

# *L'USB ET SA NORME*

**Ce document a pour but d'aider les personnes qui souhaitent se familiariser avec le bus USB. Ce document, très sommaire en fait, vu la complexité de cette norme, n'a pas du tout la prétention d'être complet mais reprend mes recherches et conclusions sur ce bus USB. Il donne un aperçu des éléments les plus importants à tenir compte lors d'une première étude.**

**Ce document en est à sa première version au 21 septembre 2002**

**Une étude plus complète est sur <http://u.s.b.free.fr> et sera mise à jour régulièrement ainsi que le téléchargement de divers logiciels permettant le développement à base du bus USB.**



**Table des matières**

<b>La norme USB .....</b>	<b>5</b>
I. Généralité sur l'USB.....	5
1. Origine de l'USB .....	5
2. Naissance de L'USB .....	5
3. Avantages de l'USB.....	6
4. Facilité d'utilisation .....	6
II. Vitesse de transfert de l'USB.....	7
1. Définitions des différentes vitesses.....	7
a. Débits théoriques.....	7
b. Débits réels.....	8
2. Raison des trois vitesses de l'USB .....	9
3. Domaine d'utilisation des différentes vitesses .....	9
4. Les débits de l'USB .....	9
III. Le câble USB .....	9
1. Définition du câble USB .....	9
2. Composition du câble USB .....	10
IV. La Norme USB .....	11
1. Introduction.....	11
2. Explication des différents chapitres de la norme USB .....	11
V. Operating System .....	13
1. L'USB et Windows.....	13
2. Compatibilité des OS avec l'USB. ....	13
3. Autre solution si le PC est trop ancien.....	13
4. Caractéristique d'un OS gérant l'USB.....	14
5. Bugs de l'USB .....	14
6. Compatibilité USB 1.1 - USB 2.0.....	14
7. USB et IEEE-1394.....	15
VI. Le Forum USB.....	15
1. Introduction du forum .....	15
2. Fonctionnement du forum.....	15
VII. Le Bus USB .....	16
1. Principe du bus USB.....	16
2. Topologie du Bus USB .....	16
3. Protocole USB .....	17
4. Type de paquet USB .....	18
a. Les paquets jetons.....	19
b. Les paquets Data .....	20
c. Les paquets « Handshake ».....	20
d. Les paquets SOF.....	21
e. Explication des différents champs des paquets d'une trame USB.....	22
VIII. Notions importantes pour commencer avec l'USB .....	24
1. Le branchement à chaud ou le «Hot Plug in» .....	24
2. Choix « Low » ou « Full » USB.....	24
3. Délai de propagation .....	25
4. Temps de connexion et de déconnexion .....	26
a. Introduction.....	26
b. Principe de fonctionnement de la connexion - déconnexion .....	26
5. Codage NRZI .....	27

6.	Consommation .....	28
7.	Alimentation USB .....	28
a.	Alimentation de périphérique USB .....	28
b.	Avantage de l'alimentation USB .....	28
c.	Différents types d'alimentation du BUS USB.....	29
8.	Les Hubs .....	29
a.	Définition d'un Hub .....	29
b.	Hub généralités .....	29
c.	Différents types d'alimentation du Hub USB.....	30
9.	Courant de veille .....	30
10.	Différents types de transfert.....	31
a.	Transfert en mode Contrôle.....	31
b.	Transfert en mode Interrupt.....	31
c.	Transfert en mode Isochrone.....	31
d.	Transfert en mode Bulk .....	31
11.	L'énumération .....	32
a.	Définition de l'énumération.....	32
b.	Principe de fonctionnement de l'énumération .....	32
12.	Les descripteurs.....	33
a.	Définition d'un descripteur .....	33
b.	Rôle des descripteurs .....	33
c.	Identification des différents descripteurs .....	34
d.	Différentes catégories des descripteurs .....	35
e.	Logiciels de Point INF .....	40
IX.	PID / VID .....	40
1.	Introduction au Product ID et au Vendor ID.....	40
2.	Normalisation des PID/VID.....	41
X.	Les fichiers d'extension INF .....	41
1.	Définition .....	41
2.	Fonctionnement des points INF.....	41
3.	Règles à respecter pour concevoir un fichier INF .....	41
4.	Applications des fichiers INF.....	42
5.	Rôle du fichier INF .....	43
6.	Création de fichier INF .....	44
XI.	Les drivers .SYS.....	44
1.	Définitions d'un drivers .SYS .....	44
2.	Création d'un driver avec le DDK .....	45
XII.	Arborescence des périphériques USB .....	45
XIII.	Kit de développement de driver .....	46
1.	Enumération des différents types existant.....	46
2.	Aperçu des assistants de création de drivers.....	47
XIV.	Quelques conseils pour installer un périphérique USB.....	49
1.	Installation d'un nouveau périphérique .....	49
2.	Vérifier qu'un périphérique est bien installé .....	49

### Table des figures

<b>Figure 1</b> : Composition d'un câble USB.....	10
<b>Figure 2</b> : Brochage des connecteurs USB de type A et B .....	10
<b>Figure 3</b> : Aspect des connecteurs USB de type A et B.....	11
<b>Figure 4</b> : Topologie du bus USB .....	16
<b>Figure 5</b> : Exemple de branchement respectant la topologie du Bus USB .....	17
<b>Figure 6</b> : Protocole USB .....	18
<b>Figure 7</b> : Transaction USB.....	19
<b>Figure 8</b> : Transaction USB.....	19
<b>Figure 9</b> : Structure des paquets USB .....	19
<b>Figure 10</b> : Format du paquet Token.....	20
<b>Figure 11</b> : Format du paquet Data.....	20
<b>Figure 12</b> : Format du paquet Handshake.....	21
<b>Figure 13</b> : Format du paquet SOF .....	21
<b>Figure 14</b> : Format réel d'un PID.....	22
<b>Figure 15</b> : Choix Low ou Full USB – Résistance de Pull Up.....	25
<b>Figure 16</b> : Temps de propagation d'une trame USB.....	25
<b>Figure 17</b> : Courbe de déconnexion des composants Low & Full USB .....	26
<b>Figure 18</b> : Courbe de connexion des composants Low USB .....	27
<b>Figure 19</b> : Courbe de connexion des composants Full USB .....	27
<b>Figure 20</b> : Principe du Codage NRZI .....	27
<b>Figure 21</b> : Alimentation des différents HUBs .....	30
<b>Figure 22</b> : Diagramme hiérarchique des descripteurs .....	34
<b>Figure 23</b> : USB Descriptor Generator .....	40
<b>Figure 24</b> : L'arborescence USB avec l'USBView .....	46
<b>Figure 25</b> : Windriver - création du point INF.....	47
<b>Figure 26</b> : Windriver - Communication avec les pipes.....	47
<b>Figure 27</b> : WinRT avec l'arborescence USB .....	48

### Table des tableaux

<b>Tableau 1</b> : Désignation des PIDs.....	22
<b>Tableau 2</b> : Récapitulatifs des différents types de transfert .....	32
<b>Tableau 3</b> : Identification des descripteurs.....	34
<b>Tableau 4</b> : Tableau récapitulatif des différents champs des Device Descriptor.....	36
<b>Tableau 5</b> : Tableau récapitulatif des différents champs des Configuration Descriptors .....	37
<b>Tableau 6</b> : Tableau récapitulatif des différents champs des Interfaces Descriptor .....	38
<b>Tableau 7</b> : Tableau récapitulatif des différents champs des Endpoint Descriptor .....	39
<b>Tableau 8</b> : Tableau récapitulatif des différents champs des HID Descriptors .....	39
<b>Tableau 9</b> : Champs des fichiers INF.....	43

### Table des annexes

<b>Annexe 1</b> : Bibliographie.....	50
<b>Annexe 2</b> : Fichier INF type .....	51

# ***La norme USB***

## **I. Généralité sur l'USB**

### **1. Origine de l'USB**

Tout au début, avant la standardisation il y avait des problèmes avec la compatibilité de l'USB, car chaque fabricant de carte mère inventait son propre protocole, il n'y avait pas encore de norme. C'est pourquoi on était très septique sur ce nouveau protocole à ses débuts. La dénomination USB qui est « Universal Serial Bus » a dérivée vers le nom « Useless Serial Bus ».

Le bus USB est donc réellement né de l'alliance en 1994 de sept partenaires industriels (Compaq, DEC, IBM, Intel, Microsoft, NEC et Northern Telecom). C'est eux qui ont commencé à créer la norme USB. Le bus USB a été conçu à l'origine pour faciliter les transferts de données en particulier définir une connectique « universelle » et « Plug & Play », utilisable aussi bien pour une souris que pour un modem ou un moniteur. Conçu également pour répondre au besoin d'intégration entre le monde du PC et celui du téléphone et enfin pour répondre au besoin d'extensions multiples en dehors du PC, en effet on peut brancher jusqu'à 127 périphériques en théorie... cela change de la liaison série habituelle.

### **2. Naissance de L'USB**

La spécification ou norme USB version 1.0 opérationnelle est sortie, après plusieurs années de développement, en janvier 1996, les premiers produits ont vu le jour en fin 1997. A cette époque, le manque de composants USB a retardé la mise à jour de la norme. En septembre 1998 paraît la dernière version officielle (version 1.1), c'est celle que l'on utilise actuellement. Cette spécification définit 2 vitesses de fonctionnement : 1,5 Mbps (Low Speed) et 12 Mbps (Full Speed).

Puis en avril 2000 est sortie la spécification 2.0. Cette nouvelle norme supporte toutes les caractéristiques de l'USB 1.1 : 1,5 Mbps (Low Speed), 12 Mbps (Full Speed), et rajoute une vitesse de 480 Mbps (High Speed) et optimise l'utilisation de la bande passante.

Par la suite, nous verrons plus en détails quelles sont les différences entre ces versions.

La version 1.0, la toute première donc, n'est plus utilisée et n'a jamais été très utilisée, c'était en quelque sorte une version de test. L'USB 1.1 corrigeait les erreurs de la version antérieure et ajoutait un nouveau type de transfert (Interrupt OUT). La version, complète, « officielle », est donc la version 1.1.

Le fonctionnement en mode High Speed possède des caractéristiques très différentes des modes Low Speed et Full Speed. Il sera présenté ultérieurement d'autant qu'il ne s'applique actuellement qu'aux applications de type vidéo, télécom et transmission haute vitesse pour lesquelles les drivers commencent à voir le jour actuellement.

### 3. Avantages de l'USB

Les avantages de l'usb sont nombreux : faible coût de l'interface, alimentation possible des dispositifs via le câble, indépendance vis à vis des machines hôtes, Hot Plug & Play (c'est à dire branchement et débranchement sans avoir besoin d'arrêter le PC), jusqu'à 127 périphériques possibles, fiabilité et sécurité (détection et correction d'erreurs), plusieurs vitesses possibles et 4 types de transferts.

### 4. Facilité d'utilisation

Le principal but du bus USB est la facilité d'utilisation qui se traduit suivant de multiples critères que voici :

Tout d'abord on peut dire que le protocole USB est une norme très souple ; une interface unique suffit pour commander plusieurs types de périphériques, il suffit juste de posséder le bon driver.

Ensuite, un point fort de l'USB est sa configuration automatique, on l'appelle aussi le « plug & play ». Cela signifie que si l'utilisateur connecte un périphérique USB, Windows détecte automatiquement ce périphérique et charge le driver approprié s'il est disponible dans les fichiers de Windows. Si ce n'est pas le cas, Windows demande d'installer le disque (CDRom) contenant ce driver afin qu'il le copie dans son répertoire de drivers (généralement c:/windows/system32/drivers) et ceci se fait une seule fois.

Lors du prochain branchement, le périphérique USB recharge automatiquement son driver, cette étape est alors transparente pour l'utilisateur. Il faut noter aussi qu'il n'est pas nécessaire, avec le protocole USB, de lancer un fichier d'installation ou de redémarrer (rebooter) le PC avant d'utiliser le périphérique.

Il n'y a pas non plus dans le protocole USB à faire le choix de l'adresse du port comme par exemple pour une liaison série, une adresse dynamique est allouée à chaque fois que l'on branche un autre périphérique, la norme USB est capable de fournir 127 adresses dynamiques différentes (codée sur 7 bits) c'est pour cela qu'on entend toujours parler de brancher jusqu'à 127 périphériques (en théorie) sur le bus USB.

Le PC alloue donc une série d'adresses de port et une commande d'interruption (IRQ) pour les interfaces USB. Contrairement aux autres périphériques qui ne sont pas USB, pour chaque ajout d'un périphérique il faut lui associer une adresse de port, souvent un IRQ ainsi qu'une carte d'extension si le PC n'en possède plus ou pas assez. Avec les périphériques USB, tous ces problèmes sont inexistant.

Un atout majeur du bus USB est qu'il est très facile à connecter, on n'a pas besoin d'ouvrir le PC à chaque fois que l'on veut rajouter un périphérique, les anciens PC possèdent au minimum deux ports USB (de nos jours il y en a davantage). On peut aussi connecter un HUB sur un port déjà existant pour pouvoir connecter encore plus de périphériques USB. Le HUB joue en quelque sorte le rôle d'une multiprise. C'est un multiplieur de port USB. Actuellement certains moniteurs, claviers ... possèdent des HUBs intégrés. On peut, évidemment, reconnecter d'autres HUBs aux HUBs cité précédemment. Nous verrons par la suite les consignes à respecter pour conserver l'esprit de la « norme USB »

Un autre avantage du bus USB est la connectique. En effet les câbles possèdent deux extrémités bien différentes pour ne pas se tromper lors du branchement, de plus les câbles USB sont très compacts comparer aux câbles pour les liaisons parallèles ou séries.

On dit également que le bus USB est « Hot pluggable », c'est à dire que l'on peu connecter et déconnecter un périphérique USB tout en ayant le PC allumé. L'Operating System installé sur le PC (Windows ...) le reconnaît ainsi immédiatement.

L'USB possède une caractéristique très intéressante : c'est le mode veille lorsque l'on n'utilise pas le périphérique. On l'appelle aussi « Power conservation ». En effet le bus USB se met en suspend après 3 ms ou il n'est plus utilisé. Pendant ce mode, le composant ne consomme que 500µA

Enfin, le dernier point fort pour l'USB est que cette norme permet d'alimenter le périphérique directement avec le PC. Il n'est pas nécessaire d'alimenter le périphérique avec une alimentation extérieure, le PC suffit

Nous verrons par la suite les conditions à respecter pour que cette « auto alimentation » soit opérationnelle

## **II. Vitesse de transfert de l'USB**

### **1. Définitions des différentes vitesses**

**L'USB supporte 3 vitesses :**

***Low Speed à 1.5Mbit/s – (USB 1.1)***

***Full Speed à 12Mbit/ s – (USB 1.1)***

***High Speed à 480Mbit/ s – (USB 2.0)***

Tout les PC supporte actuellement deux vitesses de bus, le Full Speed et le Low Speed. La vitesse High Speed a été ajoutée avec l'apparition de la spécification USB 2.0. Cependant, pour pouvoir utiliser cette vitesse de transfert, il faut être équipé de cartes mères et de contrôleurs USB supportant l'USB 2.0.

#### **a. Débits théoriques**

Ces vitesses sont en fait des vitesses approximatives. Ce sont en fait les vitesses que peuvent supporter les différents bus USB. Le taux de transfert de données réel est plus faible. En fait le bus doit faire passer, outre les données, les bits de status, de contrôles et les bits d'erreurs. Sans oublier que plusieurs périphériques peuvent se partager le bus.

## **b. Débits réels**

Pour du Low Speed le débit réel est de 800oct/s en mode Interrupt. En effet la norme permet 1 interruption toutes les 10ms, dans le meilleur des cas ( réglable de 10 à 255ms) donc 100 interruptions/s et comme la taille du paquet est de 8 octets pour cette vitesse, le débit est de 800octets/s

Pour du Full Speed, en mode Interrupt la norme permet une interruption au mieux toutes les 1ms (réglable de 10 à 255ms) c'est à dire 1000 interruptions/s, la trame pouvant s'étendre à 64 octets on obtient un débit de 64Ko/s. En mode Bulk on arrive à un débit réel d'environ 1Mo/s. La bande passante est divisée dans ce cas s'il y a plusieurs périphérique qui travail simultanément.

Et pour le High Speed le taux de transfert réel théorique est de 53Mbit/s

Vous pouvez voir qu'ici, avec les débits réels, on s'éloigne fortement des débits annoncés aux grand public. Il faut bien étudier : choisir le mode et la vitesse ( Low – Full – High Speed) en fonction de l'application pour ne pas avoir de surprises. Tout les avantages et qualités de la norme énoncés plus haut sont correctes mais il faut les placer dans les bons contextes et ne sont pas forcément pas tous compatible entre eux. De plus il faut bien préciser que les caractéristiques énoncées ci-dessous sont les caractéristiques de l'USB même. Si l'application développée fonctionne avec l'OS Windows il faut encore rajouter d'autres contraintes comme par exemple l'intervalle entre les différentes interruptions qui sont de l'ordre de 32ms au minimum...donc le débit en mode interrupt du Full Speed va être encore inférieur.

## 2. Raison des trois vitesses de l'USB

Le Low Speed a été introduit pour deux raisons essentielles, la première est que les périphériques USB ne sont pas cher, c'est à dire que tous le monde peut se le permettre. La deuxième raison est pour avoir des souris USB un peu plus pratiques. En effet les câbles USB pour le Low Speed n'ont pas besoin d'être blindés et, de ce fait, sont très souples.

Le Full Speed a été conçu pour remplacer les liaisons séries et parallèles.

Le High Speed est une vitesse supplémentaire qui a été introduit lors de la mise à jour de la norme USB 2.0. Elle permet de mettre en valeur la puissance du bus USB.

## 3. Domaine d'utilisation des différentes vitesses

Les applications Low Speed concernent essentiellement des périphériques interactifs (claviers, souris, consoles), mais aussi des afficheurs, des lecteurs (de carte à puce) et des applications en automatismes (mesure, capteurs) appelées à se développer.

En mode Full Speed on va trouver la téléphonie, les modems, les disques, les imprimantes, les fax ainsi que les scanners, certains lecteurs de carte à puce et le domaine multimédia (jeux, audio, vidéo limitée).

## 4. Les débits de l'USB

Il faut noter que dans la norme USB le débit n'est pas proportionnel à la vitesse.

L'USB « Low Speed » qui est consacré aux périphériques soit disant lents est limité à échanger au maximum 8 octets toutes les 10ms, ce qui correspond à un débit maximum de 800octets/sec soit 6400 bps. Par certains moyens détournés il est possible d'espérer atteindre 8Ko/s, mais c'est sans garantie.

L'USB « Full Speed » qui est consacré aux HUBs et aux périphériques non lents peut échanger jusqu'à 1024 octets toutes les ms, soit un débit de 1 Mo/s.

## III. Le câble USB

### 1. Définition du câble USB

Le câblage USB est relativement simple ; il a la même structure quelle que soit la vitesse de transmission. Le câble transporte deux paires de fils :

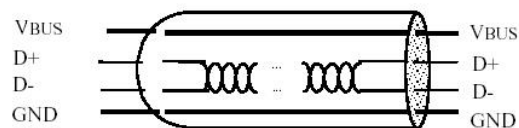
La paire de signal destinée au transfert de données D+ et D- et une seconde paire qui peut être utilisée pour la télé-alimentation GND et Vcc. La première paire est non blindée pour les périphériques lents tels que les claviers, souris fonctionnant à 1.5Mbits/s tandis que caméras, micro et autres ont recours à une paire de fils torsadée blindée pour atteindre les 12Mbits/s

## 2. Composition du câble USB

Chaque connecteur dispose de deux fils d'alimentation (5V et GND) et deux fils destinés au transfert de données (D+ et D-).

Une connexion entre deux PC est aussi possible par l'adjonction d'une interface spéciale qui déjoue la vigilance du PC maître et transforme le second PC en «esclave»

En version Low Speed le blindage n'est pas obligatoire (ce qui assure une plus grande souplesse de manipulation en particulier pour une liaison souris).



**Figure 1 :** Composition d'un câble USB

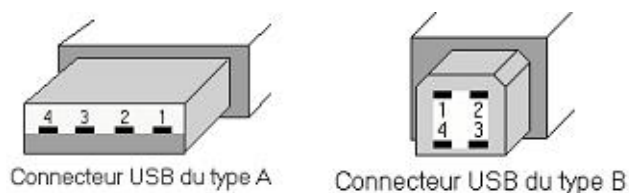
La longueur maximale autorisée par la norme est de 3m pour un câble non blindé donc généralement pour un périphérique Low USB (= 1.5Mb/s) et de 5m pour un câble blindé dans le cas d'un périphérique Full USB (=12Mb/s).

Le câble USB est composé de deux fiches bien différentes :

En amont d'une fiche appelé connecteur USB de type A, branché au host (PC).

L'extrémité aval par contre peut se retrouver en deux versions :

Connecteur USB du type B et un mini connecteur type B (appelé souvent optionnel). Ce dernier est réservé aux dispositifs de très faible dimension (ou de grande intégration) tels les appareils photo numériques.



**Figure 2 :** Brochage des connecteurs USB de type A et B

Et voici l'aspect des connecteurs en trois dimensions :



**Figure 3** : Aspect des connecteurs USB de type A et B

## IV. La Norme USB

### 1. Introduction

Je vais dégrossir dans cette partie, la norme USB pour ne pas décourager certains. Comme vous le verrez, la norme USB 1.1 est un document en anglais de plus de 320 pages. Ne parlons pas de la norme 2.0 qui fait plus de 650 pages. Vous verrez avec un peu de recul qu'il n'est pas obligé de tout lire pour débiter avec l'USB, quelques chapitres suffisent amplement pour un bon début.

### 2. Explication des différents chapitres de la norme USB

Voici un aperçu en français du contenu des différents chapitres de la norme USB 1.1.

#### Chapitre 1 : Introduction

C'est un chapitre sans grand intérêt, il explique pourquoi l'USB est apparu et dans quel but. Il n'apporte rien techniquement mais peut être intéressant pour la culture USB.

#### Chapitre 2 : Termes et abréviations

Ce chapitre est un glossaire rassemblant toutes les abréviations utilisées dans la norme USB. Je ne vous conseil pas de le lire mais il faut savoir qu'il existe et s'y rendre si au cours de la lecture vous ne comprenez pas une abréviation.

#### Chapitre 3 : Origine

Ils expliquent ici les objectifs de l'USB, notamment le Plug & Play et la simplification de cette nouvelle norme point de vue utilisateur. Ce chapitre est aussi plus ou moins inutile.

#### Chapitre 4 : L'architecture USB

Ce chapitre est important, il donne une vue d'ensemble d'un système USB d'un point de vue topologie, débit de données, type de flux existant, spécifications électriques de base.

### **Chapitre 5 : Flux de donnée USB**

Ce chapitre parle des transfert de trame USB et explique en détails les types de flux existant ( Contrôle, Interruption, Isochrone, Bulk). Il introduit également des notions très importantes tels que les Endpoints, les Pipes. Tout ce qui est dans ce chapitre est important mais à différents niveaux. A vous de voir quelle est votre utilisation de l'USB pour ne pas vous surcharger de connaissances « inutiles ».

### **Chapitre 6 : L'USB d'un point de vue mécanique**

Ce chapitre donne les normes des connecteurs USB, les dimensions exactes, les matériaux utilisés etc. Bref très inutile pour notre utilisation. Seules les premières pages sont importantes, elles donnent la différence entre les cordons Low ou Full USB ainsi que la signification des fils.

### **Chapitre 7 : L'USB d'un point de vue électrique**

Ce chapitre traite des couches de bas niveaux des signaux électriques, l'impédance de la ligne, le codage de bits, les résistances de Pull up pour différencier le Full USB du Low USB. Il n'est pas nécessaire de le lire entièrement.

### **Chapitre 8 : Protocole USB**

Ce chapitre explique comment transit les paquets USB, la synchronisation, le PID, l'adressage, l'Endpoint, les champs CRC. Je vous rassure tout de suite, il n'est pas nécessaire de connaître ce protocole pour faire fonctionner de l'USB avec un microcontrôleur par exemple, parce que c'est le microcontrôleur qui gère cette couche automatiquement sans que l'utilisateur s'en aperçoive.

### **Chapitre 9 : Mode de fonctionnement d'un composant USB**

Ce chapitre est très important, il parle de l'énumération des composants USB, c'est la toute première chose que fait l'OS lorsqu'on connecte un composant USB. Il est important de le lire en détails pour bien comprendre ce principe. Il est très utile pour les programmeurs qui écrivent le soft puisqu'il donne les commandes à utiliser pour communiquer avec le composant USB.

### **Chapitre 10 : Host USB : Hardware et Software**

Ce chapitre donne toutes les réponses concernant le maître (host) et les liens entre le PC niveau soft et le composant USB du point de vue hard. Inutile pour une première lecture.

### **Chapitre 11 : La spécification des HUB**

Les hubs sont des composants USB particuliers qui ont un protocole et une architecture différente de celle d'un composant USB classique. Ce chapitre, qui est très long détail les mécanismes des hubs, leurs configurations, leurs descripteurs...  
Ce chapitre est inutile si vous ne touchez qu'aux composants USB classiques.

Après une première analyse, on s'aperçoit que le document de 320 pages se réduit déjà à une centaine de pages utiles. Cela est plus tentant pour attaquer l'étude.

Par la suite, je vais pas forcément traiter les chapitres dans le même ordre que la norme USB, j'exposerais entièrement mon étude sur l'USB, les chapitres apparaîtrons au fur et à mesure du besoin. Le plan que j'ai adopté pour vous exposer l'USB est d'après moi plus adapté que celui ci, il traite tous les sujets auxquels je me suis heurté en développant ma carte USB.

## **V. Operating System**

### **1. L'USB et Windows**

On parle essentiellement de l'OS Windows avec l'USB, ces deux mots sont en fait très liés car le créateur de l'USB est en fait en grande partie Microsoft. Mais il faut aussi savoir que l'USB existe également sur les iMac. L'USB va être prochainement aussi opérationnel sur les plates-formes Linux, NetBSD et FreeBSD.

Il est même possible d'écrire des drivers USB sous DOS.

### **2. Compatibilité des OS avec l'USB.**

Windows 98 est le premier operating system (OS) capable de gérer correctement l'USB, les successeurs Windows 2000, Windows Millenium et Windows XP l'intègre correctement également et sont plus riches en drivers pré-installés. Le terme Plug & Play se justifie davantage. Il existe une version de Windows 95 capable de gérer l'USB : Windows 95's OEM Service Release connue sous le nom de Windows 95 OSR2.1 et 2.5.

Mais l'USB n'est, à cette époque qu'à ces premiers balbutiements et les caractères Plug & Play et fiabilité ne sont pas toujours assurés. Certains périphériques possèdent des drivers pour Windows 95 mais il ne vaut mieux pas compter là dessus puisque les mises à jour de l'OS coûte même plus cher qu'un nouvel OS.

Donc en résumé, pour utiliser de l'USB, il est préférable d'opter pour un OS Windows 98 sorti en juin 98 ou une version supérieure. La première version de Windows98 (Windows 98 Gold) est moins complète niveau compatibilité USB, la version suivante (Windows 98 Second Edition) est corrigée et fonctionne correctement avec les composants USB. Les versions ultérieures : Windows 2000, Windows Me, Windows XP ... sont des versions encore plus compatibles USB et plus stables. L'USB n'est pas supporté par Windows NT4. Pour développer de l'USB sur des versions Windows NT ultérieures, on peut utiliser l'extension BSQUARE pour écrire un driver adéquat, mais d'après les connaisseurs ce n'est pas une tâche facile...

### **3. Autre solution si le PC est trop ancien**

Si par contre la carte mère ne possède pas de connecteurs USB, il est possible d'insérer des cartes RS232-USB ou RS485-USB ou simplement des cartes USB qui se branchent sur un port PCI qui font le liens entre la sortie USB de la carte mère et la sortie USB utilisateurs. En fait les constructeurs de cartes mères ont à l'époque inclus l'USB sur leurs cartes mais n'ont pas prévu de les rendre accessibles à l'utilisateur. C'est seulement plus tard que Microsoft et Intel's PC 2001 System Design Guide ont instauré « l'obligation » de mettre au moins deux connecteurs USB sur la face arrière du PC.

## 4. Caractéristique d'un OS gérant l'USB

Il faut respecter trois conditions pour prétendre être un operating system capable de gérer l'USB.

D'une part il faut qu'il puisse gérer le branchement et le débranchement de périphérique. D'autre part il faut qu'il puisse communiquer avec tous les nouveaux périphériques qui viennent d'être branchés pour trouver le moyen le plus adéquat pour pouvoir transférer des données.

Et finalement, il doit pouvoir produire un mécanisme permettant aux drivers de communiquer avec le Host et le périphérique USB, que l'on appelle généralement l'énumération.

A un niveau plus élevé, on peut aussi dire qu'un OS gérant l'USB doit contenir des drivers pour différents périphériques, qui font le lien entre l'operating system et le composant USB. Si l'operating system ne possède pas le driver par défaut du périphérique à installer, c'est le fabricant du périphérique qui doit le fournir. Nous verrons par la suite qu'il est également possible de faire « soi même » un driver via un outil de Microsoft. (DDK : Device Development Kit).

Les drivers de composants USB utilisent le nouveau Win32 Driver Model (WDM) qui est une architecture de drivers fonctionnant sous les différentes versions de Windows à partir de Windows 98. Le but de l'utilisation de ce WDM est de fournir, à l'avenir, un driver unique. Actuellement il existe encore des drivers différents pour les différentes versions de Windows.

## 5. Bugs de l'USB

Certains doivent le savoir, mais l'USB pose quelque fois des problèmes. Le problème peut venir soit du hard, soit du soft au niveau du PC ou du périphérique USB même. Ces problèmes sont dus à la complexité du protocole USB et au fait que l'USB est assez récent. Généralement ses bugs sont corrigés lors des nouvelles versions de drivers. Mais de nos jours l'USB est quasiment Plug & Play, rare sont les périphériques qui provoquent encore des écrans bleus sous Windows.

## 6. Compatibilité USB 1.1 - USB 2.0

Comme dit précédemment, l'arrivée de l'USB 2.0 est une grosse évolution point de vue USB. Elle permet en effet le transport de données avec un débit très impressionnant ; 480 Mbit/s.

Il faut savoir que cette nouvelle norme d'USB (spécification 2.0) est entièrement compatible avec les versions précédentes. Les périphériques USB 2.0 peuvent utiliser les mêmes connecteurs et les mêmes câbles que les périphériques 1.1.

Par contre pour utiliser la vitesse High Speed il faut que le périphérique soit connecté à un host ou à un hub supportant cette vitesse. Les hubs 2.0 peuvent sans problèmes être connectés aux périphérique 1.1.

## 7. USB et IEEE-1394

L'USB high speed à 480 Mbit/s est en concurrence avec l'IEEE-1394 (Firewire). A présent il y a aussi l'IEEE-1194b qui monte à une vitesse de 3.2Gbit/s.

En ce moment les nouveaux périphériques ont le choix entre l'USB et le Firewire, création d'Apple.

L'avantage de l'IEEE-1394 est qu'il est plus rapide est plus flexible que l'USB mais par contre il est plus cher. L'IEEE-1394 est mieux adapter pour le transfert vidéo et d'autres utilisations ou la vitesse de transfert est importante. L'USB lui par contre est mieux adapté pour les périphériques de bases tel que les claviers, les imprimantes, les scanners et les lecteurs de disques durs. En clair pour des applications nécessitant une vitesse de transfert de données normale, et dont le coût doit être réduit. L'IEEE-1394 utilise une topologie « peer-to-peer », c'est à dire que chaque périphérique peut communiquer avec un autre directement sans passer par le host comme avec l'USB. On peut donc envoyer l'information à plusieurs périphériques à la fois. L'IEEE-1394 est donc de ce fait plus flexible que l'USB mais cette flexibilité est plus complexe et a un prix.

## VI. Le Forum USB

### 1. Introduction du forum

C'est LE forum de référence pour l'USB. Il a été créé par les créateurs de la norme USB, pour aider les développeurs et pour assurer une certaine qualité des composants USB. C'est sur ce site ( <http://www.usb.org> ) ou l'on peu trouver des informations sur la spécification USB. On trouve donc ici la norme USB sous format Acrobat Reader (\*.pdf).

### 2. Fonctionnement du forum

Toute personne peut librement développer une application utilisant l'USB. Mais à partir du moment ou elle vend ce composant elle est obligé de passer par ce Forum pour obtenir un Vendor ID légal. Le prix de ce Vendor ID coûte environ \$1500.

Par contre si l'on veut bénéficier des avantages de l'USB-IF ( Implementers Forum) il faut payer \$2500 par an... Bien sur avec ce prix on vous offre un Vendor ID. Les avantages à adhérer à ce forum sont multiples... vous aurez droit à une sorte de mise à jour tout les trimestres de toutes les nouveautés de l'USB, vous aurez le droit de participer aux groupes que développent des composants USB, les frais sur l'utilisation du logo USB seront également offerts et enfin vous aurez droit à avoir une petite place sur le site usb.org pour exposer vos différents composants... Je vous invite à jeter un œil sur ce site, vous aurez plus de détails sur l'utilisation et la philosophie de ce forum.

Autrement dit si vous êtes une petite entreprise, il n'est pas intéressant d'investir dans l'USB du point de vue création de composant. Par contre cela devient rentable pour les entreprises qui fabriquent de gros volume de composants. A part la norme USB, documents en anglais très complet et très lourd à digérer, il n'existe pas beaucoup de documentation sur le fonctionnement de l'USB, peut être pour inciter l'inscription au Forum USB-IF ...

## VII. Le Bus USB

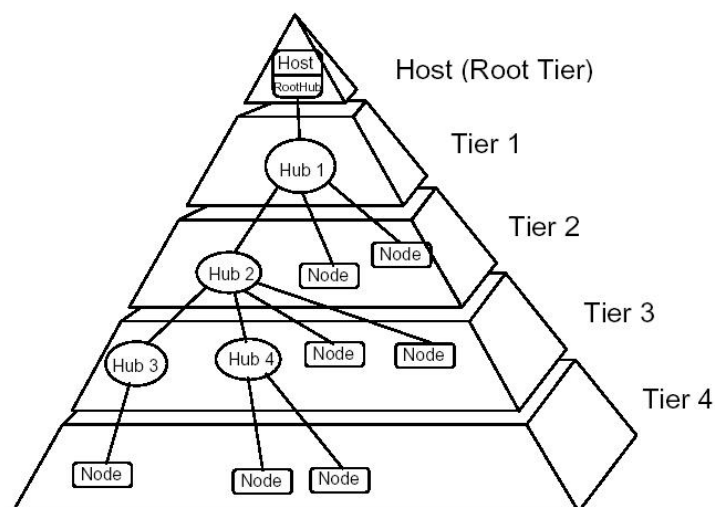
### 1. Principe du bus USB

Le bus USB est un bus fonctionnant sur la hiérarchie, commandé par un host unique. Le host utilise un protocole Maître/Esclave pour communiquer avec les périphériques USB. Cela signifie que c'est le host qui décide du transfert des données et que les différents périphériques ne peuvent pas établir de connexion entre eux tant que le maître n'a pas donné l'autorisation. On peut dire que cela peut être un inconvénient par rapport aux autres protocoles mais il ne faut pas oublier que l'USB a été conçu avec des compromis de coût et de performance. Le fait que le bus USB fonctionne avec le protocole Maître/Esclave résout implicitement ces problèmes comme par exemples les problèmes de collision ou d'arbitrage de périphériques.

### 2. Topologie du Bus USB

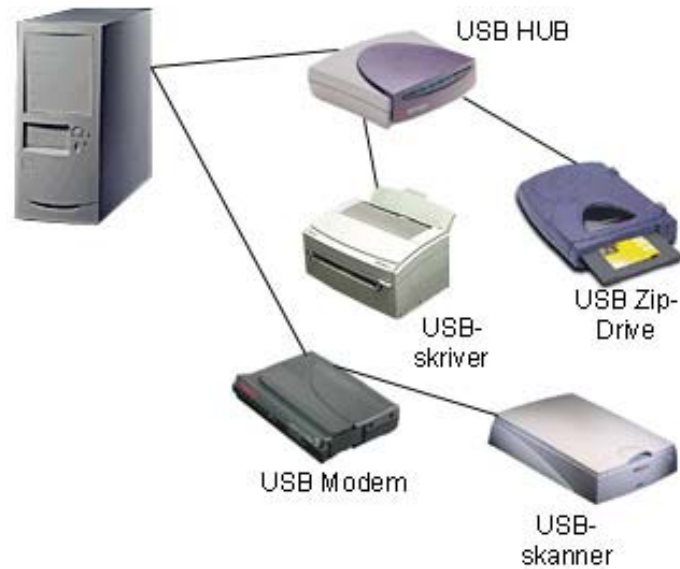
La topologie est une topologie « étoile série » ou encore « tiered star » qui tolère jusqu'à 5 niveaux de concentrateurs. La spécification limite le nombre de périphérique à 127. Un dispositif conforme à la norme 2.0 peut-être relié à un HUB 1.1 mais dans ce cas le trafic sera celui défini par la norme 1.1. Les divers cas possibles sont précisés sur la figure ci-dessous.

Notons qu'à l'initialisation d'un "device" (ou d'une fonction selon la terminologie utilisée par le groupement USB) celui-ci reçoit une adresse. Les transferts Full Speed déclenchés par le "host" parcourent tous les tronçons, mais seule la fonction adressée va évidemment répondre. Ainsi s'il s'agit d'un ordre d'impression il sera reçu par tous les périphériques, mais seule l'imprimante le comprendra.



**Figure 4 :** Topologie du bus USB

Et voici en image un exemple de branchement avec le topologie étoile de l'USB.



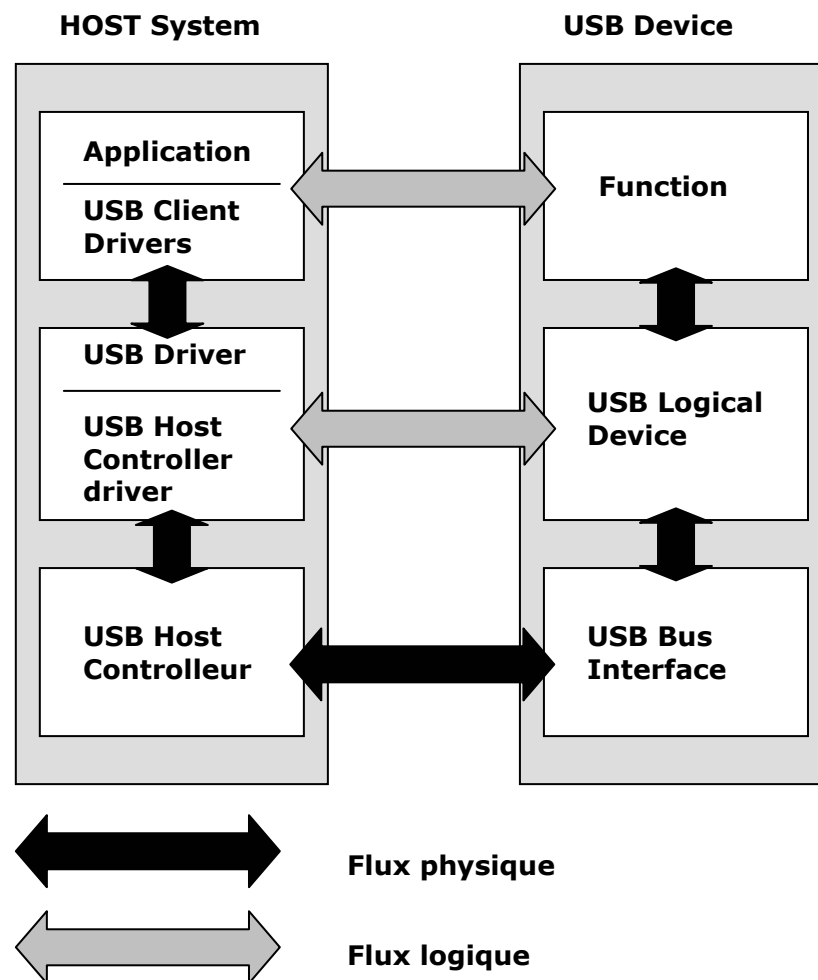
**Figure 5 :** Exemple de branchement respectant la topologie du Bus USB

### 3. Protocole USB

Le protocole USB est, comme tout les autres protocoles un protocole à encapsulation. Mais avant de s'intéresser à la trame proprement dite, voyant ce qu'il en est du protocole.

Le client driver communique les demandes de transfert des applications via des IRP ( I/O Packet). Puis, l'USB driver traduit chaque transfert en une suite de transactions. Ensuite l'USB Host Controller driver regroupe les transactions en trames et finalement l'USB Host Controller traduit les transactions en paquets et enchaîne les trames.

Le synoptique ci dessous traduit de façon imagée les liens entre les différents éléments.



**Figure 6 : Protocole USB**

#### 4. Type de paquet USB

Contrairement à la liaison série RS232 et des interfaces séries similaires où le format des données envoyées n'est pas défini, l'USB lui est composé de plusieurs couches de protocoles bien définis. La plupart des circuits intégrés USB s'occuperont de la couche inférieure, la rendant ainsi presque invisible au regard du concepteur final. Mais il est tout de même intéressant d'en connaître les grandes lignes.

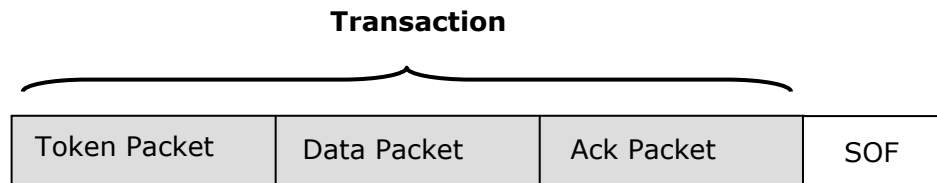
L'USB a quatre types différents de paquet :

- Token ( En-tête)
- SOF (Start of frame)
- Data ( Optionnel )
- Acknowledge ( Handshake)

Les **paquets jetons** indiquent le type de la transaction qui va suivre et a pour but de transporter l'adresse USB et le sens du transfert. Les **paquets de données** contiennent les données utiles. Les **paquets "Handshake"** sont utilisés pour valider les données ou rapporter les erreurs. Et les **paquets début de trame (SOF)** indiquent le commencement d'une nouvelle trame.

L'entité de transfert USB est appelée *transaction*, elle est généralement constituée de paquets juxtaposés, la transition étant un paquet SOF qui indique le début d'une autre transaction. On dit alors qu'un transfert est composé d'une succession de transactions

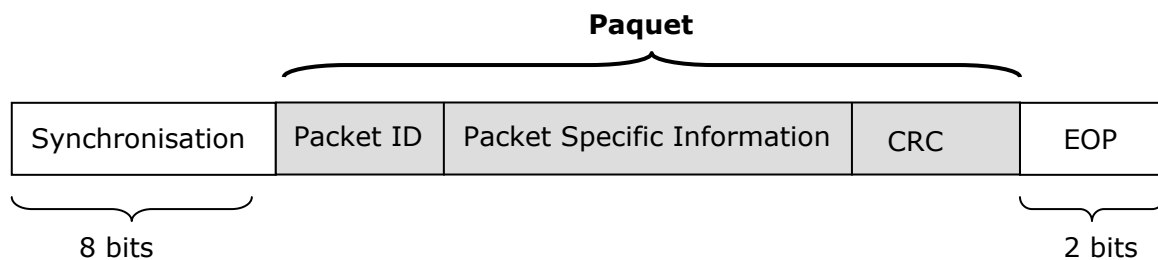
Voici ci dessous le détail d'une transaction :



**Figure 8 : Transaction USB**

Le premier bit transmit est le bit LSB.

Les différents paquets ont une structure bien définie que voici:



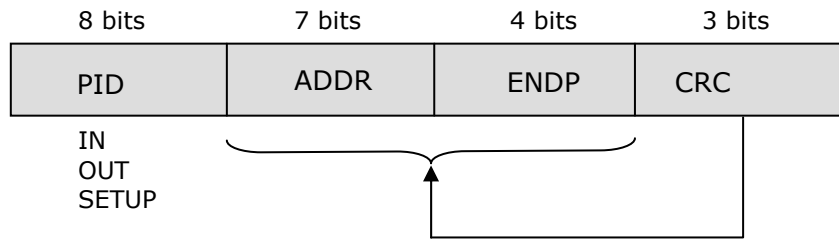
**Figure 9 : Structure des paquets USB**

La structure des paquets étant identique il faut préciser que le format est différent selon la nature du paquet. Voici ci dessous les différents formats de paquet. Les trames ont une durée de 1ms, elles sont marquées par le Token SOF ( Start of Frame). Avec le système *d'encapsulation*, il faut bien noter que les transactions sont regroupées à l'intérieur des trames sans qu'il y ait de chevauchement. Notons également qu'une trame peut contenir une ou plusieurs transactions destinées à un même périphérique.

### a. Les paquets jetons

- Il y a 3 sortes de paquets Jetons,
  - **In** : Informe l'appareil USB que l'hôte veut lire des informations.
  - **Out** : Informe l'appareil USB.que l'hôte veut envoyer des informations.
  - **Setup** : Utilisé pour commencer les transferts de commande.

Les paquets *Token* doivent se conformer au format suivant,



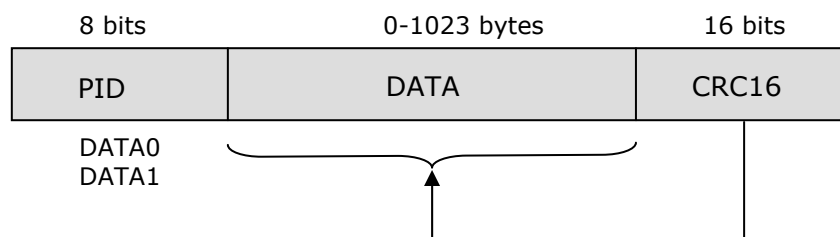
**Figure 10** : Format du paquet *Token*

### b. Les paquets *Data*

Il y a 2 sortes de paquets de données pour la norme 1.1 ( *Data0* et *Data1*)  
 Le mode High Speed définit 2 autres PIDs de données, *DATA2* et *MDATA*.

- La taille maximale de données " données utiles " pour les périphériques Low Speed vitesse est de 8 octets.
- La taille maximale de données " données utiles " pour les périphériques Full Speed est de 64 octets.
- La taille maximale de données " données utiles " pour les périphériques High Speed est de 1024 octets.
- Les données doivent être envoyées en multiple d'octets.

Les paquets de *Data* ont le format suivant :



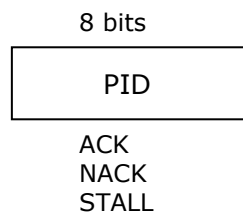
**Figure 11** : Format du paquet *Data*

### c. Les paquets « *Handshake* »

- Il y a 3 sortes de paquets *Handshake* qui font simplement partie du PID.
  - **ACK** - validant que le paquet a été reçu correctement.

- **NAK** - Indique que l'appareil ne peut temporairement ni envoyer ou recevoir des données. Aussi utilisé pendant les transactions d'interruptions pour avertir l'hôte qu'il n'a pas de données à envoyer.
- **STALL** (Bloqué) - L'appareil se retrouve dans un état qui va exiger l'intervention de l'hôte.

Les paquets *Handshake* ont le format suivant :

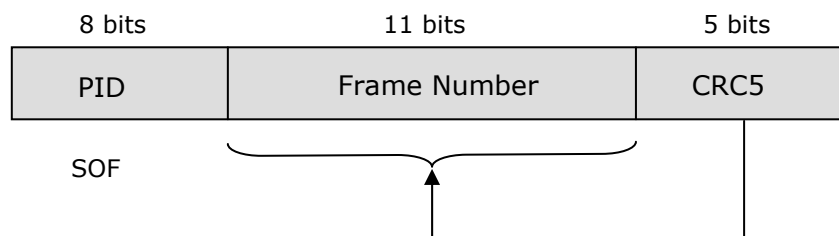


**Figure 12** : Format du paquet Handshake

#### d. Les paquets SOF

Le paquet SOF composé d'une trame de 11 bits est envoyé par l'hôte toutes les  $1\text{ms} \pm 500\text{ns}$  sur un bus Full Speed vitesse ou bien toutes les  $125\mu\text{s} \pm 0,0625\mu\text{s}$  sur un bus High Speed.

Les paquets *SOF* ont le format suivant :



**Figure 13** : Format du paquet SOF

Le standard USB assure une grande fiabilité des transferts par la détection de nombreuses erreurs au niveau hardware. :

- Erreur de paquets
  - Paquet ID
  - Bit Stuff
  - CRC
  - Data Toggle
- Time-out ( absence de réponse )
- 'Babbling'
- LOA (Loss of Activity)

Toute détection d'une erreur de paquet est traduit par la non réponse du périphérique, c'est à dire un Time Out.

### e. Explication des différents champs des paquets d'une trame USB

- **Sync**

Tous les paquets doivent commencer avec un champ Sync. Le champ Sync fait de 8 bits de long pour la basse et pleine vitesse ou 32 bits pour la haute vitesse est utilisé pour synchroniser l'horloge du récepteur avec celle de l'émetteur / récepteur. Les 2 derniers bits indiquent l'endroit où le champ PID commence.

- **PID**

PID signifie Paquet ID. Ce champ est utilisé pour identifier le type de paquet qui est envoyé. Le tableau suivant montre les valeurs possibles.

Groupe	Valeur PID	Identificateur Paquet
<b>Token</b>	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
<b>Data</b>	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
<b>Handshake</b>	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET (No response Yet)
<b>Special</b>	0000	PREmbule
	1100	ERR
	1000	Split
	0100	Ping

**Tableau 1** : Désignation des PIDs

Il y a 4 bits pour le PID, toutefois pour s'assurer qu'il a été reçu correctement, les 4 bits sont complétés et répétés faisant un PID de 8 bits au total. Le format résultant figure ci-dessous :

PID0	PID1	PID2	PID3	nPID1	nPID1	nPID2	nPID3
------	------	------	------	-------	-------	-------	-------

**Figure 14** : Format réel d'un PID

Voici ci dessous les explications des différentes abréviations utilisées :

**SOF** = Start Of Frame

**SETUP** = Configuration

**ACK** = ACKnowledge ; Validation

**NAK** = No Acknowledge ; Pas de validation

**STALL** = Bloqué

**PREamble** = Synchronisation initiale

**Split** = Partager

**Ping** = S'assure d'une bonne connexion

- **ADDR**

Le champ adresse détermine à quel appareil le paquet est destiné. Sa longueur de 7 bits, lui permet de supporter 127 appareils. L'adresse 0 n'est pas valide, tant qu'un appareil qui n'a pas encore d'adresse attribuée, doit répondre aux paquets envoyés à l'adresse 0.

- **ENDP**

Le champ de terminaison est composé de 4 bits, autorisant 16 terminaisons possibles. Les appareils Low Speed, toutefois peuvent seulement avoir 2 terminaisons additionnelles au dessus du canal de communication par défaut ( 4 terminaisons maximales)

- **CRC**

Les Contrôles à Redondance Cyclique sont exécutés sur les données à l'intérieur du paquet de charge utile. Tous les paquets jetons ont un CRC de 5 bits tandis que les paquets de données ont un CRC de 16 bits.

- **EOP**

Fin de Paquet. Signalé par une sortie unique zéro (SE0) pendant une durée approximative de 2 bits suivie par un " J " d'une durée de 1 bit.

## **VIII. Notions importantes pour commencer avec l'USB**

### **1. Le branchement à chaud ou le «Hot Plug in»**

Un total de 127 interfaces peuvent être reliées à un ordinateur USB. La configuration des appareils branchés à une telle prise s'effectue à chaud, c'est-à-dire qu'il n'est plus nécessaire d'éteindre le PC pour brancher ou débrancher un appareil. Lors du branchement un échange d'informations, dénommé énumération dans le jargon USB, permet de reconnaître et de qualifier le nouveau venu.

Deux types de périphériques cohabitent sur le bus USB : ceux qui exigent une bande passante garantie (flux isochrone pour micro, flux «interrupt» pour souris) et ceux qui vont se partager le solde (flux «bulk» pour imprimante).

A l'opposé des chaînes SCSI, les éléments compatibles USB se connectent en étoile à un PC, c'est-à-dire que chaque entité a une connexion directe ou via un hub à la prise USB du PC. Nous verrons cette topologie par la suite

Lors du branchement d'un nouvel élément, une nouvelle adresse (#1... #127) est assignée par le passage d'un jeton, à l'image de Token Ring.

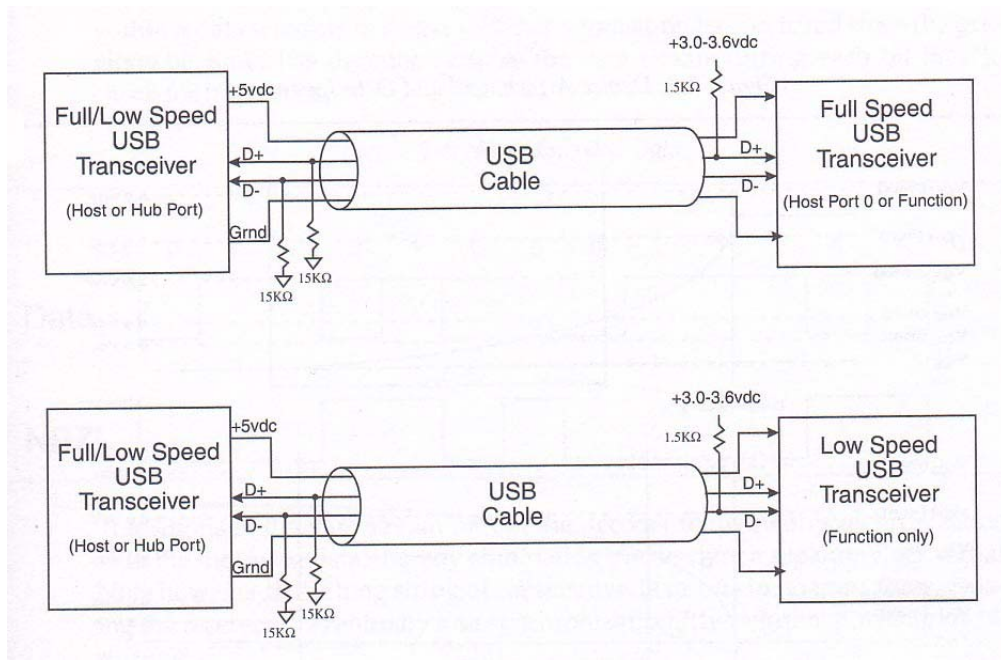
Les périphériques USB communiquent donc entre eux via le PC qui fait office de chef d'orchestre. En retour à chaque périphérique branché sera assigné un driver (ou pilote).

### **2. Choix « Low » ou « Full » USB**

Pour faire le choix entre la version USB Low Speed et Full Speed, il suffit de placer une résistance de tirage sur l'interface d'entrée (Fonction ou Hub). Cette résistance de tirage (Pull-Up de 1.5kOhm) est placée soit sur D- dans le cas du Low Speed ou sur D+ dans le cas du Full Speed.

Elle est indispensable pour permettre au HUB amont de détecter la connexion ou la déconnexion. Par contre côté sortie du HUB il y a une résistance sur chacune des lignes D+ et D- (Pull Down de 15kOhm)

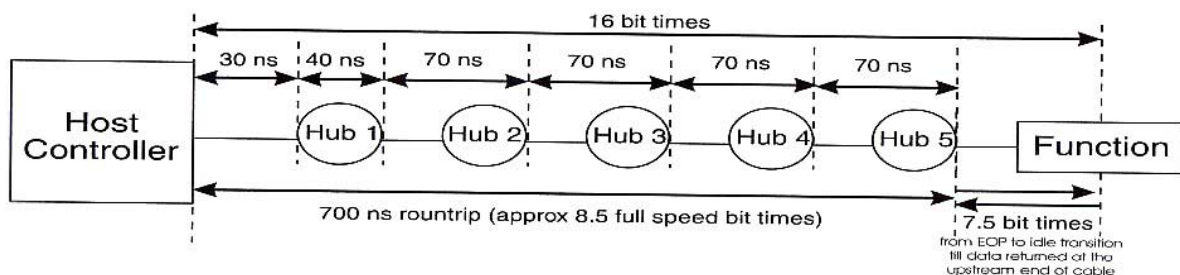
On voit sur le schéma ci dessous que s'il n'y a pas de résistance de Pull up, les deux câbles de données sont tirés à la masse par les deux résistances de Pull down. Les résistances Pull down étant de 15kOhm et les résistances de Pull up de 1.5kOhm (en fonction de la vitesse choisie) il y aura toujours une ligne de donnée qui sera à 90% de Vcc. Si le HUB détecte qu'une ligne de donnée avoisine les 90% de Vcc il en conclura qu'un composant USB est connecté ainsi que des informations sur la vitesse du composant, Low ou Full Speed. L'énumération peut commencer, c'est parti pour la grande aventure de l'USB.



**Figure 15 :** Choix Low ou Full USB – Résistance de Pull Up

### 3. Délai de propagation

La longueur maximum exploitable est conditionnée par les temps de propagation le long du câble (30ns par segment) et à travers les hub (40ns par HUB), et les temps de réponse de la fonction adressée (700ns maxi) sachant que le temps de propagation (aller-retour) entre le host et ladite fonction ne doit pas dépasser 1300ns. En outre, lorsqu'on dispose plusieurs HUBS en cascade, il ne faut pas sous-estimer les contraintes d'alimentation et la chute de tension le long du trajet : s'il s'agit d'un « device » alimenté via le bus USB il est clair qu'il doit pouvoir fonctionner avec une tension notablement inférieure à 5V



**Figure 16 :** Temps de propagation d'une trame USB

## 4. Temps de connexion et de déconnexion

### a. Introduction

Tout d'abord une petite remarque, les connecteurs USB (de type A ou B), ont les deux pattes d'alimentation un peu plus longues que les pattes de données. Ceci est volontaire, d'une part pour que le périphérique USB soit alimenté avant que transitent les données. Il est normal qu'avant de transmettre des données, que le périphérique soit reconnu et que les bons drivers ont été chargés. L'USB permet de faire cette étape automatiquement. D'autre part cette différence de longueur entre les pattes d'alimentations et de données est aussi pour protéger les composants USB, car un composant doit d'abord être alimenté avant de recevoir des données

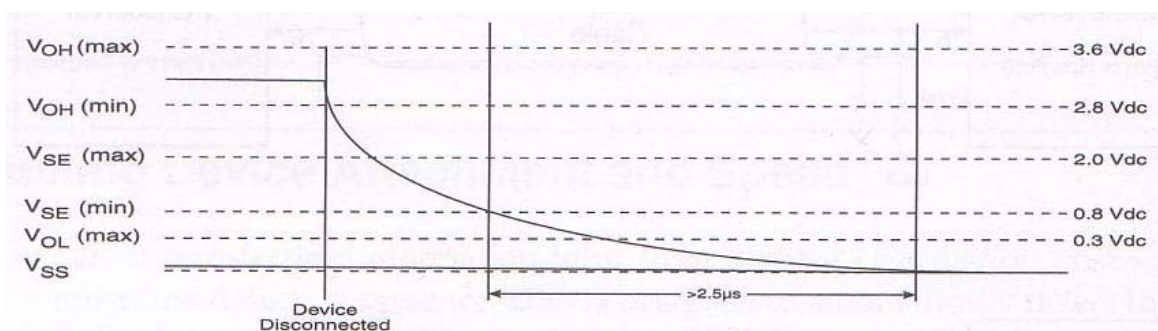
### b. Principe de fonctionnement de la connexion - déconnexion

Lorsqu'on branche un composant USB une différence de potentiel se crée et c'est cette différence de potentiel qui démarre le processus d'énumération. Sachant que le soft du host scanne continuellement les ports et les HUBs USB.

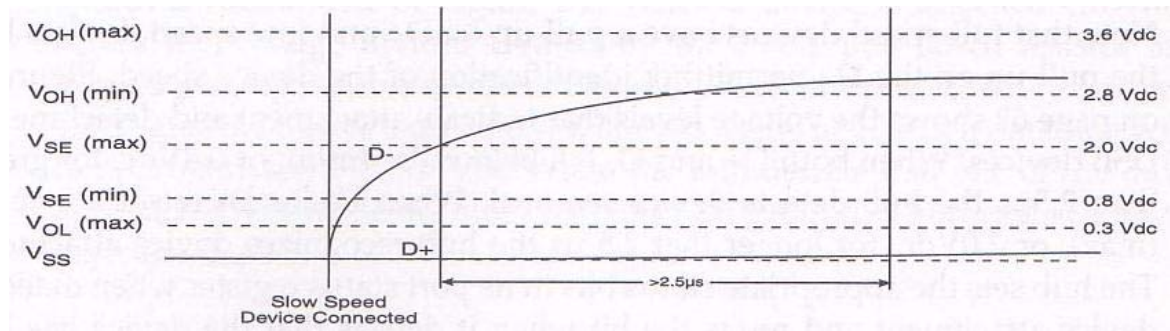
Les composants USB ont deux états, l'un pour les composants Low-Speed et l'autre pour les composants Full-Speed. Ces deux états sont opposés l'un par rapport à l'autre.

Ces états sont appelés état de référence K et J. Essayons à présent d'expliquer succinctement le fonctionnement. Comme on le sait déjà, les données D+ et D- sont des données différentielles, donc si un composant USB est branché, un câble de donnée ( D+ par exemple pour le Low-Speed) est proche de Vcc et l'autre proche de GND. On appelle cet état « l'état J ».

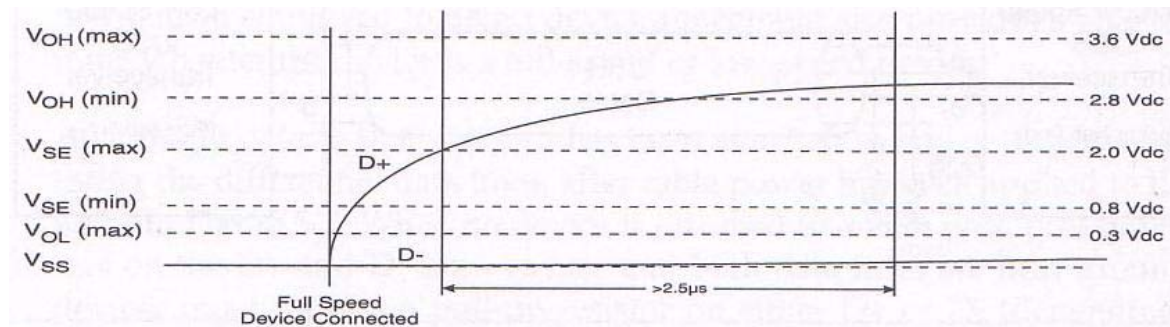
Si on débranche le composant, les deux lignes de données changent d'état et on appelle cet état l'état K. En s'aidant des schémas ci-dessous on s'aperçoit qu'un composant est déconnecté si les deux lignes de données ( D+ et D-) chute sous la valeur de 0.8Vdc sur une durée supérieure a 2.5µs. Et de même pour la connexion : si une ligne de données (D+ ou D- selon la vitesse) dépasse 2.0Vdc sur une durée supérieure à 2.5µs, le host considère que le composant est attaché. Ci-dessous les courbes de connexion et de déconnexion des périphériques USB.



**Figure 17** : Courbe de déconnexion des composants Low & Full USB



**Figure 18 :** Courbe de connexion des composants Low USB



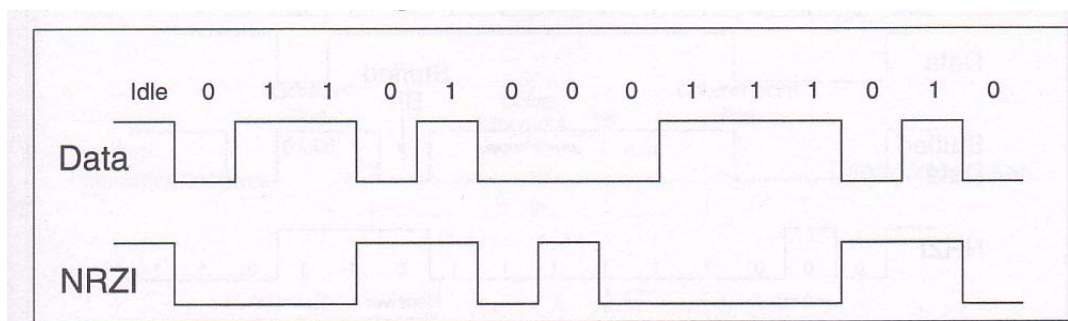
**Figure 19 :** Courbe de connexion des composants Full USB

## 5. Codage NRZI

Pour transmettre les données, l'USB utilise le codage NRZI ( Non Retour à Zéro Inversé). On ne va pas s'étendre longuement là dessus, je pense que cela n'est pas indispensable pour comprendre le fonctionnement de l'USB.

Le principe de ce codage est simple, Un «1» logique est représenté par un non changement d'état en NRZI et un «0» logique est représenté par un changement d'état. Le codage va encore plus loin et utilise le Bit Stuffing ; c'est le fait de mettre un «0» après 6 «1» logique consécutif pour forcer une transition dans le code NRZI ( pour éviter les pertes de données).

Ci dessous un exemple ce codage NRZI. Ce type de codage est uniquement utilisé pour le transport à travers le cordon USB. Il doit être décodé lors de la réception pour pouvoir retraiter les données.



**Figure 20 :** Principe du Codage NRZI

## 6. Consommation

Aucune fonction (Device ou HUB) ne peut consommer plus de 100mA sur le bus avant d'être énumérée. Après énumération un Device peut consommer jusqu'à 500mA pour un device « High power device » ou rester à 100mA pour un Low Power Device.

Chaque Hub doit pouvoir fournir 100mA par port en aval pour pouvoir alimenter d'autres périphériques. Un HUB « self powered » c'est à dire un HUB qui est alimenté par l'extérieur doit pouvoir fournir 500mA par port aval. Notons également qu'un HUB self powered doit posséder un limiteur de courant sur ses ports aval (5A max.) et signaler les anomalies au host.

On peut donc en déduire de cela qu'un périphérique High power doit posséder un dispositif d'alimentation séquentiel (100mA, puis 500mA). Par ailleurs, un bus powered HUB ne peut supporter que des Low Power Devices. Un High power Device ne peut être connecté qu'à un Self-powered HUB.

Notons également qu'on ne peut pas cascader directement 2 Bus powered HUBs.

## 7. Alimentation USB

### a. Alimentation de périphérique USB

Pour simplifier l'explication de l'alimentation des périphériques USB, la norme a prévu deux niveaux d'alimentations, le premier niveau consomme une unité d'énergie, l'autre consomme cinq unités. Une unité vaut 100mA. C'est à dire qu'il existe des composants qui consomment 100mA et d'autres 500mA. Généralement les composants Low USB consomment une unité d'énergie et les composants High USB consomment jusqu'à cinq unités. Par défaut tous les composants consomment une unité et c'est par soft qu'on lui demande de consommer plus si l'application le nécessite, dans le cas d'un composant High USB bien sur. Toutes ces informations sont contenues dans les descripteurs, le composant ne pourra jamais consommer plus que ce qui est prescrit dans son descripteur. On peut donc en déduire de cela qu'un périphérique High power doit posséder un dispositif d'alimentation séquentiel (100mA, puis 500mA).

Aucune fonction (Device ou hub) ne peut consommer plus de 100mA sur le bus avant d'être énumérée. Après énumération un Device peut consommer jusqu'à 500mA pour un device « High power device » ou rester à 100mA pour un Low Power Device.

L'USB est donc assez flexible et peut supporter plusieurs types d'alimentation. Certains composants peuvent être entièrement alimentés par le bus USB. (Bus powered) .

### b. Avantage de l'alimentation USB

Le fait de pouvoir alimenter un périphérique USB avec le même câble qui transporte les données a de multiples avantages. D'une part cela évite déjà à l'utilisateur de brancher le périphérique sur une prise de courant extérieure, ceci rend déjà le périphérique plus léger et moins encombrant. D'autre part d'un point de vue du concepteur, cela réduit le prix de fabrication.

### c. Différents types d'alimentation du BUS USB

- **Low-power bus-powered functions**

Les périphériques utilisés par un bus Low-Power puisent toutes leurs puissances de VBUS et ne peuvent consommer qu'une unité d'énergie. Les périphériques Low power alimentés par un Low Power Bus sont aussi conçus pour travailler avec une tension de VBUS s'échelonnant entre 4.4 V et 5.25V. Mais une tension de 4.4V suffit pour l'énumération. Beaucoup d'appareils fonctionnant à 3.3V doivent être muni d'un régulateur.

- **High-power bus-powered functions**

Les périphériques alimentés par un bus High-Power puisent toutes leurs puissances de VBUS et ne peuvent puiser qu'une unité d'énergie avant d'être configuré. Après la configuration, ils peuvent consommer jusqu'à 500mA. La seule condition est que ce soit défini dans les descripteurs. Les périphériques Low et High Power alimentés par un High Power Bus doivent eux aussi être détectés avec une tension s'échelonnant entre 4.75V et 5.25V.

- **Self-powered functions**

Ce type de périphérique à une alimentation mixte, c'est à dire qu'il peut absorber une unité d'énergie sur le Bus USB et le reste est tiré d'une alimentation extérieure. Il faut prévoir alors dans ce cas un peu de réserve car le bus ne délivrera pas plus qu'une unité. Ces types de périphériques « mixtes » sont les plus faciles à concevoir puisque la détection et l'énumération du périphérique peut se faire sans alimentation externe, puisqu'une unité suffit.

## 8. Les Hubs

### a. Définition d'un Hub

Un hub est une sorte de multiprise réseau qui permet de multiplier le signal. La différence entre un hub et un switch est que le hub renvoie les paquets sur tous les ports, alors que le switch identifie le destinataire du paquet et ne l'envoie qu'à lui.

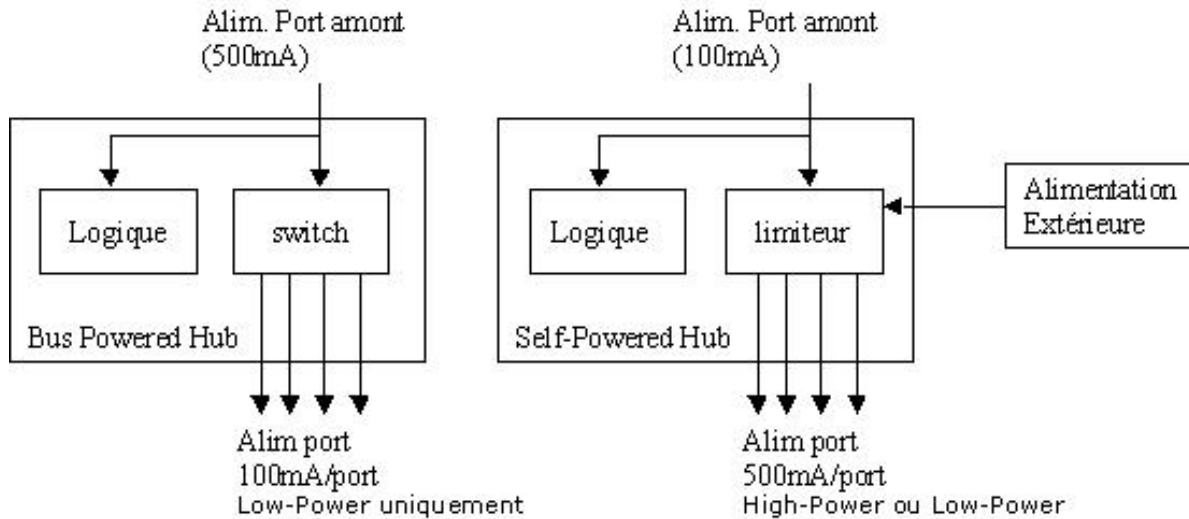
### b. Hub généralités

Chaque Hub doit pouvoir fournir 100mA par port en aval pour pouvoir alimenter d'autres périphériques. Un Hub « self powered » c'est à dire un hub qui est alimenté par l'extérieur doit pouvoir fournir 500mA par port aval. Notons également qu'un hub self powered doit posséder un limiteur de courant sur ses ports aval (5A max.) et signaler les anomalies au host.

On peut aussi dire qu'un périphérique High power doit posséder un dispositif d'alimentation séquentiel (100mA, puis 500mA).

Par ailleurs un bus powered Hub ne peut supporter que des Low Power Devices. Un High power Device ne peut être connecté qu'à un Self-powered Hub.

On ne peut cascader directement 2 Bus powered Hubs.



**Figure 21** : Alimentation des différents HUBS

### c. Différents types d'alimentation du Hub USB

Il existe dans la norme USB différents types de hubs et par conséquent en fonction de leurs utilisations ils doivent s'alimenter différemment. Tout comme les périphériques USB, les Hubs sont aussi rangés suivant différentes classes. Le Hub est alimenté différemment suivant la classe ou il se trouve. Voici les principaux types de Hubs :

- **Root port Hubs**

Avec ce type d'alimentation, les composants sont directement attachés au USB host Controller. C'est le contrôleur USB situé dans le PC qui fournit l'alimentation. Si un périphérique possède une alimentation extérieure, le hub devra fournir au moins cinq unités sur chaque port. Ces ports sont alors appelés des ports High Power. Les ports qui peuvent seulement fournir une unité sont appelés des ports Low Power.

- **Bus-powered Hubs**

Ce type de hubs consomme une unité pendant la configuration et cinq unités lors de son fonctionnement. L'alimentation est divisée, une partie pour le Hub et l'autre pour l'alimentation des ports. Le schéma ci-dessous vous explique les différentes caractéristiques des Bus-powered Hubs.

- **Self-powered Hubs**

L'alimentation du Hub ne provient pas de VBUS, Ce type de Hub doit pouvoir fournir cinq unités de d'alimentation sur chaque port. Le schéma ci-dessous montre bien les différences entre les Hubs les plus utilisés.

## 9. Courant de veille

Pour limiter la consommation un mode suspend a été imaginé : toute fonction y passe après 3ms d'inactivité du bus qui la relie au hub amont (consommation inférieure à

500µA dont 200 réservés à la seule résistance de tirage sur D).

Le courant de veille est proportionnel à l'unité de charge, pour un appareil consommant une unité, le courant de veille est de 500µA, essentiellement dû aux résistances de Pull Up sur le bus.

## **10. Différents types de transfert**

### **a. Transfert en mode Contrôle**

Ce mode de transfert est compatible avec le Low et Full Speed USB. Il est utilisé pour les opérations d'initialisations et de configurations. Il est éventuellement utilisable pour les transferts standard. Le mode contrôle est aussi utilisé pour tenter d'obtenir un débit Low Speed acceptable, ou pour utiliser le driver de classe HID standard.

### **b. Transfert en mode Interrupt**

Ce mode de transfert est également compatible avec le Low et Full Speed USB. Il est destiné à des échanges limités et périodiques, il garantit la fréquence de scrutation ainsi que la reprise sur les erreurs. Il est utilisé pour des transferts à l'initiative du périphérique (asynchrones) et pour des transferts périodiques ou permanents comme les claviers.

### **c. Transfert en mode Isochrone**

Ce mode de transfert est uniquement compatible avec le Full USB. La bande passante est garantie (début, latence), par contre dans ce mode il n'y a pas de reprise sur erreur. Il est utilisé pour des transferts nécessitant un flux régulier de données comme par exemple les caméras ou les téléphones ... La bande passante réclamée et non utilisée est perdue.

### **d. Transfert en mode Bulk**

Ce mode de transfert est uniquement compatible avec le Full USB. Ce mode est réservé pour les gros transferts de données (ex : imprimantes...) Le débit est variable et dépend de la disponibilité. Ce mode assure la reprise sur les erreurs.

Les échanges isochrones sont les plus privilégiés dans le sens où le host leur réserve une bande passante garantie. Celui-ci peut refuser l'accès au bus à un périphérique s'il juge que les ressources qu'il requiert ne sont pas disponibles.

Voici un tableau récapitulatif des caractéristiques des 4 types de transfert :

	<b>Control Transfert autorisé aux périphériques low speed</b>	<b>Isochronous Transfert interdit aux périphériques low speed</b>	<b>Interrupt Transfert autorisé aux périphériques low speed</b>	<b>Bulk Transfert interdit aux périphériques low speed</b>
<b>Mode</b>	Message (format prédéfini)	Stream (format non prédéfini)	Stream (format non prédéfini)	Unidirectionnelle (format non prédéfini)
<b>Direction</b>	Bidirectionnelle	Unidirectionnelle	Unidirectionnelle	Unidirectionnelle
<b>Contrainte sur la taille maximale du bloc de données</b>	Full speed : 8, 16, 32, 64 octets par trame Low speed : 8 octets par trame	1023 octets full speed par trame	Full speed : 64 octets par trame Low speed : 8 octets pas trame	Full speed : 8, 16, 32, 64 octets par trame
<b>Accusé de réception, reprise sur erreur</b>	Oui	Non	Oui	Oui
<b>Bande réservée</b>	10% de la trame « best effort »	90% de la trame « guaranteed »		Non « good effort »

**Tableau 2 : Récapitulatifs des différents types de transfert**

## 11. L'énumération

### a. Définition de l'énumération

Le terme «énumération» désigne un processus USB par lequel le système identifie et configure le périphérique en lui donnant une adresse unique. C'est une gestion dynamique de la connexion et de la déconnexion des périphériques reliés à un bus USB.

### b. Principe de fonctionnement de l'énumération

En effet lors de la connexion (ou déconnexion) il y a une phase de détection et une phase d'identification effectuée par l'hôte qu'on appelle énumération. Lors de cette phase, le périphérique fournit à l'hôte une suite de descripteurs qui permettent son identification complète. Lors de cette phase d'énumération, on assigne une adresse unique (Unique ID) au périphérique, on charge le driver correspondant et on positionne le composant dans la configuration qui lui a été donné par les descripteurs. Il n'est pas indispensable de connaître parfaitement le processus d'énumération et le système de descripteurs pour pouvoir faire fonctionner un composant USB mais il est bon d'en connaître les grandes lignes pour pouvoir, au besoin, changer les descripteurs ou bien par simple culture générale. Notons également pour ceux qui ne le savent pas encore, que cette phase d'énumération est totalement transparente et automatique pour l'utilisateur.

Lors de l'énumération initiale (à la mise sous tension du PC) les HUBs et périphériques sont initialisés de proche en proche.

Ainsi le HUB racine signal que sur ses ports A et B il a des périphériques non initialisés. L'hôte initialise alors une liaison occupée et la place dans sa liste de scrutation, puis passe la liaison suivante. Puis c'est au tour du deuxième HUB donc l'hôte va initialiser successivement les liaisons occupées et va les placer dans sa liste de scrutation. Puis on

passera au HUB suivant s'il il en a un, etc. Jusqu'à ce que tous les périphériques connectés aient été initialisés.

## 12. Les descripteurs

### a. Définition d'un descripteur

On peut définir les descripteurs comme étant des blocs d'informations pré formatés. Tous composants USB doit obligatoirement posséder les descripteurs standards. Tous les transferts d'informations durant cette phase d'énumération se font suivant le type Control. Il va de soi que tout composant USB doit pouvoir être capable de supporter ce type de transfert. Nous verrons par la suite que ce n'est pas le cas pour tous les autres types de transfert que nous définirons.

### b. Rôle des descripteurs

Il existe sur le marché de nombreux périphériques USB. Il a fallu, lors de la création de la norme USB, trouver un dispositif pour reconnaître chaque composant USB. Cela était indispensable puisque l'USB devait être un dispositif plug & play. Lors du branchement du périphérique, le « host » autrement dit plus communément le PC, doit reconnaître tous les périphériques qui lui sont branché. Tout le processus d'énumération se fait grâce aux descripteurs qui sont rassemblés dans un fichier texte (fichier assembleur par exemple): en général, un fichier assembleur, qui est ensuite programmé dans le système USB. Lorsque l'on connecte ou déconnecte un périphérique, celui ci fournit à l'hôte toutes les informations nécessaires à son identification, c'est à dire ces descripteurs. Ils sont très utiles pour l'hôte puisqu'il peut, de ce fait, connaître les caractéristiques périphériques comme par exemple la puissance utile, le type de périphérique, le dispositif de transfert des données, le module de gestion ... etc.

Généralement, dans la plupart des périphériques, toutes ces informations sont stockées dans la ROM des composants, et lors de l'énumération, le périphérique envoie simplement ce fichier pour se faire connaître.

Les descripteurs « standards » sont regroupés en 4 catégories :

- **Device descriptor**
- **Configuration descriptor**
- **Interface descriptor**
- **Endpoint descriptor**

Ces quatre types de descripteurs sont indispensables, il y a juste une petite exception pour l'EndPoint descriptor qui n'est pas nécessaire si on n'utilise que l'EndPoint 0.

Voici une énumération des autres descripteurs possibles :

- Les *device\_qualifier descriptor* qui sont uniquement nécessaires pour les composants USB supportant le Full et le High Speed.
- Les *other\_speed\_configuration descriptor* également nécessaire pour les composants USB supportant le Full et le High Speed.

- Les *string descriptor* si l'on veut stocker du texte comme par exemple le nom du vendeur ou du produit.
- Les *interface\_power descriptor* uniquement compatible avec l'USB 2.0 et permet d'avoir des informations sur la gestion d'énergie.

### c. Identification des différents descripteurs

Chaque descripteur contient une valeur qui identifie le type de descripteur :

Type	Valeur (hexa)	Descriptor
<b>Standard</b>	01	Device
	02	Configuration
	03	String
	04	Interface
	05	Endpoint
	06	Device_qualifier
	07	Other_speed_configuration
	08	Interface_power
<b>Class</b>	21	HID
	29	Hub
<b>Spécifique aux classes HID</b>	22	Report
	23	Physical

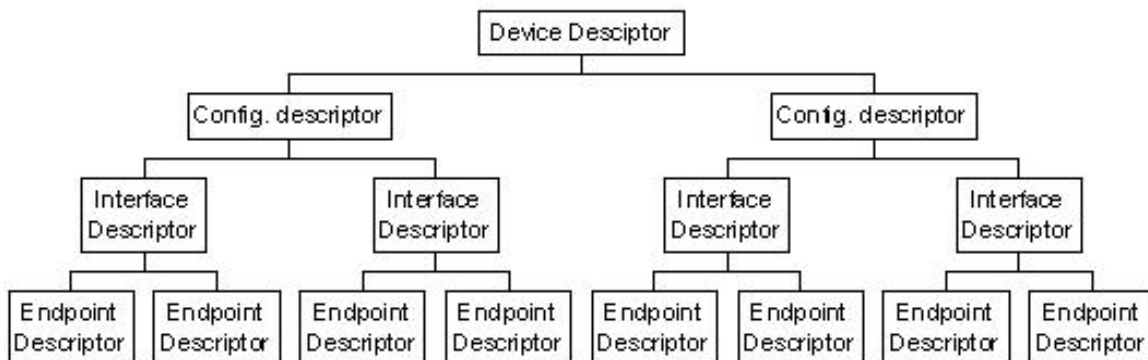
**Tableau 3** : Identification des descripteurs

Il existe deux types d'informations dans les descripteurs :

Les informations standards à tous périphériques USB et les informations spécifiques à chaque périphérique. Nous expliquerons dans la suite le rôle de chaque type de descripteur.

Avant de continuer la lecture, observez bien la hiérarchie des différents descripteurs.

Diagramme hiérarchique des descripteurs



**Figure 22** : Diagramme hiérarchique des descripteurs

Expliquons à présent plus en détails les différentes catégories de descripteurs et leurs rôles.

## d. Différentes catégories des descripteurs

### Première catégorie : Device Descriptor

Ce type de descripteur donne les informations générales. C'est le premier descripteur que vient lire le host. Comme dit précédemment il y a une très grande diversité de composants USB, il y a des propriétés communes à chaque dispositif d'USB comme par exemple le numéro de spécification USB qui est présent dans toutes les configurations, de même pour les numéros d'identifiant de produit et de vendeur. (Product ID et Vendor ID). Un Ordinateur peut, uniquement avec ces informations contenues dans le « Device Descriptors », reconnaître un composant USB. il donne également des renseignements sur le *pipe* de communication par défaut qui est utilisé pour la configuration.

Voici un exemple des différents descripteurs faisant partie de la catégorie « Device Descripteur». Dans le champ description il y a la signification de chaque descripteur, cela peut être très utile si l'on commence à s'intéresser plus longuement sur ce sujet.

Offset	Nom	Taille	Description
0	Blength	1	Taille du descripteur, toujours égale à 18 pour un device descriptor.
1	BdescriptorType	1	Type du descripteur, toujours 01h pour un device descriptor.
2	BcdUSB	2	C'est la version de la spécification USB (codée en DCB) exemple : Version 1.0 -> 0100h Version 1.1 -> 0110h Version 2.0 -> 0200h
4	BDeviceClass	1	Si le composant appartient a une classe, ce champ renseigne sur cette classe. Si il n'appartient à aucune classe le champ doit être mis à 0. L'adresse FFh est réservée et signifie que la classe est spécifique au vendeur
5	BdeviceSubClass	1	Si un composant appartient déjà à une classe, ce champ permettra de restreindre encore une fois en définissant une sous classe. Si bDeviceClass = 0, BdeviceSubClass doit forcément être égal à 0 aussi.
6	BdeviceProtocol	1	Ce champ renseigne sur le protocole choisi par la classe ou la sous classe. C'est dans ce champ qu'un Hub USB 2.0 dira s'il supporte à la fois le Full et le High Speed.
7	BmaxPacketSize0	1	Taille maximum des paquets de l'Endpoint 0. Low Speed : 8 Full Speed : 8 - 16 - 32 - 64 High Speed : 64
8	IdVendor	2	Ce champ renseigne sur le Vendor ID. Un chapitre spécial à été réservé pour expliquer son utilité.
10	IdProduct	2	Ce champ renseigne sur le Product ID. Un chapitre spécial à été réservé pour expliquer son utilité.
12	BcdDevice	2	(Device release number) Codé en BCD, ce champ, optionnel, renseigne sur le driver à charger par

			exemple.
14	IManufacturer	1	C'est un index qui pointe sur un String qui donne le nom du Constructeur. Ce champ est optionnel. 0 si inutilisé.
15	Iproduct	1	C'est un index qui pointe sur un String qui donne des informations sur le produit. Ce champ est optionnel. 0 si inutilisé.
16	ISerialNumber	1	C'est un index qui pointe sur un String qui donne des informations sur le numéro de série du composant. Ce champ est optionnel. 0 si inutilisé. Ce champ peut être intéressant si l'on utilise plusieurs périphériques identique possédant les même PID et PID.
17	BnumConfiguration	1	Nombre de configuration que supporte le composant.

**Tableau 4 :** Tableau récapitulatif des différents champs des Device Descriptor

### Deuxième catégorie : Configuration descriptor

Un descripteur de configuration renseigne sur les différents états dans lequel peut se trouver le composant USB.

Ces descripteurs de configuration définissent par exemple l'origine de l'alimentation, elle peut soit provenir d'une alimentation extérieure, soit directement du bus USB.

Il se peut qu'il y ait deux configurations différentes selon le type d'alimentation. En effet, si le système USB est directement alimenté par le bus USB il se peut en raison des conditions de puissance d'énergie, que le dispositif pourrait invalider quelques paramètres.

La numérotation des configurations commence à 1, la configuration 0 est une configuration réservée. Si le périphérique est dans cette configuration là, on dit qu'il est n'est pas configuré « unconfigured » et ne peut pas communiquer avec la host tant qu'il n'est pas sorti de cet état.

Voici un exemple des différents descripteurs faisant partie de la catégorie « Configuration descriptor »

Offset	Name	Size	Description
0	BLengt	1	Taille du descripteur, toujours égale à 9 pour un configuration descriptor.
1	BDescriptorType	1	Type du descripteur, toujours 02h pour un configuration descriptor.
2	WTotalLength	2	Longueur total en bytes de toutes les données dans cette configuration. Cette longueur inclus les descripteurs d'interfaces, HIDs, reports et les EndPoint.
4	BNumInterfaces	1	Nombre d'interfaces supportée dans cette configuration.
5	BconfigurationValue	1	Identifiant de Set_Configuration et de Get_Configuration.
6	Iconfiguration	1	C'est un index qui pointe sur un String qui décrit cette configuration. Ce champs est optionnel. 0 si

			inutilisé.
7	BmAttributes	1	Précise si le bus comment est alimenté le périphérique. Self Power ou Bus. Précise également si le périphérique supporte le remote wakeup.
8	MaxPower	1	Consommation du composant USB (en mA /2)

**Tableau 5 :** Tableau récapitulatif des différents champs des Configuration Descriptors

### Troisième catégorie : Interface descriptor

Une interface peut être considérée comme un ensemble d'«Endpoint». Ce mot n'a pas de traduction très réaliste mis à part « point final ». Un Endpoint est en quelque sorte l'extrémité d'un « pipe ». Un « pipe » est une sorte de tuyau par lequel transitent les données via le host. Un descripteur d'interface communique une information unique à tous ses Endpoints.

Si une configuration est choisie, toutes ses interfaces sont actives et par conséquent, aucun Endpoint ne peut être relié à des interfaces différentes sous la même configuration. Les interfaces sous différentes configurations peuvent partager des Endpoints.

Voici un exemple des différents descripteurs faisant partie de la catégorie « Interface descriptor»

Offset	Name	Size	Description
0	Blength	1	Taille du descripteur, toujours égale à 9 pour un interface descriptor.
1	BdescriptorType	1	Type du descripteur, toujours 04h pour un interface descriptor.
2	BinterfaceNumber	1	Dans un device, une configuration possède plusieurs interfaces qui peuvent être actives en même temps. Ce champs spécifie pour pouvoir reconnaître les interfaces de la même configuration.
3	BAlternateSetting	1	Cette valeur sélectionne par contre une interface bien précise.
4	BnumEndpoints	1	Nombre d'EndPoint supporté par l'interface. L'EndPoint 0 n'est pas compter. Si c'est le seul supporté BnumEndpoints = 0.
5	BinterfaceClass	1	Si le composant appartient à une classe, ce champs renseigne sur cette classe. L'adresse FFh est réservée et signifie que la classe est spécifique au vendeur La class HID a le code 03h. L'adresse 00h est réservée.
6	BInterfaceSubClass	1	Si un composant appartient déjà à une classe, ce champs permettra de restreindre encore une fois en définissant une sous classe. Si BInterfaceClass = 0, bInterfaceSubClass doit forcément être égal à 0 aussi. Si BinterfaceClass à une adresse entre 01h et FEh, bInterfaceSubClass doit être un code

			définit par la spécification USB. Une valeur FFh signifie que c'est une classe spécifique à un vendeur bien précis.
7	BinterfaceProtocol	<b>1</b>	Ce champs renseigne sur le protocole choisi par la classe ou la sous classe.
8	Interface	<b>1</b>	C'est un index qui pointe sur un String qui décrit cette interface.

**Tableau 6 :** Tableau récapitulatif des différents champs des Interfaces Descriptor

#### Quatrième catégorie : Endpoint descriptor

Un descripteur d'Endpoint indique la direction du transfert ( IN ou OUT), ses types de transfert (ISOCRONOUS, BULK, INTERRUPTION ou CONTROL), ainsi que d'autres informations qui sont regroupé dans le tableau suivant.

En fait, l'ordinateur « le host » communique uniquement avec ces Endpoints. Tous les transferts de paquet de données transitant sur le bus proviennent d'un Endpoint ou sont envoyés à un Endpoint. Généralement les Endpoints correspondent aux Entrées-Sorties ou au registre du dispositif USB.

Le nombre maximum d'Endpoint est différent selon que l'on utilise de l'USB Low Speed ou de l'USB High speed. Un dispositif USB High Speed peut supporter jusqu'à 15 Endpoints tandis qu'un dispositif USB Low Speed ne peut que supporter 3 Endpoints. Nous verrons par la suite les différentes contraintes sur l'utilisation des ces Endpoints pour un dispositif USB Low Speed. Par contre, il est possible que deux Endpoints partagent le même numéro. Un Endpoint transitant les données dans un sens (IN par exemple) et l'autre pour les données transitant dans l'autre sens ( OUT ). Dans ce cas, il est nécessaire de définir deux descripteurs différents.

L'Endpoint 0 est un Endpoint particulier, c'est le seul qui est bi-directionnel et présent dans tous les dispositifs. Il est utilisé par le host pour contrôler le système. Il n'est cependant pas nécessaire de lui affecter un descripteur particulier. L'EndPoint 0 fonctionne toujours en mode *Control*.

Voici un exemple des différents descripteurs faisant partie de la catégorie « Endpoint descriptor»

Offset	Name	Size	Description
0	BLength	1	Taille du descripteur, toujours égale à 7 pour un Endpoint descriptor.
1	BdescriptorType	1	Type du descripteur, toujours 05h pour un EndPoint descriptor.
2	BendpointAddress	1	Renseigne sur le numéro de l'EndPoint et la

			direction.
3	BmAttributes	1	Renseigne sur le type de transport utilisés. 00 = Contrôle    01 = Isochrone 10 = Bulk        11 = Interrupt
4	WMaxPacketSize	2	Spécifie la taille maximale des paquets Bits 0 à 9 : 0 à 1023 Bits 10 et 11 : indique le nombre de transactions supplémentaire par microframe (uniquement pour le High Speed) Bits 12 à 15 : réservé.
6	Binterval	1	Indique le temps maximal (en ms) entre le polling des EndPoints.

**Tableau 7 :** Tableau récapitulatif des différents champs des Endpoint Descriptor

Il existe cependant encore une catégorie de descripteur très utilisée, c'est la catégorie HID ( Human Interface Device ). C'est la catégorie la plus utilisée. En fait, elle regroupe tous les appareils qu'utilisent directement les personnes c'est à dire les souris, les claviers, les Gamepads, les écrans, etc. D'après la spécification HID 1.1, les descripteurs relatifs aux HID sont envoyés après les descripteurs d'interface et avant les descripteurs d'Endpoints.

#### **Autres catégories : HID descripteur**

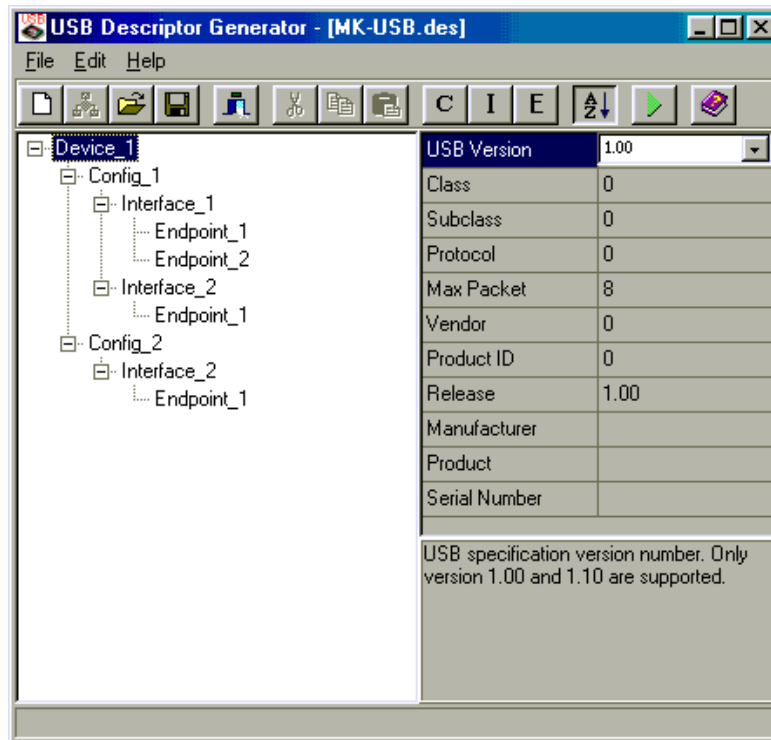
Voici un exemple des différents descripteurs faisant partie de la catégorie « HID descripteur»

Offset	Name	Size	Description
0	BLength	1	La taille varie selon le nombre de descripteur
1	BdescriptorType	1	Type du descripteur, toujours 21h pour un HID descripteur.
2	BcdHID	2	Valeur codée en DCB renseignant le type HID
3	BcountryCode	1	Optionnel, permet de localiser l'interface.
4	BnumDescriptor	1	Nombre de Classe de descripteur.
5	BdescriptorType1	1	Type de descripteur de la première class de descripteur
6	WDescriptorLength1	2	Longueur de la première classe de descripteur.

**Tableau 8:** Tableau récapitulatif des différents champs des HID Descriptors

## e. Logiciels de Point INF

Microchip met à disposition gratuitement un petit logiciel USB Descriptor Generator permettant d'écrire aisément ce genre de descripteur.



**Figure 23 :** USB Descriptor Generator

## IX. PID / VID

### 1. Introduction au Product ID et au Vendor ID

Lors de l'installation d'un nouveau périphérique USB, les descripteurs fournissent les informations le concernant à savoir un PID (Product ID) et un VID (Vendor ID). C'est grâce à ces deux valeurs que le PC peut reconnaître l'identité du composant. Comme dit précédemment la réglementation des VID est très stricte, cher et est délivré par le forum USB-IF accessible depuis le site <http://www.usb.org>. Chaque fabricant possède un VID et c'est grâce à cette valeur codée sur 16 bits que l'on peut retrouver le fabricants du composant. Chaque fabricant ayant plusieurs produits à leurs actifs, ils les différencient avec le PID codé également sur 16bits. L'allocation des PID, contrairement aux VID, est faite par le constructeur du dispositif. Il n'y a aucune contrainte administrative de la part du forum USB-IF.

## 2. Normalisation des PID/VID

Autrement dit chaque couple PID/VID doit être unique sur le marché.

Il existe un site répertoriant la longue liste des VIDs et des constructeurs correspondant. Il est même possible de connaître tout les PIDs des même constructeurs, c'est à dire qu'il est possible de retrouver le modèle exact du périphérique. Voici le site, il pourra vous être utile un jour... <http://www.yourvote.com/pci>. Un petit moteur de recherche vous permettra de trouver rapidement le constructeur en vous réorientant vers son site. Ce site répertorie les PID/VID du bus PCI. En ce qui concerne l'USB il faut s'adresser au forum.

## X. Les fichiers d'extension INF

### 1. Définition

L'abréviation INF provient du mot « Information » et comme son nom l'indique, renseigne sur les informations de configuration de périphérique utile pour le Plug & Play. Les points INF ne sont pas seulement utilisés pour les composants USB, les périphériques PCI l'utilisent également. Un fichier INF est un fichier texte, vous pouvez l'éditer avec n'importe quel éditeur de texte standard.

### 2. Fonctionnement des points INF

En règle générale, les fichiers INF utiles à l'installation sont déjà présents dans les fichiers INF par défaut dans Windows. Les Fichiers INF par défaut sont des fichiers INF génériques c'est à dire qu'il sont capable de débiter l'installation de plusieurs types de périphérique. Nous verrons par la suite comment est structurer un point INF et quels sont les critères qui permettent aux fichiers INF de reconnaître des périphériques.

Un fichier INF est organisé en plusieurs sections. Chaque section (il en existe une vingtaine) possède une fonction particulière. Ces fonctions vont de la simple manipulation de fichiers jusqu'à la modification d'entrées dans la base de registres en passant par les fichiers INI.

### 3. Règles à respecter pour concevoir un fichier INF

Les fichiers INF doivent respecter les règles suivantes pour ne pas générer d'erreur lors de la création de fichier INF :

- Les sections commencent avec un nom de section entouré de crochets.
- Chaque fichier INF doit contenir la section [Version] qui identifie la compatibilité du fichier avec Windows 95 ou Windows NT.
- L'utilisation de variable est possible en utilisant la syntaxe %nom\_de\_la\_variable%.

- Les variables sont définies dans la section [Strings]. Pour utiliser le caractère % dans une chaîne, il faut utiliser la syntaxe suivante : %%
- Les lignes commençant par un point-virgule sont des commentaires.
- Un anti-slash ( \ ) en fin de ligne indique que la suite de la ligne continue à la ligne suivante sauf s'il est entouré de guillemet ( "\ " )
- Les lignes vides sont ignorées.
- Le fichier INF ne peut pas dépasser 64 KO.
- Un seul fichier INF est exécuté à la fois !
- Penser à utiliser les guillemets " ", voire des doubles guillemets, surtout pour les textes avec espaces.
- Le chemin d'accès du fichier INF, ne doit pas être au format Nom Longs de Windows 95..
- Utilisez des majuscules pour les noms de fichiers courts (8+3).
- Le texte est entouré de % (%exemple\_de\_texte%).

#### 4. Applications des fichiers INF

Les fichiers INF ont diverses applications que nous connaissons déjà. Ils servent pour :

- L'installation d'un driver (Détection d'un nouveau périphérique) ou d'un module Windows.
- L'Installation automatique d'un programme (notamment pour installer un programme à l'insu d'un utilisateur).
- La Modification d'un INI ou d'une entrée dans la base de registre lors du script de connexion.

Types de section	Description
Add Registry	Ce type de section permet d'ajouter des entrées dans la base de registres.
ClassInstall32	Ce type de section permet d'installer une nouvelle classe.
Copy Files	Ce type de section permet de copier une sélection de fichiers.
Delete Registry	Ce type de section permet de supprimer des entrées dans la base de registres.
Delete Files	Ce type de section permet de supprimer une sélection de fichiers.
DestinationDirs	Cette section permet de définir le répertoire de destination de chaque sélection de fichiers.
Device	Cette section donne les spécifications pour l'installation d'un périphérique.
EventLog Install	Permet d'ajouter ou de supprimer un message d'évènement dans la base de registre.
Ini File to Registry	Déplace une ligne ou une section d'un fichier INI vers la base de registre.
Install	Cette section permet d'identifier les sections du fichier INF.

Log Config	Cette section permet de définir les paramètres du périphérique (IRQ, DMA,...) à installer.
Manufacturer	Cette section permet d'identifier le constructeur du périphérique à installer.
Rename Files	Ce type de section permet de renommer une sélection de fichiers.
Service Install	Cette section installe les services spécifiées dans la section Service.
Services	Ce type de section permet d'ajouter ou de supprimer un service au système.
Strings	Cette section permet d'initialiser les variables utilisées dans les autres sections.
Update INI Fields	Ce type de section permet de modifier une partie d'une entrée dans une section d'un fichier INI.
Update INI File	Ce type de section permet de modifier une entrée complète dans une section d'un fichier INI.
Version	Cette section est l'entête OBLIGATOIRE dans tous les fichiers INF.

**Tableau 9 : Champs des fichiers INF**

Pour alléger cette étude, j'ai mis en annexe un fichier INF avec les explications de toutes les sections. Cette annexe est très intéressante, je vous conseil vivement d'aller la consulter pour avoir un petit aperçu du monde des fichiers INF.

## 5. Rôle du fichier INF

Lors du branchement d'un périphérique USB, les descripteurs transmettent le couple PID/VID au PC. A ce moment là, le PC scanne dans tous ces répertoires s'il trouve un fichier « d'information » (INF), qui comporte le couple PID/VID qu'il à reçu par les descripteurs. S'il ne trouve pas les fichiers concernés, il ouvrira une fenêtre Windows classique que certains d'entre vous connaissent sûrement en disant qu'il n'a pas réussi à trouver le driver et que vous devez le chercher manuellement. Dans ce cas, soit vous avez ce fichier INF sur CD-ROM ou sur disquette et vous lui indiquez le chemin qu'il copiera, ainsi que d'autres fichiers placés dans ses répertoires de drivers. Soit vous ne posséder pas ce fichier et le périphérique ne s'installera pas. Le fichier INF est le point de départ de toute installation, s'il n'est pas là, il ne peut pas indiquer au système d'exploitation quel driver charger pour faire fonctionner l'application.

Précédemment je parlais de fichier INF standard, en effet dans C:/Windows/INF il y a une série de fichiers INF par défaut qui regroupe un bon nombre de couple PID/VID. De ce fait, si vous installez un périphérique a peu près standard, le PC le reconnaîtra et l'installation débutera. Elle pourra même être transparente pour l'utilisateur si tout le reste de la phase d'installation se passe bien. En ce qui concernent l'USB, Windows 98 possède déjà pas mal de fichiers INF, d'ou l'aspect Plug & Play ... Les versions ultérieures de Windows en possèdent encore davantage. Actuellement si on désire installer un tout nouveau périphérique sous Windows 98, il faudra peut être faire quelques étapes d'installation à la main, tandis qu'avec Windows 2000 ou Windows XP, tout sera transparent.

Ensuite il faut aussi noter qu'il existe des fichiers INF, déjà présent dans le système d'exploitation, qui n'ont pas de couple PID/VID bien précis, mais des plages de valeurs pour les PID et les VID. Cela permet en fait d'utiliser un seul fichier INF pour différents périphériques de même type (donc utilisant le même driver) mais de constructeurs différents. A présent on comprend un peu mieux le caractère un peu strict sur la détermination et l'obtention des VID de la part Forum USB-IF.

Une fois que le fichier INF a été trouvé, il exécute toutes les commandes qui lui sont dictées. Le fichier INF explicatif en annexe est un fichier INF très complet pour pouvoir vous montrer tous les cas possibles. Mais un fichier INF moins complet fait bien l'affaire s'il s'agit juste d'installer un périphérique. Les fichiers d'extension « INF » ont tous une commande qui demande de charger le driver en question. Ce driver a pour extension l'extension SYS. C'est ce fichier qui fait le lien entre le hardware c'est à dire le composant physique et le software, la partie de commande. Cet aspect est détaillé dans un autre chapitre. C'est le fichier INF qui reconnaît le composant et qui ordonne à l'OS d'installer son driver.

## 6. Création de fichier INF

Comme pour les descripteurs il existe quelque logiciels permettant d'« écrire » très facilement des Fichiers INF. Windriver est un logiciel de la société JUNGO ( [www.jungo.com](http://www.jungo.com) ) qui écrit en quelques clics un fichier INF. En fait ce logiciel scanne les descripteurs du périphérique en question et en écrit directement un fichier INF. Nous parlerons de Windriver plus longuement dans un autre chapitre.

Microsoft possède également de logiciels pour créer des fichier INF, Infedit pour Windows 98, Geninf, ChkINF et InfCatReady pour Windows 2000.

Il existe également dans le monde du Web des petits Freeware qui sont des éditeurs assistés pour pouvoir écrire aisément ce genre de fichiers.

## XI. Les drivers .SYS

### 1. Définitions d'un drivers .SYS

Comme dit précédemment, un driver a le plus souvent l'extension SYS. Ce driver doit être fourni avec le périphérique à installer... et c'est le plus souvent le cas. A chaque fois que vous installez chez vous votre imprimante ou scanner ou autre périphériques, il vous faut la première fois un CD d'installation. Comme les .INF il existe aussi des .SYS par défaut.

Par contre, contrairement au fichier INF, il n'est pas si évident de concevoir un WDM driver. (Windows Driver Model). Il nécessite des compétences en programmation C et des connaissances sur la communication entre Windows, le hardware et les applications. Par contre, les connaisseurs disent qu'il est beaucoup plus facile d'écrire un drivers USB qu'un driver ISA.

Dans la nature, un fichier SYS est un fichier compilé et il est impossible de le modifier pour l'adapter à un autre type de périphérique. Donc si vous voulez débiter dans le développement de périphérique USB, assurez vous que vous disposez d'un tel fichier. Sans celui-ci, le composant ne sera pas installé et ne pourra pas communiquer avec le PC. Bien sur, dans la plupart des cas, si le périphérique à installer est standard, un point SYS par défaut dans "C:/Windows/SYSTEM32/Drivers" résout bien les problèmes. Il se peut par contre qu'il ne sera pas optimum mais vous donnera une performance suffisante pour une première tentative.

## 2. Création d'un driver avec le DDK

Il existe une solution pour créer des points SYS, c'est en utilisant le DDK ( Device Developer's Kit), un produit de Microsoft Developer's Network (MSDN). Il faut par contre, pour pouvoir utiliser le DDK, posséder Microsoft Visual C++ qui est capable de compiler des drivers WDM (Windows Driver Model).

Le DDK est en fait une sorte de base de donnée de code et de documentation sur différents types de périphériques. En ce qui concerne l'USB par exemple, le DDK inclut des fichiers, qui en les compilant, génère un fichier bulkusb.sys. Ce driver est un driver qui est capable de piloter un composant USB avec le type de transfert BULK. Un autre par exemple, isousb.sys qui permet de piloter de l'USB de type ISOCHRONE. Le DDK est téléchargeable sur le site de Microsoft. Pour les intéressés de la programmation de Driver WDM je vous conseille des ouvrages qui traitent cela avec plus de détails :

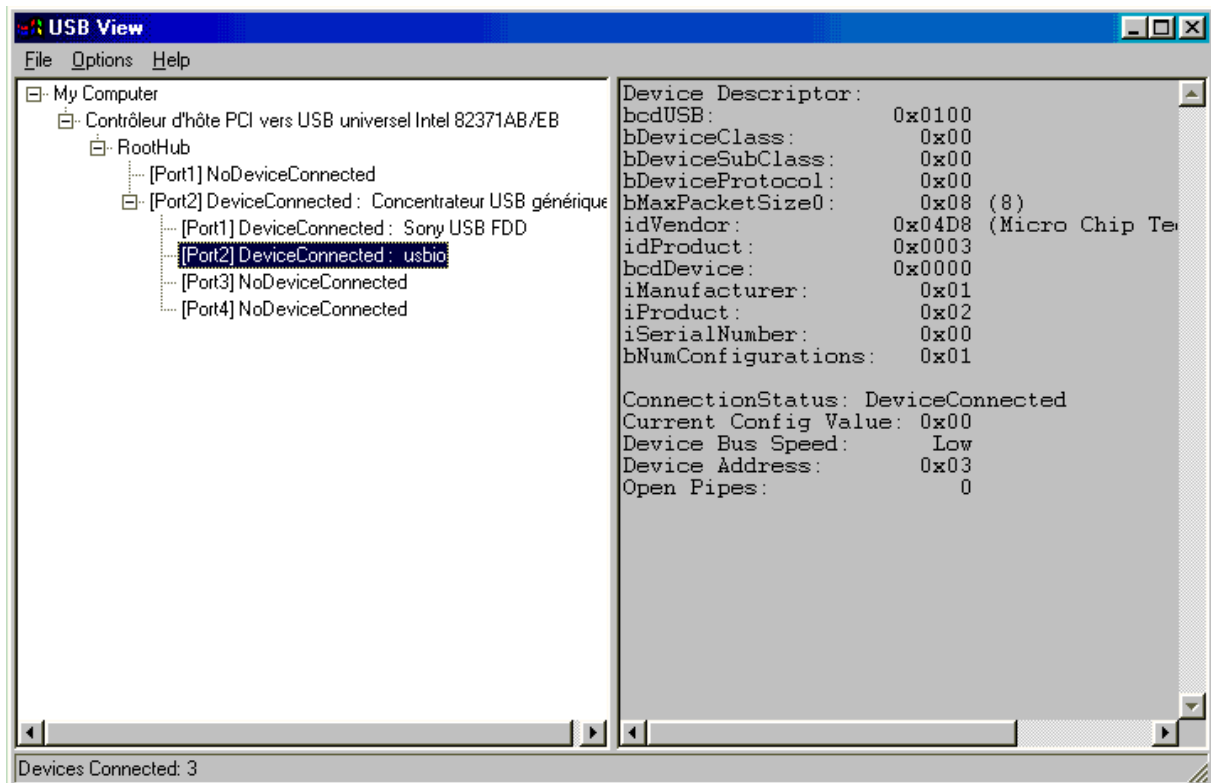
- Programming the Microsoft Windows Driver Model de Walter Oney
- Writing Windows WDM Device Drivers de Chris Cant
- Developing Windows NT Device Drivers d'Edward N. Dekker et Joseph M. Newcommer

## XII. Arborescence des périphériques USB

Windows DDK inclus les codes sources pour l'USBView, c'est une petite application qui permet de scanner le bus USB. Elle répertorie tout les HUBs ainsi que tout les périphériques USB qui sont banchés et donne également les descripteurs de chacun d'eux.

Les logiciels de développement tel que WinDriver ou WinRT fournissent également un aperçu du bus USB, mais il faut noter que ces logiciels sont des logiciels payant. La petite application USBView fait très bien l'affaire pour comprendre la logique de l'arborescence USB.

Voici ci dessous un aperçu de celle ci :



**Figure 24 :** L'arborescence USB avec l'USBView

## XIII. Kit de développement de driver

### 1. Enumération des différents types existant

Il existe sur le marché quelques kits de développement pour faire des drivers USB.

**BSQUARE – WinRT**  
**JUNGO Ltd. – Windriver**  
**Compuware NuMega - DriverStudio**

Ces trois assistants de drivers se divisent en deux catégories. La première consiste à utiliser un driver générique qui est capable de communiquer avec tout les types de pipes. Le fichier INF est également crée par ce type d'assistant.

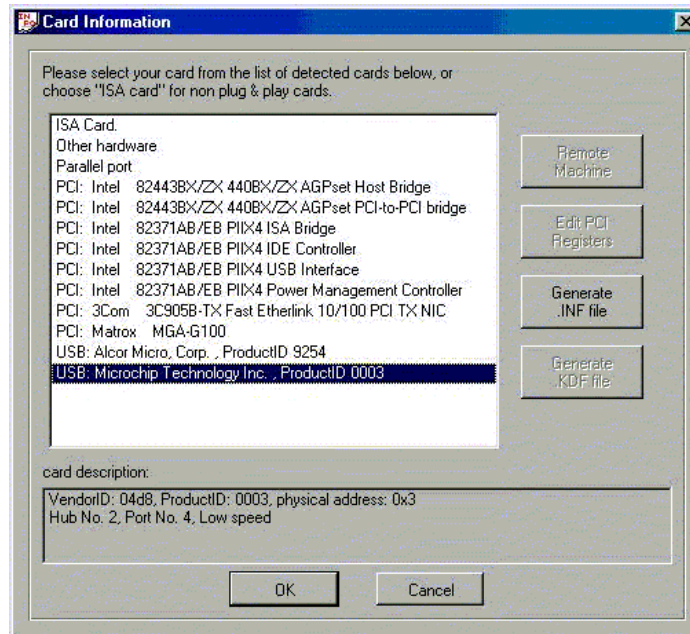
Cette approche est relativement rapide et ne nécessite pas de programmation mais ne s'utilise pas dans tout les cas. La deuxième catégorie fournit des bibliothèques et d'autres outils qui assistent pour l'écriture du code d'un driver spécifique. Cette approche est plus flexible mais nécessite la connaissance de la programmation de driver.

Les deux assistants rejoignant la première catégorie sont Windriver et WinRT, celui que j'ai utilisé est Windriver de la société Jungo. Il est très facile à utiliser, il permet en quelques clics d'avoir un petit exécutable permettant d'ouvrir et fermer les différents pipes et ainsi communiquer avec le périphérique USB.

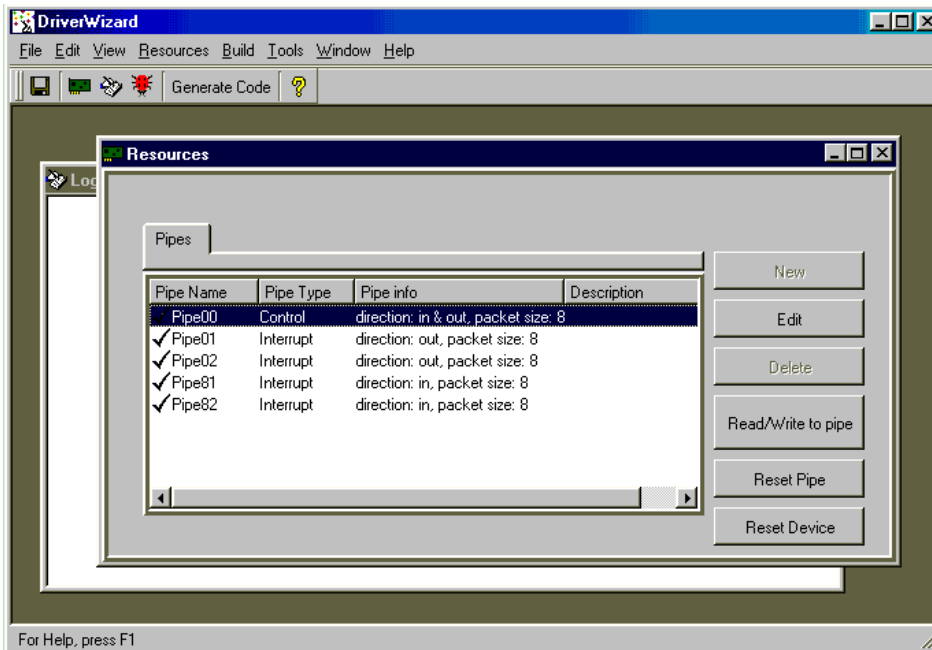
Les versions d'évaluation de ces deux logiciels ne sont que valide pendant un mois.

## 2. Aperçu des assistants de création de drivers

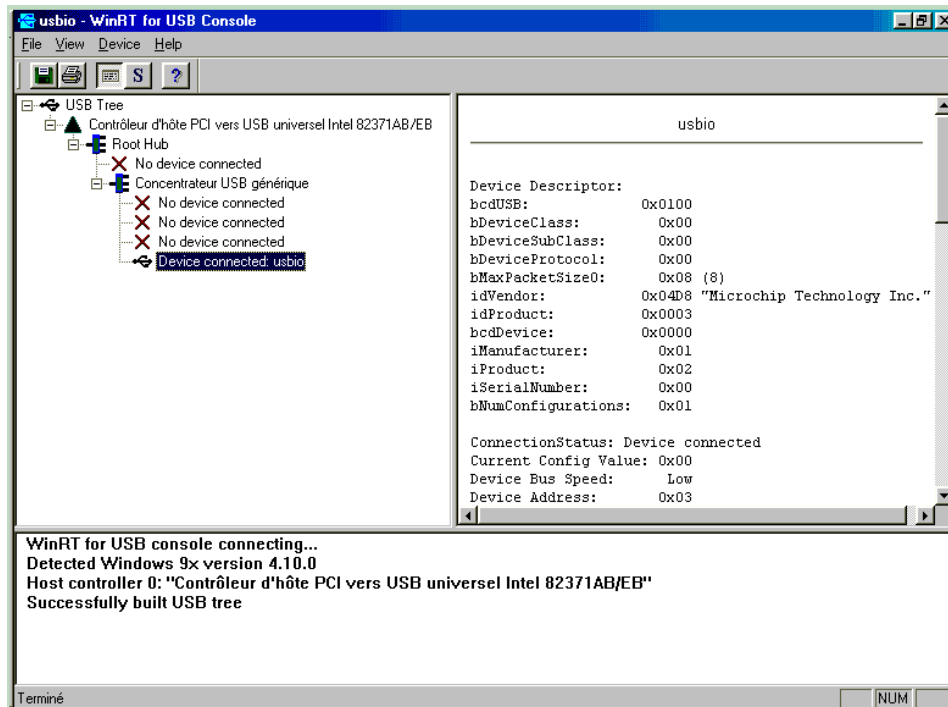
Voici ci-dessous un aperçu des deux logiciels les plus faciles à utiliser.



**Figure 25 :** Windriver - création du point INF



**Figure 26 :** Windriver - Communication avec les pipes



**Figure 27 :** WinRT avec l'arborescence USB

## **XIV. Quelques conseils pour installer un périphérique USB**

### **1. Installation d'un nouveau périphérique**

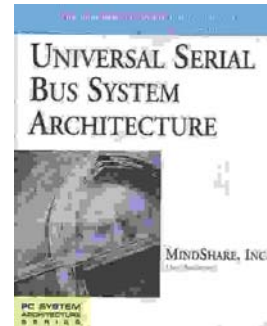
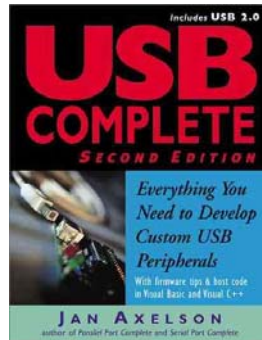
L'installation d'un nouveau périphérique se fait, avec les nouveaux OS, de manière totalement automatique. Puisque les nouveaux OS contiennent les données des périphériques actuels du marché. L'installation est donc transparente. Par contre s'il s'agit d'un OS plus ancien comme Windows98 par exemple, ou bien s'il s'agit d'un périphérique qui n'est pas reconnu par le PC, il faut procéder à l'installation manuel, Il faut pour cela démarrer l'assistant d'installation de nouveau matériel et indiqué manuellement au PC ou ce trouvent les drivers.

### **2. Vérifier qu'un périphérique est bien installé**

Pour vérifier qu'un périphérique USB est bien installé, il suffit d'aller dans les propriétés systèmes de votre Windows et de regarder s'il s'affiche correctement dans la liste. S'il y a un point d'exclamation jaune devant le nom du périphérique USB c'est qu'il est détecté physiquement mais que l'installation du driver a échoué. Il faut le réinstaller, pour cela cliquer sur Propriété -> puis l'onglet Pilote -> puis mettre à jour le Pilote. Si vous avez une croix rouge devant le nom du périphérique, c'est qu'il y a eu un problème lors de l'installation du driver, un gros conflit avec le ou les drivers et le hard, le bus USB ne fonctionnera plus. Pour résoudre ce problème il faut redémarrer l'ordinateur, l'installation du driver reprendra automatiquement. Si toutefois cela n'arrange pas les choses, sélectionnez le composant et supprimez-le. Puis Actualiser, l'OS va re-scanner à nouveau tout le bus USB et résoudra vos problèmes.

### **Annexe 1 : Bibliographie**

Voici les trois livres qui m'ont servi à apprendre et à comprendre le fonctionnement du bus USB



Comme support papier il y avait bien entendu également la norme USB en elle-même.

Puis quelques sites Internet qui évoquaient aussi l'existence de ce bus :

<http://www.usb.org>

<http://www.usb-by-example.com/>

<http://www.beyondlogic.org/usbnutshell/usb1.htm>

<http://www.usbman.com>

[http://membres.lycos.fr/michelhubin/CONSEILS/pc\\_52.htm](http://membres.lycos.fr/michelhubin/CONSEILS/pc_52.htm)

[http://www.pulsewan.com/data101/usb\\_basics.htm](http://www.pulsewan.com/data101/usb_basics.htm)

**Annexe 2 : Fichier INF type**

; Documentation sur les fichiers INFs

```
; *****
; ***** Déclaration d'un fichier inf. *****
; *****
```

[version]

```
signature="$CHICAGO$" ; Ceci est donc un fichier INF!
; La clé signature indique pour quel genre d'OS est destiné le .INF
; Pour les périphériques qui utilise les drivers WDM, cette clé peut être $Windows98$,
; $WindowsNT$ ou $Chicago$. Chicago est en fait un nom « standard » qui à été
; introduit pendant le développement de Windows 95 et à été repris par la suite avec les
; autres version de Windows.
```

; Les lignes commençant par un point-virgule sont des commentaires.

; Les lignes vides sont ignorées.

; Le fichier INF ne peut pas dépasser 64 KO

; Un seul fichier INF est exécuté à la fois !

; Penser à utiliser les guillemets " ", voire des doubles guillemets, surtout pour les textes avec espaces.

; Le chemin d'accès du fichier Inf, ne doit pas être au format Nom Longs de Windows 95...

; Utilisez des majuscules pour les noms de fichiers courts ( 8+3 )

```
; *****
; ***** Sections "Titres" que le fichier Inf va exécuter *****
; *****
```

; Le lancement des fichiers INF, se fait par l'appel d'une section qui définit l'ensemble des actions à réaliser, via l'appel des autres sections, qui font le boulot.

[BaseWinOptions] ; Section à installer lors de l'installation de Windows.

[Install.Section] ; Section à Installer par programme ou appel:

; ex: "RunDll setupx.dll,InstallHinfSection Install.Section 4 Chemin\Didacticiel.inf"

[DefaultInstall] ; Section à installer par défaut à partir du menu contextuel.

; Ces sections appelleront d'autres sections qui feront le boulot...

CopyFiles = Demo.Copy ; Copier des fichiers

DelFiles = ; Effacer des Fichiers

RenFiles = ; Renommer des fichiers

AddReg = Demo1.Reg, Demo2.Reg ; Ajouter des clés, variables, ou valeurs dans la base de registre.

DelReg = Demo.Efface ; Supprimer des clés, variables, ou valeurs ...

UpdateInis = ; Ajouter, modifier, supprimer des entrées d'un fichier INI

```
; *****
; *** Comment copier, effacer, renommer des fichiers ***
; *****
```

; Dans un 1er temps, déclarer les sections, chacune concernant un ou plusieurs fichiers

; à copier vers la même destination...

[Install.Section]

CopyFile = Demo.Copy, Demo2.Copy ; Appel des sections de copie de fichiers

DelFiles = Demo.Delete ; Appel de la section effacer des fichiers

RenFiles = Demo.Rename ; Appel de la section renommer des fichiers

CopyFiles = @Fichier.txt ; Copier le seul fichier désigné

; Ensuite, définir la destination des fichiers de chaque section

[DestinationDirs]

Demo.Copy2 = "C:\TEMP" ; Chemin d'accès de la destination en clair

```

Demo.Delete = 11,shellxt           ; 11 pour %11% ou Windows\System, et sous-répertoire
\SHELLEXT\                          ; ou chemin des fichiers pour effacer ou renommer
DefaultDestDir = 30,Temp           ; Chemin par défaut pour le reste des sections, et les copies
directes (@file.ext)              ; S'il n'est pas défini, ce sera 10 pour Windows\

; Il faut aussi définir la (les) source(s) de la copie
[SourceDisksNames]
55="Nom quelquonque", "", 1        ; [N° disque / disquette] = [Description] ,
                                   ; [Nom disquette] , [n° de série disquette]

[SourceDisksFiles]
Winrar.exe=55                      ; [Nom du fichier] = [n° disque ci-dessus]

[Demo.Delete]
file1                               ; Nom du fichier à effacer

[Demo.Rename]
file41,file42,fileInterm, Flag     ; Nouveau nom du fichier, Ancien nom, Nom intermédiaire,
Flag

[Demo.Copy]                         ; Section liste de fichiers
fichier.ext                         ; Nom de destination
fichier2.ext, file2.ext             ; Nom du fichier source facultatif, s'il diffère du nom de
destination
fichier3.ext, file3.ext , File.$$$ ,flag ; Nom temporaire facultatif, sera rechangé lors du
prochain redémarrage
                                   ; Utile pour remplacer des fichiers ouverts par Windows

; Flags:
; 1 ?
; 2 ? A voir avec DOS?
; 4 Force la copie quel que soit le conflit de version.
; 8 Force le renommage si le fichier est ouvert, pour copier quand même. (redémarrage)
; 16 Interdirait d'écraser un fichier même plus ancien.
; 32 Supprime le dialogue, si conflit de version, mais n'écrase pas un fichier plus
récent.
; Combinaisons: 9, 12, 33, 40

; *****
; *** Entrer des données dans la base de registre ***
; *****
[Demo1.Reg]
; Valeurs de type Alphanumérique ( par défaut )
HKCR,_XXX,"Valeur 1",,"Alpha123x445" ; Alphanumerique par défaut
HKCR,_XXX,"Valeur 2",0,"Alpha123455x" ; 0 ou rien pour écraser la valeur précédente
HKCR,_XXX,"Valeur 3",2,"Alpha123555x" ; 2 préserve une valeur existante

; Valeur de type Binaire
HKCR,_XXX,"Valeur binaire 1",1,00,00,00,04
HKCR,_XXX,"Valeur binaire 2",3,00,00,00,00,A2,FA,D4

; Valeur de type DWord
HKCR,_XXX,"Valeur Dword",65537,134 ; Valeur écrite en décimal

; Valeur par défaut d'une clé
HKCR,_XXX,,,"Valeur par défaut"
; NB: Il faut mettre en guillemets s'il y a des espaces.

; *****
; *** Supprimer des entrées de la base de registre ***
; *****
[demo.efface]
HKCR,_YYY                          ; Supprime la clé et ses sous-clés

```

```

; *****
; *** Travailler avec les fichiers INI ***
; *****
[update-ini.Section]
; Fichier INI, Section, [ancienne entrée], [nouvelle entrée], [flags]
%11%\Win.ini, Section1,, Valeur1=2 ; Ajouter une entrée
%11%\Win.ini, Section2, Valeur3=*, ; Effacer une entrée
%11%\Win.ini, Section4, Valeur=1, %valeur% ; Remplacer une entrée

```

```

; *****
; *** Utilisation de variables ***
; *****

```

; Certaines variables peuvent être définies dans la section [Strings], et être utilisées avec leur nom entre %.

; Le but sera soit de rendre plus lisible, soit de faciliter une relocalisation.

```
[Strings]
```

```
Nom4 = "Texte du nom"
```

```
Dossier = "sous_répertoire\etc"
```

```
[Demo2.Reg]
```

```
HKCR,_XXX,%30%\%Dossier%,%Nom4%
```

; Un certain nombre de variables ( LDID ), sont prédéfinies, et représentent des dossiers système, ; quelque soit leur position effective ( ça marchera aussi bien si on a installé Windows en E:\Win98\ ).

```

;
; %01% Répertoire courant
; %10% Windows
; %11% Windows\System
; %13% Windows\Command (DOS)
; %17% Windows\INF
; %18% Windows\Help
; %20% Windows\Font
; %24% Applications
; %30% Racine du lecteur de boot
; %34% Ancien répertoire DOS s'il existe.
; %28700% Program Files

```

```

; *****
; *** Comment créer un répertoire ***
; *****

```

```
[update-ini.Section]
```

```
setup.ini, progman.groups,, "group1=""Programmes\Démarrage"" ; "NomUnique = ""Chemin\répertoire""
```

; par défaut de chemin, c'est dans le menu

démarrer

```
setup.ini, progman.groups,, "group2=""%10%\Bureau\dossier "" ; Sinon, préciser le chemin complet
```

; Il semble qu'il faille y créer un raccourci pour que ça marche

```

; *****
; *** Comment ajouter un Raccourci ***
; *****

```

```
[update-ini.Section]
```

```
setup.ini, progman.groups,, "group4=""Programmes\Démarrage""
```

```
setup.ini, group4,, " ""Texte du raccourci"", ""Chemin\Prog.Exe ou .pif"", ""Fichier icone"", n° d'icone,, ""Chemin démarrer en"" "
```

; NB on devrait aussi pouvoir le supprimer:

```
setup.ini, group4,, " ""Texte du raccourci"" "
```

```

; *****
; *** Comment lancer un exécutable ? ***
; *****
HKLM,Software\Microsoft\Windows\CurrentVersion\RunOnce,"Texte à
Afficher",,"%10%\Programme.exe"
; NB: Le nom de la clé est libre, unique, dans ce fichier INF, et la valeur alphanumérique est
exécutée

; *****
; *** Comment renommer fichiers et répertoires ***
; *****
[rename.reg]
HKLM,"Software\Microsoft\Windows\CurrentVersion\RenameFiles\Nom_bidon",,"%25% ;
Répertoire amont
HKLM,"Software\Microsoft\Windows\CurrentVersion\RenameFiles\Nom_bidon",INF,,"INF2"
; Attributs du répertoire: R=1; H=2; S=4; HS=6; SHR=7
HKLM,"Software\Microsoft\Windows\CurrentVersion\RenameFiles\Nom_bidon2",,"%25%
HKLM,"Software\Microsoft\Windows\CurrentVersion\RenameFiles\Nom_bidon2","NOTEPAD.EXE",,"
NOTEOLD.EXE,2"

; Utiliser les majuscules pour que le système reconnaisse un nom court

; *****
; *** Désinstallation automatisée ***
; *****
; Elle se fait par une entrée de la base de registre, qui lancera le programme chargé de la
désinstallation
; Ce programme de désinstallation peut aussi être un fichier INF, dont on appelle une section titre
; qu'on peut nommer [Default.Uninstall] , et qui exécute les opérations de désinstallation

; Créer l'entrée de désinstallation: [DefaultInstall]
; AddReg = Uninst.Reg
[Uninst.Reg]
HKLM,SoftWare\Microsoft\Windows\CurrentVersion\Uninstall\"Nom unique",,,,
HKLM,SoftWare\Microsoft\Windows\CurrentVersion\Uninstall\"Nom unique",,"DisplayName",,"Texte
descriptif"
HKLM,SoftWare\Microsoft\Windows\CurrentVersion\Uninstall\"Nom
unique",,"UninstallString",,"rundll setupx.dll,InstallHinfSection Default.Uninstall 4
%Chemin%\Didacticiel.inf"

; Créer la section de désinstallation:
[Default.Uninstall]
DelReg = Prog.Reg, Uninst.Reg ; Penser à supprimer l'entrée ci-dessus !
DelFiles = Prog.Files

; *****
; *** Afficher un titre général ***
; *****
; l'astuce consiste à utiliser RunOnce\Setup, qui va afficher un texte

[Titre.Reg]
HKLM,Software\Microsoft\Windows\CurrentVersion\RunOnce\setup," Titre de mon Fichier
INF",0,""
HKLM,Software\Microsoft\Windows\CurrentVersion\RunOnce\setup,"
",0,""
HKLM,Software\Microsoft\Windows\CurrentVersion\RunOnce\setup," ",0,""
HKLM,Software\Microsoft\Windows\CurrentVersion\RunOnce\setup," ",0,""
HKLM,Software\Microsoft\Windows\CurrentVersion\RunOnce\setup,"Lancement de
l'installation...",0,"RunDll32.exe advpack.dll,LaunchINFSection Didacticiel.inf, Install.Section"

[Install.Section]
; Section réelle d'installation

```

```
; *****  
;  
; *** Question: Poursuivre Oui/Non ? ***  
; *****  
; Uniquement dans une section appelé indirectement (pas DefaultInstall)  
; c'est donc une fonction à associer avec la précédente !
```

```
[Install.Section]
```

```
BeginPrompt = Question
```

```
[Question]
```

```
Prompt="Vous allez tester l'installation! Poursuivre ?"
```

```
ButtonType=YESNO
```

```
Title="Explication d'un .INF"
```

```
; *****  
;  
; *** Notes diverses ***  
; *****  
; InstallType = 14 ; 1 = compact, 2 = typical, 4 = portable, 8 = custom  
; Reboot = 1 ou 2 ; force un redémarrage du système...
```