



HACKING THE USB WORLD WITH FACEDANCER

INTERFACES, ENDPOINTS, AND TRANSFERS

KATE TEMKIN & DOMINIC SPILL



ENDPOINTS

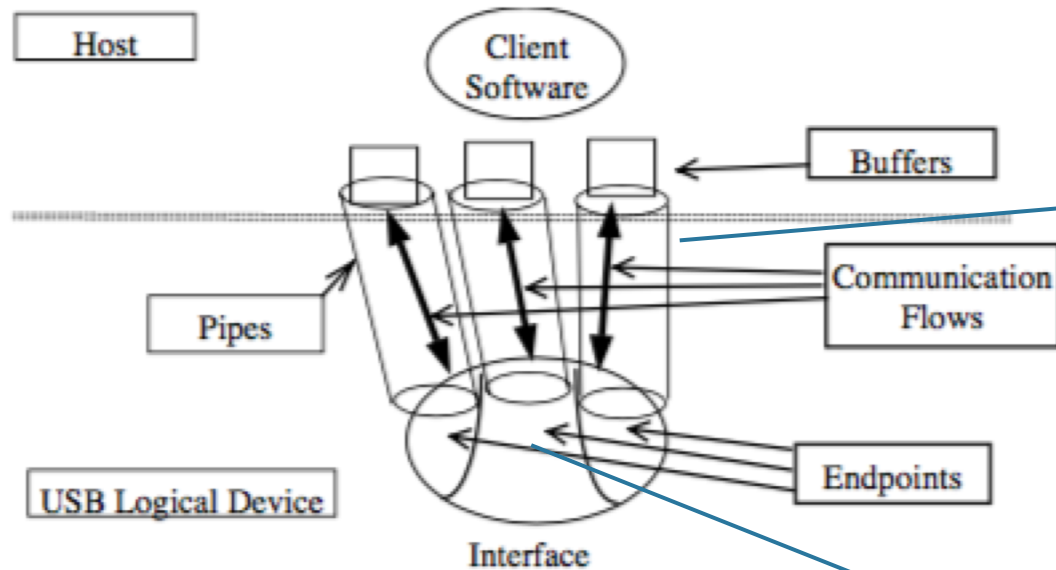


Figure 5-10. USB Communication Flow

Most USB devices use a number of **communication channels**, which are described in terms of their **endpoints**.

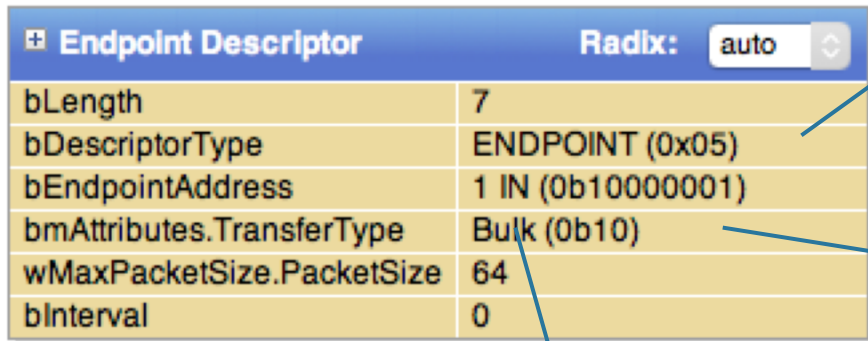
Endpoints both help to provide **conceptual channels** and help the host to **schedule packets**.

ENDPOINT TYPES

Type	Purpose
Control	Communications channel used for standard communications and simple packetized back-and-forth . Used for initial device discovery and setup. <i>Only transport that also specifies a packet format. EP0 is always a control endpoint.</i>
Bulk	Transport for shipping bytes ‘in bulk’. Bulk endpoints tend to be assigned the leftover bandwidth on the bus.
Interrupt	Transport for short bursts of latency-sensitive data. Used in cases that are similar to when you’d trigger an interrupt (e.g. keyboard keypress state).
Isochronous	Transport for data that grows “stale” if not delivered quickly— such as video frames from a camera.

ENDPOINT DESCRIPTORS

With the exception of the control endpoint, which is always present, each endpoint is described by an endpoint descriptor.



Endpoint Descriptor		Radix: auto
bLength	7	
bDescriptorType	ENDPOINT (0x05)	
bEndpointAddress	1 IN (0b10000001)	
bmAttributes.TransferType	Bulk (0b10)	
wMaxPacketSize.PacketSize	64	
bInterval	0	

The descriptor contains the endpoint's vital statistics, including its address and type. Note that the endpoint's directed is encoded in its address.

The MSB of the endpoint number specifies its address.

0x81 = endpoint one IN

0x80 = endpoint one OUT

INTERFACES

```
▼ Low Speed device @ 16 (0x14210000): ..... Composite device: "Endura Pro Keyboard "
```

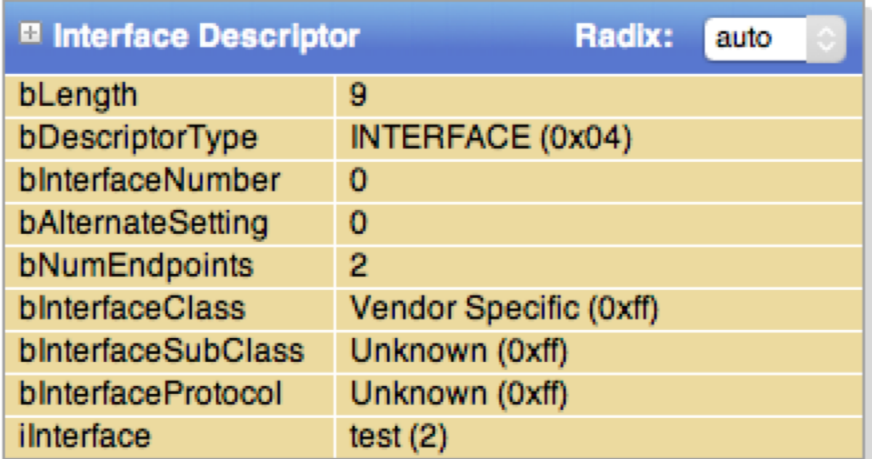
▶ Port Information:	0x0018
▶ Number Of Endpoints (includes EP0):	
▶ Device Descriptor	
▼ Configuration Descriptor (current config):	"HID KB/MS"
▶ Length (and contents):	59
Number of Interfaces:	2
Configuration Value:	1
Attributes:	0xA0 (bus-powered, remote wakeup)
MaxPower:	98 mA
▼ Interface #0 - HID/Boot Interface	"HID Keyboard"
Alternate Setting	0
Number of Endpoints	1
Interface Class:	3 (HID)
Interface Subclass;	1 (Boot Interface)
Interface Protocol:	1
▶ HID Descriptor	
▶ Endpoint 0x81 - Interrupt Input	
▼ Interface #1 - HID/Boot Interface	"HID Mouse"
Alternate Setting	0
Number of Endpoints	1
Interface Class:	3 (HID)
Interface Subclass;	1 (Boot Interface)
Interface Protocol:	2
▶ HID Descriptor	
▶ Endpoint 0x82 - Interrupt Input	

USB devices can present more than one logical function— for example, a device can be both a **keyboard** and **mouse**.

To help **organize endpoints**, and to allow the host to use **multiple drivers**, endpoints are organized into **interfaces** based on function.

INTERFACE DESCRIPTORS

Each interface is described by an **interface descriptor**, which describes the grouping for the **logical function**.



The screenshot shows a window titled "Interface Descriptor" with a "Radix:" dropdown set to "auto". The window contains a table with the following fields and values:

bLength	9
bDescriptorType	INTERFACE (0x04)
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	2
bInterfaceClass	Vendor Specific (0xff)
bInterfaceSubClass	Unknown (0xff)
bInterfaceProtocol	Unknown (0xff)
iInterface	test (2)

Interfaces can contain a description of the **device's class**, allowing the host to figure out if it has drivers for an **individual interface**.

CONFIGURATION DESCRIPTORS

Configuration Descriptor		Radix: auto
bLength	9	
bDescriptorType	CONFIGURATION (0x02)	
wTotalLength	32	
bNumInterfaces	1	
bConfigurationValue	1	
iConfiguration	None (0)	
bmAttributes.Reserved	0	
bmAttributes.RemoteWakeup	RemoteWakeup Supported (0b1)	
bmAttributes.SelfPowered	Bus Powered (0b0)	
bMaxPower	44mA (0x16)	

A device can opt to provide multiple collection of interfaces, and **allow the host to switch between those**. These collections are known as configurations, and are described using a **configuration descriptor**.

Each configuration provides a summary of its **expected power consumption**, allowing hosts to select configurations that **match power requirements**.

This is rarely used.

GETTING CONFIGURATION DESCRIPTORS

A `GET_DESCRIPTOR` can read a configuration and all of its 'subordinate' descriptors at once, if the length requested is long enough.

Get Configuration Descriptor	Index=0 Length=32
SETUP txn	80 06 00 02 00 00 20 00
IN txn [2 POLL]	09 02 20 00 01 01 00 A0
IN txn [1 POLL]	16 09 04 00 00 02 FF FF
IN txn [1 POLL]	FF 02 07 05 81 02 40 00
IN txn [1 POLL]	00 07 05 02 02 40 00 00
OUT txn	

Configuration Descriptor	Radix: auto
bLength	9
bDescriptorType	CONFIGURATION (0x02)
wTotalLength	32
bNumInterfaces	1
bConfigurationValue	1
iConfiguration	None (0)
bmAttributes.Reserved	0
bmAttributes.RemoteWakeup	RemoteWakeup Supported (0b1)
bmAttributes.SelfPowered	Bus Powered (0b0)
bMaxPower	44mA (0x16)

Interface Descriptor	Radix: auto
bLength	9
bDescriptorType	INTERFACE (0x04)
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	2
bInterfaceClass	Vendor Specific (0xff)
bInterfaceSubClass	Unknown (0xff)
bInterfaceProtocol	Unknown (0xff)
iInterface	test (2)

Endpoint Descriptor	Radix: auto
bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	1 IN (0b10000001)
bmAttributes.TransferType	Bulk (0b10)
wMaxPacketSize.PacketSize	64
bInterval	0

Endpoint Descriptor	Radix: auto
bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	2 OUT (0b00000010)
bmAttributes.TransferType	Bulk (0b10)
wMaxPacketSize.PacketSize	64
bInterval	0

SCOPING OUT A DEVICE

Device Descriptor		Radix: auto
bLength	18	
bDescriptorType	DEVICE (0x01)	
bcdUSB	1.10 (0x0110)	
bDeviceClass	Defined in Interface (0x00)	
bDeviceSubClass	Defined in Interface (0x00)	
bDeviceProtocol	Defined in Interface (0x00)	
bMaxPacketSize0	8	
idVendor	0x0403	
idProduct	0x6001	
bcdDevice	4.00 (0x0400)	
iManufacturer	ftdi (1)	
iProduct	test (2)	
iSerialNumber	ftE2G0FR (3)	
bNumConfigurations	1	

Configuration Descriptor		Radix: auto
bLength	9	
bDescriptorType	CONFIGURATION (0x02)	
wTotalLength	32	
bNumInterfaces	1	
bConfigurationValue	1	
iConfiguration	None (0)	
bmAttributes.Reserved	0	
bmAttributes.RemoteWakeup	RemoteWakeup Supported (0b1)	
bmAttributes.SelfPowered	Bus Powered (0b0)	
bMaxPower	44mA (0x16)	

Interface Descriptor		Radix: auto
bLength	9	
bDescriptorType	INTERFACE (0x04)	
bInterfaceNumber	0	
bAlternateSetting	0	
bNumEndpoints	2	
bInterfaceClass	Vendor Specific (0xff)	
bInterfaceSubClass	Unknown (0xff)	
bInterfaceProtocol	Unknown (0xff)	
iInterface	test (2)	

Endpoint Descriptor		Radix: auto
bLength	7	
bDescriptorType	ENDPOINT (0x05)	
bEndpointAddress	1 IN (0b10000001)	
bmAttributes.TransferType	Bulk (0b10)	
wMaxPacketSize.PacketSize	64	
bInterval	0	

Endpoint Descriptor		Radix: auto
bLength	7	
bDescriptorType	ENDPOINT (0x05)	
bEndpointAddress	2 OUT (0b00000010)	
bmAttributes.TransferType	Bulk (0b10)	
wMaxPacketSize.PacketSize	64	
bInterval	0	

The device descriptor contains the number of configurations present, which can then be used to issue a **GET_DESCRIPTOR** for each configuration.

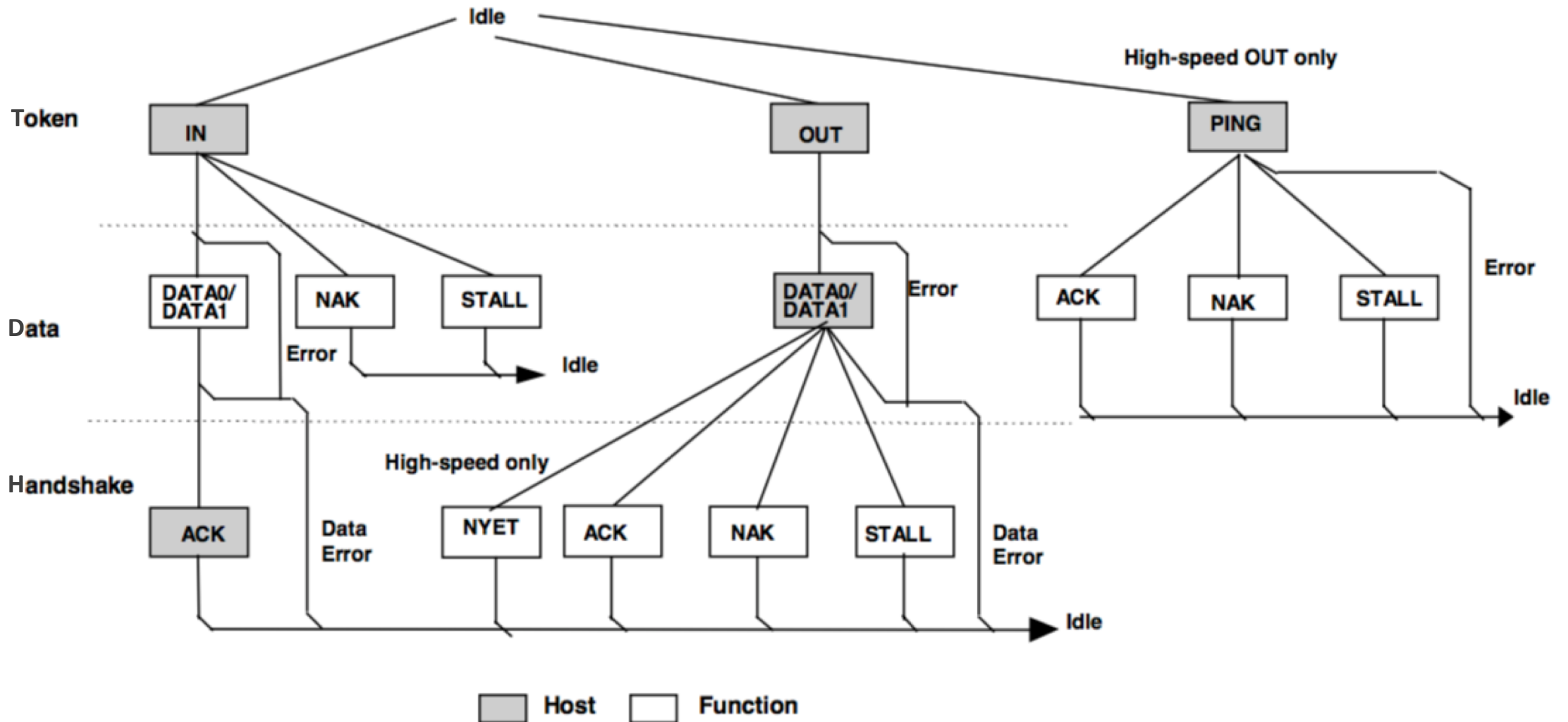
SET_CONFIGURATION REQUESTS

Standard Request		Radix: auto
bmRequestType.Recipient	Device (0b0)	
bmRequestType.Type	Standard (0b0)	
bmRequestType.Direction	Host-to-Device (0b0)	
bRequest	Set Configuration (0x9)	
wValue	Configuration Value (0x1)	
wIndex	0x0	
wLength	0x0	

Before any endpoints other than EP0 can be used, a configuration **must be selected** using a SET_CONFIGURATION request.

Setting configuration zero marks the device as **unconfigured**.

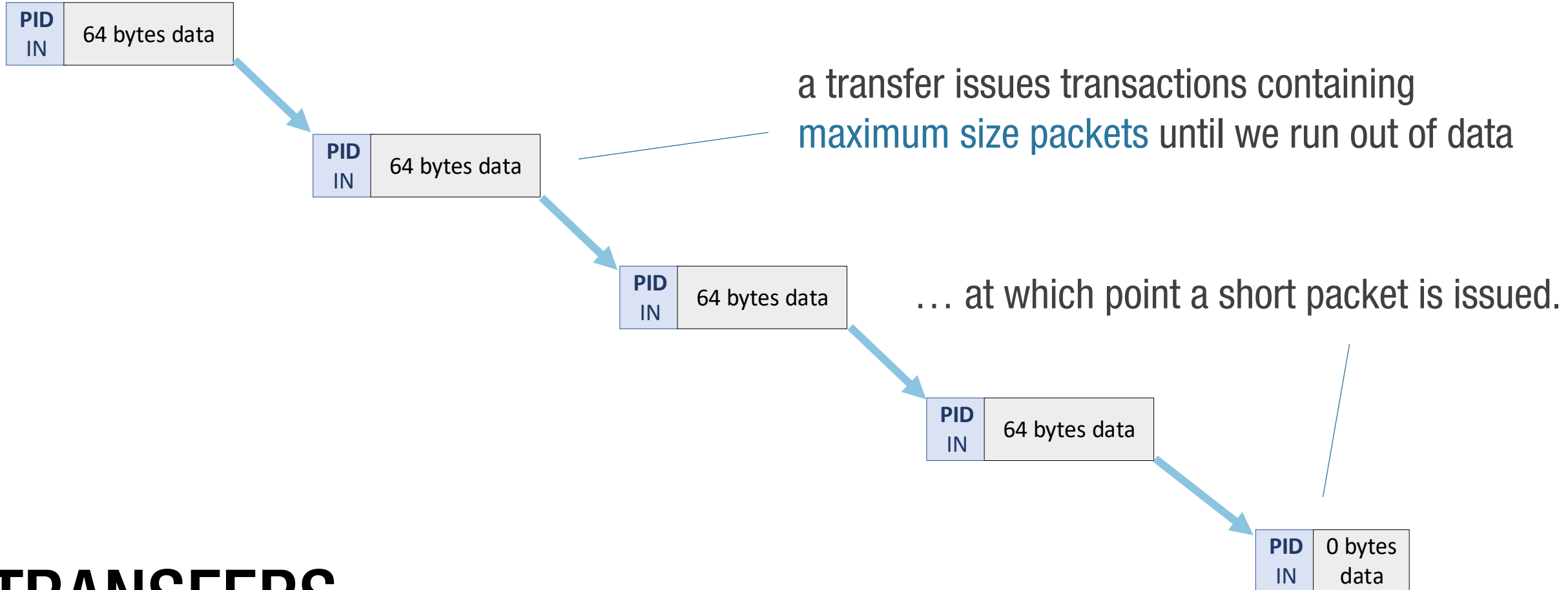
▼ Set Address	Address=27
▶ SETUP txn	00 05 1B 00 00 00 00 00
▶ IN txn	



BULK TRANSACTIONS

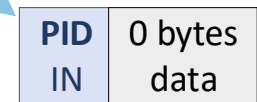
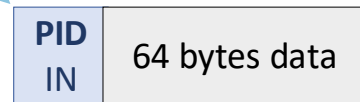
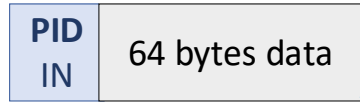
All diagrams from the USB 2.0 specification.

Field	Value	Field	Value	Field	Value	Field	Value	Field	Value
Length	256	Length	192	Length	128	Length	64	Length	0
Address	0x1000	Address	0x1040	Address	0x1080	Address	0x10C0	Address	0x1100



TRANSFERS

Field	Value	Field	Value	Field	Value	Field	Value	Field	Value
Length	256	Length	192	Length	128	Length	64	Length	0
Address	0x1000	Address	0x1040	Address	0x1080	Address	0x10C0	Address	0x1100



Usually, both device and host controllers automate breaking transfers into transactions.

This has some interesting security consequences.

TRANSFERS