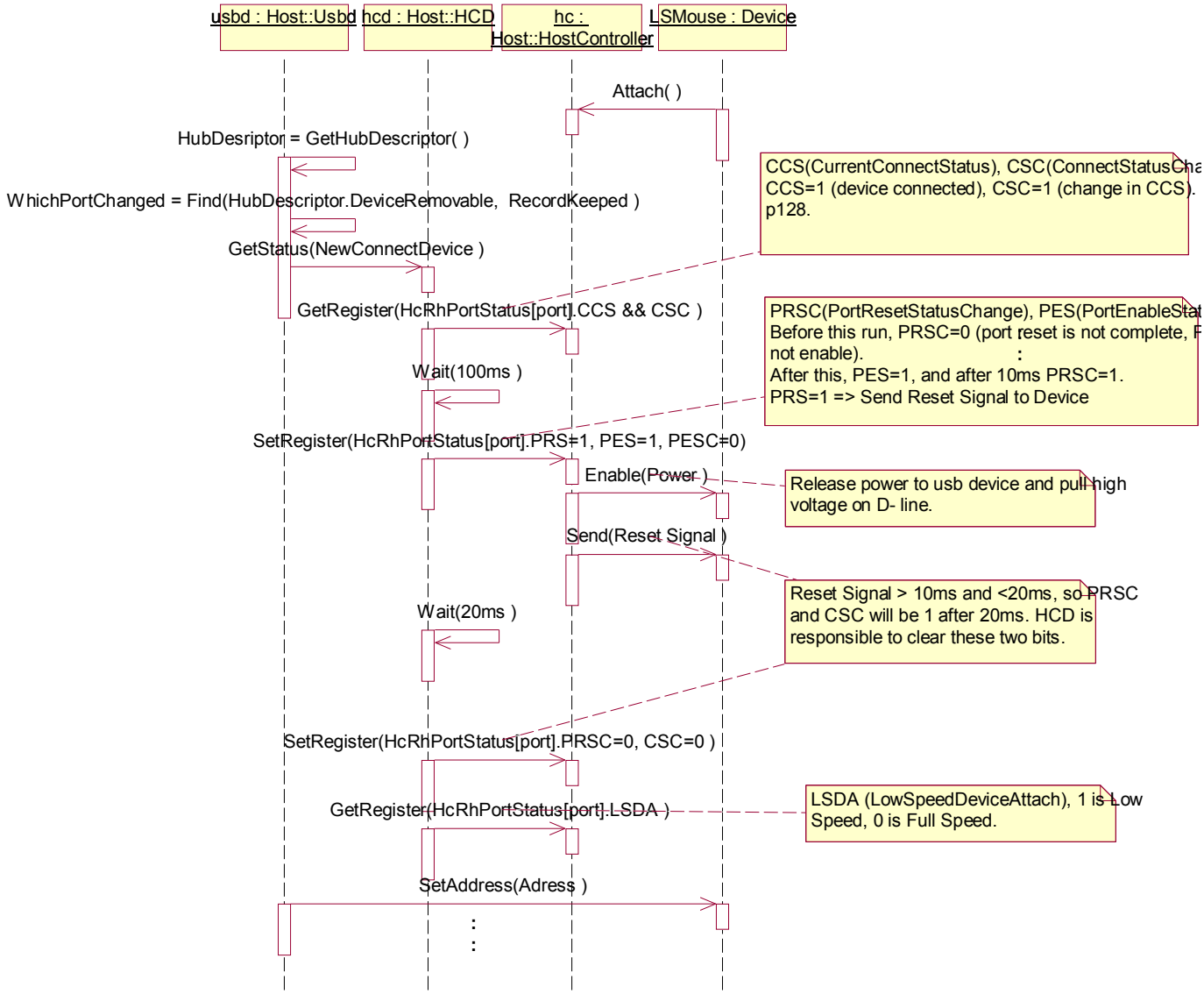

usbd : Host::Usbd
hcd : Host::HCD
hc : Host::HostController
LSMouse : Device
Attach( )
HubDesriptor = GetHubDescriptor( )
WhichPortChanged = Find(HubDescriptor.DeviceRemovable, RecordKeeped)
GetStatus(NewConnectDevice)
GetRegister(HcRhPortStatus[port].CCS && CSC)
Wait(100ms)
SetRegister(HcRhPortStatus[port].PRS=1, PES=1, PESC=0)
Enable(Power)
Send(Reset Signal)
Wait(20ms)
SetRegister(HcRhPortStatus[port].PRSC=0, CSC=0)
GetRegister(HcRhPortStatus[port].LSDA)
SetAddress(Adress)
CCS(CurrentConnectStatus), CSC(ConnectStatusCha
CCS=1 (device connected), CSC=1 (change in CCS).
p128.
PRSC(PortResetStatusChange), PES(PortEnableStat
Before this run, PRSC=0 (port reset is not complete, F
not enable).
After this, PES=1, and after 10ms PRSC=1.
PRS=1 => Send Reset Signal to Device
Release power to usb device and pull high voltage on D- line.
Reset Signal > 10ms and <20ms, so PRSC and CSC will be 1 after 20ms. HCD is responsible to clear these two bits.
LSDA (LowSpeedDeviceAttach), 1 is Low Speed, 0 is Full Speed.

Bus Enumeration (p243).

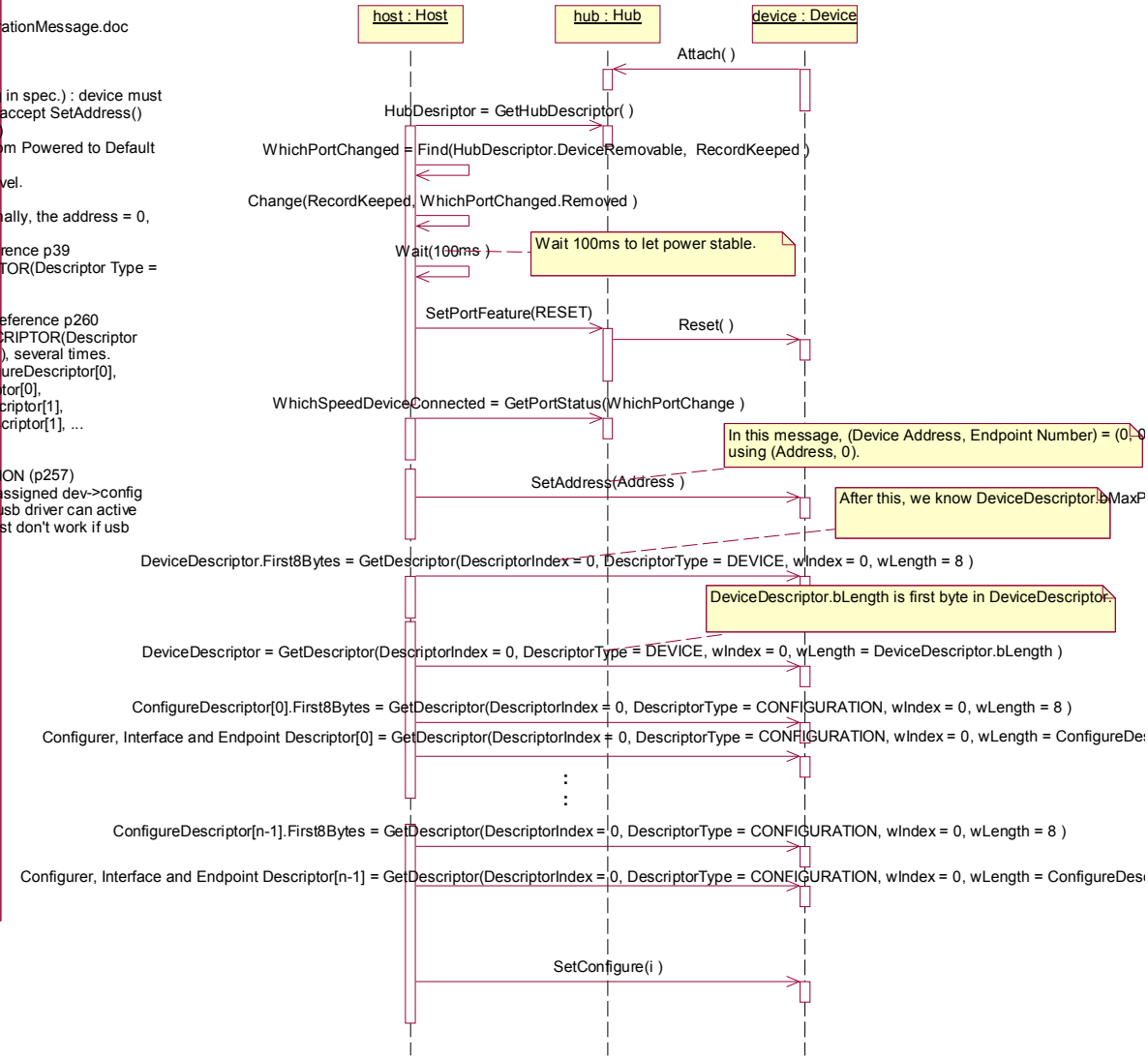
The Message contents ref. BusEnumerationMessage.doc

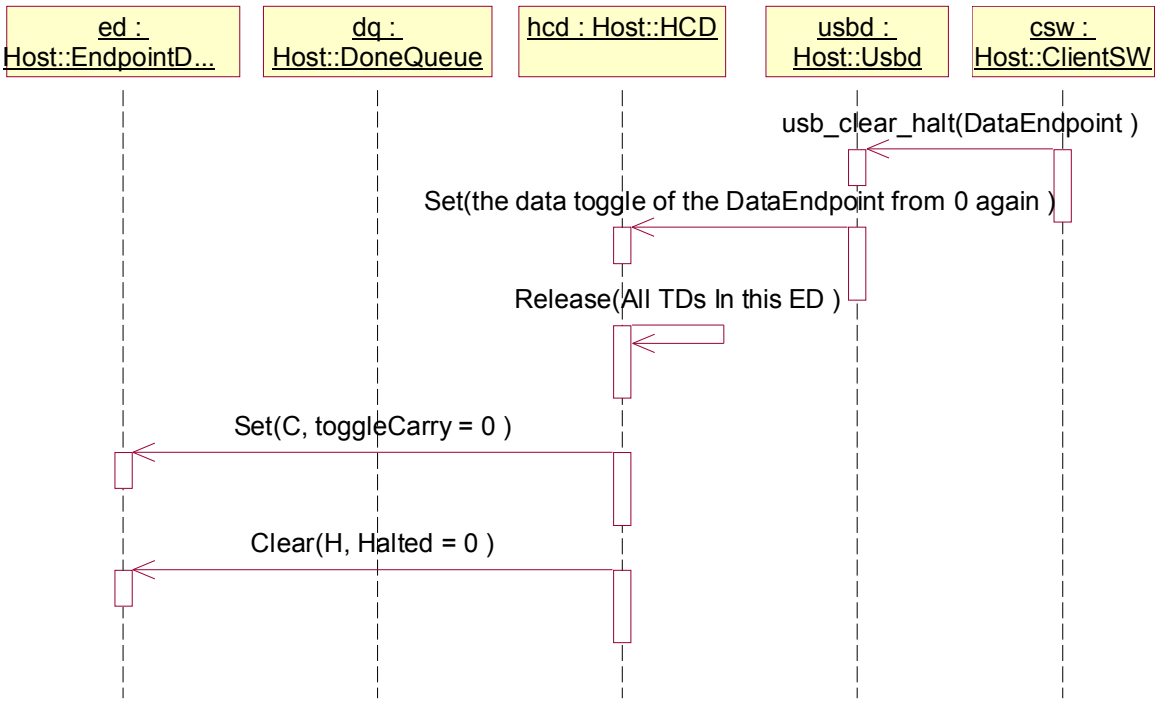
SetPortFeature(RESET) = Reset (using in spec.) : device must finish Reset in 10ms, and device must accept SetAddress() request after Reset. (p313, p153, p145)
After Reset, the device change state from Powered to Default (p243) and hub enable this port (p313).
Reset is done under Host Controller Level.

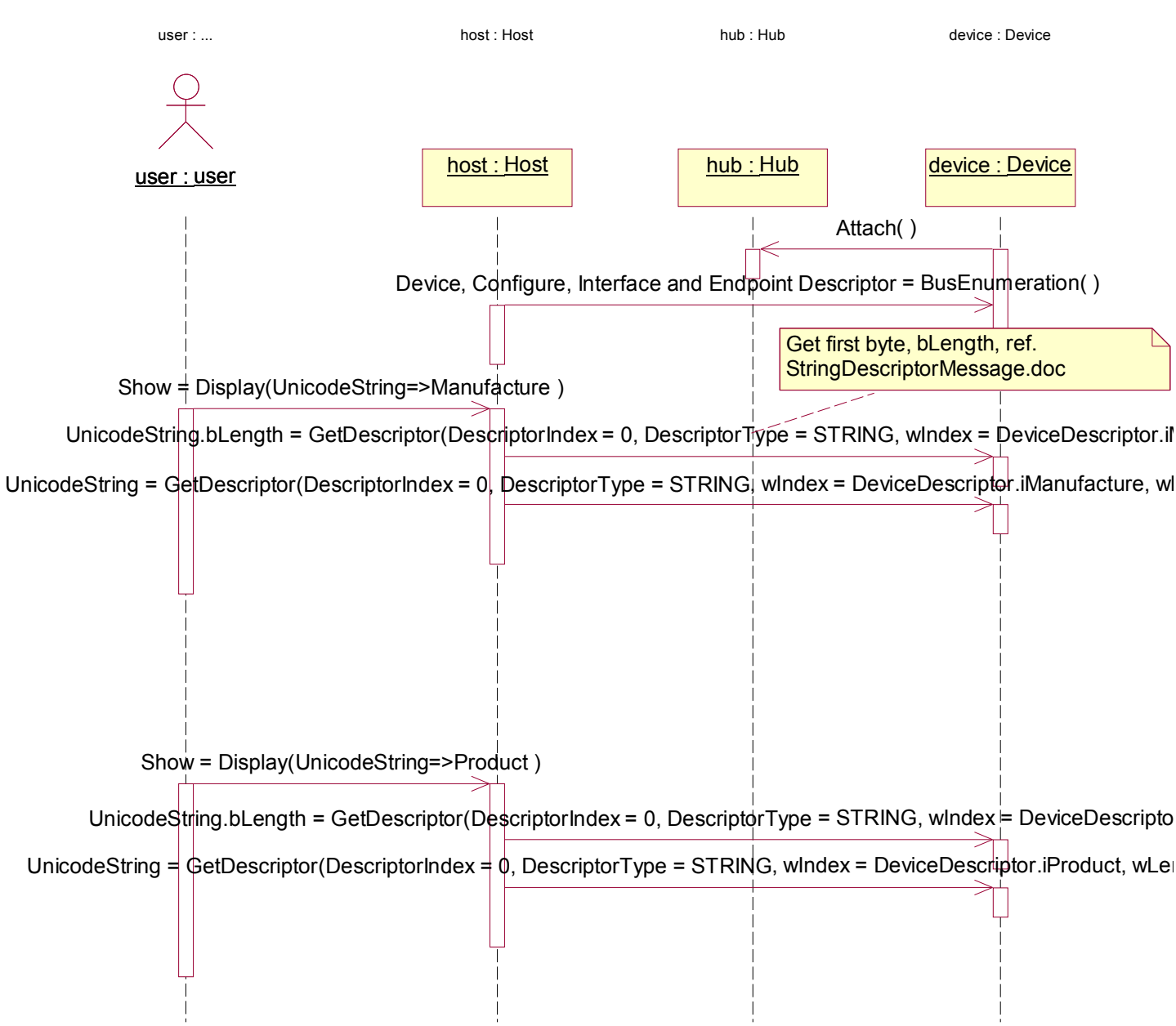
SetAddress() = SET_ADDRESS : Normally, the address = 0, endpoint = 0 (p247, p256)
GetDeviceDescriptorFirst8Bytes() : reference p39
GetDeviceDescriptor() = GET_DESCIPTOR(Descriptor Type = DEVICE) (p251, p253)

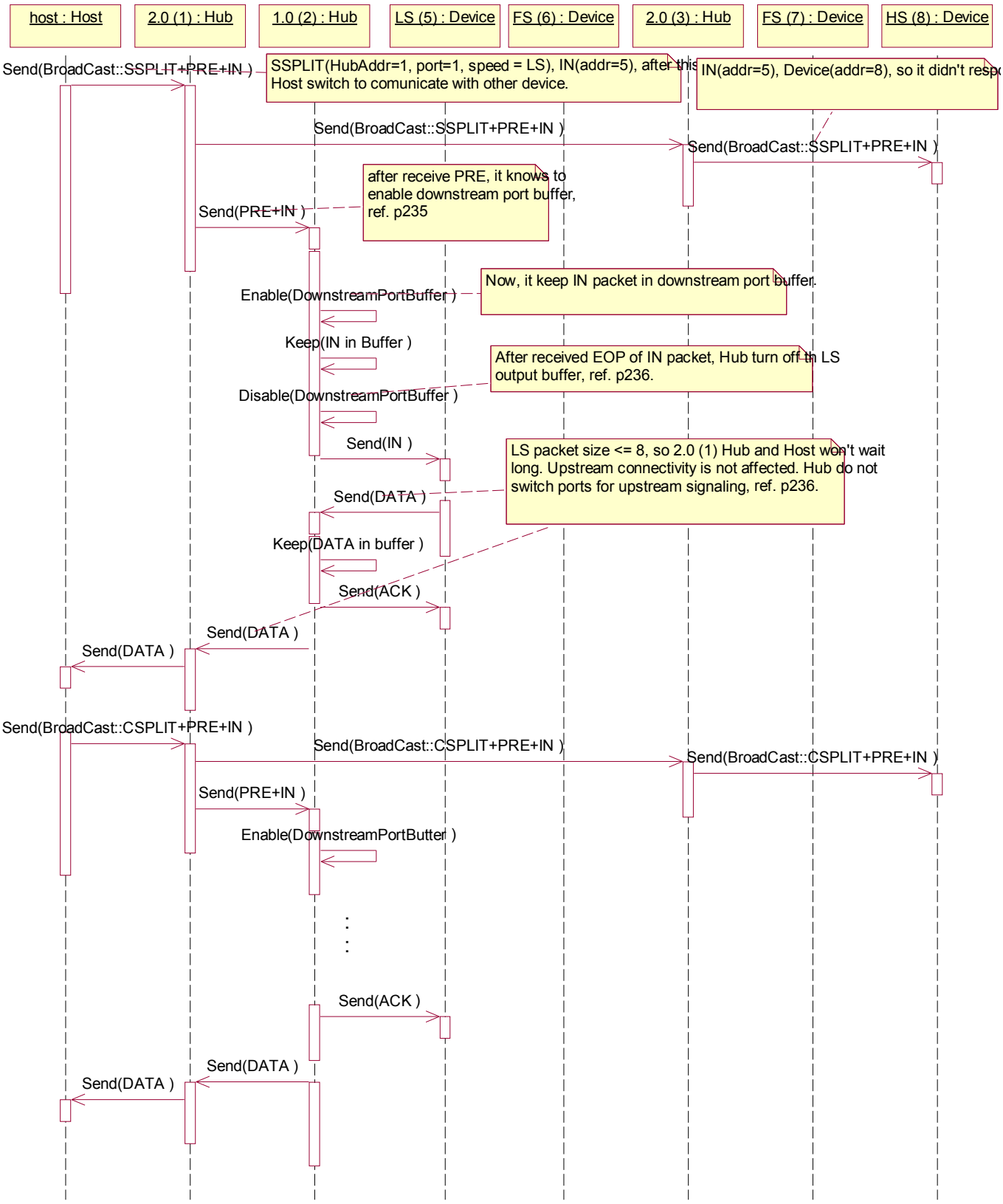
GetConfigureDescriptorFirst8Bytes() : reference p260
GetConfigureDescriptor() = GET_DESCRIPTOR(Descriptor Type = CONFIGURATION) (p251, p253), several times.
appear sequence in contents is (ConfigureDescriptor[0], InterfaceDescriptor[0], EndpointDescriptor[0], EndpointDescriptor[1], ..., InterfaceDescriptor[1], EndpointDescriptor[0], ...,ConfigureDescriptor[1], ...
reference p253

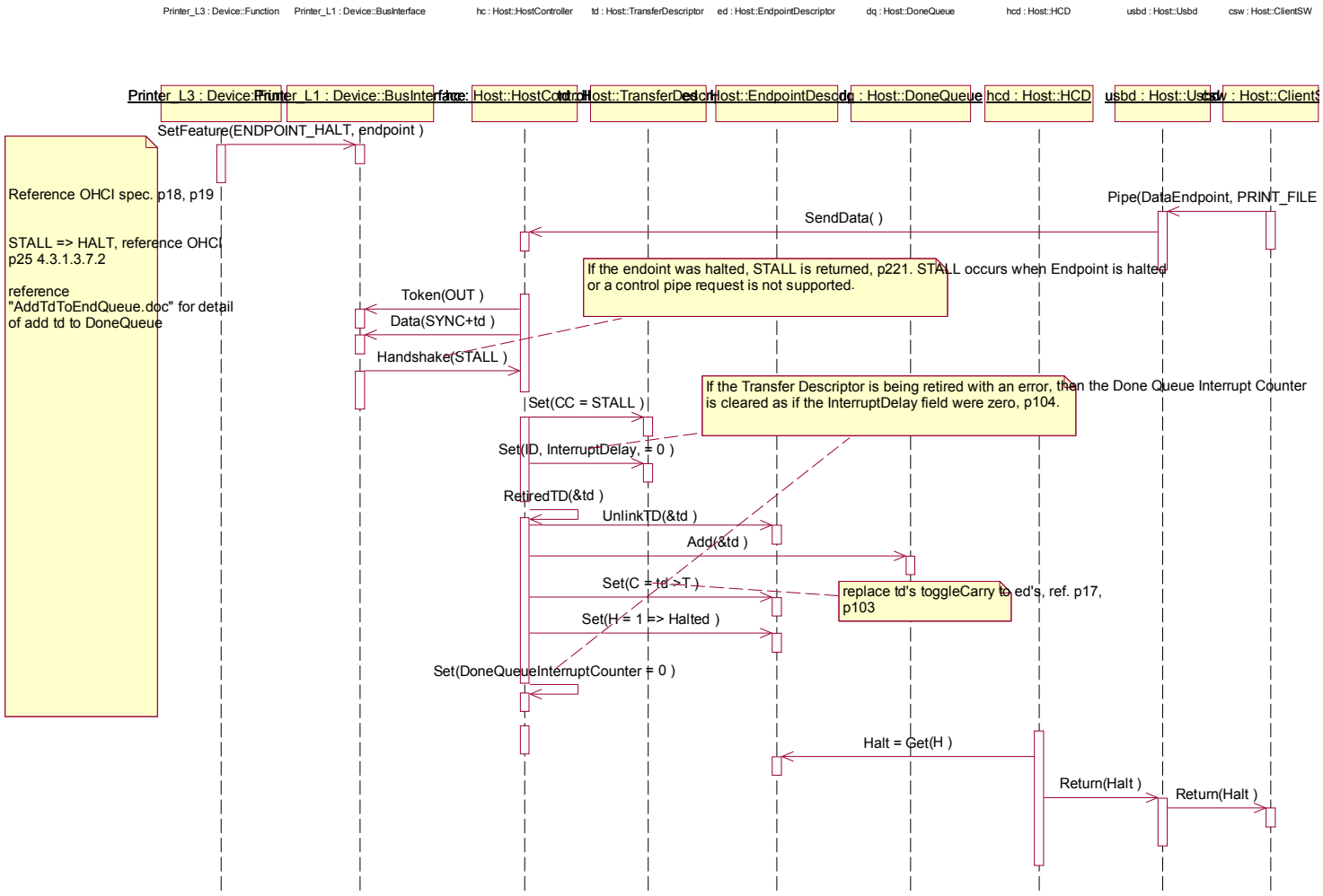
SetConfigure() = SET_CONFIGUREATION (p257)
I think, linux-usb SystemSoftware has assigned dev->config [0].bConfigurationValue as default, so usb driver can active SetConfigure() before use (I think it's just don't work if usb driver didn't do SetConfigure().)

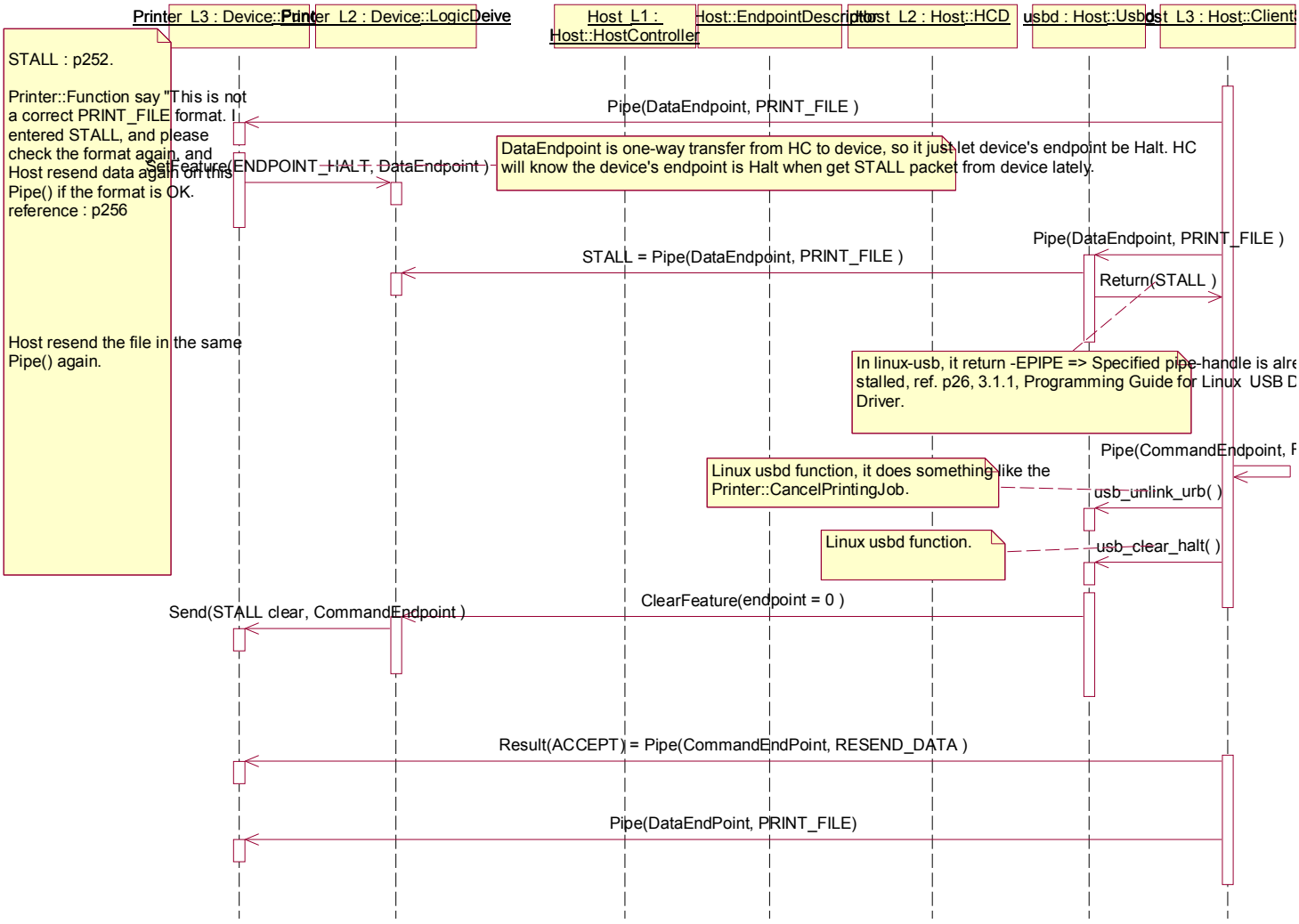


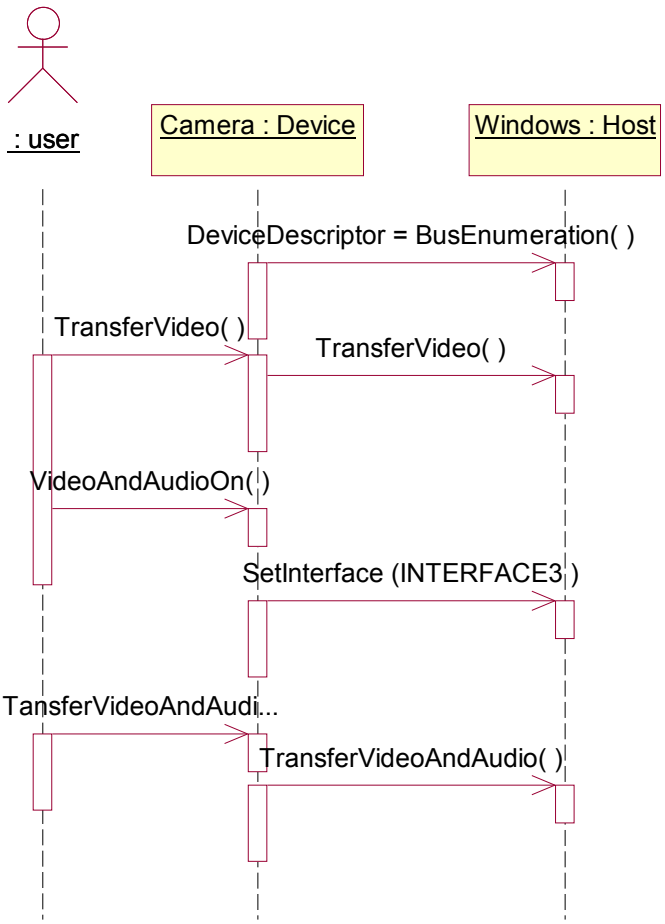

ed : Host::EndpointD...
dq : Host::DoneQueue
hcd : Host::HCD
usbd : Host::Usbd
csw : Host::ClientSW
usb_clear_halt(DataEndpoint)
Set(the data toggle of the DataEndpoint from 0 again)
Release(All TDs In this ED)
Set(C, toggleCarry = 0)
Clear(H, Halted = 0)


user : ...
host : Host
hub : Hub
device : Device
user : user
host : Host
hub : Hub
device : Device
Attach( )
Device, Configure, Interface and Endpoint Descriptor = BusEnumeration( )
Get first byte, bLength, ref. StringDescriptorMessage.doc
Show = Display(UnicodeString=>Manufacture)
UnicodeString.bLength = GetDescriptor(DescriptorIndex = 0, DescriptorType = STRING, wIndex = DeviceDescriptor.iI
UnicodeString = GetDescriptor(DescriptorIndex = 0, DescriptorType = STRING, wIndex = DeviceDescriptor.iManufacture, wI
Show = Display(UnicodeString=>Product)
UnicodeString.bLength = GetDescriptor(DescriptorIndex = 0, DescriptorType = STRING, wIndex = DeviceDescripto
UnicodeString = GetDescriptor(DescriptorIndex = 0, DescriptorType = STRING, wIndex = DeviceDescriptor.iProduct, wLe


host : Host
2.0 (1) : Hub
1.0 (2) : Hub
LS (5) : Device
FS (6) : Device
2.0 (3) : Hub
FS (7) : Device
HS (8) : Device
Send(BroadCast::SSPLIT+PRE+IN)
SSPLIT(HubAddr=1, port=1, speed = LS), IN(addr=5), after this Host switch to comunicate with other device.
IN(addr=5), Device(addr=8), so it didn't resp
Send(BroadCast::SSPLIT+PRE+IN)
Send(BroadCast::SSPLIT+PRE+IN)
after receive PRE, it knows to enable downstream port buffer, ref. p235
Send(PRE+IN)
Enable(DownstreamPortBuffer)
Now, it keep IN packet in downstream port buffer.
Keep(IN in Buffer)
After received EOP of IN packet, Hub turn off th LS output buffer, ref. p236.
Disable(DownstreamPortBuffer)
Send(IN)
LS packet size <= 8, so 2.0 (1) Hub and Host won't wait long. Upstream connectivity is not affected. Hub do not switch ports for upstream signaling, ref. p236.
Send(DATA)
Keep(DATA in buffer)
Send(ACK)
Send(DATA)
Send(DATA)
Send(BroadCast::CSPLIT+PRE+IN)
Send(BroadCast::CSPLIT+PRE+IN)
Send(BroadCast::CSPLIT+PRE+IN)
Send(PRE+IN)
Enable(DownstreamPortButter)
Send(ACK)
Send(DATA)
Send(DATA)


Printer_L3 : Device::Function
Printer_L1 : Device::BusInterface
hc : Host::HostController
td : Host::TransferDescriptor
ed : Host::EndpointDescriptor
dq : Host::DoneQueue
hcd : Host::HCD
usbd : Host::Usbd
csw : Host::ClientSW
SetFeature(ENDPOINT_HALT, endpoint)
Reference OHCI spec. p18, p19
STALL => HALT, reference OHCI p25 4.3.1.3.7.2
reference "AddTdToEndQueue.doc" for detail of add td to DoneQueue
Pipe(DataEndpoint, PRINT_FILE
SendData( )
If the endoint was halted, STALL is returned, p221. STALL occurs when Endpoint is halted or a control pipe request is not supported.
Token(OUT)
Data(SYNC+td)
Handshake(STALL)
If the Transfer Descriptor is being retired with an error, then the Done Queue Interrupt Counter is cleared as if the InterruptDelay field were zero, p104.
Set(CC = STALL)
Set(ID, InterruptDelay, = 0)
RetiredTD(&td)
UnlinkTD(&td)
Add(&td)
Set(C = td->T)
replace td's toggleCarry to ed's, ref. p17, p103
Set(H = 1 => Halted)
Set(DoneQueueInterruptCounter = 0)
Halt = Get(H)
Return(Halt)
Return(Halt)

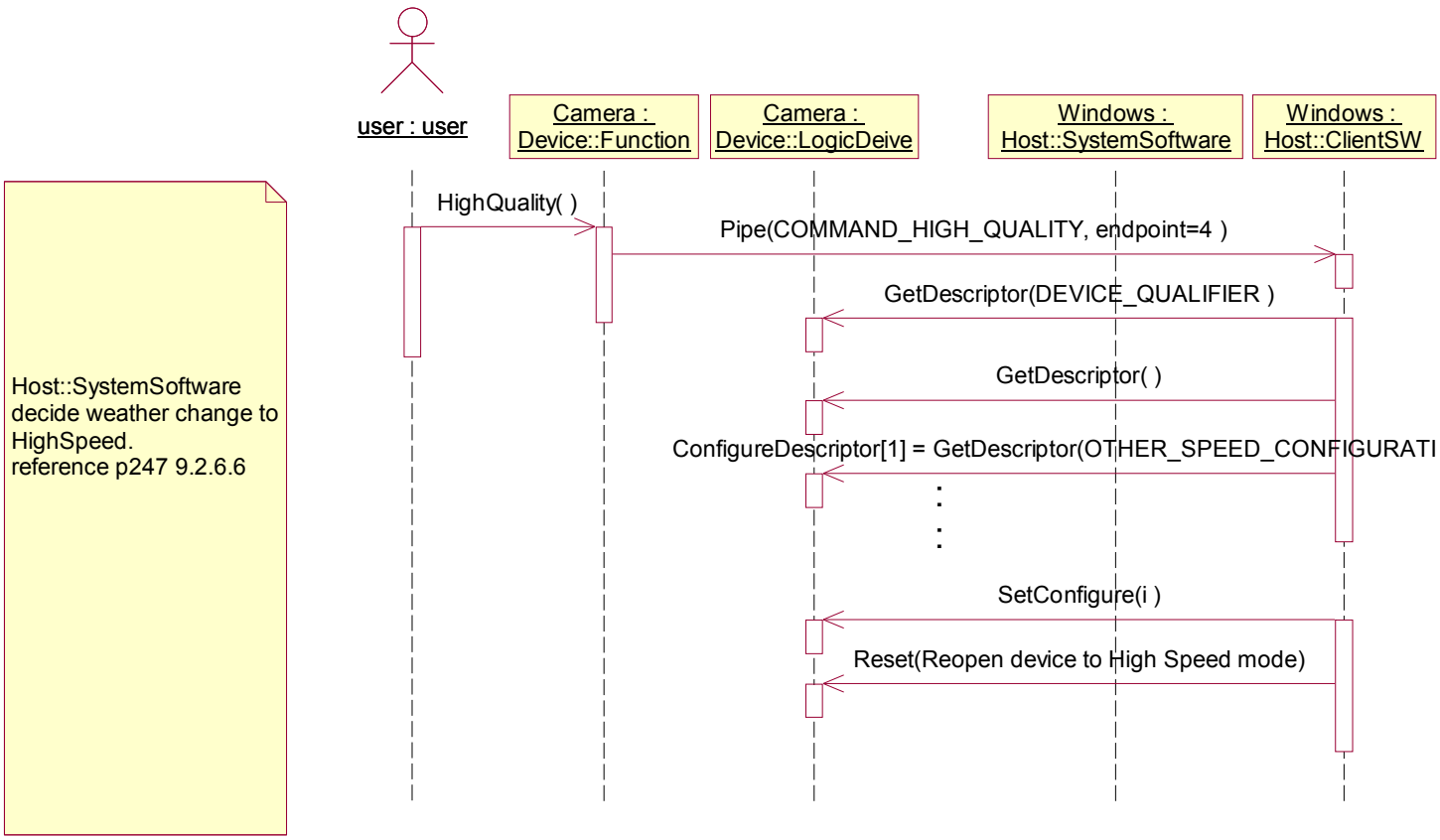

Printer_L3 : Device::Printer
Printer_L2 : Device::LogicDeive
Host_L1 : Host::HostController
Host::EndpointDescriptor
Host_L2 : Host::HCD
usbd : Host::Usbd
Host_L3 : Host::ClientSW
STALL : p252.
Printer::Function say "This is not a correct PRINT_FILE format. I entered STALL, and please check the format again, and Host resend data again on this Pipe() if the format is OK.
reference : p256
Host resend the file in the same Pipe() again.
Pipe(DataEndpoint, PRINT_FILE)
SetFeature(ENDPOINT_HALT, DataEndpoint)
DataEndpoint is one-way transfer from HC to device, so it just let device's endpoint be Halt. HC will know the device's endpoint is Halt when get STALL packet from device lately.
Pipe(DataEndpoint, PRINT_FILE)
STALL = Pipe(DataEndpoint, PRINT_FILE)
Return(STALL)
In linux-usb, it return -EPIPE => Specified pipe-handle is alre
stalled, ref. p26, 3.1.1, Programming Guide for Linux USB D
Driver.
Pipe(CommandEndpoint, F
Linux usbd function, it does something like the Printer::CancelPrintingJob.
usb_unlink_urb( )
Linux usbd function.
usb_clear_halt( )
ClearFeature(endpoint = 0)
Send(STALL clear, CommandEndpoint)
Result(ACCEPT) = Pipe(CommandEndPoint, RESEND_DATA)
Pipe(DataEndPoint, PRINT_FILE)

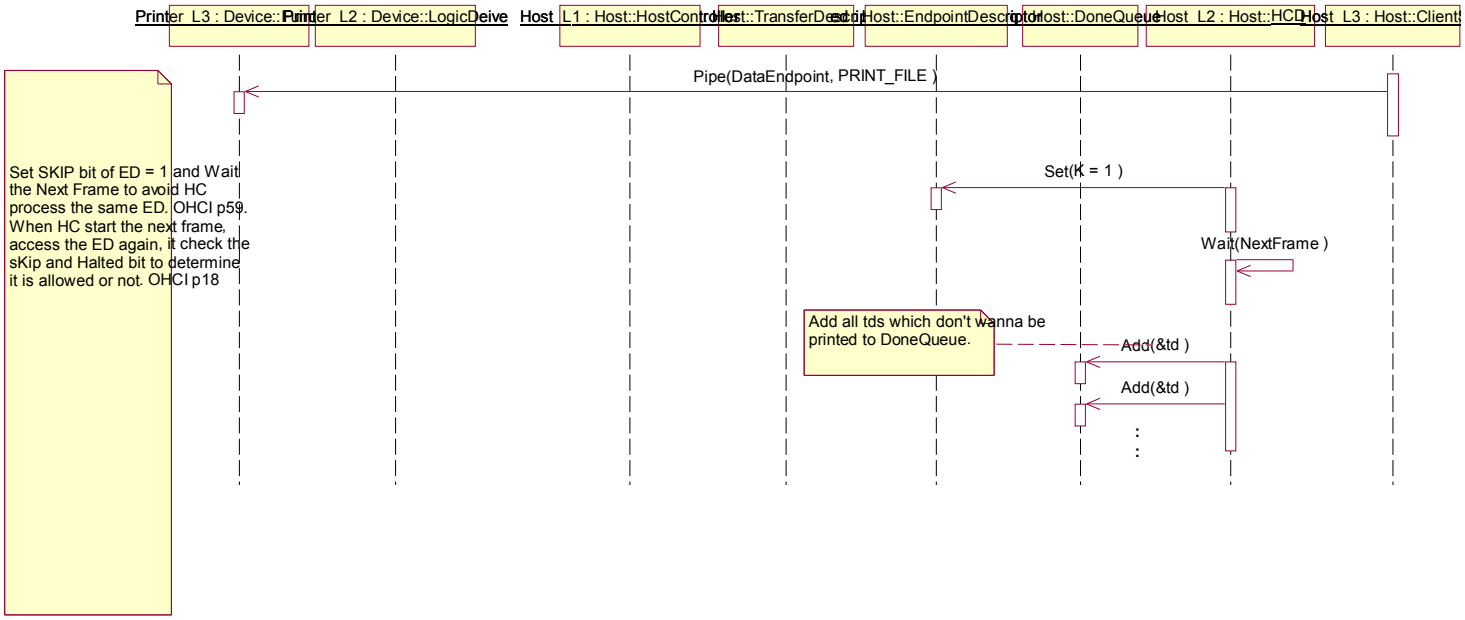


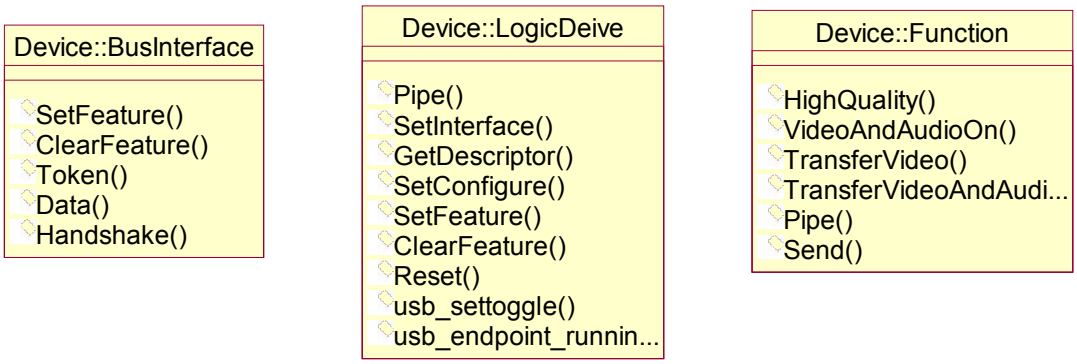
INTERFACE1 : Camera activated, endpoint 1.
INTERFACE2 : Microphone activated, endpoint 2
INTERFACE3 : Camera and microphone activated, endpoint 1 and 2

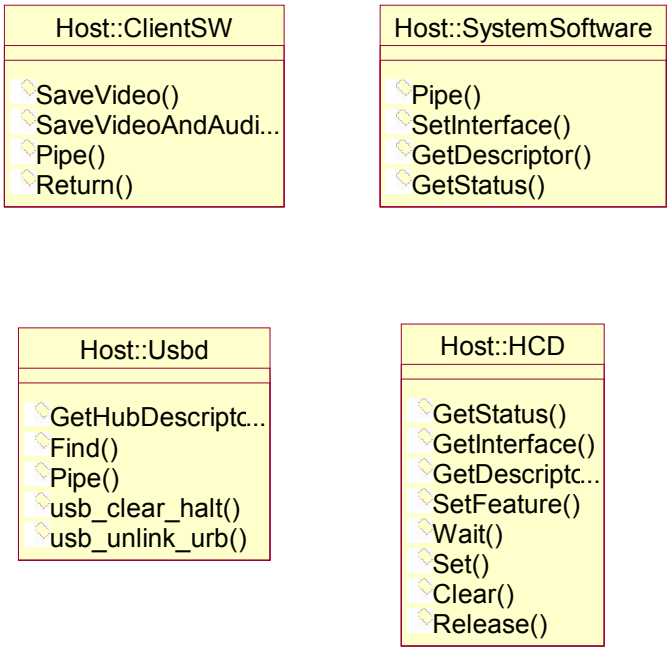
At beginning, TransferImage() using default interface, INTERFACE1. INTERFACE1 has 1 endpoint to pipe Video Data.

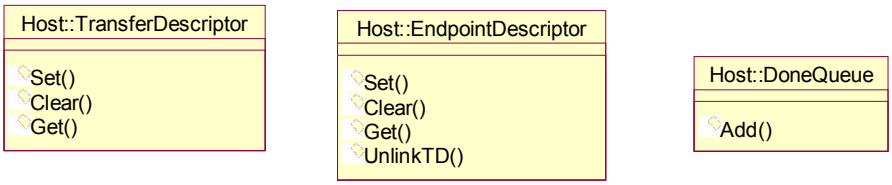
INTERFACE3 has 2 endpoints, one for Video Data, the other for Audio Data.

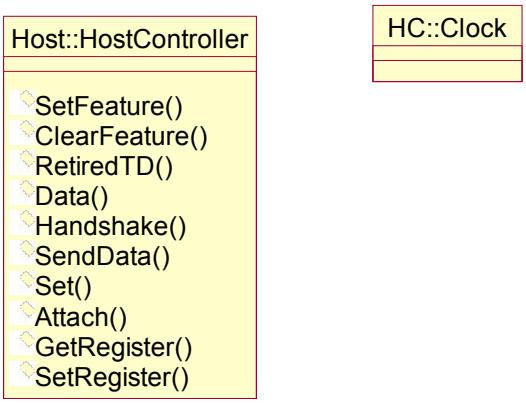

user : user
Camera : Device::Function
Camera : Device::LogicDeive
Windows : Host::SystemSoftware
Windows : Host::ClientSW
HighQuality( )
Pipe(COMMAND_HIGH_QUALITY, endpoint=4)
GetDescriptor(DEVICE_QUALIFIER)
GetDescriptor( )
ConfigureDescriptor[1] = GetDescriptor(OTHER_SPEED_CONFIGURATI
SetConfigure(i)
Reset(Reopen device to High Speed mode)
Host::SystemSoftware
decide weather change to
HighSpeed.
reference p247 9.2.6.6


Printer_L3 : Device::Function
Printer_L2 : Device::LogicDeive
Host_L1 : Host::HostController
td : Host::TransferDescriptor
ed : Host::EndpointDescriptor
dq : Host::DoneQueue
Host_L2 : Host::HCD
Host_L3 : Host::ClientSW
Pipe(DataEndpoint, PRINT_FILE)
Set SKIP bit of ED = 1 and Wait the Next Frame to avoid HC process the same ED. OHCI p59. When HC start the next frame, access the ED again, it check the sKip and Halted bit to determine it is allowed or not. OHCI p18
Set(K = 1)
Wait(NextFrame)
Add all tds which don't wanna be printed to DoneQueue.
Add(&td)
Add(&td)

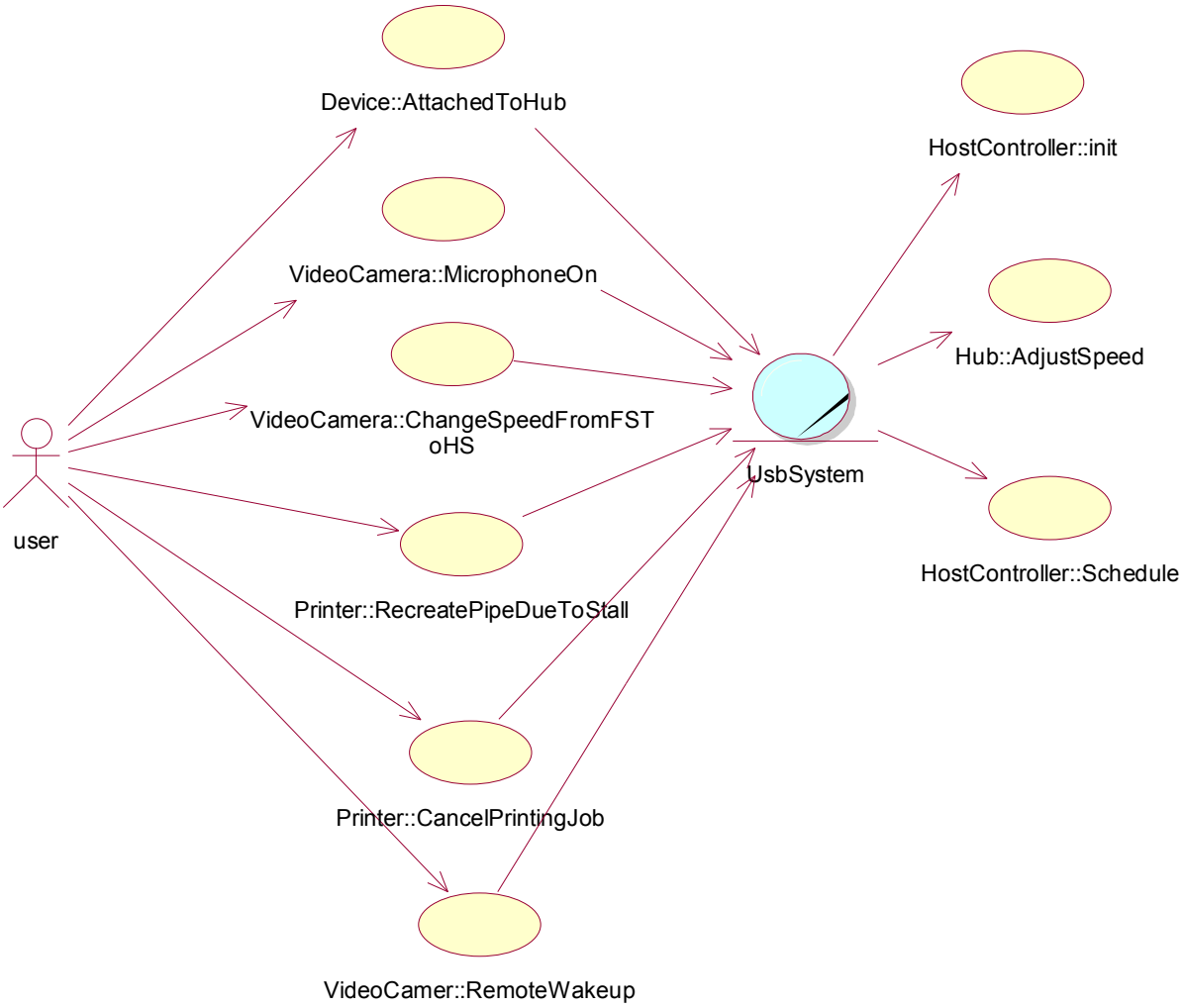

Device::BusInterface
SetFeature()
ClearFeature()
Token()
Data()
Handshake()
Device::LogicDeive
Pipe()
SetInterface()
GetDescriptor()
SetConfigure()
SetFeature()
ClearFeature()
Reset()
usb_settoggle()
usb_endpoint_runnin...
Device::Function
HighQuality()
VideoAndAudioOn()
TransferVideo()
TransferVideoAndAudi...
Pipe()
Send()

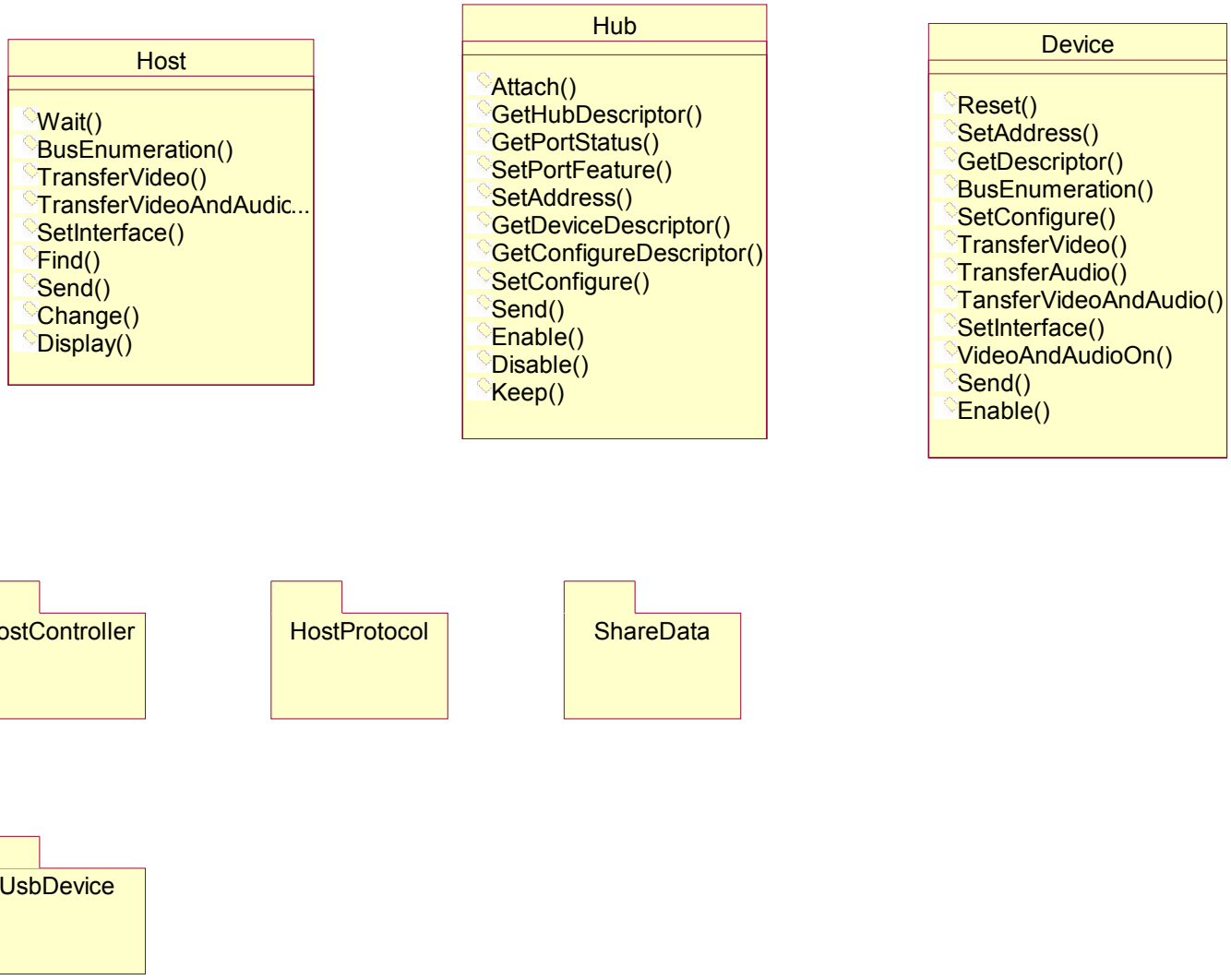

Host::ClientSW
SaveVideo()
SaveVideoAndAudi...
Pipe()
Return()
Host::SystemSoftware
Pipe()
SetInterface()
GetDescriptor()
GetStatus()
Host::Usbd
GetHubDescriptc...
Find()
Pipe()
usb_clear_halt()
usb_unlink_urb()
Host::HCD
GetStatus()
GetInterface()
GetDescriptc...
SetFeature()
Wait()
Set()
Clear()
Release()

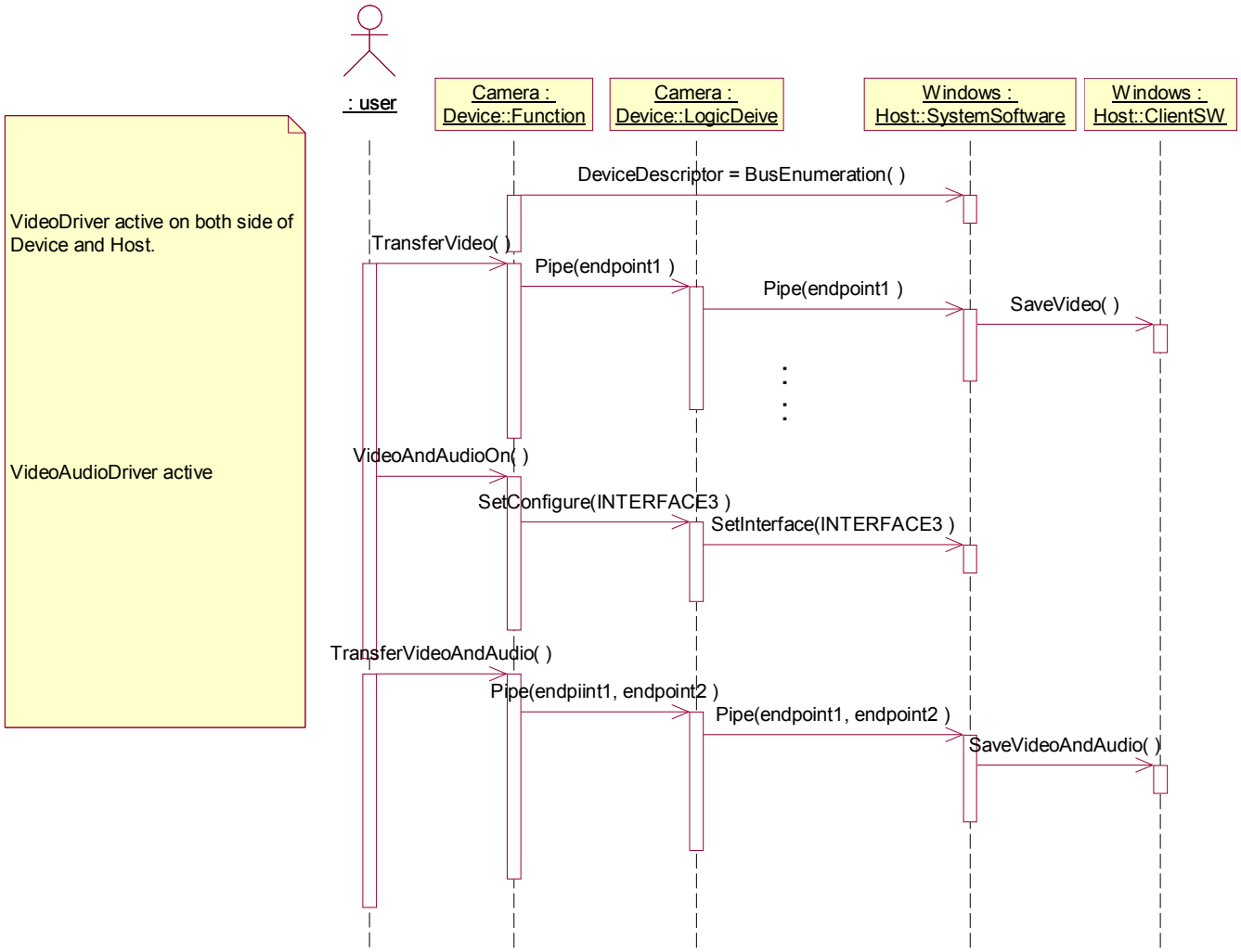

Host::TransferDescriptor
Set()
Clear()
Get()
Host::EndpointDescriptor
Set()
Clear()
Get()
UnlinkTD()
Host::DoneQueue
Add()


Host::HostController
SetFeature()
ClearFeature()
RetiredTD()
Data()
Handshake()
SendData()
Set()
Attach()
GetRegister()
SetRegister()

HC::Clock


Device::AttachedToHub
HostController::init
VideoCamera::MicrophoneOn
Hub::AdjustSpeed
VideoCamera::ChangeSpeedFromFST
oHS
UsbSystem
user
HostController::Schedule
Printer::RecreatePipeDueToStall
Printer::CancelPrintingJob
VideoCamer::RemoteWakeup


Host
Wait()
BusEnumeration()
TransferVideo()
TransferVideoAndAudic...
SetInterface()
Find()
Send()
Change()
Display()
Hub
Attach()
GetHubDescriptor()
GetPortStatus()
SetPortFeature()
SetAddress()
GetDeviceDescriptor()
GetConfigureDescriptor()
SetConfigure()
Send()
Enable()
Disable()
Keep()
Device
Reset()
SetAddress()
GetDescriptor()
BusEnumeration()
SetConfigure()
TransferVideo()
TransferAudio()
TansferVideoAndAudio()
SetInterface()
VideoAndAudioOn()
Send()
Enable()
HostController
HostProtocol
ShareData
UsbDevice


: user
Camera : Device::Function
Camera : Device::LogicDeive
Windows : Host::SystemSoftware
Windows : Host::ClientSW
VideoDriver active on both side of Device and Host.
VideoAudioDriver active
DeviceDescriptor = BusEnumeration( )
TransferVideo( )
Pipe(endpoint1)
Pipe(endpoint1)
SaveVideo( )
VideoAndAudioOn( )
SetConfigure(INTERFACE3)
SetInterface(INTERFACE3)
TransferVideoAndAudio( )
Pipe(endpiint1, endpoint2)
Pipe(endpoint1, endpoint2)
SaveVideoAndAudio( )