DOSUSB driver documentation

Contents

1. Introduction

DOSUSB is an USB driver package for DOS. It consists of DOSUSB.COM - a TSR implementing a low level USB host controller driver (HCD/USBD), supporting EHCI, OHCI and UHCI controllers, and additional higher level drivers and code examples for storage media, printers and other devices.

DOSUSB.COM provides a software layer so application programs and USB device drivers can be used supporting specific USB devices. With help of DOSUSB.COM it is possible for DOS application programs to communicate with all sorts of USB devices and to implement
higher level USB device drivers for DOS. DOSUSB can also be used with WIN 3.11.

The DOSUSB.COM driver is supplied together with a DOS mass storage device driver for flash disks and memory card readers, a DOS printer device driver for printers connected via USB and a DOS device driver for USB serial port adapters.

The specification for USB 2.0 describes that the EHCI controller shall have no direct USB ports but communicates via OHCI or UHCI controllers and their USB ports with the devices. Therefore you will almost always find EHCI controllers together with OHCI or UHCI controllers on the computer's mainboard.

An exception to this, which is also mentioned in the USB 2.0 specification, is a connection of a 2.0 HUB directly to an EHCI controller. Then there are no OHCI/UHCI

controllers needed on the mainboard just one or more HUBs instead. Intel chose that approach with its new P55 etc. chipset boards for the Intel CORE i5 and CORE i7 processors. This version of DOSUSB supports these boards and their RMHs. The RMHs will be listed as unidentified devices since they do not have product descriptors.

2 Demo Version

The free version for download on the internet is a demo version of DOSUSB. This version will stop working after 20 minutes of operation. Calls by an application program will be unsuccessful then after that period. DOSUSB can also be unloaded and loaded again for five times only. After that the PC has to be booted again before loading DOSUSB.

An unrestricted version is available for purchase at the WWW.DOSUSB.NET website.

3 Installation

Extract the zip file you have downloaded into a directory on your hard disk e.g. c:\dosusb. This directory will then contain three subdirectories and the following files:

```
DOSUSB.COM        The driver
usbview.exe       Application which displays the descriptors of a USB
                  device
listdevs.exe      Displays all devices and their addresses
lpt1usb.sys       Device driver for printers replacing the LPT1: device
usbdisk.sys       Device driver for mass storage devices
serdrv.sys        Device driver for serial port adapters
```

Additionally there are example files for programming in the examples directory. These are listed and discussed at the end of this documentation in the Examples section.

To start DOSUSB enter DOSUSB.COM at the DOS prompt. DOSUSB will then enumerate all the connected devices and install itself as a resident program. It will display the controllers and devices found and show the address, speed plus Information about the devices.

DOSUSB itself requires about 30k of computer memory. However, it also requests additional memory for each controller installed in the computer.

DOSUSB will work on a computer featuring EHCI, UHCI or OHCI controllers. It will determine whether these controllers are installed and report this.

On laptops, the number of ports per controller may exceed the number of USB sockets installed. DOSUSB will then show more ports than can be used with devices.

4. Printer, Serial Port and Mass Storage Device Drivers

DOSUSB includes three device drivers for DOS: a printer driver, a serial port driver and a mass storage driver. For other devices drivers can be developed using the URB programming interface. Examples for this are included.

4.1 Printer driver

There are two printer drivers available. LPT1USB.SYS installs as a LPT1: device while PRNUSB.SYS installs as the PRNUSB: device. So you with the latter driver you can use LPT1: for other printers if needed, while with LPT1USB.SYS you can print from the DOS Editor or other application programs directly.

Both are printer device drivers which use DOSUSB to print to USB printers. They will work with a HP 2420d Laserprinter, a HP 880C, a HP 5940, a HP 460C inkjet, a HP Deskjet 980cxi, a Lexmark E250 or a KYOCERA FS-1100 printer.
They will also work with parallel to USB adapters. Check the endpoints and use the setalt utility to set the interface before printing with such an adapter.

LPT1USB.SYS and PRNUSB.SYS are loaded with a "device=" statement in the config.sys file or with the DEVLOAD utility. Load DOSUSB then, do not use the /L command line option here! The default device address is one, the default device endpoint for OUT is two. Use the command line options e.g. /D2 /E1 to select device address two and OUT endpoint number one. Use USBView to determine the right settings for your printer.

These drivers cannot be used with low cost GDI printers, since these do not accept ASCII code for printing. They expect data which controls their print head directly. This protocol is vendor specific and usually not available. If a printer accepts ASCII code via its parallel interface, it usually can also do this via its USB interface.

However, since drivers for GDI printers are available for WIN 3.11, if you use DOSUSB and PRNUSB.SYS with WIN 3.11, you can also use these GDI printers with PRNUSB.SYS.There are no drivers for these printers in DOS, however.

Use "type dosusb.txt > lpt1" or "type dosusb.txt > prnusb" to print the file dosusb.txt from the DOS prompt. Or open "lpt1" or "prnusb" as a file in your application program.

4.2 Flash disk and memory card reader driver

USBDISK.SYS is a device driver for flash drives and other mass storage devices such as USB hard disks and PCMCIA cards in USB card readers.

The driver is loaded with a "device=" statement in the config.sys file such as: device=c:\dosusb\usbdisk.sys provided the file usbdisk.sys in the c:\dosusb directory on your hard disk. You can also use the DEVLOAD utility.

If you have more than one USB device connected to the PC and the flash drive is not the first device to get address number one, you can use the /D command line option. This way you can specify the address of the flash drive. DOSUSB will always give the same address to each device connected to the PC as long as no device is removed or added. So if you found out, which address the flash drive gets from DOSUSB you can specify that with the /D option. Without the /D option USBDISK.SYS assumes address one. USBDISK.SYS reads the device descriptor to

determine the endpoints so these do not need to be set by a command line option.

Then boot the PC to have DOS load usbdisk.sys as specified in config.sys.

When DOS boots and loads the USBDISK.SYS driver, the driver will display the drive letter DOS has assigned to the mass storage device.

Plug in your flash drive now. Always use the same slot, to avoid to get a different address than before.

On newer PCs the BIOS will detect the flash drive and start communicating with it. Wait till the LED on the flash disk stops flashing indicating that the BIOS is done.

Start dosusb (without the /L option for log-file creation)

Now you can do a "dir" command on the flash drive and copy files from the drive to the hard disk and from the disk to the drive. You can also delete files etc.

Even if there are only few or no files in the directory, DOS may have to read a large amount of data when evaluating the FAT for reporting the free space on the volume, causing a long delay. Using the "DIR" command with the switch "/B" (not on DR-DOS or EDR-DOS) will provide just a minimalized listing without reporting free space thus avoiding the delay.

Windows opens a small window showing how much data has been copied. DOS does not do that. So with large files it seems that nothing is happening - but DOS does copy! You could also use a file manager for copying which usually wil provide a progress indicator.

Remarks:

DOS 6.22 does not support FAT32 disks. So flash disks formatted as FAT32 cannot be used with DOS 6.22 which is the majority now. Use DOS 7.0 or FreeDOS with FAT32.

If you want to boot the PC from a floppy disk or CD, DOS will often not assign a drive letter which is correct when usbdisk.sys is loaded using the config.sys file. In this case you should not put usbdisk.sys into the config.sys file but use the devload.com utility to load usbdisk.sys later after loading DOSUSB.COM.

USBDISK.SYS does not detect by itself that the mass storage device has changed. If you connect a different flash disk to be on the safe side you should reboot your PC to reload USBDISK.SYS or your data could be corrupted.

However, if you insert a flash disk of the same type and size if is sufficient to unload and load DOSUSB again and leave USBDISK.SYS in memory. After swapping the DISK make a "dir" command first to have DOS read the disk's directory again.

If the mass storage device is not formatted it is not supported by USBDISK.SYS. It will not return that the device is unformatted. Also it has been observed that DOS needs a partition table while Windows will handle the flash disk without it.

USBDISK.SYS will just access the first partition of a USB hard disk.

If your flash disk has a write protect switch, this will not be queried by USBDISK.SYS. So DOS can copy to the flash disk even when the write protect switch is set.

If you want to boot from a flash disk and then run DOSUSB to access this, you first have to copy the contents of the flash disk (e.g. DOS command.com) to a RAM disk. DOSUSB will prohibit the BIOS from accessing the disk when loaded. DOS may look for files on the missing boot drive after being copied to the RAM disk.

Flash disks feature highly different read and write speeds and have different firmware implementations. Try different flash disks with DOSUSB and take the one that works best with your PC and DOSUSB.

On older PCs DOSUSB may work too fast for some flash drives. You can try the /S command line switch to adjust that.

4.3 Serial port adapter driver

SERDRV.SYS is a device driver for USB serial port adapters. It has been tested with Prolific adapters and will work with newer versions of these devices. The Prolific chipset is used by many adapters and you can use USBVIEW or check in Windows wether it is such a device.

It does not emulate an UART and therefore cannot be used with many DOS serial communication programs which hook into the serial interrupt to read the data from the serial port. This device driver provides a DOS device called "SERDRV". It can be opened like a standard file in DOS. Then you can read and write to this device using DOS read and write file commands. See the SIMCOM.BAS sample provided. The baud rate can be selected with a command line option.

This is the first version of this driver. USB serial adapters use vendor defined commands which are not published. So the driver will not work with any adapter.

5. The URB Programming Interface

Using the USB request block (URB) programming interface any program can use DOSUSB to communicate with almost any USB device. At www.usb.org there are many standard protocols for communication with USB devices documented. See the included samples for writing such programs.

To request the execution of a transaction by DOSUSB an application program or a device driver has to use a USB Request Block (URB) of the following structure:

```
urbstruc struc
   transaction_type  byte  ;control (2Dh), in (69h), out (E1h) as hex code
   chain_end_flag    byte  ;reserved
   dev_add           byte
   end_point         byte
   error_code        byte  ;returned by dosuhci
   status            byte  ;returned by dosuhci
   transaction_flags word  ;reserved
   buffer_off        word  ;for in/out
   buffer_seg        word  ;for in/out
   buffer_length     word  ;for in/out
   actual_length     word  ;for in/out, (maximum length / bytes returned)
```

```
   setup_buffer_off  word  ;for control
   setup_buffer_seg  word  ;for control
   start_frame       word  ;reserved
   nr_of_packets     word  ;isochronous if set to one
   int_interval      byte  ;reserved
   error_count       byte  ;number of retries
   timeout           word  ;reserved
   next_urb_off      word  ;reserved
   next_urb_seg      word  ;reserved
urbstruc ends ;32 byte long
```

This URB structure is similar to the Linux URB for USB.

Using this URB it is possible to schedule control, input and output transactions. For control transactions, DOSUSB can do the following in and out transactions required to retrieve the requested data.

5.1 Elements of the URB structure

transaction_type

select the type of transaction scheduled, this can be either a control, in or out transaction. For a control transaction enter 2D hex, for an in transaction enter 69 hex and for an out transaction enter E1 hex.

You can also send commands to DOSUSB, if you enter FF hex as the transaction code. The commands available are explained below.

chain_end_flag

this element is currently not used

dev_add

enter the USB device address here of the device the scheduled transaction shall be send to.

The device driver can either request the device descriptor for all addresses from one to invalid to find the device in question. Or you can use usbview to find the device manually.

end_point

each device has several endpoints. The device driver has to select the right endpoint address for the control, in and out transactions. There are different endpoints for bulk, interrupt, control or isochronous transactions.

error_code

in this element DOSUSB will return an error code, if it encounters a problem with the scheduled transaction. The following codes are currently defined:
1 - invalid device address, 2 - internal error,
3 - invalid transaction_type, 4 - invalid buffer length
5 - device has been disconnected

status

this is the status byte returned by the USB controller after executing the scheduled transaction. If a control transaction has been scheduled, this element has the status byte of the last transaction in the queue.

See details about this byte in the Intel UHCI design guide. In short:
Bit 0: Reserved, Bit 1: Bitstuff error, Bit 2: CRC/Timeout, Bit 3: NAK,
Bit 4: Babble detected, Bit 5: Data Buffer error, Bit 6: Stalled,
Bit 7: Active
So e.g. 88 hex means a NAK received, since the active bit is set the data packet can be retried, 44 hex means a CRC/Timeout error which caused a stall. The device may need a reset.

The OHCI controller returns different completion codes than the UHCI controller. To simplify driver development, the DOSUSB driver translates the OHCI completion codes to UHCI status codes. However, in the dosusb.log file the OHCI codes are specified.
OHCI completion code 1 is translated to 44h, 2 to 42h, 3 to 44h, 4 to 40h, 5,6 and 7 to 44h, 8 to 50h, 9 to 00h, 12 to 20h, 13 to A0h and 14 to 88h.

transaction_flags

this element is currently not used

buffer_off / buffer_seg

enter a pointer to a buffer here, where either the data to be send by an out transaction or the data to be returned by an in or control transaction shall be placed

buffer_length

specify either the length of the out buffer or the expected length of the data to be returned by an in or control transaction here.

If you set buffer_length greater than actual_length (maximum length) DOSUSB will schedule the number of in transactions required to retrieve the amount of data specified in buffer_length. DOSUSB accepts a buffer_length up to 1024 bytes for UHCI and OHCI and 16384 bytes for EHCI. This currently does not work for full/low speed devices connected to an 2.0 Hub.

actual_length

to schedule a transaction, enter the maximum length supported by the addressed endpoint here.

After the transaction is completed this element will have the number of bytes returned.

For an EHCI or OHCI controllers and also UHCI controllers, the number is the real length in bytes.

Transfers to an EHCI control endpoint of endpoint address zero have a fixed size of 64 bytes while for other endpoints its 512 bytes. For UHCI and OHCI transfers the maximum size is 64 and depends on the device.

setup_buffer_off / setup_buffer_seg

for control transactions, enter a pointer to the request here. This can e.g. be a device descriptor request

nr_of_packets

if this element has a value greater zero, DOSUSB will schedule a transaction for isochronous traffic.
Isochronous support is not yet implemented for EHCI and OHCI!

error_count

this instructs the UHCI controller to retry unsuccessful transactions. Valid values are: 0 - default (3 retries), 1 - 1 retry, 2 - 2 retries, 3 - 3 retries. For OHCI this is set to three and cannot be modified.
DOSUSB will not have the host controller retry if a NAK is received.
These transactions are terminated with 88h in the status element and have to be retried by the application. This is not the case with EHCI controllers.

All other elements are currently unused. They are provided so a device driver that is written for the current version of DOSUSB will run without changes with future versions too.

5.2. Scheduling control transactions

In the case of a control transaction DOSUSB will return the input in the buffer pointed to buffer_seg and buffer_off, if buffer_seg and buffer_off are not zero and buffer_length is greater than zero.

For a control transaction such as a device descriptor request DOSUSB will return the device descriptor in buffer and acknowledge the receipt to the device with a zero out transaction. In this case the actual_length field has to be equal to the maximum packet length and buffer_length has to specify the requested length of the descriptor which shall be returned.

If buffer_length is set to zero, DOSUSB will do an in transaction but no out transaction. So for a set configuration transaction you have to set buffer_length to zero since this transaction does not return data from the device and therefore requires no out transaction.

If you set buffer_seg and buffer_off to zero, DOSUSB will just do an control transaction. The driver calling DOSUSB can then request the in and out transactions required following the control transaction.

5.3. Calling DOSUSB to execute the URB

To have DOSUSB execute the transaction defined by the URB, you have to place a pointer to the URB in the DS:DX registers and call interrupt 65h. The DS register has to contain the segment part of the pointer and the DX register the offset part of it.

With the /I commandline parameter you can specify a different interrupt to be used by DOSUSB. For example /I64 will ask DOSUSB to use interrupt 64h. Use the range 60h-67h.

Using the URB you can only schedule a transaction to a single device specified by

the device address. So you can send and receive data from a mass storage device or send data to a printer. However, after scheduling a transaction to a mass storage device you can schedule the next transaction to a printer and then again to the mass storage device.

If you have a camera, which sends a constant stream of data, you have to retrieve that data quickly. Otherwise, depending on the buffer size in the camera, some data may get lost.

6. Commands to DOSUSB

If you enter FFh as the transaction type, you can specify in the ax register the following commands:

ax=1: If dev_add is zero, DOSUSB will reset all devices and enumerate them again. It will not enumerate new devices and thus all devices keep their address. If dev_add is greater than zero, DOSUSB will just reset and enumerate the device with that address. It will not output anything to screen in that case.
The result value of this call will be returned in the "transaction_type" field of the urb. Possible values are 30hex = "0" for success and 31hex = "1" for failure.

ax=2: enables debug output to screen.

ax=3: disables debug output to screen.

ax=4: displays the URB received from an application program or device driver on the screen. You also have to specify a valid device address in the URB here. This is intended for testing when developing a new driver.

ax=5: set data toggle to zero, e.g. after a clear_feature(endpoint_halt) command. You also have to specify the device address and endpoint in the URB here.

ax=6: allows to check if DOSUSB is installed. Will return "G" or 47h in the transaction type field of the URB.
Check if the interrupt is initialized (not zero at 0:194h) first before using this.
A safer procedure is the following:
DOSUSB can be installed on any INT in the range from 60h to 67h inclusive, the default is 65h. To detect whether DOSUSB is installed, read the pointer in the interrupt table for e.g. interrupt 65h. If you add 2 bytes to the offset of the pointer you have the address for the string "DOSUSB" - if you find this string then DOSUSB is installed. Above this string you find 3 more byte's holding the release date as ((YYYY-2000),MM,DD). See the icheck.bas or listdevs.asm example.

ax=7: allows to check whether an EHCI, UHCI or OHCI controller is installed.
A valid device address in the URB is required.
Will return in the transaction type field of the URB a "U" or 55h if UHCI, an "O" or 4Fh if OHCI and an "E" or 69h if EHCI has been determined by DOSUSB.
For the device address specified the speed of the device is returned in the field next_urb_off. It will read "H" for high speed, "F" for full speed and "L" for low speed.

ax=8: DOSUSB will check all ports for devices connected and reset and enumerate all devices it finds. If there are any additional devices or devices have been removed, all devices may receive a new address.
The device on port zero of the first controller will get address number

one and so forth. If this device has not changed, it will keep its address. However, if there has been no device on that port before, all devices will get a new address.

ax=9: allows to determine the type of USB controller installed. Will return the PCI vendor ID in the field next_urb_off of the URB and the PCI device ID in the field next_urb_seg.

7. Commandline options

DOSUSB currently supports the following command line options:

/L write debug output to dosusb.log file (used to be /D in the previous version)

/I change interrupt to call DOSUSB URB - usbdisk.sys will only work on int 65h.

/R use D100:0000 as I/O address for OHCI or EHCI

If you use a protected mode DOS version with an EHCI or OHCI controller, you can use the /R option to avoid DOSUSB to access memory above 1 MB.

/1 use UHCI or OHCI only, no EHCI.

/E use EHCI only, no UHCI/OHCI - see details below

If you only want to communicate with high speed devices according to USB 2.0, you can use the /E option to save the memory required to additionally support OHCI and UHCI controllers. If you have both OHCI and UHCI controllers installed, e.g. a mainboard with UHCI controllers and an extension card with OHCI controllers, you can also use this option since DOSUSB does not support both OHCI and UHCI controllers simultaneously.

/P test option - see details below

/S delay option - e.g. /S10 will insert a 10 msec delay after each transaction

If you have a slow PC it may be necessary to set e.g. /S1 or /S3 if DOSUSB works too fast for that processor.

/X exclude controller

The **/X option** allows to exclude a USB controller, which will not be used by DOSUSB. E.g. /X0000EC00 will exclude the UHCI controller at I/O address EC00 hex.
For an OHCI or EHCI controller you have to specify e.g. /XFF00EC00 hex.

If you want a USB mouse or keyboard to be controlled by the BIOS while using DOSUSB to communicate with another device at a different controller, you can use this command to exclude the EHCI controller where the mouse and keyboard are connected or the flash disk you have booted from. If you only have one EHCI controller this will not work. In this case you can use the /E option to leave all full/low speed devices to the BIOS. Full/low speed devices connected to a 2.0 hub will still work.

Otherwise you have to set DOSUSB into 1.1 mode with the /1 option.

Without the /1 option, no UHCI/OHCI controllers should be excluded since the EHCI controllers works together with the UHCI/OHCI controllers.
The /1 option cannot work if you have a motherboard with no OHCI/UHCI controllers but RMHs instead.

/Y exclude device

The /Y option allows to exclude a complete EHCI controller plus the UHCI or OHCI controllers it is using to support full and low speed devices. E.g. /Y1A excludes all controllers having device 1A. DOSUSB displays the device numbers on its startup screen. If your PC has two EHCI controllers you can exclude the one your USB keyboard is connected to.

/N reduce transaction timeout

If a device just sends NAKs for a long time DOSUSB will timeout after a certain period. Since serial adapters send NAKs when they have not received any data, it sometimes improves the performance when this timeout period is reduced. This is especially the case if a serial adapter is connected to a high speed hub. The value following the /N option divides the timeout period by the power of two. E.g. /N8 divides the period by 256. The value is to be specified in hexadecimal, so /NF is valid too. The value to choose depends on the speed of the PC. The default value used by DOSUSB for split transactions is 7. The maximum value is 11hex which will usually stop DOSUSB to work properly.

/P exclude EHCI port

The **/P option** is intended as a test option. It allows to exclude a single port of an EHCI controller and the connected device will then be seen as a full speed device by DOSUSB. E.g. /PFF00EC0001 will exclude the port 01 at the I/O address FF00EC00. The port number follows immediately the eight digit I/O address specified.
In combination with the /E option the /P option allows to exclude a high speed device e.g. a boot disk and leave that to the BIOS if the PC has uhci/ohci controllers installed.

8. Debug mode

DOSUSB offers a trace of the transactions performed via the UHCI, OHCI or EHCI interface. This allows to monitor whether the scheduled transactions where successful.

If DOSUSB is called with the /L command line, the debug output is written into the dosusb.log file while DOSUSB is running.

Using the transaction descriptor FF hex and the command 02 you can turn on the trace any time and using the command 03 you can turn it off again. In this case, the debug output is written to the screen. After each transaction the trace stops and asks for a keypress. If you enter "c" here, the screen will be cleared.

For each UHCI transaction the trace contains the following lines:

a) the bytes which make up the transaction descriptor as it is returned by the UHCI

interface. There are two differences of this transaction descriptor to the one written to the UHCI interface: the status byte has been updated by the controller and DosUHCI puts the value 500h into the actual length field to check if it has been updated by the controller. Following the transaction descriptor the transaction type is given (IN, OUT or CTRL), the bits set (LS=low speed, SPD=short packet detect, IOS=isochronous traffic , IOC=interupt on complete) and the error limit specified, usually three errors.

Above the transaction descriptor there are numbers which shall allow to locate the bytes in the descriptor or AL for actual length, ST for status, PI for PID and AD for device address.

b) Then the result of the transaction(s) is displayed. Following this there are specified: the device address, the endpoint, the data toggle bit and the maximum packet length.

If buffer_length is greater than actual_length (maximum length), there will be buffer_length divided by actual_length number of transactions. These will be displayed beneath each other here.

c) The final line shows the bytes in hexadecimal of the data sent or returned by this transaction.

For control transactions, the request is also displayed at the beginning of the transaction.

For OHCI transactions the trace is set up differently.

For control transactions, the request is also displayed at the beginning of the transaction.

An OHCI transaction is set up by an endpoint descriptor (ED:) and an transaction descriptor (TD:). The trace will list both after the transaction.

It will also show the TC field of the endpoint descriptor, the value in the toggle field of the transaction descriptor and the completion code in numeric and text form followed by the number of bytes transfered.

9. Planned features for the next release

- support for isochronous transfers in EHCI and OHCI.

- support mouse/keyboard at high speed hubs

- support low speed hubs connected to high speed hubs

10. USB reference material

Writing USB device drivers requires a good understanding of the USB specifications. Here are some documents to get started:

"USB in a nutshell"
A good introduction to the USB specifications. Available from:
www.beyondlogic.org

"USB Made Simple"
Another fine introduction by MQP Electronics.
http://www.usbmadesimple.co.uk/

"Universal Host Controller Interface (UHCI) Design Guide" by Intel.

This is the interface DOSUSB is based on for its UHCI part.
www.intel.com/technology/usb/

"OpenHCI Open Host Controller Interface Specification for USB" by Compay, Microsoft, National Semiconductor. This is the specification for the OHCI part of DOSUSB.
http://download.microsoft.com/download/whistler/hwdev1/1.0/wxp/en-us/hci_1.exe
http://h18000.www1.hp.com/productinfo/development/openhci.html

"USB 2.0 specifications"
Once you've read "USB in a nutshell" you have to consult this reference.
www.usb.org/developers

"USB complete" by Jan Axelson
Probably the most important book on USB. The author's web site is:
www.lvr.com/usb.htm

"USB Tracker" by Ellisys
This is a good USB scope and currently one with a relatively low price. If you download their demonstration software, you get excellent examples of USB communications with different devices.
www.ellisys.com

To write a device driver for a floppy drive or mass storage device you will usually only need the class specifications from www.usb.org. However, many devices have vendor specific commands which often are not made available. Some of these specifications can be gathered from the Linux USB driver implementations.

11. USB analysers

If you develop a device driver using DOSUSB, a USB protocol analyser will be of great value.

The best choice is a hardware monitor such as the USB Tracker by Ellisys.

Besides this device, there are software USB analysers e.g.:
USBinfo at http://lpt.usbfireinfo.com
or SourceUSB at http://www.sourcequest.com.

There is also a free software USB sniffer at http://sourceforge.net/projects/usbsnoop or at http://benoit.papillault.free.fr/usbsnoop

These tools are cheaper than the USB Tracker but only allow you to monitor the traffic between Windows and a USB device. However, you can see how windows communicates with the device.

Finally there are devices which allow you to use a logic analyser to monitor USB traffic: www.futureplus.com.

12. Examples

The package contains examples of programs which use the DOSUSB driver. These are mostly not complete device drivers, they shall only provide a starting point to develop drivers which use DOSUSB.

Most of these examples are not yet changed to work with EHCI USB 2.0, so you

have to enter the /1 command line option to use them. The examples already adapted are placed into the samples20 directory, while the other examples are placed in the samples11 directory.

After the extraction of the DOSUSB.ZIP file the examples directory will contain the source-code for the following examples:

samples20:

```
usbview.bas     Source code for usbview.exe in Powerbasic
listdevs.bas    Source code for listdevs.exe in Powerbasic
diskspy.bas     Powerbasic source code for diskspy.exe
stick.bas       Sample to read sector zero of a USB flash drive.
enum.bas        Powerbasic sample to retrieve a device descriptor
prnusb.asm      Source code for the printer device driver
```

samples11:

```
mouse.bas       Source code for mouse.exe in Powerbasic
keyboard.bas    Source code for reading a USB keyboard in Powerbasic
hplaser.bas     Source for an application to print the file dosusb.txt
                to an HP Laserjet printer
lq590.bas       Source for a sample to print to an Epson LQ-590 printer
setalt.bas      Source code to set alternate interface for a device
enum.c          Turbo C program to retrieve a device descriptor
enum_gcc.c      GCC program to retrieve a device descriptor
enum_ser.c      DJGPP program to retrieve the serial number of a device
enum.bas        Freebasic sample to retrieve a device descriptor
enum.asm        Assembler sample to retrieve a device descriptor
hplaser.pas     Pascal sample to print dosusb.txt to an HP Laserjet printer
hplaserv.bas    VBDOS sample to print dosusb.txt to an HP Laserjet printer
hplaserv.mak    VBDOS project file for hplaserv.bas
hplaserq.bas    Quickbasic sample to print dosusb.txt to an HP Laserjet
                printer
reset.bas       Powerbasic source for a reset command
reset.pas       Pascal source for reset command
restart.bas     Powerbasic source for restart.exe
restart.pas     Pascal source for restart command
icheck.bas      Powerbasic source for an installation check
cable.bas       Sample to send data via a PC connect USB cable
reader.bas      Sample to read sector zero of a memory card
stick.pas       Sample in Pascal to read sector of a USB flash drive
floppy.bas      Sample to read sector zero of a floppy drive
camera.bas      Read isochronous traffic from a camera
scanner.bas     Sample which reads the identity of an Epson 1670 scanner
serial.bas      Sample terminal program using a USB/serial adapter
simcom.bas      Sample using the SERDRV.SYS device driver for USB/serial
                adapters
```

USBVIEW is a Powerbasic program which will display all descriptors of a device.

LISTDEVS is a utility in Powerbasic which displays all connected devices and their device addresses.

DISKSPY is a utility to read selected sectors of a USB disk and display the contents in hex and ascii much like the DOS debug program. Thus allows to read the partition table.

SERIAL.BAS is a simple terminal program which demonstrates communicating via a USB serial adapter using the URB interface directly.

SIMCOM.BAS is a terminal program which uses the SERDRV.SYS device driver to communiate with a USB serial adapter.

MOUSE is a demo program which reads the input from a USB mouse. Will run with UHCI only.

LQ590.bas is a sample which prints a short text on an Epson LQ-590 printer.

HPLASER.bas will print the file DOSUSB.TXT on a HP Laserprinter. It uses PCL commands to set up the printer for text output.

Most low cost USB printers are GDI or Windows printers which will not print ASCII codes sent to them.

SETALT.bas is a sample to set an alternate interface of a device.

ENUM.c is a Turbo C sample to retrieve a device descriptor.

ENUM_GCC.c is a GCC sample to retrieve a device descriptor

ENUM_SER.c is a DJGPP sample to retrieve the serial number of a USB device.

ENUM.bas is a Powerbasic sample to retrieve a device descriptor.

ENUMF.bas is a Freebasic sample to retrieve a device descriptor.

ENUM.asm retrieves a device descriptor using assembler.

HPLASER.pas is a sample in Pascal to print dosusb.txt to an HP Laserjet printer. Michel LECLERC translated HPLASER.bas to this Pascal sample.

HPLASERV.bas is a Visual Basic for DOS sample to print dosusb.txt to an HP Laserjet printer. Converted from hplaser.bas for Powerbasic.

RESET.bas resets and enumerates all connected devices. This will not check for additionally connected devices and not remove the addresses for disconnected devices. So the addresses for all devices will not change.

RESTART.bas resets and enumerates all connected devices. This will assign addresses to additionally connected devices and remove the addresses for disconnected devices. The addresses for other devices may change.

ICHECK.bas checks whether the DOSUSB driver is installed and running.

KEYBOARD.bas reads the keys typed on a USB keyboard. The USB HID to PS/2 scan code translation table can be found at the microsoft site in the document: translate.pdf

CABLE.bas is a sample which sends data via a "PC connect" USB cable from one USB port to a second USB port.

READER.bas reads a sector of a memory card in a USB card reader.

STICK.BAS is very similar to READER.bas and reads a sector of a USB memory stick.

STICK.PAS, RESTART.PAS and RESET.PAS are samples converted to Pascal by Joe da Silva

ADDRESS.PAS is a sample converted to Pascal by Ladislav Lacina

FLOPPY.bas will read the boot sector of a floppy in a USB floppy drive.

CAMERA.bas reads isochronous data from a Trust 120 Spacecam camera and saves it in the file image.yuv. Will run with UHCI only.

SCANNER.bas is a sample which reads the identity of an Epson Perfection 1670 scanner. This does not use the ESC/I commands.

30th March 2010 Georg Potthast