**<PIPE>**

* 속도에는 저속, 전속력, 고속이 있다.LOW-SPEED(키보드), FULL-SPEED(오디오), HIGH-SPEED(비디오)
* 여러 대역폭 장치에 적합, 다중연결 가능
* 여러 데이터 및 메시지 스트림 전송 지원(메시지? 데이터?)
* 광범위한 패킷 크기 지원
* macOS 에서는 항상 데이터 캐싱을 사용하기 때문에 제거를 꼭 누르고 뽑아야 한다.
* 완전 양방향 통신을 지원한다. 2.0까지는 반이중통신을 지원하나, => 버퍼가 입출력 따로 있음
* isochronous 등시성 => 전송에 타이밍이 중요
* 예컨대 버퍼에 쌓인걸 위로 올려야한다 =>버퍼자체가 위에서는 파이프라 볼수잇따
* USBD는 특정 파이프를 통해 데이터를 전송하라는 요청으로 구성된 I/O 요청 패킷(IRP) 형식의 데이터 전송 메커니즘을 제공합니다.
* => 프로토콜에서 그러하듯 논리적인 파이프 개념 => 아랫단의 피지컬 연결은 모른다
* => 그렇다면 피지컬 연결과 논리적 연결의 매핑이 필요하겠군
* • 호스트와 USB 장치 간의 USB 표준 제어 흐름 관리
* • 호스트와 USB 장치 간의 데이터 흐름 관리
* USB 시스템 소프트웨어는 여러 요청이 동시에 메시지 파이프로 전송되지 않도록 합니다.
* 끝점 번호가 0인 두 끝점으로 구성된 파이프를 기본 제어 파이프라고 합니다.
* 통신 모드:
* • 스트림: 파이프를 통해 이동하는 데이터에는 USB 정의 구조가 없습니다.
* • 메시지: 파이프를 통해 이동하는 데이터에는 일부 USB 정의 구조가 있습니다.<시그널링 정보같은 개념인듯>
* 또한 파이프에는 다음과 같은 관련 항목이 있습니다.
* • USB 버스 액세스 및 대역폭 사용에 대한 주장.
* • 전송 유형.
* • 방향성 및 최대 데이터 페이로드 크기와 같은 연결된 끝점의 특성. 데이터 페이로드는
* 버스 트랜잭션 내에서 데이터 패킷의 데이터 필드에 전달됩니다
* USB 시스템 소프트웨어는 기본 제어 파이프의 "소유권"을 유지하고 다른 클라이언트 소프트웨어의 파이프 사용을 중재합니다. USB 시스템 소프트웨어는 여러 요청이 동시에 메시지 파이프로 전송되지 않도록 합니다.
* USB는 네 가지 전송 유형을 정의합니다.
* • 제어 전송: 일반적으로 명령/상태 작업에 사용되는 버스트, 비주기적, 호스트 소프트웨어 시작 요청/응답 통신입니다.
* • Isochronous Transfers: 일반적으로 시간 관련 정보에 사용되는 호스트와 장치 간의 주기적이고 지속적인 통신입니다.
* 이 전송 유형은 데이터에 캡슐화된 시간 개념도 보존합니다. 그러나 이것이 그러한 데이터의 전달 요구가 항상 시간 결정적이라는 것을 의미하지는 않습니다.
* <싱크채널같은 느낌>
* • 인터럽트 전송: 저주파, 제한된 대기 시간 통신.
* • 대량 전송: 일반적으로 사용 가능한 모든 대역폭을 사용할 수 있고 대역폭을 사용할 수 있을 때까지 지연될 수 있는 데이터에 사용되는 비주기적인 대규모 패킷 버스트 통신입니다.
* USB는 파이프를 통해 전달하는 데이터의 내용을 해석하지 않습니다. 메시지 파이프에서 USB 정의에 따라 데이터를 구성해야 하지만 데이터의 내용은 USB에서 해석되지 않습니다.
* => 등시성은 화질이 깨져도 일정한 시간으로 전송하나보다
* 실제 구현체는 버퍼이며 클라이언트의 파이프에 매핑된다. 파이프의 속성에 따라 버퍼 도메인의 설정이 달라진다.
* The endpoints with endpoint number zero are always accessible once a device is attached, powered, and has received a bus reset.
* 파이프는 다음을 나타냅니다.메모리 버퍼와 장치의 끝점을 통해 호스트의 소프트웨어 간에 데이터를 이동하는 기능.
* •스트림: 파이프를 통해 이동하는 데이터에는 USB 정의 구조가 없습니다. 단방향 •메시지: 파이프를 통해 이동하는 데이터에는 USB 정의 구조가 있습니다.USB는 파이프를 통해 전달하는 데이터의 내용을 해석하지 않습니다. 양방향
* •USB 버스 액세스 및 대역폭 사용에 대한 주장.•전송 유형입니다.•방향성 및 최대 데이터 페이로드 크기와 같은 연결된 끝점의 특성입니다.
* 소프트웨어 클라이언트는 일반적으로 I/O 요청 패킷(IRP)을 통해 파이프로 데이터 전송을 요청한 다음 다음 중 하나를 수행합니다.완료되면 기다리거나 알림을 받습니다. IRP에 대한 세부 정보는 운영 체제에 정의되어 있습니다. => IRP는 운영체제에 정의되어 있짐나 버퍼에 데이터를 쓰는건 동일. 호스트에서는 관여하지 않는다.
* IRP, 입력에서 완전히 채우지 않는 짧은 패킷(즉, 최대 크기의 데이터 페이로드 미만)IRP 데이터 버퍼는 클라이언트의 기대에 따라 두 가지 의미 중 하나를 가질 수 있습니다.
* •클라이언트는 IRP에서 다양한 크기의 데이터를 기대할 수 있습니다. 이 경우 전송되지 않는 짧은 패킷IRP 데이터 버퍼 채우기는 "데이터 단위 끝"을 나타내는 대역 내 구분 기호로 간단히 사용할 수 있습니다.
* => 데이터의 끝을 내가 감지해야하나? 타이니보고 알아봐야할 듯
* 파이프일 때끝점에 대해 설정되면 대부분의 파이프 전달 특성이 결정되고 고정된 상태로 유지됩니다.
* 클라이언트 소프트웨어의 파이프 번들과 USB SYSTEM SW의 Default Pipe(Endpoint 0)가 이거네, USB Wire의 매핑관계가 잇는가? 있다면 저장해야 하는가?
* => 흠...파이프 번들은 시스템 소프트웨어는 관리하면 안되나?
* => 각각의 버퍼를 논리적 개념의 Pipe같은 도메인으로 인식해서 실제로는 버퍼에 쓰고 읽지만 엔드포인트와 통신하는 느낌을 주자
* 􀁺 USB 드라이버(USBD) : IRP를 디바이스 엔드포인트 Call로 변환
* 그림 5-17을 보면 클라이언트의 드라이버가 버퍼에 데이터를 쓰고 호스트는 전송이 끝나면 인터럽트를 날려준다
* 파이프는 전송할 데이터의 전달 요구 사항에 따라 몇 가지 특성을 갖습니다.이러한 특성의 예는 다음과 같습니다.•데이터를 전송해야 하는 속도•데이터가 일정한 속도로 제공되는지 또는 산발적으로 제공되는지 여부•데이터 전송이 지연될 수 있는 기간•전송 중인 데이터의 손실이 치명적인지 여부USB 장치 끝점은 특정 파이프에 필요한 특성을 설명합니다. 끝점이 설명되어 있습니다.USB 장치의 특성화 정보의 일부로. 자세한 내용은 9장을 참조하십시오 => 세부구현때
* 파이프는 각각 유휴 상태여야 합니다 => 스테이트가 있다.
* 클라이언트는 자신의 데이터를 연속적인 직렬 스트림으로 보고 이를 유사한 방식으로 관리합니다. => 큐방식 피포
* 인터럽트 파이프의 간격. 지정된 요청이 완료되면 클라이언트에 알립니다.
* 클라이언트는 인터페이스의 각 파이프를 다음과 같이 초기화합니다.해당 파이프에 대한 정책을 설정합니다. 여기에는 IRP당 전송할 최대 데이터 양이 포함되며,파이프의 최대 서비스 간격. 서비스 간격은 밀리초로 표시되며 다음을 설명합니다.IRP의 데이터가 등시성 파이프에 대해 전송되는 간격입니다.
* 다음 데이터 요소는 요청에 대한 관련 정보를 정의합니다.•요청과 연결된 파이프의 ID입니다. 이 파이프를 식별하면 정보도 설명됩니다.이 요청에 대한 전송 유형과 같은.•특정 클라이언트에 대한 알림 식별입니다.•전송 또는 수신할 데이터 버퍼의 위치 및 길이입니다.•요청의 완료 상태입니다. 요약 상태와 필요에 따라 거래별 세부 정보상태를 제공해야 합니다.•작업 공간의 위치와 길이. 이것은 구현에 따라 다릅니다
* USBD 파이프는 정확히 다음 상태 중 하나에 있습니다.•활성: 파이프의 정책이 설정되었으며 파이프가 데이터를 전송할 수 있습니다. 클라이언트는 다음과 같이 쿼리할 수 있습니다.특정 파이프에 대해 미결 상태인 IRP가 있는지 여부. 미결제 IRP가 없는 파이프새 IRP를 수락할 수 있는 한 여전히 활성 상태로 간주됩니다.•중지됨: 파이프에 오류가 발생했습니다. 이 상태는 또한 해당장치에서 중지된 끝점입니다.파이프 및 끝점은 장치가 구성되고 파이프 및/또는 끝점이 다음과 같을 때 활성으로 간주됩니다.정지되지 않았습니다. 클라이언트는 다음과 같은 방법으로 파이프 상태를 조작할 수 있습니다.•파이프 중단: 파이프에 대해 예약된 모든 IRP는 즉시 폐기되고 클라이언트에게 반환됩니다.중단되었음을 나타내는 상태로 표시됩니다. 호스트 상태도 반영된 엔드포인트 상태도 아닙니다.파이프의 영향을 받습니다.•파이프 재설정: 파이프의 IRP가 중단됩니다. 호스트 상태가 활성으로 이동됩니다. 반사된 경우USBD 클라이언트에서 명시적으로 명령해야 하는 엔드포인트 상태를 변경해야 합니다.•중단된 파이프 지우기: 파이프의 상태가 Halted 에서 Active 로 지워집니다 .•파이프 중지: 파이프의 상태가 Halted 로 설정됩니

**<제어pipe>**

* 엔드포인트0를 제어 파이프라고 명명하자.
* Stream Pipe는 USB에서 정의 되지 않은 구조의 파이프를 통해서 데이터를 전송하지만, Message의 경우에는 USB에서 정의된 구조의 파이프를 통해서 데이터를 전송하게 된다. Endpoint 0을 포함한 두개의 Endpoint를 기본 제어 Pipe라고 부른다.
* USB시스템 S/W는 기본제어 Pipe의 오너쉽을 가지고 있으며 다른 클라이언트 S/W가 이 Pipe를 사용할 수 있도록 중재한다.
* 만약에 구조라면,
* 컨트롤 파이프, 데이터 파이프 구분 / 단방향 파이프, 양방향 파이프
* => 이럴수있겠고 두개의 인터페이스를 상속해야하고 ISP이다.
* => 또 스트림 파이프는 스트림 Pipe는 Bulk, Isochoronous, interrupt의 전송 형태에 사용된다. 이렇게 전송 형태에 따라 상속할 수 있다. => Interface의 extends
* USB 시스템은 호스트와 디바이스간의 제어 전송을 지원하기 위하여 최대한 노력하여야 한다. Function과 그의 클라이언트 S/W는 제어 전송을 위하여 특별한 버스 접근 주파수 혹은 대역폭을 요청할수 없다. USB 시스템 S/W만이 이들을 제약 할 수 있다.
* USBD는 자신이 소유하고 사용하는 기본 파이프를 클라이언트와 공유하는 것을 지원합니다. 또한장치와 관련된 다른 제어 파이프에 대한 액세스를 제공합니다
* 기본 파이프는 클라이언트가 직접 액세스하지 않지만 일부를 수행하는 데 자주 사용됩니다.명령 메커니즘을 통해 중계되는 클라이언트 요청

**<Negotiator>**

* 정의를 디바이스의 요구사항으로부터 적절한 리소스를 계산하여주는 모듈
* USB 장치는 다음과 같이 가정합니다.더 많은 대역폭이 필요한 경우 더 큰 버퍼를 제공할 수 있습니다. USB 아키텍처의 목표는버퍼링으로 인한 하드웨어 지연이 몇 밀리초 이내로 제한되도록 합니다.
* • Isochronous Data Transfers: Occupy a prenegotiated amount of USB bandwidth with a prenegotiated
* USB 시스템은 호스트와 호스트 간의 제어 전송을 지원하기 위해 "최선의 노력"을 기울일 것입니다.장치. 기능 및 해당 클라이언트 소프트웨어는 특정 버스 액세스 주파수 또는 대역폭을 요청할 수 없습니다.제어 전송. USB 시스템 소프트웨어는 장치가 사용할 수 있는 버스 액세스 및 대역폭을 제한할 수 있습니다.통제 이전에 대한 욕구. 이러한 제한 사항은 섹션 5.5.3 및 섹션 5.5.4에 정의되어 있습니 제어 전송은 메시지 파이프를 통해 양방향 통신 흐름을 지원한다. 제어파이프가 "구성되면" 입출력 엔드포인트를 모두 사용ㅎ나다.
* USB 시스템은 지원에 필요한 리소스를 확인합니다.사용 가능한 경우 필요한 대역폭을 할당합니다.
* 정책을 설정한 결과 USB 시스템이 취하는 조치 중 다음을 결정합니다.클라이언트가 제공한 데이터 버퍼 공간 이상으로 필요한 버퍼 작업 공간의 양입니다. 크기필요한 버퍼의 수는 클라이언트가 선택한 사용량과 전송당 필요에 따라 다릅니다.
* 클래스별 전원 제어 기능

**<UsbClassChecker? 나중에 이름변경>**

* 수용가능한 클래스코드를 감지할 수 있어야 한다. 예를 들어 차량용 허브인데 프린터 연결하면 뭐함?

**<Configurator> => 뭔가 애매한 도메인인데 나중ㅇ ㅔ생각**

* USB 디스크립터, 일시 중단, 재개 및 원격 깨우기와 같은 저전력 기능을 동적으로 변경하여 여러 장치 구성을 지원합니다. 다음 디바이스 클래스가 지원됩니다.
* 여러가지를 조합해서 유니크한 코드를 만들어내고 그거에 대해서 바로 접속 가능할지
* vid pid
* USB 전원소스가 바뀌면 기기를 재열거해야한다.
* • USB 장치의 부착 및 제거 감지
* • 현황 및 활동 통계 수집
* 전력을 절약하기 위해 USB 장치는 지정된 기간 동안 버스 트래픽이 관찰되지 않으면 자동으로 일시 중단 상태로 들어갑니다. 일시 중단되면 USB 장치는 주소 및 구성을 포함한 모든 내부 상태를 유지합니다.
* USB 장치는 기능을 사용하기 전에 구성해야 합니다. 호스트는 USB 장치 구성을 담당합니다. 호스트는 일반적으로 USB 장치의 구성 정보를 요청하여 장치의 기능을 결정합니다.
* 9.1.2 버스 열거 => 세부적으로 나중에 봐야할 듯 => 시퀀스이다
* 뎁스 제한이 있지만 뎁스 제한 내에서는 동등한 논리적인 디바이스로 인식한다.
* 허브에 대해서 아에 무시할까? 처음 등록때 조건만 체크하는거고 허브가 요구하는만큼의 전력을 공급하고 허브에 달린 애들은 그냥 동등한 논리적 디바이스로 인식한다.
* 해제에 대해서도 각각에 대해서 인식해서 할수있나? 허브가 한번에 뽑히면?
* 240페이지(268페이지) 상태다이어그램
* => 뭐,,즉 연결되고 최소전력 들어오고 주소할당받고 설정완료라는거
* => 노운호스트라면 한방에 안되나? 쓰던 주소로
* 9.1.1.4 주소모든 USB 장치는 처음 전원이 공급되거나 장치가 재설정된 후에 기본 주소를 사용합니다. 각USB 장치는 연결 후 또는 재설정 후 호스트에 의해 고유한 주소가 할당됩니다. USB 장치일시 중단되는 동안 할당된 주소를 유지합니다.USB 장치는 장치가 현재 고유하게 할당되었는지 여부에 관계없이 기본 파이프의 요청에 응답합니다.주소 또는 기본 주소를 사용 중입니다
* 9.1.2 참고
* 9.4, 9.6, 9.6.3, 9.6.6은 나중에 세부구조때 활용하자

**<PowerSupplier>**

* 최종 사용자와 분리된 전기 세부 정보(클라이언트가 신경안써도 되게 호스트단에서 끝 vbus랑 통신 네고)
* VBUS는 선이다. 외부 액터로 표현이 가능한가? 아닌거같음
* USB를 통해 전원을 공급하는데 이 경우 버스 전원장치라고 하며, 자체 전원이 있는 장치의 경우 자체 전원 장치라고 한다.
* => 상위 클라이언트가 전력량에 대해서 몰라야된다 => 이것도 제약사항인데 뭔가 구조에 영향을 줄때 하라는데 역할을 호스트가 하게되니 영향을 주는거겠지?
* Electrical details isolated from end user (e.g., bus terminations)
* USB의 Power Delivery가 버전업 되면서 포트 하나로 고용량의 전력을 제공할 수 있게 되었다. 특히 USB-Type-C는 무려 100W까지 지원하므로 이젠 굳이 파워 케이블을 따로 사용해야할 이유가 없을 지경에까지 이르렀다. 이거는 버전에 따라…
* USB 파워 딜리버리 기술은 기본적으로 충전기와 충전대상 기기가 서로 데이터를 주고받으며 전압과 전류를 조절하는 게 기본이 된다.
* 암단자에 회로가 달린 형식이 되어 중간에 USB-Type-C to C 케이블을 꽂아 사용할 수 있는 형태도 있다. 회로에 전압을 선택할 수 있는 조절장치가 있으며, 사용자는 적당히 마감을 하여 전압을 선택하고 적당한 DC출력 케이블을 납땜해 자유롭게 출력을 얻는 방식으로 사용할 수 있다. => 설명에 포함
* 디바이스의 필요한 전류량을 얻어와야하고 파워로부터 달라고 요청을 해야 한다.
* enumeration 과정 중 기기측의 컴피규레이션 디스크립터의 bMaxPower에 기록된 전류량을 확인한다.
* => 하지만 실제로는 걍 100mA 혹은 500mA를 할당하는 경우가 많다.
* • 제한된 양의 전원 제공을 포함하여 호스트 컨트롤러와 USB 장치 간의 전기 인터페이스 제어
* 전력제어를 담당하는 논리적인 개념의 도메인이다. 하드웨어에서 자체 전원이 있는 것을 가정하여 능동적으로 가능할것으로 보인다.
* => 전원을 끄거나 서스펜드로 만드는일도 유에스비 드라이버가 해야함 => 깨우는것도(원격웨이크업 Remote Wakeup)
* 이 도메인은 각 디바이스의 전력에 대한 정책을 결정하는 도메인으로 바운더리가 아니라 컨트롤러이다.

**<OperationSystem>**

* 자가 식별 주변 장치, 드라이버 및 구성에 대한 기능 자동 매핑
* 플러그앤플레이 아키텍처와 일치
* 기존 운영체제 인터페이스 활용

**<HostController>**

* 오류 복구
* 시스템에서 여러 호스트 컨트롤러를 지원하도록 아키텍쳐 업그레이드 가능
* 특정 HCDI는 다양한 호스트 컨트롤러 구현을 지원하는 각 운영 체제에 의해 정의됩니다.
* Controller Driver Interface (HCDI)
* This interface is never available directly to clients
* 그림보니 하드웨어 버스랑 ipc로 통신한다 이런게 있네(세로 화살표) 통신을 감춰야지
* 프로토콜에는 각 패킷에 CRC가 포함된다 => 근데 이건 파이 레벨일듯. 허브단계에서의 오류 검증은? => 퍼포몬스랑 상충될텐데? AD
* 전제조건으로 파이단에서 신뢰할수있는 데이터가 온다고 가정(CRC등)
* 등시성 전송때는 손실이 있을 수 있으나 그외의 경우에는 데이터 무결성이 전제되는데 이미 파이단에서 된다고 가정하자
* 10.4 호스트 컨트롤러 드라이버호스트 컨트롤러 드라이버(HCD)는 호스트 컨트롤러 하드웨어와 호스트 컨트롤러의USB를 통한 데이터 전송 보기. HCDI는 다음 요구 사항을 충족합니다.•호스트 컨트롤러 하드웨어의 추상화를 제공합니다.•USB 상호 연결을 통한 호스트 컨트롤러의 데이터 전송에 대한 추상화를 제공합니다.•지원하기 위해 호스트 컨트롤러 리소스의 할당(및 할당 해제)을 위한 추상화를 제공합니다.USB 장치에 대한 보장된 서비스

**<Device(Entity)>**

* 루트 허브로부터 계층구조가 가능하지만 뎁스는 7까지만 가능하다.
* => 연결시 서브 연결도 받아들이는데 뎁스랑 개수를 입력받고 127개를 안넘는지 뎁스가 안큰지 확인해야 한다.
* 저장이 가능한 하드웨어가 있다면 노운 장치 저장해서 컨피그 빨리 안되나
* 허브는 디바이스를 연결해서 시스템에 포함시킬 수 있는 기능. 뭔가 엔티티가 있긴 해야하네
* 장치의 버전에 따라 어떻게 처리할 것인가? 어댑터 제공
* 직렬 포트이기 때문에, 이론적으로 한 PC에서 127개까지 인식할 수가 있다고 한다.
* 원리는 USB 기기의 펌웨어를 조작하여 본래의 기능과는 다른 장치(예를 들어 키보드나 마우스 등의 입력장치)로 일시적으로 인식시켜서 사용자가 의도하지 않은 행동을 하도록 신호를 보내는 것.[15] 이게 가능한 이유는 USB 단자의 작동 방식상 호스트는 연결된 장치가 깔라고 시키는 드라이버를 일단 깔아서 실행시켜보게 되어있다.
* USB 대용량 저장소 장치 유형은 연결되는 장치에서 사용되는 파일 시스템을 특정하고 있지는 않고, 대신 하드 디스크의 낮은 수준의 인터페이스처럼 데이터를 섹터 단위로 읽고 쓰는 수준의 단순한 인터페이스만을 규정한다. 따라서 운영 체제는 USB 드라이브를 하드 드라이브처럼 다룰 수 있고 그들이 운영 체제에서 제공하는 파일 시스템으로 포맷하는 것도 가능한다.
* USB는 장치의 기능을 식별하는 데 사용되는 클래스 코드 정보를 정의합니다. 그리고 명목상 그 기능을 기반으로 하는 장치 드라이버를 로드합니다.(장치로부터)
* => 스테이트 개념도 있다.
* => ConnectedDeviceEntity라는 개념이 있어서 거기에 업데이트
* 허브가 제거되면 연결된 모든 디바이스가 사라진다.
* => 트리구조는 있어야 자식을 싹 제거가능하겠네 맞나?
* 각 논리 장치에는 장치 연결 시 시스템에서 할당한 고유한 주소가 있습니다.
* 허브는 호스트 컨트롤러에서 허브디바이스로 인식된다. USB에서는 각 디바이스를 식별하기 위해 어드레스를 할당하지만 그 어드레스는 7비트를 사용하므로 USB 시스템은 127개의 디바이스를 사용할 수 있다.(허브까지 포함)
* USB 논리적디바이스(Endpoint), (내가 쓴거)
* LogicalUsbDevice는 스펙상 Endpoint에 해당하는 논리적인 도메인이다. 스펙의 용어를 그대로 도메인에 사용하는 경우 스펙에 매우 익숙한 개발자에게는 이해가 쉬울 수 있으나 직관적인 구조의 이해를 이해 명명하였다. 즉, USB Host Controller, USB System SW, Client SW라는 3개의 레이어 중 USB System SW 내에서만 적용되는 논리적인 개념이다.
* => 아닌데 스펙에도 논리적 디바이스라고 말하는거 같은데 확인
* => USB 논리적 디바이스는 USB 시스템에서 하나의 Endpoint로 표현되며, Endpoint는 한 개의 적용할 인터페이스에 그룹화 되어 Endpoint 집합으로 주어진다.
* USB의 논리적 디바이스는 USB시스템에게는 Endpoint들의 집합으로 보인다.
* 모든 USB 장치는 Suspended 상태를 지원해야 합니다

**<EndPoint>**

* 뭔가 개념이 디바이스하고 1:1매핑은 아닌거 같다. 고민
* 엔드포인트는 통신 흐름의 종점인 USB 장치의 고유하게 식별 가능한 부분입니다
* 끝점은 호스트와 장치 간의 통신 흐름의 종단인 USB 장치의 고유하게 식별 가능한 부분입니다. 각 USB 논리 장치는 독립적인 끝점 모음으로 구성됩니다.
* 끝점은 다음과 같이 자신을 설명합니다.
* • 버스 액세스 빈도/대기 시간 요구 사항
* • 대역폭 요구 사항
* • 엔드포인트 번호
* • 오류 처리 동작 요구 사항
* • 엔드포인트가 보내거나 받을 수 있는 최대 패킷 크기
* • 엔드포인트의 전송 유형(자세한 내용은 섹션 5.4 참조)
* • 엔드포인트와 호스트 간에 데이터가 전송되는 방향
* 끝점 번호가 0인 끝점 이외의 끝점은 구성되기 전에 알 수 없는 상태에 있으며 구성되기 전에 호스트에서 액세스할 수 없습니다.
* 인터페이스는 장치의 단일 기능 또는 기능을 호스트에 제공하는 관련 엔드포인트 세트입니다. => 펑션을 디바이스가 가지고 있고 펑션은 몇 개의 엔드포인트로 구성될수 있나보다
* 장치의 각 끝점에는 끝점이라고 하는 고유한 장치 결정 식별자가 디자인 타임에 제공됩니다.숫자. 각 끝점에는 장치가 결정한 데이터 흐름 방향이 있습니다
* 장치의 조합주소, 끝점 번호 및 방향을 통해 각 끝점을 고유하게 참조할 수 있습니다
* 끝점은 다음과 같이 자신을 설명합니다.•버스 액세스 빈도/대기 시간 요구 사항•대역폭 요구 사항•엔드포인트 번호•오류 처리 동작 요구 사항•엔드포인트가 보내거나 받을 수 있는 최대 패킷 크기•엔드포인트의 전송 유형(자세한 내용은 섹션 5.4 참조)•엔드포인트와 호스트 간에 데이터가 전송되는 방향
* 하나의 Endpoint는 통신의 흐름의 종단이며 각 고유의 ID를 갖고 USB 디비이스 내에 존재하게 된다. 각 USB 논리적 디바이스는 Endpoint의 집합체이다. 각 USB 논리적 디바이스는 시스템에 연결될 때 시스템에 의하여 독립적인 주소를 할당받는다. 그리고 각 Endpoint는 설정된 시간 내에 번호를 할당받게 된다. Endpoint는 데이터 전송의 방향의 정보 (input device – host, output host – device)를 갖는다. 그리고 디바이스의 주소, 번호, 방향의 구성체라 할 수 있다. 또한 전송의 형태를 정의한다.
* 3-4 Non-Endpoint 0
* Function들은 구현을 위해서 필요한 부가적인 Endpoint들을 가질 수 있다. 낮은 속도(Low Speed)의 Function들은 2개의 선택적인 Endpoint를 가질 수 있고, 풀 속도(Full Speed)의 Function들은 프로토콜에 따라 입력 Endpoint들과 출력 Endpoint들에 대해서 각각 최대 15개의 Endpoint를 가질 수 있다.
* 대량전송의 경우
* 엔드포인트가 다음 중 하나를 수행하면 대량 전송이 완료됩니다.•예상되는 데이터의 양을 정확히 전송했습니다.•페이로드 크기가 wMaxPacketSize 보다 작은 패킷을 전송하거나 길이가 0인 패킷을 전송합니다

**<UsbClass> => 하위 개념 도메인**

**<LEDControler>**

* 데이터 전송속도에 따라 외부 LED의 점멸 속도가 다르게 보이게 하고 싶다
* 데이터 전송없이 충전중일때는 다른색으로 외부 LED가 점멸하면 좋겠다.

**<Client>**

* 도메인이라 할 수 있을지 모르겠네
* USB 장치에 대한 클라이언트용 인터페이스(USBDI)
* 어플리케이션(클라이언트)에서 쉽게 개발하기위해 기능을 많이 가지는것도 좋다.
* (어플리케이션단에서도 파이프가 있는데 제공해준다던지)
* 어플리케이션에서 여러 usb를 사용한다던가 그런 시나리오도 있을 수 잇ㅇ므
* => 매핑정보를 저장해야하나?
* 장치에서 USBD 클라이언트의 데이터 전송에 사용할 인터페이스 모음을 반환합니다. 각인터페이스는 단일 클라이언트가 소유한 파이프 모음입니다.
* 설명자 가져오기USBDI는 다음과 같이 표준 장치, 구성 및 문자열 설명자를 검색하는 메커니즘을 제공해야 합니다.클래스 또는 공급업체별 설명자도 포함됩니다. •모든 대체 정보를 포함하여 장치에 저장된 모든 구성 설명자 정보모든 인터페이스에 대한 설정•인터페이스에 대한 대체 설정이 활성화되어 있는 표시기•인터페이스에 대한 활성 대체 설정의 끝점에 대한 파이프 핸들•인터페이스에 대한 활성 대체 설정의 엔드포인트에 대한 실제 wMaxPacketSize 값또한 지정된 파이프에 대해 USBDI는 다음을 나타내는 wMaxPacketSize 를 반환하는 기능을 제공해야 합니다.현재 파이프에서 사용 중입니다