode or raw mode) or translated using a keyboard mapping table (known as the keymap) before being sent to the application (known as keycode or intepreted mode).

X uses the raw scancodes, which are known as keycodes in X documentation. These are translated inside the X server to a functional equivalent, known as a key symbol or keysym using a mapping table, although this is not the same format and performs a subtly different function. This means that translations in console mode and under X are not the same.

3.2.1. Console mode keyboard mappings

To be completed, but basically you use the **showkey -s** to get the raw scancodes, and then use **setkeycodes** to modify the keymap or **loadkeys** to load a new keymap. Then you can assign the translated keycode to whatever makes sense for the application program.

3.2.2. X keyboard mappings

To be completed, but basically you use the **showkey -s** or **xev** to get the raw scancodes, and then use **xmodmap** to modify the keyboard map.

3.3. Communicating with the Handspring Visor

To be done.

3.4. Communicating with the Diamond Rio 500

To be done.

3.5. Using a USB Scanner

As noted in the installation part, the scanner driver provides a link for a suitable user space program to access the scanner hardware. This section deals with getting the user space program (SANE) set up correctly.

If you don't have SANE installed, you can get it from from the any of the sites referenced at http://www.mostang.com/sane/source.html.

At this point, you need to do different things depending on what sort of scanner you have. Mostly you need to change configuration files. If you selected the default install location, SANE's configuration files are located in `/usr/local/etc/sane.d/`. If SANE came installed with your distribution, the configuration files are probably in `/usr/etc/sane.d/`.

3.5.1. Hewlett Packard Scanners

Not all Hewlett Packard scanners are supported by the sane-hp backend driver. The only ones with a reasonable chance of working are HP4100C, HP5200C, HP6200C and HP6300C. The Photosmart S20 should also work. Note that the HP3300C and HP4200C are not included in the devices with a working backend.

For a HP scanner, you need to change `/usr/local/etc/sane.d/hp.conf` to contain just the following two lines:

```
/dev/usb/scanner0
```

```
option connect-device
```

3.5.2. Epson Scanners

The Perfection 636U and Perfection 1200U scanners have a reasonable chance of working with the sane-epson backend driver.

For an Epson scanner, you need to `/usr/local/etc/sane.d/epson.conf` to contain the following line:

```
usb /dev/usb/scanner0
```

3.5.3. AGFA ScapScan 1212U

This scanner comes in two versions. The version 2 model needs its firmware downloaded. Both version 1 and version 2 need patches to SANE.

Check the driver disk that came with your scanner for a file named `SnapScan 1212U_2.bin`. If there is no file by that name, you probably have a version 1 scanner and can proceed on. Otherwise you need to copy that file off, and obtain a suitable firmware downloader, such as the agfafirm tool from http://hem.fyristorg.com/henrikj/snapscan/downloads/agfafirm. With this tool, you then do **agfafirm /dev/usb/scanner0 "SnapScan 1212U_2.bin"**. The status lamp on the scanner should flash.

You need to modify the SANE sane-snapscan backend. This requires the source code. Complete the following steps:

- Obtain the source code from ftp://ftp.mostang.com/pub/sane/sane-1.0.1.tar.gz.
- unpack the sources with the following command: **tar xvzf sane-1.0.1.tar.gz**
- change into the sane source directory: **cd sane-1.0.1**

- obtain the required patch from http://hem.fyristorg.com/henrikj/snapscan/downloads/sane-1.0.1-usb.diff.
- Apply the patch with the following command: **patch -p1 < sane-1.0.1-usb.diff**
- Build and install SANE with the following command: **./configure; make; make install**

Note that this patch will not apply cleanly to SANE 1.0.2, but there you can either apply the rejects by hand or use a development patch. I assume that if you can do this, you won't need detailed instructions.

3.5.4. Prisa Acerscan Scanners

In addition to SANE changes, the Prisa Acerscan 620U and 640U need firmware downloaded.

Check the driver disk that came with your scanner for a file named `u96v057.bin` in the `DRIVERS/DISK1/BIN/` directory. Copy this file off the driver disk.

You need to obtain a suitable firmware downloader, such as the acerfirm tool from http://utopia.knoware.nl/~joey/linux/acerfirm. With this tool, you then do **acerfirm /dev/usb/scanner0 "u96v057.bin"**. The status lamp on the scanner should flash.

You need to modify the SANE sane-snapscan backend. This requires the source code. Complete the following steps:

- Obtain the source code from ftp://ftp.mostang.com/pub/sane/sane-1.0.1.tar.gz.
- unpack the sources with the following command: **tar xvzf sane-1.0.1.tar.gz**
- change into the sane source directory: **cd sane-1.0.1**
- obtain the required patch from http://hem.fyristorg.com/henrikj/snapscan/downloads/sane-1.0.1-usb.diff.
- Apply the patch with the following command: **patch -p1 < sane-1.0.1-usb.diff**
- Hand edit the following area in `backend/snapscan.h`

```
static struct SnapScan_Model_desc scanners[] =
{
  /* SCSI model name -> enum value */
  { "FlatbedScanner_4", VUEGO310S },
  { "SNAPSCAN 1212U", SNAPSCAN1236S },
  { "SNAPSCAN 1212U_2", SNAPSCAN1236S },
  { "SNAPSCAN 1236",    SNAPSCAN1236S },
  { "SNAPSCAN 310",     SNAPSCAN310 },
  { "SNAPSCAN 600",     SNAPSCAN600 },
  { "SnapScan",         SNAPSCAN300 },
};
```

to include the following additional entry:

```
 { "FlatbedScanner13",  SNAPSCAN600 },
```

- Build and install SANE with the following command: **./configure; make; make install**

Note that this patch will not apply cleanly to SANE 1.0.2, but there you can either apply the rejects by hand or use a development patch. I assume that if you can do this, you won't need detailed instructions.

3.5.5. Using SANE

You should now be able to use a SANE frontend (**`xscanimage`** or **`scanimage`**) to scan images. You may need to specify what device you are using as a command line option (-d) to **`scanimage`**. Much more information is available in the various SANE man pages, including how to get GIMP and **`xscanimage`** working together.

Chapter 4. Frequently Asked Questions

There certain questions that keep reappearing. This section tries to answer those questions.

How do I report a bug?

In addition to the information required in `/usr/src/linux/REPORTING-BUGS`, it also helps enormously if you provide the output of `/proc/bus/usb/devices`. If you are having problems with your host controller being recognised, it helps to have the output of `lspci -v`. If it is supposed to be UHCI, it helps to know whether interrupts are being generated every second.

What is a good book on USB

I have only read one, and thought it was very poor. I suggest that you get the specifications from http://www.usb.org, which are at no cost, are quite readable, and are up to date.

How do I make USB work on my Sony Vaio laptop?

You may need to turn off the BIOS option for Plug-n-Play operating system support on some models of Vaio. Then USB should work fine, although you may have problems with using the modem. This is only needed on some models.

How do I make USB be detected on my machine?

If you are sure that you actually have a suitable hardware setup, look for a BIOS option that could be applicable. It might be labelled as USB, or it might be more obscure, discussing Plug-n-Play, or having options for various types of operating systems. You may need to try various combinations.

I've upgraded and USB doesn't work on my machine anymore

If you have a motherboard with BIOS set to MPS 1.4, try MPS 1.1.

My Asus motherboard doesn't work, or works with only one device

Some early Asus P2B-DS and P2B-D motherboards have a design flaw that means that the chipset doesn't get sufficient voltage. I suggest returning the motherboard. Alternatively, check the Asus website at http://www.asus.com.tw/supportnews/english/mainboard/p2bx/17695.html for a hardware solution.

II. About this Guide

Chapter 5. Contributors

I am indebted to all the developers who have worked on Linux USB.

I would like to thank everyone who provided content or corrections to this document. Unfortunately I do not have a definitive list of those people, but those that I can remember are:

- Georg Acher
- David Brownell
- John Cagle
- Matthew Dharm
- Randy Dunlap
- Johannes Erdfelt
- Deti Fliegl
- Ron Forrester
- Pam Huntley
- Henrik Johansson
- Karl Heinz Kremer
- Greg Kroah-Hartman
- David Nelson
- Vojtech Pavlik
- Thomas Sailer
- Eric Sandeen
- Joey Schim
- Christophe VG

- Thomas Vogels

If you helped out, I would like to acknowledge your contribution. Please contact me if your name is not already listed above.

Chapter 6. Availability and licensing

6.1. Obtaining updates and translations

You can get the latest version of this Guide from http://LinuxUSBGuide.sourceforge.net/.

It is also mirrored at http://linuxkb.cheek.com/pages/usb-howto.

You can get the SGML source by emailing the author, or from ftp://LinuxUSBGuide.sourceforge.net/

You can get a Japanese translation of this Guide at http://www.kyoto.vasara.co.jp/usb/.

You can get a Korean translation of this Guide at http://kldp.org/HOWTO/html/USB/USB-HOWTO.html

Translation into French is in progress.

Translation may differ from this version.

6.2. License

Chapter 7. Corrections

Please send comments on this document to the author, preferably by E-Mail (<bradh@dynamite.com.au>), including the version number: ($Revision: 1.0.7 $).