Manual for Communicating to the USB port of the cutters of Summa with the OS X operating system or Linux.

Manual Revision History

June 2003 Original Issue.

Feb 2008-02-26 Additon of summa USB port 2,3 and 4

Introduction

This document describes a way to communicate to the USB port on OSX. This has only been tested at Summa with OS X 10.1. As we are no Macintosh specialists, we only provide some guidelines and information.

Summa does not provide any USB driver, the communication with the USB port takes place at user level, not at kernel level.

You should first read the document “Working with USB Device Interfaces” that can be found at http://developer.apple.com.

You should also download the USB SDK for OSX found at http://developer.apple.com. This SDK provides some sample code that can easily be adapted for the cutters.

The rest of this document gives some details about the implementation of the USB protocol on the cutters from Summa and gives some details on how to adapt the sample code from Apple.

USB implementation for Summa Cutters

- The cutter does not belong to any class, so the Composite driver described on page 9 of “working with USB device interfaces”, will not be used by OSX. This means that you will have to select a configuration in order to communicate with the USB port.

- The cutter has only one USB configuration and one USB interface. The USB interface has several pipes. The driver should only work over 3 interrupted pipes (PIPE00 and PIPE01 and PIPE02).

- Now Follows a dump of the configuration of the USB port on the cutters from Summa (the bold text contains useful information)

NOTE : throughout this document the pipes are numbered from 0 to 3 (Zero based indexing) but apple uses 0 as index for the “control pipe” and the rest of the pipes are 1 based indexing. So PIEP00 will have index 1 when dealing with pipes in the Mac software!

Device Descriptor:
==========
bcdUSB: 0x0100
bDeviceClass: 0xFF
bDeviceSubClass: 0x00
bDeviceProtocol: 0x00
bMaxPacketSize0: 0x10 (16)
idVendor: 0x099F
idProduct: 0x0100
bcdDevice: 0x0100
iManufacturer: 0x00
iProduct: 0x00
iSerialNumber: 0x00
bNumConfigurations: 0x01

USB_CONFIGURATION_DESCRIPTOR
==========
bLength = 0xa, decimal 10
bDescriptorType = 0x2 (USB_CONFIGURATION_DESCRIPTOR_TYPE)
wTotalLength = 0x34, decimal 52
bNumInterfaces = 0x1, decimal 1
bConfigurationValue = 0x1, decimal 1
iConfiguration = 0x0, decimal 0
bmAttributes = 0x60 (USB_CONFIG_SELF_POWERED)
MaxPower = 0x1, decimal 1

USB_INTERFACE_DESCRIPTOR #0
==========
bLength = 0xa
bDescriptorType = 0x4 (USB_INTERFACE_DESCRIPTOR_TYPE)
bInterfaceNumber = 0x0
bAlternateSetting = 0x0
bNumEndpoints = 0x4
bInterfaceClass = 0xff
bInterfaceSubClass = 0x0
bInterfaceProtocol = 0x0
bInterface = 0x0

USB_ENDPOINT_DESCRIPTOR for Pipe00
==========
bLength = 0x8
bDescriptorType = 0x5 (USB_ENDPOINT_DESCRIPTOR_TYPE)
bEndpointAddress= 0x81 (INPUT)
bmAttributes= 0x3 (USB_ENDPOINT_TYPE_INTERRUPT)
wMaxPacketSize= 0x10, decimal 16
bInterval = 0x2, decimal 2

USB_ENDPOINT_DESCRIPTOR for Pipe01
==========
bLength = 0x8
bDescriptorType = 0x5 (USB_ENDPOINT_DESCRIPTOR_TYPE)
bEndpointAddress= 0x1 (OUTPUT)
bmAttributes= 0x3 (USB_ENDPOINT_TYPE_INTERRUPT)
wMaxPacketSize= 0x10, decimal 16
bInterval = 0x2, decimal 2

USB_ENDPOINT_DESCRIPTOR for Pipe02
==========
bLength = 0x8
bDescriptorType = 0x5 (USB_ENDPOINT_DESCRIPTOR_TYPE)
bEndpointAddress= 0x82 (INPUT)
bmAttributes= 0x2 (USB_ENDPOINT_TYPE_INTERRUPT)
wMaxPacketSize= 0x10, decimal 16
bInterval = 0x2, decimal 2

USB_ENDPOINT_DESCRIPTOR for Pipe03
==========
bLength = 0x8
bDescriptorType = 0x5 (USB_ENDPOINT_DESCRIPTOR_TYPE)
bEndpointAddress= 0x2 (OUTPUT)
bmAttributes= 0x2 (USB_ENDPOINT_TYPE_BULK)
wMaxPacketSize= 0x10, decimal 16
bInterval = 0xa, decimal 10

Each USB device (in our case the cutter) communicates to the host software (= software on Pc or Mac) through what are called pipes. The cutter has several pipes implemented. This means that there are several communications channels between the cutter and the host software.
The first one is called the Default Control Pipe, the operating software uses this pipe for plug and play. This pipe is used to get information about the USB configuration of the cutter.

PIPE01 is a **unidirectional** pipe that transports data from the host computer to the cutter. This data is the cut data (DM/PL or HP-GL).
PIPE00 is also **unidirectional** and carries information from the cutter to the host. For example you can poll the size of the media through this pipe (in DM/PL for example).
As an example to get the paper size of the cutter (in DM/PL) you will have to send ‘ ;: ER ’ on PIPE01 and you will have to read the response on PIPE00.
The third pipe has been called PIPE02. Reading from this pipe returns the free space in the internal buffer of the cutter. This pipe is used to implement a software handshake method.

Summa uses interrupt pipes and not bulk pipes for the communication. Interrupt pipes does not mean that you need an interrupt to transfer the data.

- **Writing Data To Cutter.**
  You must send the data in little packets (e.g. 256 bytes). This gives much better performance.

  When the data is split in packets of 256 bytes, the last packet will probably be smaller than 256 bytes. This is no problem. However make the last packet a multiple of 32 by padding the data with zero's.

- **Reading Data From Cutter**
  The "timeout" principle doesn't exists on the USB bus, at least not in the same way as in the serial port. So when querying info from the cutter, you normally first send data on PIPE01 and then read data on PIPE00. The cutter needs some time to put data on the USB bus after receiving the request. While the data is not ready the cutter will respond to any query with a zero length packet. This means the ReadPipe() function will return with 0 bytes read. You will have to call ReadPipe() until you get the desired response from the cutter. So when getting an answer from the cutter of zero bytes or less than expected does not mean that no data is available anymore. The cutter needs some time to put the data in its output buffer.

  Reading data must be done by multiples of 16.

- **Handshaking:**
  On Mac OS 9, there was a hardware handshake problem, maybe this is solved on some versions of OSX? However I advise to implement software handshaking. This is done by first checking if there is enough space in the buffer of the cutter (just read the data on PIPE02). If this value is high enough then send data to PIPE01.

- **Differences for Summa USB port 2,3 and 4:**
  Each cutter can be setup to 4 different port numbers, Summa USB port 1 is the standard port, the descriptor of this port is included in this document (zee above). The other ports differ only in the idProduct of the device descriptor. These are: :

  For SUMMAUSB PORT 1 : 0x100

  For SUMMAUSB PORT 2 : 0x102

  For SUMMAUSB PORT 3 : 0x103

  For SUMMAUSB PORT 4 : 0x104

- **Adapting the sample code from the SDK to communicate with the cutter.**

The easiest way to quickly test communication with the USB port is by adapting the sample code that is provided in the USB SDK by apple. The sample code to start from can be found in the folder : Developper/examples/Iokit/usb/USBSimpleExample.

I did adapt the following lines in order to send some data to the cutter.

1. In the function TransferData replace the lines

```
for(I=0;I<12;I++)
  outBuf[I] = 'R';
err = (*intf)->WritePipeAsync(intf,OutpipeRef,outbuf,12,…);
```

with

```
sprintf(outBuf,";: A D 100,100 U 0,0 Z");
err = (*intf)->WritePipeAsync(intf,OutpipeRef,outbuf,strlen(outBuf),…);
```

2. In the function dealWithPipes remove the lines

```
If(transferType != kUSBBulk)
{
        printf(…);
}
```

and change the lines:

```
// if(inPipeRef && outPiperef)
//transferData(intf, inPipeRef,outPipeRef);
```

with

```
if(inPipeRef && outPiperef)
transferData(intf, inPipeRef,2);
```

3. in the function main replace the lines

```
Sint32   idVendor = 1351;
Sint32   idProduct = 8193;
```

with

```
Sint32   idVendor = 0x099F;
Sint32   idProduct = 0x0100;
```