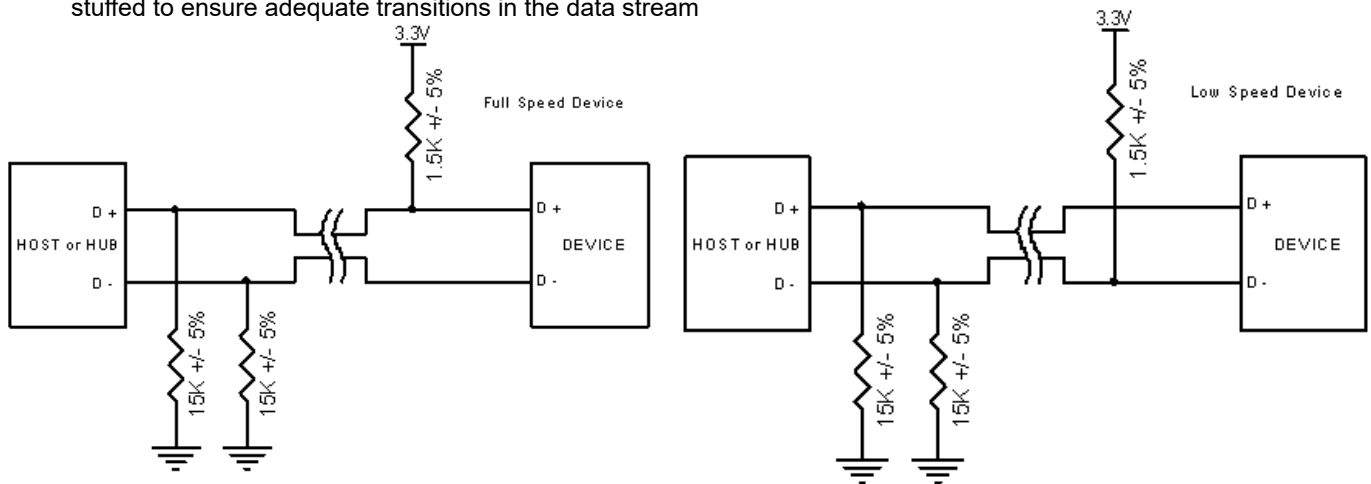


Physical layer

USB uses a differential transmission pair for data. This is encoded using NRZI and is bit stuffed to ensure adequate transitions in the data stream



The states: LOW speed: J= D+: 0, D-: 1, K= D+: 1, D-: 0
 HIGH/FULL speed: J= D+: 1, D-: 0, K= D+: 0, D-: 1
 SE0 = D+: 0, D-: 0,
 SE1 = D+: 1, D-: 1

The protocol

Common USB Packet Fields

PID stands for Packet ID. This field is used to identify the type of packet that is being sent. First 4 bits - essential. Last 4 - the same, for error correcting.

Token	OUT Token	0001	Start the frame
	IN Token	1001	
	SOF Token	0101	
	SETUP Token	1101	
Data	DATA0	0011	
	DATA1	1011	
	DATA2	0111	
	MDATA	1111	
Handshake	ACK Handshake	0010	
	NAK Handshake	1010	
	STALL Handshake	1110	
	NYET (No Response Yet)	0110	
Special	PREamble	1100	
	ERR	1100	
	Split	1000	
	Ping	0100	

SYNC .The sync field is 8 bits long at low and full speed or 32 bits long for high speed and is used to synchronise the clock of the receiver with that of the transmitter.The format is: KJKJKJKK

ADDR The address field specifies which device the packet is designated for. Being 7 bits in length allows for 127 devices to be supported.

ENDP The endpoint field is made up of 4 bits, allowing 16 possible endpoints.

CRC Cyclic Redundancy Checks are performed on the data within the packet payload.

EOP End of packet.

USB Packet Types

1.Token packets:

In -A host Informs the USB device "I want to read".

Out - A host Informs the USB device "I want to write".

Setup - For designation of the begin control transfers.

Sync	PID	ADDR	ENDP	CRC5	EOP
------	-----	------	------	------	-----

2.Data packets

Maximum payload:

low speed - 8bytes,

full speed - 1023 bytes

high speed - 1024 bytes

SYNC	PID	DATA	CRC16	EOP
------	-----	------	-------	-----

3.Handshake packets

ACK - a packet has been received successfully

NAK - a device can't send/receive data temporary.

It also used in interrupt transaction to inform the host "there is no data to send"

STALL- the device finds in a state that is requires interventionfrom a host

SYNC	PID	EOP
------	-----	-----

4.Start of frame packet

Sends by a host :

full speed - $1 \pm 0.5\text{mS}$

high speed: $125\text{uS} \pm 0.0625\text{uS}$

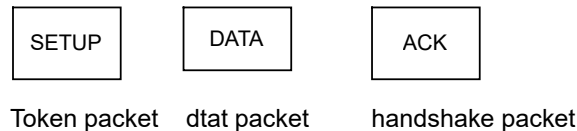
sync	PID	Frame Number	CRC5	EOP
------	-----	--------------	------	-----

Endpoints

Each device must have the endpoint 0: IN and OUT. Endpoint /transfer may be the following types:

✓ 1. Control transfers

Set up an USB device with "standard device requests" functions, also in enumeration phase. The transfer consists of three stages: SETUP, DATA, HANDSHAKE



✓ 2. Interrupt transfers

- Guaranteed latency,
- Stream pipe-bidirectional (IN, OUT endpoints),
- Error detection and period retry

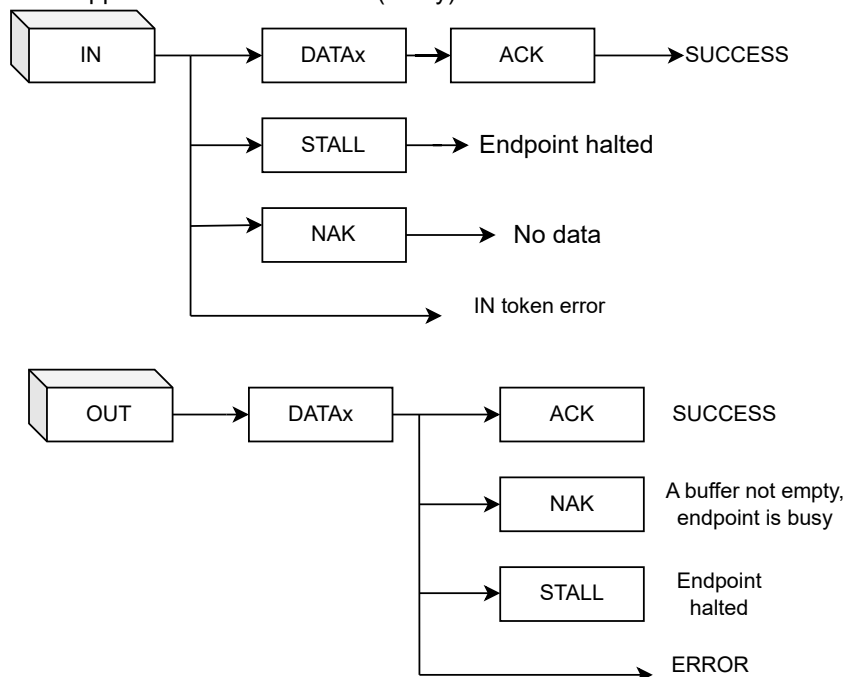
NOTE: in all the types of transfers the **Host is always the initiator of a transaction, it always begins. Interrupt NOT means that a device interrupts a host. The device MUST wait until enumeration has finished until any Tx/Rx actions - to avoid errors.**

Payload (max): low sp.-8bytes, full sp.-64bytes, high sp.1024bytes.

The bInterval may be 1-255

IN: The host periodically polling to a device with "IN" token. The device is waiting for this token. When a device has a data for the host, it send the data to the host and waiting for ACK/NACK host's response. The period of polling described in the endpoint descriptor, **bInterval**

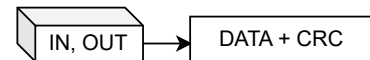
OUT: The host sends OUT token with followed data. When a device has not empty buffer or busy now - it sends NAK. When there a data error happened - STALL returned (rarely)



CONCLUSIONS: interrupt transfers used for time sensitive precise transactions: the host sending the "IN" tokens-asks every bInterval (1-250mS) time, so a packet sends each bInterval exactly on-time. The maximum packet size is 64bytes. A Packet size is FIXED and described in the endpoint descriptor - wMaxPacketSize? **Every interrupt transaction must use this exact size or smaller.** Delivery guaranteed because of ACK. Interrupt used for small periodic time sensitive data transmissions. Avoid large data - it can leads to blocking other devices on USB bus

✓ 3. Isochronous transfers

- Guaranteed access to USB bandwidth
- Stream pipe-bidirectional (IN, OUT)
- Error detection via CRC? but no retry guarantee to delivery
- No ACK, NAK, STALL handshake - only data and CRC



payload: high speed - 1024 Bytes, full speed - 1023 Bytes

Used for audio/video stream, or application where allows data loose

Interval must be 1 for FS and 1-16 for HS

NOTE: The device MUST wait until enumeration has finished until any Tx/Rx actions - to avoid errors.

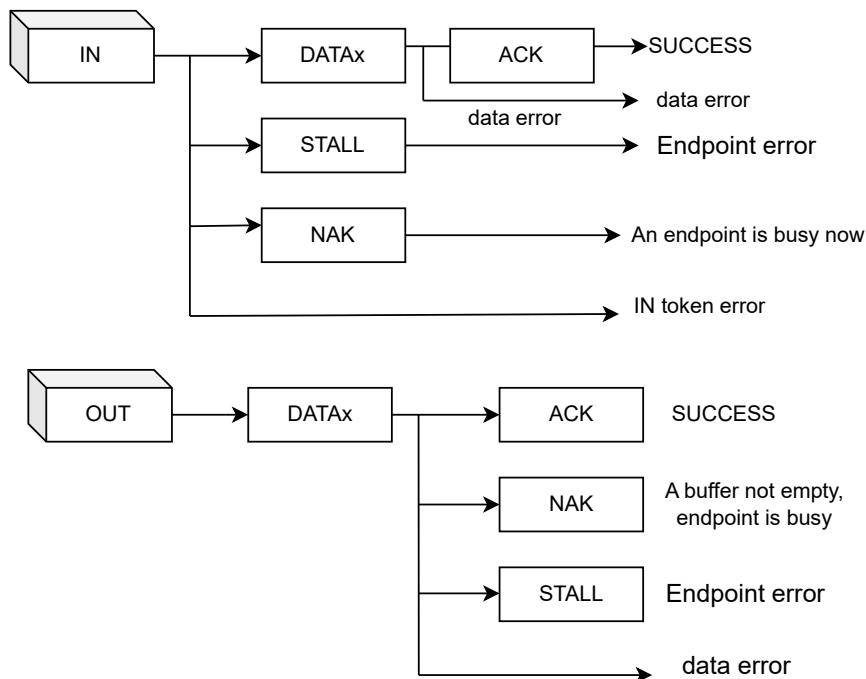
✓ 4. Bulk transfers

- Used to transfer large bursty data.
- Error detection via CRC, with guarantee of delivery.
- No guarantee of bandwidth or minimum latency.
- Stream Pipe - Unidirectional
- Full & high speed modes only

The maximum packet size : Full Speed - 8,16,32,64Bytes,

High Speed - 512Bytes

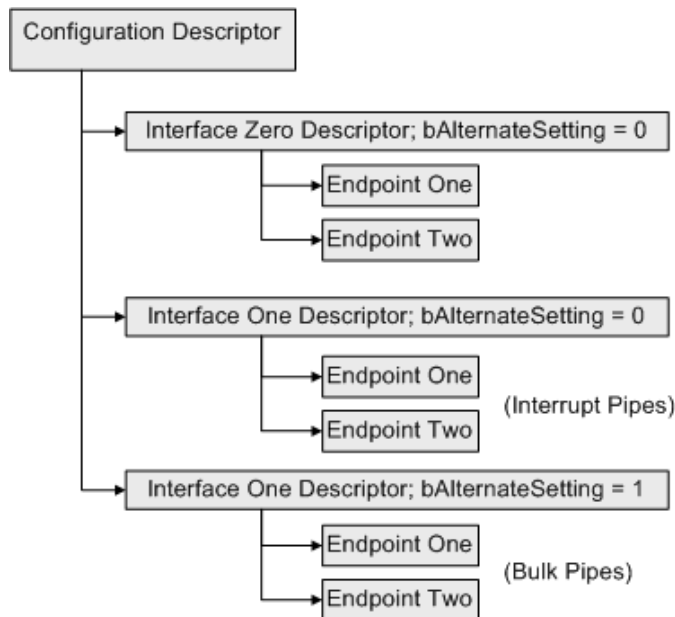
NOTE: the data size can be less than maximum defined amount, even no data (zero)



CONCLUSIONS: bulk transfers used for not time sensitive transactions: the host sending the "IN" when has a scheduled time, so there may be random delays. The maximum packet size is 64bytes. A Packet size is FIXED and described in the endpoint descriptor - wMaxPacketSize? **Every interrupt transaction must use this exact size or smaller.** Delivery guaranteed because of ACK. Bulk used for large data transmission.

USB descriptors

A USB device can have only one device descriptor



DEVICE DESCRIPTOR

One USB device can have only one device descriptor

bLength = 0x12 , // Size of this descriptor (always 18 bytes).

bDescriptorType = 0x01 // Tells host “this is a device descriptor.”

bcdUSB = 0x0200 // The USB specification version device complies with.

bDeviceClass = 0x00 //0 means “I have multiple interfaces; check them individually.” But when not 0, it can be: 0x02 = CDC, 0x03 = HIDm 0x08 = Mass Storage

bDeviceSubClass = 0x00 // Class-specific. Here unused → 0.

bDeviceProtocol = 0x00 // Protocol-specific. Here unused → 0.

bMaxPacketSize0 = 0x40 //Maximum packet size on **endpoint 0** (control).

Configuration descriptor

Total power need, bus powered or selfpowered, how many interfaces belongs to this config, total size of all descriptors in this config

bLength = 09 //Size of this descriptor in bytes.

bDescriptorType = 02 //Descriptor type = **Configuration** descriptor.

wTotalLength = 0x0020 //Total length of entire configuration, including:
 *Configuration descriptor (9 bytes)
 *Interface descriptor
 *Endpoint descriptors
 *Class descriptors (if any)

```

bNumInterfaces = 01           //Number of Interfaces inside this configuration

bConfigurationValue = 01       //The ID of this configuration.Passed later to
                                SET_CONFIGURATION(1)

iConfiguration = 00           //Index of a string descriptor describing this configuration., 0-
no                               string

bmAttributes = 0x80           //
                                | Bit | Meaning |
                                | --- | --- |
                                | 7 | Must be 1 |
                                | 6 | Self-powered (1 = yes, 0 = bus-powered) |
                                | 5 | Remote wakeup capability |
                                | 4-0 | Reserved |

bMaxPower = 0x32             //(50 × 2mA = 100mA)

```

Interface descriptor

It is a "function" inside the device, for example: mouse, keyboard, audio...

-class(HID, CDC, Audio...)

-Subclass/Protocol

-How many endpoints belong to this function

```

bLength = 09 //A Size of this descriptor in bytes (always 9 for an interface descriptor)

bDescriptorType = 04 //Descriptor type 4 = Interface Descriptor

bInterfaceNumber = 00 //Index of this interface inside the configuration.
                        If your device has multiple interfaces (CDC, HID, Audio) they get numbers
                        0, 1, 2....

bAlternateSetting = 00 //Allows different settings for the same interface. Usually 0 unless
                        // using ISO endpoints with different bandwidths

bNumEndpoints = 01 //Number of endpointsDoes NOT count endpoint 0.Here: 1 endpoint
                        //(usually interrupt IN for keyboard).

bInterfaceClass = 03 //USB Class Code. 0x03 = HID (Human Interface Device).

bInterfaceSubClass = 01 //HID subclass.0x01 = Boot Interface (keyboard or mouse that BIOS can
                        use).

bInterfaceProtocol = 01 //Protocol ID. 0x01 = Keyboard (0x02 would be mouse)

iInterface = 00 //Index of a string descriptor describing this interface.
                  0 → no string.

```

Endpoint descriptor

Description of data-pipe inside an interface.

- Direction (IN/OUT),
- Transfer type (Interrupt, isochronous, bulk , control)
- Maximum packet size
- pooling interval (for interrupt/isochronous transfers)

```
bLength = 0x09           //Length of this descriptor (always 9 bytes)

bDescriptorType = 0x05   //Value 5 means: Endpoint Descriptor

bEndpointAddress = 0x81

                               //Binary: 1000 0001
                               Bit 7 = direction
                               1 = IN (device → host)
                               0 = OUT (host → device)
                               Bits 0-3 = endpoint number → Endpoint 1.

bmAttributes = 0x02       //Bits meaning:
                               Bits 0–1 = transfer type
                               00 = Control
                               01 = Isochronous
                               10 = Bulk
                               11 = Interrupt
                               Bits 2–7 = reserved

wMaxPacketSize = 0x0040 //Maximum size of one DATA packet.At Full
                               Speed: Bulk/Interrupt max = 64 bytes

bInterval = 0x00         //Used only by Interrupt and Isochronous endpoints. For Bulk
                               endpoints: ignored.
```

*

String descriptor

Format breakdown

- **bLength**: A 1-byte field indicating the total length of the descriptor in bytes.
- **bDescriptorType**: A 1-byte field with a value of 0x03 to identify it as a string descriptor.
- **bString**: A variable-length field containing the Unicode string data. Each character in the string uses two bytes (UTF-16LE), and a 0x00 byte is added after each standard ASCII character to convert it to UTF-16LE.

Standard USB device requests

1. **GET_STATUS** (0x00) A host asks the device :
 - are you self powered or bus powered?
 - are you remote wake-up enable?
 - For an n endpoint: is it halted?

2. **CLEAR_FEATURE** (0x01)

Tells device to disable some feature
most commonly used to clear "ENDPOINT_HALT"
This re-enable an endpoint

3. **SET_FEATURE** (03h)

opposite to CLEAR_FEATURE. Most commonly used for:
-SLALL an endpoint,
-enable remote wake-up

4. **SET_ADDRESS** (05h)

A host assigns to a device a unique address
(1-127). Mandatory for all the USB devices.

5. **GET_DESCRIPTOR** (06h) A host requests one of three descriptors:

- Device
- Interface
- Configuration
- Endpoint
- String
- HID or class specific

6. **SET_DESCRIPTOR** (07h) most never used

7. **GET_CONFIGURATION** (08h)

A host asks a device: which configuration is active now?

8. **SET_CONFIGURATION** (09h)

A host select, which configuration on a device should
It activates all the interfaces and endpoints in select

9. **GET_INTERFACE** (0Ah)

For devices with alternate interfaces. A host asks, which interface

10. **SET_INTERFACE** (0Bh)

Select an alternate settings. Used for example in USB Audio
packet size or bandwidth.

11. **SYNC_FRAME** (0x0C) Used only in isochronous endpoints. A host asks - what is your current

Enumeration

The process identical for all USB devices, no matter the transform type is. Enumeration used the
Endpoint 0 and Address 0

1) The device plug-in. The host resets the device, a device entry into default state, address=0

2) A host sends GET_DESCRIPTOR (Device, it is recommended first 8 bytes but not mandatory 8), a device responds with 8 bytes of the Device Descriptor. Important fields:

- USB version, -PID,
- VID, -DeviceClass,
- number of configurations, maximal endpoint 0 packet size

3) A host reset the device

4) A host sends the request SET_ADDRESS in range 1-127, the device switches in addressed mode

5) A host sends GET_DESCRIPTOR (Device, full) request. Now a host asks for 18 byte Device Descriptor Used to determine:

- Which driver category (CDC, HID...)
- Whether the device is composite:
- How many configuration exists

6) A host sends the request GET_DESCRIPTOR (Configuration). This descriptor includes whole hierarchy:

Configuration descriptor:

Interface0:

Endpoint 1

Endpoint 1

(maybe class specific)

Interface 1:

Endpoint 2

Endpoint 2

It tells to a host - how many interfaces, type of each interface, (HID, CDC, e.t.c.), what endpoints exists, their direction and transfer type, polling interval, maximal packet size.

7) Host sends the request SET_CONFIGURATION. A host selects a configuration number (usually 1). The device switches to the Configured state. Now:

- All endpoints are in active state
- Drivers can start using them
- The device is ready for normal operation

Enumeration example

