USB Made Simple

Back

Part 5 - Example Device

Forward

Typical Human Interface Device (HID)

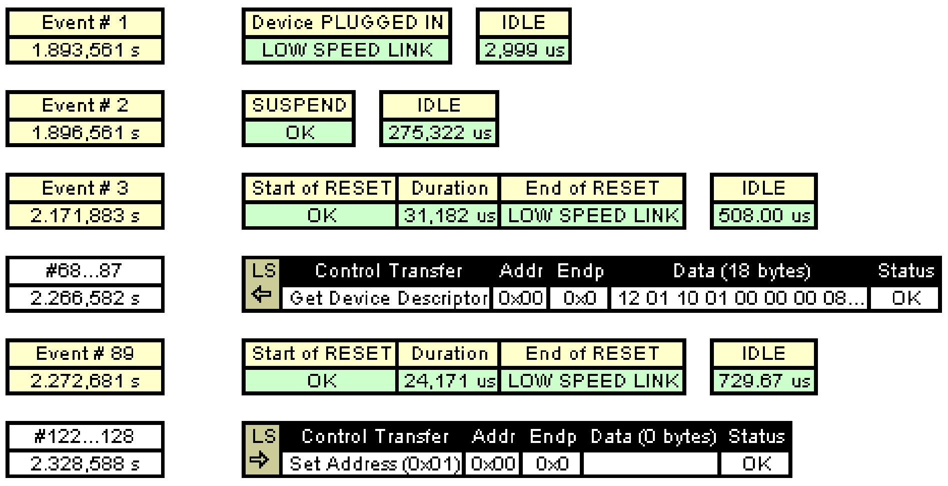
A Mouse

We are going to look at a typical enumeration and subsequent operation of one of the simplest USB devices around; the mouse. Below you will see the output of a hardware bus analyser which is capturing all the USB traffic involved when a mouse is plugged in. Let us emphasise straight away that this is just a typical sequence, which can vary widely depending on the host, which in this case was Windows XP.

For the display shown below, the analyser has been set to display only Bus States, and the highest level Transfers. Each Control Transfer shown is made up of a series of transactions, each of which is made up of a series of packets, as we shall see later.

The capture begins 1.9 seconds before the mouse is plugged in. The analyser indicates that the device has been plugged in, and that it a low speed device (because the pull-up resistor is on D-). After 3 ms, because the host is not yet allowed to send any data, the device is SUSPENDED because of not seeing any activity on the data lines. Shortly afterwards, the host issues a RESET which in this case lasted 31 ms.

It then performs a Get Descriptor request (to the default address 0), to discover the maximum packet size defined for the control endpoint. Having got this information, it resets the device again, and then sends a Set Address request, setting device address to 1 in this example.



Event # 1	Device PLUGGED IN	IDLE
1.893,561 s	LOW SPEED LINK	2,999 us

Event # 2	SUSPEND	IDLE
1.896,561 s	OK	275,322 us

Event # 3	Start of RESET	Duration	End of RESET	IDLE
2.171,883 s	OK	31,182 us	LOW SPEED LINK	508.00 us

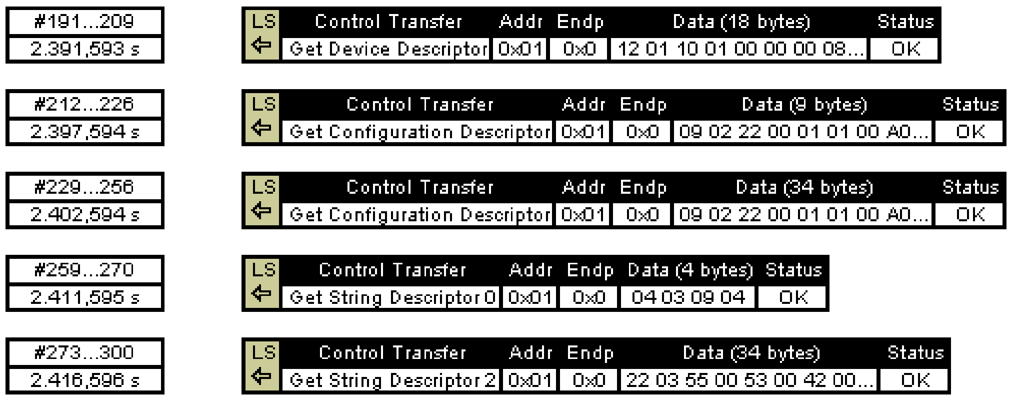
#68...87	LS	Control Transfer	Addr	Endp	Data (18 bytes)	Status
2.266,582 s	⇦	Get Device Descriptor	0x00	0x0	12 01 10 01 00 00 00 08...	OK

Event # 89	Start of RESET	Duration	End of RESET	IDLE
2.272,681 s	OK	24,171 us	LOW SPEED LINK	729.67 us

#122...128	LS	Control Transfer	Addr	Endp	Data (0 bytes)	Status
2.328,588 s	⇨	Set Address (0x01)	0x00	0x0		OK

All the requests from now on are sent to device address 1. The host has now requested the device descriptor. (Click here to see analysis of the device descriptor in a separate window). It has also requested the first nine bytes of the configuration descriptor collection. Remember that when the host requests the configuration descriptor, it will also get, following it, the interface and endpoint descriptors and maybe others. But the host doesn't know the total length of this collection. It does, however, know that the configuration descriptor itself is 9 bytes long, and that this descriptor contains a value for the total length of the descriptor collection. So it starts by requesting just the configuration descriptor using Get Configuration Descriptor with a required length of 9. (Click here to see an analysis of the configuration descriptor collection in a separate window.)

Additionally the host has requested String Descriptor 0, for the list of supported string languages, and descriptor index 2, which in this case has been assigned to the product description string.



#191...209	LS	Control Transfer	Addr	Endp	Data (18 bytes)	Status
2.391,593 s	⇦	Get Device Descriptor	0x01	0x0	12 01 10 01 00 00 00 08...	OK

#212...226	LS	Control Transfer	Addr	Endp	Data (9 bytes)	Status
2.397,594 s	⇦	Get Configuration Descriptor	0x01	0x0	09 02 22 00 01 01 00 A0...	OK

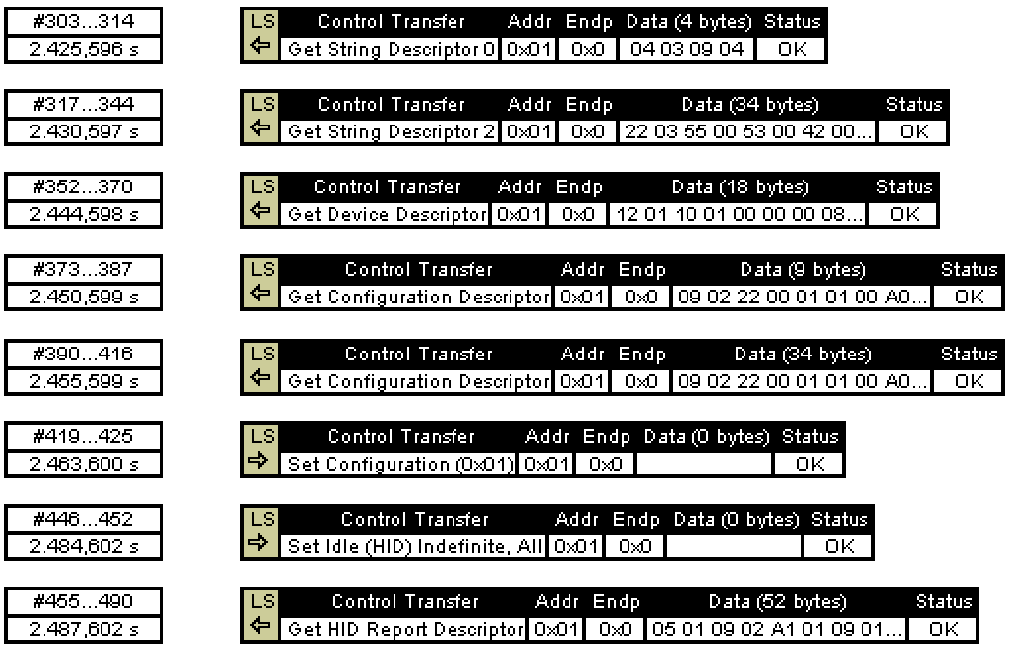
#229...256	LS	Control Transfer	Addr	Endp	Data (34 bytes)	Status
2.402,594 s	⇦	Get Configuration Descriptor	0x01	0x0	09 02 22 00 01 01 00 A0...	OK

#259...270	LS	Control Transfer	Addr	Endp	Data (4 bytes)	Status
2.411,595 s	⇦	Get String Descriptor 0	0x01	0x0	04 03 09 04	OK

#273...300	LS	Control Transfer	Addr	Endp	Data (34 bytes)	Status
2.416,596 s	⇦	Get String Descriptor 2	0x01	0x0	22 03 55 00 53 00 42 00...	OK

After this it can be seen that the host has asked for much of the information again. This can occur for various reasons, such as the different drivers in the stack each asking the same questions for their own purposes.

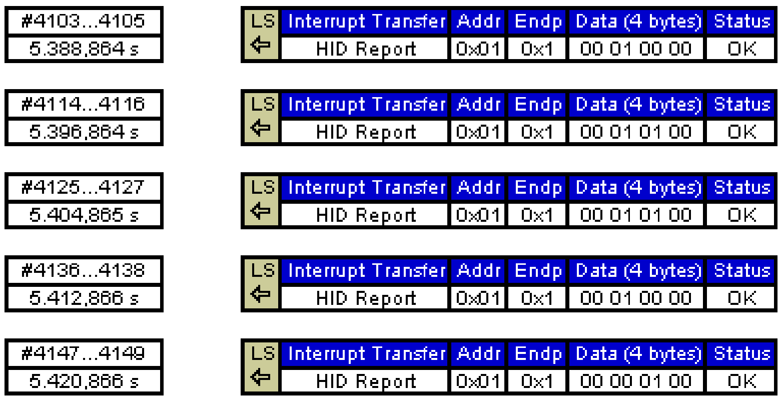
The host has then sent the Set Configuration request, and from that point in time, the device is 'configured' and is able to perform its purpose in life. The host is now able to send the HID class request 'Set Idle', to tell the device only to respond to an interrupt IN transaction if a new event occurs. (*In any case when an IN request is sent and there is no change to report, the device will respond with a NAK packet. The analyser has been set not to display these 'NAKed' transactions so we will not see them here*) It then also requests the HID class report descriptor, which in this case informs the appropriate driver to expect to receive a 4 byte report of mouse events on its interrupt IN endpoint.



Packet	Time	Speed	Control Transfer	Addr	Endp	Data	Status
#303...314	2.425,596 s	LS ←	Get String Descriptor 0	0x01	0x0	(4 bytes) 04 03 09 04	OK
#317...344	2.430,597 s	LS ←	Get String Descriptor 2	0x01	0x0	(34 bytes) 22 03 55 00 53 00 42 00...	OK
#352...370	2.444,598 s	LS ←	Get Device Descriptor	0x01	0x0	(18 bytes) 12 01 10 01 00 00 00 08...	OK
#373...387	2.450,599 s	LS ←	Get Configuration Descriptor	0x01	0x0	(9 bytes) 09 02 22 00 01 01 00 A0...	OK
#390...416	2.455,599 s	LS ←	Get Configuration Descriptor	0x01	0x0	(34 bytes) 09 02 22 00 01 01 00 A0...	OK
#419...425	2.463,600 s	LS →	Set Configuration (0x01)	0x01	0x0	(0 bytes)	OK
#446...452	2.484,602 s	LS →	Set Idle (HID) Indefinite, All	0x01	0x0	(0 bytes)	OK
#455...490	2.487,602 s	LS ←	Get HID Report Descriptor	0x01	0x0	(52 bytes) 05 01 09 02 A1 01 09 01...	OK

At this point the driver starts sending interrupt IN requests, and when any event is available to be reported the interrupt data transfer succeeds and 4 bytes of data are transferred.

You may notice that there was a nearly 3 second gap before the mouse was first moved. In the meantime there were still regular interrupt IN transactions, which were NAKed by the device, as it had nothing to report. To avoid the display being swamped by these NAKed transactions, the analyser has been set not to display them.



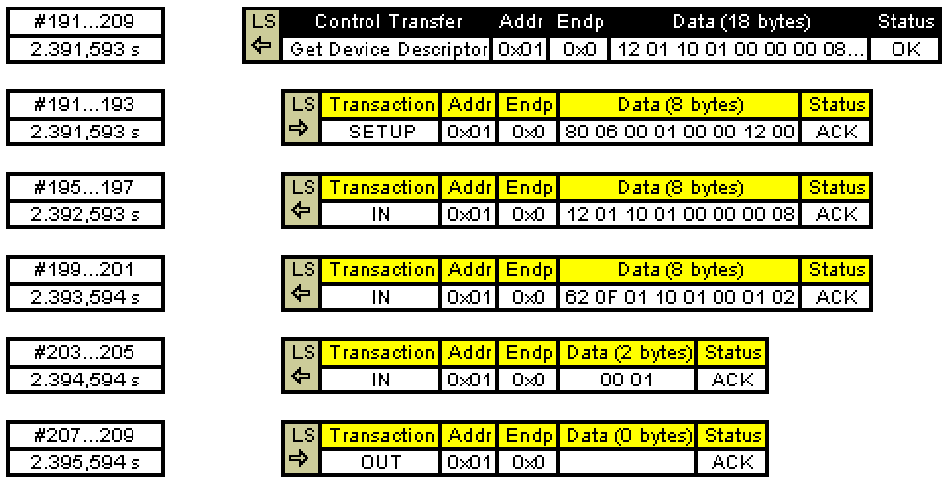
Packet	Time	Speed	Interrupt Transfer	Addr	Endp	Data (4 bytes)	Status
#4103...4105	5.388,864 s	LS ←	HID Report	0x01	0x1	00 01 00 00	OK
#4114...4116	5.396,864 s	LS ←	HID Report	0x01	0x1	00 01 01 00	OK
#4125...4127	5.404,865 s	LS ←	HID Report	0x01	0x1	00 01 01 00	OK
#4136...4138	5.412,866 s	LS ←	HID Report	0x01	0x1	00 01 00 00	OK
#4147...4149	5.420,866 s	LS ←	HID Report	0x01	0x1	00 00 01 00	OK

We are now going to examine one of these control transfers in more detail. By clicking a button on the analyser we can reveal the transactions which make up a particular control transfer. We will select a Get Device Descriptor control transfer to examine.

As you can see the Get Device Descriptor is made up, in this case, of five transactions. The first transaction (SETUP) comprises the setup stage.

The next three (IN) transactions represent the data stage, during which the 18 bytes of the descriptor are transferred back to the host.

The final (OUT) transaction is the status stage, to acknowledge successful completion.



#191...209	LS	Control Transfer	Addr	Endp	Data (18 bytes)	Status
2.391,593 s	⇦	Get Device Descriptor	0x01	0x0	12 01 10 01 00 00 00 08...	OK

#191...193	LS	Transaction	Addr	Endp	Data (8 bytes)	Status
2.391,593 s	⇨	SETUP	0x01	0x0	80 06 00 01 00 00 12 00	ACK

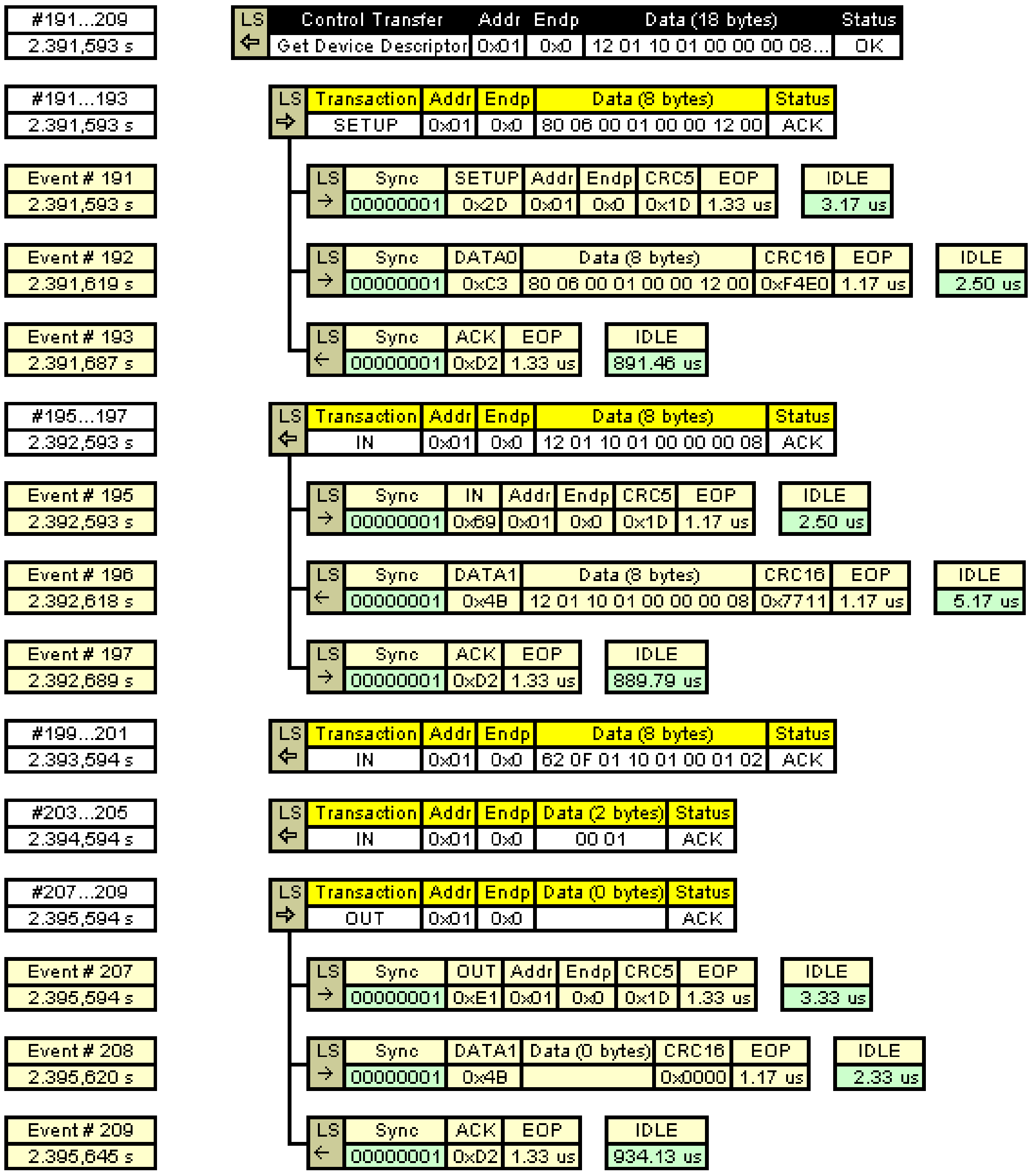
#195...197	LS	Transaction	Addr	Endp	Data (8 bytes)	Status
2.392,593 s	⇦	IN	0x01	0x0	12 01 10 01 00 00 00 08	ACK

#199...201	LS	Transaction	Addr	Endp	Data (8 bytes)	Status
2.393,594 s	⇦	IN	0x01	0x0	62 0F 01 10 01 00 01 02	ACK

#203...205	LS	Transaction	Addr	Endp	Data (2 bytes)	Status
2.394,594 s	⇦	IN	0x01	0x0	00 01	ACK

#207...209	LS	Transaction	Addr	Endp	Data (0 bytes)	Status
2.395,594 s	⇨	OUT	0x01	0x0		ACK

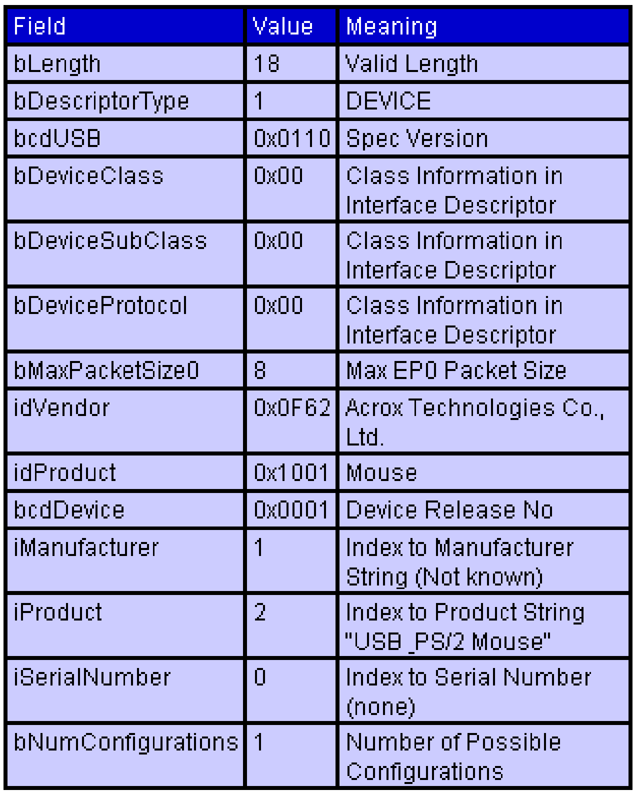
Each of these transactions is made up of packets. By selecting some of these transactions to expand, we can see the details of the packets. Notice that we have chosen to expand only one of the three IN transactions, to keep the picture smaller. Notice how, for example the SETUP transaction is made up of three packets. You should be able to recognise the component parts of each packet by now.



```
#191...209    LS  Control Transfer       Addr  Endp  Data (18 bytes)             Status
2.391,593 s   ⇦   Get Device Descriptor  0x01  0x0   12 01 10 01 00 00 00 08...  OK

#191...193    LS  Transaction  Addr  Endp  Data (8 bytes)           Status
2.391,593 s   ⇨   SETUP        0x01  0x0   80 06 00 01 00 00 12 00  ACK

Event # 191   LS  Sync      SETUP  Addr  Endp  CRC5  EOP      IDLE
2.391,593 s   →   00000001  0x2D   0x01  0x0   0x1D  1.33 us  3.17 us

Event # 192   LS  Sync      DATA0  Data (8 bytes)           CRC16   EOP      IDLE
2.391,619 s   →   00000001  0xC3   80 06 00 01 00 00 12 00  0xF4E0  1.17 us  2.50 us

Event # 193   LS  Sync      ACK   EOP      IDLE
2.391,687 s   ←   00000001  0xD2  1.33 us  891.46 us

#195...197    LS  Transaction  Addr  Endp  Data (8 bytes)           Status
2.392,593 s   ⇦   IN           0x01  0x0   12 01 10 01 00 00 00 08  ACK

Event # 195   LS  Sync      IN    Addr  Endp  CRC5  EOP      IDLE
2.392,593 s   →   00000001  0x69  0x01  0x0   0x1D  1.17 us  2.50 us

Event # 196   LS  Sync      DATA1  Data (8 bytes)           CRC16   EOP      IDLE
2.392,618 s   ←   00000001  0x4B   12 01 10 01 00 00 00 08  0x7711  1.17 us  5.17 us

Event # 197   LS  Sync      ACK   EOP      IDLE
2.392,689 s   →   00000001  0xD2  1.33 us  889.79 us

#199...201    LS  Transaction  Addr  Endp  Data (8 bytes)           Status
2.393,594 s   ⇦   IN           0x01  0x0   62 0F 01 10 01 00 01 02  ACK

#203...205    LS  Transaction  Addr  Endp  Data (2 bytes)  Status
2.394,594 s   ⇦   IN           0x01  0x0   00 01           ACK

#207...209    LS  Transaction  Addr  Endp  Data (0 bytes)  Status
2.395,594 s   ⇨   OUT          0x01  0x0                   ACK

Event # 207   LS  Sync      OUT   Addr  Endp  CRC5  EOP      IDLE
2.395,594 s   →   00000001  0xE1  0x01  0x0   0x1D  1.33 us  3.33 us

Event # 208   LS  Sync      DATA1  Data (0 bytes)  CRC16   EOP      IDLE
2.395,620 s   →   00000001  0x4B                   0x0000  1.17 us  2.33 us

Event # 209   LS  Sync      ACK   EOP      IDLE
2.395,645 s   ←   00000001  0xD2  1.33 us  934.13 us
```

A good analyser will be able to show you the data from the descriptor in an analysed form, to save you the trouble. (Click here to see analysis of the device descriptor in a separate window).

Device Descriptor Analysis



Field	Value	Meaning
bLength	18	Valid Length
bDescriptorType	1	DEVICE
bcdUSB	0x0110	Spec Version
bDeviceClass	0x00	Class Information in Interface Descriptor
bDeviceSubClass	0x00	Class Information in Interface Descriptor
bDeviceProtocol	0x00	Class Information in Interface Descriptor
bMaxPacketSize0	8	Max EP0 Packet Size
idVendor	0x0F62	Acrox Technologies Co., Ltd.
idProduct	0x1001	Mouse
bcdDevice	0x0001	Device Release No
iManufacturer	1	Index to Manufacturer String (Not known)
iProduct	2	Index to Product String "USB PS/2 Mouse"
iSerialNumber	0	Index to Serial Number (none)
bNumConfigurations	1	Number of Possible Configurations

Data Content

```
0000:   12  01  10  01  00  00  00  08      ........
0008:   62  0F  01  10  01  00  01  02      b.......
0010:   00  01                              ..
```

In a similar way, we might have selected the configuration descriptor set to display in analysed form. The interface descriptor tells us, and the host, that it is a HID class device. The bInterfaceSubClass is usually 0 in HID class devices except for a mouse or a keyboard which meet the simplified protocol requirements for being operated by the BIOS code, before the usual USB drivers have been loaded. In this case we notice that the device will work with the boot interface and (from bInterfaceProtocol) that it is a mouse. The usual HID driver will learn about this in another way; by parsing the HID report descriptor.

We notice that this device has a single Interrupt IN endpoint in addition to the default control endpoint, and that it is set to be interrogated once every 10 ms and expects the host to read 4 bytes each time. (Click here to see a fuller analysis of the configuration descriptor collection in a separate window.)

As with any HID device the descriptor following the interface descriptor is the HID descriptor whose main job is to tell the host where to find the HID Report Descriptor. This is the means by which the device can specify what it is and the detailed content of reports it may send and/or receive.

Control Transfer

Get Configuration Descriptor

A request for a configuration descriptor returns the configuration descriptor, all interface, endpoint, OTG, class- or vendor-specific descriptors for all of the class- or vendor-specific descriptors for all of the interfaces in a single request.

Field	Value	Meaning
bLength	9	Valid length
bDescriptorType	2	CONFIGURATION
wTotalLength	34	Total combined size of this set of descriptors
bNumInterfaces	1	Number of interfaces supported by this configuration
bConfigurationValue	1	Value to use as an argument to the SetConfiguration() request to select this configuration
iConfiguration	0	Index of string descriptor describing this configuration
bmAttributes (Self-Powered)	0	Bus-Powered
bmAttributes (Remote Wakeup)	1	Yes
bmAttributes (Other bits)	0x80	Valid
bMaxPower	100 mA	Maximum Current Drawn by Device in This Configuration

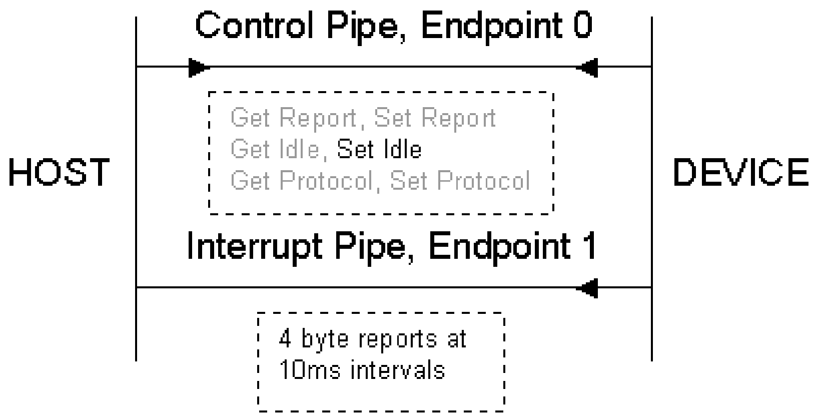
Field	Value	Meaning
bLength	9	Valid length
bDescriptorType	4	INTERFACE
bInterfaceNumber	0	Zero-based Number of this Interface.
bAlternateSetting	0	Value used to select this alternative setting for the interface identified in the prior field
bNumEndpoints	1	Number of endpoints used by this interface (excluding endpoint zero).
bInterfaceClass	0x03	HID
bInterfaceSubClass	0x01	Boot Interface
bInterfaceProtocol	0x02	Mouse
iInterface	0	Index of string descriptor describing this Interface

Field	Value	Meaning
bLength	9	Valid length
bDescriptorType	0x21	HID
bcdHID	0x0110	HID Class Spec Version
bCountryCode	0	Not Supported
bNumDescriptors	1	Number of Descriptors
bDescriptorType	REPORT	Descriptor Type 34
wDescriptorLength	52	Descriptor Length

wDescriptorLength	52	Descriptor Length

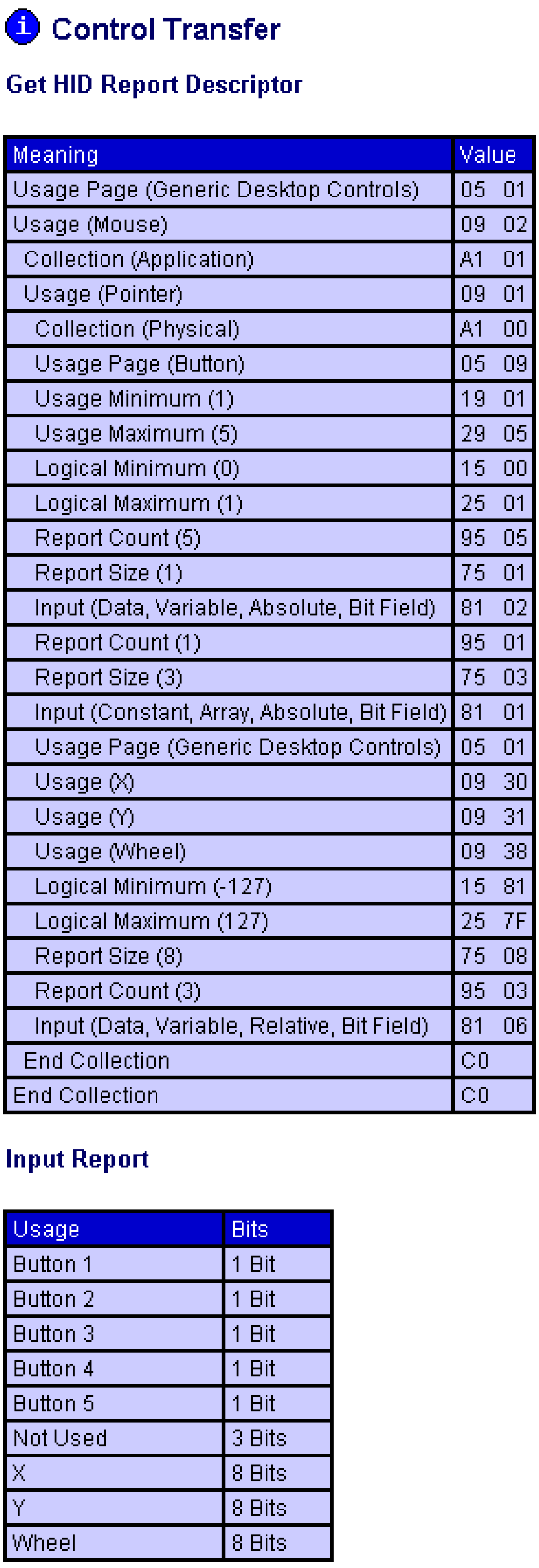
Field	Value	Meaning
bLength	7	Valid length
bDescriptorType	5	ENDPOINT
bEndpointAddress	0x81	Endpoint 1 - IN
bmAttributes	0x03	Interrupt. Data Endpoint.
wMaxPacketSize	0x0004	Maximum Packet Size is 4
bInterval	0x0A	10 Frames (10 ms)

We have learned that when the device has been configured, the host will start an IN interrupt (on endpoint 1 IN) to read a report or reports up to 4 bytes long at intervals of 10 ms (or possibly less, typically it would be 8 ms with Windows).


Control Pipe, Endpoint 0
HOST
Get Report, Set Report
Get Idle, Set Idle
Get Protocol, Set Protocol
DEVICE
Interrupt Pipe, Endpoint 1
4 byte reports at 10ms intervals

So once the mouse is configured, the picture above represents the communication channels possible in the mouse. It still has the bi-directional control endpoint, and can receive the six HID class requests shown in the picture, and it has the interrupt IN endpoint, for sending its reports of mouse events. Typically, and in our example here, the only class request out of the 6 defined HID requests which is used, is the Set Idle request.

We can now select the Get HID Report descriptor to analyse. The display below shows the contents and significance of the HID report descriptor, which, using a form of coding specified in the HID specification, defines one or more reports which are the means of transferring information to or from a HID device. The parsed form of the report desriptor is shown below. Parsing a HID report is a fairly complex operation, so the analyser has helped out by displaying the defined reports, or in this case the one report defined. It is a single Input report, with 5 buttons, and X and Y movement, and a wheel movement, which make up a total of 4 bytes to match the maximum size of the interrupt endpoint.



Control Transfer

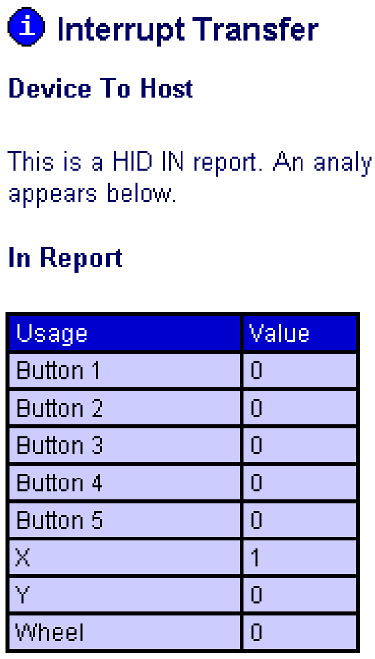
Get HID Report Descriptor

Meaning	Value
Usage Page (Generic Desktop Controls)	05 01
Usage (Mouse)	09 02
Collection (Application)	A1 01
Usage (Pointer)	09 01
Collection (Physical)	A1 00
Usage Page (Button)	05 09
Usage Minimum (1)	19 01
Usage Maximum (5)	29 05
Logical Minimum (0)	15 00
Logical Maximum (1)	25 01
Report Count (5)	95 05
Report Size (1)	75 01
Input (Data, Variable, Absolute, Bit Field)	81 02
Report Count (1)	95 01
Report Size (3)	75 03
Input (Constant, Array, Absolute, Bit Field)	81 01
Usage Page (Generic Desktop Controls)	05 01
Usage (X)	09 30
Usage (Y)	09 31
Usage (Wheel)	09 38
Logical Minimum (-127)	15 81
Logical Maximum (127)	25 7F
Report Size (8)	75 08
Report Count (3)	95 03
Input (Data, Variable, Relative, Bit Field)	81 06
End Collection	C0
End Collection	C0

Input Report

Usage	Bits
Button 1	1 Bit
Button 2	1 Bit
Button 3	1 Bit
Button 4	1 Bit
Button 5	1 Bit
Not Used	3 Bits
X	8 Bits
Y	8 Bits
Wheel	8 Bits

To complete the picture we now examine the content of one of the interrupt transfers (events 4103 to 4105 in the first sequence we looked at). The bytes transferred were 00 01 00 00. If we select the Interrupt Transfer to analyse we will see how the meaning matches up with the parsed report descriptor analysis.



Interrupt Transfer

Device To Host

This is a HID IN report. An analysis of the report contents appears below.

In Report

Usage	Value
Button 1	0
Button 2	0
Button 3	0
Button 4	0
Button 5	0
X	1
Y	0
Wheel	0

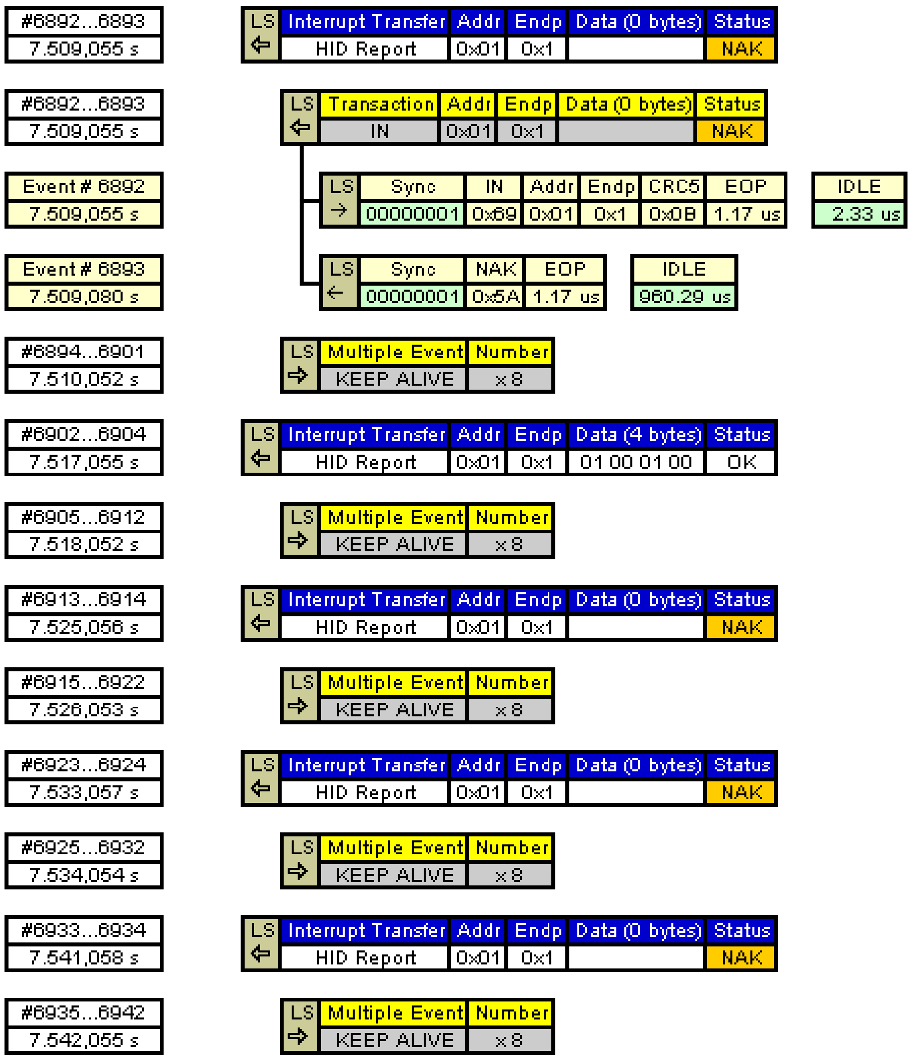
We can tell from this that the mouse moved 1 unit in the X direction.

What have we not examined?

We have not seen any Keep Alive signals in our displays. There is at least one Keep Alive signal every frame, or every 1 ms. This makes for a very cluttered display, even when they are grouped together by the analyser. So we have set the analyser not to display the Keep Alive pulses for the sake of this discussion.

In the same way, the interrupt IN transfers have only been shown when successful. Every 8 ms there was an attempt to perform an IN transaction by the host, and most of these were NAKed by the device. The analyser was set not to shown these NAKed transactions.

Here is a section of the capture containing these elements with the analyser set to reveal them. In addition, one of the NAKed interrupts has been expanded so you can see what it is built up from.



```
#6892...6893  7.509,055 s   LS ← Interrupt Transfer: HID Report | Addr 0x01 | Endp 0x1 | Data (0 bytes) | Status NAK
#6892...6893  7.509,055 s     LS ← Transaction: IN | Addr 0x01 | Endp 0x1 | Data (0 bytes) | Status NAK
Event # 6892  7.509,055 s       LS → Sync 00000001 | IN 0x69 | Addr 0x01 | Endp 0x1 | CRC5 0x0B | EOP 1.17 us | IDLE 2.33 us
Event # 6893  7.509,080 s       LS ← Sync 00000001 | NAK 0x5A | EOP 1.17 us | IDLE 960.29 us
#6894...6901  7.510,052 s     LS → Multiple Event: KEEP ALIVE | Number x 8
#6902...6904  7.517,055 s   LS ← Interrupt Transfer: HID Report | Addr 0x01 | Endp 0x1 | Data (4 bytes) 01 00 01 00 | Status OK
#6905...6912  7.518,052 s     LS → Multiple Event: KEEP ALIVE | Number x 8
#6913...6914  7.525,056 s   LS ← Interrupt Transfer: HID Report | Addr 0x01 | Endp 0x1 | Data (0 bytes) | Status NAK
#6915...6922  7.526,053 s     LS → Multiple Event: KEEP ALIVE | Number x 8
#6923...6924  7.533,057 s   LS ← Interrupt Transfer: HID Report | Addr 0x01 | Endp 0x1 | Data (0 bytes) | Status NAK
#6925...6932  7.534,054 s     LS → Multiple Event: KEEP ALIVE | Number x 8
#6933...6934  7.541,058 s   LS ← Interrupt Transfer: HID Report | Addr 0x01 | Endp 0x1 | Data (0 bytes) | Status NAK
#6935...6942  7.542,055 s     LS → Multiple Event: KEEP ALIVE | Number x 8
```

Summary

We have examined the actual transfers involved during enumeration and operation of a real device, a mouse.

Coming up...

Next we will look at High Speed USB which uses a data rate of 480 Mb/s on the same cables as Full Speed USB.

Forward

- Detailed timing information
- Full analysis of all standard events
- Results can be printed
- Optional class analysis modules

Click for more information