V

GS /OS

5.0 Introduction à GS/OS & Prodos 8

Nous allons étudier dans cette partie deux systèmes d'exploitation fonctionnant sur Apple II GS : GS/OS (GS Operating System), le système d'exploitation 16 bits de l'Apple II GS, et le ProDos 8 (Professional Disk Operating System), le système d'exploitation 8 bits de l'Apple II et de l'Apple II GS.

GS/OS est le premier système d'exploitation qui exploite l'Apple II GS doté de ROM 01 ou de ROM supérieure. Il ne fonctionne sur aucun autre modèle de la famille Apple que l'Apple II GS. GS/OS exploite les caractéristiques les plus avancées du microprocesseur 16 bits 65C816 du II GS. C'est le système d'exploitation qui a remplacé ProDos 16, un système d'exploitation d'attente qu'Apple a fourni à l'introduction sur le marché de l'Apple II GS de Septembre 1986 à Septembre 1988. Pour des raisons de compatibilité, GS/OS reconnait toutes les commandes de ProDos 16, afin que les applications écrites pour fonctionner sous ProDos 16 continuent de fonctionner correctement sous GS/OS. ProDos 8 fonctionne sur les ordinateurs APPLE II+, IIe, et IIc. Il fonctionne aussi sur Apple II GS quand celui-ci est en mode émulation; vous pouvez passer du GS/OS au ProDos 8 si GS/OS est le système d'exploitation utilisé pour booter le GS. ProDos 8 est un système d'exploitation écrit en 8 bits fonctionnant avec les microprocesseurs 6502 et 65C02. La plupart des commandes ProDos 8 ont un équivalent sous GS/OS, mais la méthode pour mettre en oeuvre la commande est différente, du moins pour les programmes en langage machine.

GS/OS et ProDos 8, comme tous les autres systèmes d'exploitation gèrent les données allant et provenant d'un support magnétique

comme par exemple une disquette 5.25, une disquette 3.50 ou un disque dur. Ils font ce travail en traduisant des commandes disque de haut niveau utilisées par un programme d'application en instructions de bas niveau nécessaires pour communiquer directement avec le contrôleur de disque. Le système d'exploitation utilise aussi des groupes de données structurées appelés fichiers.

GS/OS et ProDos 8 fonctionnent parfaitement avec tous les périphériques disque qu'Apple a vendu pour la famille Apple II : le Drive 5.25 (et ses prédecesseurs), le disque dur HD20SC, l'UniDisk 3.5, le lecteur Apple 3.5, la carte d'extension mémoire Apple II (utilisée en RamDisque), et le lecteur APPLE CD SC CD-ROM.

5.0.1 Les systèmes d'exploitation sur Apple II - Une longue histoire

Quand Apple s'est lancé dans l'aventure en 1977, le magnétophone et ses cassettes était le seul "périphérique" pour stocker "en masse" les données des utilisateurs. La raison en est simple : le premier Apple II disposait d'un port cassette, ce qui facilitait la connection d'un magnétophone à cassette jusqu'à l'invention du lecteur de disquettes et de sa carte contrôleur.

Travailler avec une cassette pour stocker ses données n'était pas de tout repos. Les temps de chargement étaient très lents, et l'utilisateur n'était jamais certain de retrouver ses précieuses données. Pour rendre l'Apple II de ces temps héroiques aussi pratique qu'un ordinateur compatible d'aujourd'hui, on ne pouvait pas donner de nom aux fichiers et pour le chargement d'un fichier en particulier, il fallait au préalable positionner méticuleusement la cassette à l'endroit stratégique du début de l'enregistrement du dit fichier.

Steve Wozniak, l'inventeur de l'Apple II fut bien évidemment le premier que cette situation rendait extrèmement nerveux. Au cours de l'hiver 1977-1978, il a conçu une carte d'extension capable d'interfacer un Apple II et un lecteur de disquettes qu'on a appelé plus tard le Disk II. Au même moment, Bob Shepardson, et plus tard Randy Wigginton, Dick Huston, et Rick Auricchio, étaient en train d'écrire un système d'exploitation qui permettrait aux programmeurs de créer, organiser, et accéder à des fichiers sur les disques souples 5.25 que ce lecteur (le Disk II) allait utiliser.

Apple a en fait livré le Disk II, sa carte contrôleur, et la première version du système d'exploitation (DOS 3.1) au tout début de l'été 1978. Ce lecteur, le Disk II a été bien plus tard renommé Unidisk, puis lecteur Apple 5.25. Ce fut probablement l'évènement le plus important de la courte histoire d'Apple, car cela signifiait que pour la première fois des logiciels professionnels pouvaient être écrits pour

un ordinateur de type Apple II. De tels logiciels nécessitent de créer et de manipuler des fichiers de données importants rapidement et facilement, une prouesse qu'il était impossible de réaliser avec une cassette et un magnétophone.

Plusieurs modifications ont été apportées au DOS 3.1 dans les mois qui ont suivi sa sortie pour corriger les inévitables bugs qui frétillaient dans tout le système d'exploitation. Le DOS s'est finalement stabilisé à la version 3.2.1 au milieu de l'année 1979. Cette première version du DOS formattaient des disquettes en 35 pistes (l'utilisation d'une 36ème piste est une particularité du Disk II qui n'a séduit que quelques partisans avides de succès ...), chacune contenant 13 secteurs de 256 octets (soit donc un total de 133.75 Ko, où1 Ko = 1024 octets). En fait, le programme situé sur la Rom de cette carte contrôleur ne pouvait booter que les disquettes utilisant ce format spécifique en 13 secteurs.

C'est également en 1979 qu'Apple a sorti son système d'exploitation Pascal. Ce système gère les fichiers d'une manière tout à fait différente des systèmes DOS 3.x ou plus tard ProDos. Pour transférer des fichiers texte Pascal sur une disquette DOS (ou inversement), vous pouvez utiliser des programmes utilitaires disponibles dans le commerce ou auprès des clubs d'utilisateurs.

Apple a mis à jour le DOS 3.2.1 en 1980 afin qu'il puisse gérer le nouveau système de formatage en 16 secteurs (ce n'est pas Apple qui a inventé le 18 secteurs ...) utilisé par son système d'exploitation Pascal. Le résultat fut le DOS 3.3, une version du DOS encore en cours quand Apple a sorti ProDos 8 au tout début de 1984. Le formatage étant différent, cela a nécessité un changement de ROM sur la carte contrôleur. Le principal avantage de ce changement de formatage fut de pouvoir utiliser 26.25 Ko supplémentaire sur le disque 5.25 (pour un total de 140 Ko : ceux des lecteurs qui n'ont pas saisis peuvent immédiatement retourner à la lecture des manuels Apple ou s'adresser à E. S. chez Apple France). L'inconvénient de taille était que le DOS 3.3 ne pouvait pas directement lire les fichiers situés sur une disquette formatée en DOS 3.2.1; de plus, il n'était pas possible de booter les disques formatés en DOS 3.2.1 sur une carte contrôleur équipée de cette nouvelle Rom. Heureusement, Apple a crée un programme appelé MUFFIN pour transférer des fichiers de l'ancien format (13 secteurs) au nouveau format (16 secteurs). Il est intéressant de constater que c'est à partir de MUFFIN que les pirates de l'époque ont puisé nombre de connaissances pour déplomber les disquettes utilisant une protection altérant le formatage. Apple a créer un deuxième programme BOOT13 afin que l'utilisateur puisse booter des disques formatés en DOS 3.2.1 avec le nouveau contrôleur de disques.

Apple a sorti ProDos 8 que l'on appelle maintenant ProDos en janvier 1984. Il fonctionne sur tous les Apple IIe, Apple IIc, ou Apple II GS ou sur un Apple II Plus muni d'une carte d'extension mémoire de 16 Ko en slot 0. Il fonctionne aussi sur l'Apple II de l'age de pierre muni d'une carte d'extension Ram de 16 Ko, seulement si le Basic Applesoft, et non pas le BASIC Integer est présent en ROM. A la sortie de ProDos 8, Apple a précisé qu'il ne sortirait plus de logiciels utilisant le DOS 3.3; Apple a recommandé aux développeurs indépendants d'en faire de même. 6 ans après, certains éditeurs de logiciels et constructeurs de disques durs ne connaissent toujours pas ProDos ...

Une carte contrôleur compatible ProDos 8 pour le disque dur ProFile qu'Apple avait réalisé deux ans auparavant pour être utilisé sur Apple III a été mise sur le marché en Janvier 1984. ProDos 8 est capable de gérer des volumes d'une taille allant jusqu'à 32 Méga-octets. Ce n'est que plus tard qu'Apple a remplacé le ProFile par le disque dur HD20SC d'une capacité de 20 Méga-octets, un périphérique SCSI. On le connecte à l'ordinateur au moyen d'une carte d'interface SCSI.

En Septembre 1985, le lecteur UniDisk 3.5 a fait sa première apparition. Il utilise des disquettes au format 3.50 pouces enveloppées d'une protection en plastique; leur capacité de stockage est de 800 Ko. ProDos 8 reconnait automatiquement la carte contrôleur de ce type de lecteur de disquettes, aucun Driver n'est à installer. C'est à cette époque qu'Apple a commencé à fabriquer les Apple IIc avec un contrôleur de lecteur 3.50 intégré. Apple a aussi développé à la même époque une carte d'extension mémoire que ProDos 8 reconnait en Ram Disque au moment du boot.

Apple a annoncé l'Apple II GS en Septembre 1986. A ce moment, Apple a renommé ProDos par ProDos 8, et a réalisé ProDos 16 un système d'exploitation spécifique à l'Apple II GS. Bien que ProDos 16 formate les disques et stocke les fichiers de la même manière que ProDos 8 (ce qui signifie que les deux systèmes d'exploitation peuvent co-exister sur le même disque), les deux systèmes sont tout à fait incompatibles au niveau de la programmation. Apple a réalisé le ProDos 16 pour exploiter pleinement la capacité d'adressage du 65C816; ProDos 8 travaille seulement avec une capacité mémoire de 64 Ko. En même temps que l'Apple II GS, le lecteur Apple 3.5, un autre lecteur utilisant des disquettes 3.50 pouces a fait son apparition. La différence entre Apple 3.5 et l'UniDisk 3.5 est que le premier n'a pas de processeur intelligent intégré, et que par conséquent il ne fonctionne que sur l'Apple II GS.

Une autre version de l'Apple IIc est sortie en Septembre 1986. Il comprend un connecteur sur lequel on peut enficher une carte d'extension mémoire. ProDos 8 reconnait cette carte en RamDisque.

En Septembre 1988, Apple a réalisé GS/OS, un nouveau système d'exploitation pour le II GS destiné à remplacer ProDos 16. GS/OS est bien entendu compatible avec le ProDos 16. Il comprend des nouvelles commandes qui sont plus puissantes que celles de ProDos 16. Une caractéristique très importante est que GS/OS peut accéder à des disques formatés sous ProDos et à des disques formatés avec un autre système d'exploitation comme par exemple le High Sierra (pour les CD Roms), le HFS (utilisé par le Macintosh), ou le MS-DOS. Pour avoir accès à un autre système d'exploitation, il suffit de placer un module FST (File System Translator) sur le disque système GS/OS. Pour le moment, Apple a fourni les modules FST pour les systèmes ProDos et High Sierra.

Un autre modèle d'Apple IIc, l'Apple IIc Plus a également vu le jour en Septembre 1988. Il incorpore un lecteur de disquettes 3.5 qui fonctionne avec ProDos 8.

Les premières versions de ProDos avaient des bugs mineurs mais embêtants qui furent corrigés dans les versions ultérieures. Au moment de la rédaction de ces lignes, la version ayant cours est la 1.7. GS/OS, système d'exploitation beaucoup plus complexe, n'est pas encore très stable au contraire de ProDos 8. Apple va en sortir de nouvelles versions environ deux fois par an...

5.0.2 Comparaison du ProDos 8 et du DOS 3.3

Le DOS 3.3 est constitué de deux modules principaux : le driver I/O (Input/Output) qui communique directement avec le contrôleur de disque 5.25, et l'interpréteur de commande Applesoft qui analyse et exécute les commandes disque du Basic Applesoft que le DOS 3.3 permet (OPEN, READ, CATALOG, etc ...). Les modules équivalents dans ProDos 8 sont séparés en deux fichiers programme appelés PRODOS (le driver I/O) et BASIC.SYSTEM (l'interpréteur de commande Applesoft). Dans la plupart des applications, ProDos 8 charge directement BASIC.SYSTEM au moment où le disque boote. Le tableau suivant décrit brièvement les commandes disque permises sous BASIC.SYSTEM et DOS 3.3. La plupart de ces commandes sont disponibles sous les deux systèmes, certaines étant spécifiques à l'un ou l'autre des systèmes d'exploitation. En général, les commandes BASIC.SYSTEM sont plus puissantes que leur équivalent en DOS 3.3 car elles acceptent un nombre important de paramètres. Faites attention, car quelques commandes ne jouent pas le même rôle d'un système d'exploitation à l'autre.

5.0.3 Comparaison des commandes disque Applesoft.

BASIC.SYSTEM-DOS 3.3

COMMANDES	DESCRIPTION	P8	DOS 3.3
APPEND	Ouvrir un fichier pour y ajouter des datas	OUI	OUI
BLOAD	Charger un fichier	OUI	OUI
BRUN	Charger et exécuter un prog. binaire	OUI	OUI
BSAVE	Sauvegarder un fichier	OUI	OUI
CATALOG	Lister les fichiers du volume (forme longue)	OUI	OUI
CLOSE	Fermer un fichier	OUI	OUI
DELETE	Effacer un fichier	OUI	OUI
EXEC	Exécuter des commandes d'un fichier Texte	OUI	OUI
IN#	Rediriger l'entrée de caractères	OUI	OUI
LOAD	Charger un programme Applesoft	OUI	OUI
LOCK	Vérouiller un fichier	OUI	OUI
NOMON	(Permis mais ignorer sous ProDos 8)	OUI	OUI
OPEN	Ouvrir un fichier	OUI	OUI
POSITION	Préparer lecture ou écriture à une position	OUI	OUI
PR#	Rediriger la sortie de caractères	OUI	OUI
READ	Lecture à partir d'un fichier	OUI	OUI
RENAME	Renommer un fichier	OUI	OUI
RUN	Charger et exécuter un prog. Applesoft	OUI	OUI
SAVE	Sauvegarder un prog. Applesoft	OUI	OUI
UNLOCK	Déverrouiller un fichier	OUI	OUI
VERIFY	Vérifiez l'existence d'un fichier	OUI	OUI

WRITE	Ecrire un fichier	OUI	OUI
"	Exécute un fichier Applesoft, BIN, TXT, SYS	OUI	NON
BYE	Transférer le contrôle à un autre prog. SYS	OUI	NON
CAT	Lister les fichiers du volume (forme courte)	OUI	NON
CHAIN	Transférer le ctrl à un autre prog. Applesoft	OUI	NON
CREATE	Créer un fichier	OUI	NON
FLUSH	Ecrire les contenus des buffers fichier	OUI	NON
FRE	Faire le ménage des chaines en mémoire	OUI	NON
PREFIX	Donner le nom du catalogue actif	OUI	NON
RESTORE	Récupérer les variables d'un prog. Basic	OUI	NON
STORE	Sauvegarder les variables d'un prog. Basic	OUI	NON
FP	Initialiser le mode Applesoft	NON	OUI
INIT	Formater une disquette	NON	OUI
INT	Initialiser le mode BASIC Integer	NON	OUI
MAXFILES	Créer de l'espace pour les buffers fichier	NON	OUI
MON	Permet l'affichage des opérations du DOS	NON	OUI

Il n'est pas surprenant de constater, que l'environnement PRODOS + BASIC.SYSTEM est plus gourmand en mémoire que le DOS 3.3. Le système PRODOS + BASIC SYSTEM utilise presque deux fois plus de mémoire que le DOS 3.3. Heureusement, la plupart du ProDos 8 réside dans une Bank Ram de 16 Ko commutable et n'est donc pas en conflit avec l'espace mémoire utilisé par l'interpréteur Applesoft. Cet espace mémoire commutable est intégré sur les Apple IIe, IIc et II GS et peut -être ajouté sur les Apple II ou Apple II Plus en installant une carte d'extension mémoire de 16 Ko dans le slot 0 de l'ordinateur : il s'agit de la carte langage, créée au moment du développement du système d'exploitation Pascal sur Apple II qui nécessitait de la mémoire supplémentaire. Il y a deux effets secondaires dus à l'utilisation de la carte langage par le ProDos. Premièrement, ProDos ne peut pas co-exister avec un programme qui utilise la carte langage par

lui même, ou pour y ranger des données. Deuxièmement, le Basic Integer ne peut plus être utilisé, car il se loge en carte langage : paix à ses cendres.

Une autre grande différence entre DOS 3.3 et BASIC.SYSTEM est dans la manière de prendre en charge les buffers pour les fichiers. Un buffer fichier est une zone de mémoire qu'un fichier va utiliser; ce buffer contient les données contenues dans la partie active de ce fichier ainsi que l'information nécessaire à la localisation de ce fichier sur disque. Quand le DOS 3.3 se lance, il initialise automatiquement trois de ces buffers; un nombre différent de buffers peut être réservé (de 1 à 16) en utilisant la commande MAXFILES. Les buffers fichier du DOS 3.3 ont chacun une longueur de 595 octets et sont situés entre le haut du programme Applesoft (cette adresse est stockée en $73/$74 et est appelée HIMEM) et le début du DOS 3.3 en $9D00.
ProDos 8, de son côté n'inialise pas de buffer fichier; il assigne ou déassigne dynamiquement des buffers fichiers au moment où les fichiers sont ouverts et fermés. Quand un fichier est ouvert, ProDos 8 diminue HIMEM de 1024 octets, et attribue ce buffer de 1024 octets à partir de la zone mémoire commençant en HIMEM + 1024. Quand un fichier est fermé, les buffers fichiers sont repositionnés, et HIMEM est augmenté de 1024 octets. Un total de huit fichiers peuvent être ouverts simultanément. Puisque ProDos utilise cette méthode d'allocation dynamique pour les buffers fichiers, il n'est pas possible d'utiliser la technique qui en DOS 3.3 consiste à réserver un espace mémoire pour y loger un programme machine en faisant diminuer HIMEM et en stockant le programme machine dans la zone mémoire ainsi libérée.

5.0.4 Caractéristiques importantes de ProDos 8 et de BASIC.SYSTEM

Un environnement PRODOS + BASIC.SYSTEM intègre de nombreuses caractéristiques utiles qui améliorent la vitesse d'exécution d'un programme et qui permettent de prendre en compte facilement des périphériques non Apple dans le système. Voici quelques une de ces importantes caractéristiques.

Interface Langage Machine (MLI)
Il est probable que la caractéristique la plus importante du ProDos 8 est l'interpréteur de commande disque appelé MLI (Machine Language Interface), qui permet un accès facile aux fichiers en utilisant les techniques de programmation du langage assembleur. Le DOS 3.3 ne dispose pas d'une telle interface et il est assez malcommode de programmer en langage assembleur dans son environnement. Avec les commandes MLI, on peut faire de la manipulation de fichiers comme ouvrir, lire, écrire, et fermer un fichier. Les paramètres correspondants à chacune des commandes ont clairement été définis par Apple. Nous examinerons plus tard les diverses commandes du MLI.

5.0.5 Datage des fichiers

A chaque fois que ProDos 8 crée ou écrit un fichier, il lit la date et l'heure courantes à partir de l'horloge installée dans le système (à supposer qu'il y en ait une), et range ces informations dans l'entrée du catalogue de ce fichier. Quand le disque est catalogué, la date et l'heure de création et la dernière modification apparaissent après le nom de ce fichier. ProDos 8 travaille avec l'horloge intégré de l'Apple II GS et avec toutes les cartes horloge qui émulent le jeu de commandes de la Thunderclock de Thunderware.

5.0.6 Carte contrôleur de disque et protocoles pour les drivers de périphériques

Une des grandes contrainte du DOS 3.3 est qu'il est très difficile de lui intégrer et de lui faire reconnaitre des périphériques disque autres que ceux développés par Apple (Lecteurs de disquettes haute densité, Disques durs compatibles, etc...). Il n'en est pas de même avec ProDos 8. Apple a publié un protocole de communication reconnu par le ProDos 8 qui permet de faire reconnaitre au système les périphériques disque autres que ceux qu'Apple commercialise. Ce protocole définit les adresses de la ROM de la carte contrôleur faisant référence à la taille de ce volume, les caractéristiques de ce volume, et l'adresse de la sous routine du driver responsable des opérations I/O de ce périphérique. Apple a également défini la technique pour passer ces paramètres à une sous routine driver disque ProDos 8 et comment le driver retourne des codes d'erreurs au programme appelant.

5.0.7 Une gestion améliorée des Interruptions

ProDos 8 installe automatiquement sa propre routine de gestion des interruptions qui prend le contrôle du système à chaque fois qu'un périphérique I/O génère un signal IRQ (Interrupt Request). Cette sous routine peut alors appeler d'autres sous routines permettant de gérer ces interruptions. Cela signifie qu'il est assez facile d'intégrer une sous routine d'interruption même si une autre est déjà active.

5.0.8 Structure hiérarchisée du catalogue

En utilisant ProDos 8, il est possible de créer plusieurs catalogues, chacun d'entre eux pouvant contenir plusieurs fichiers, le tout sur le même disque. Cela permet de ranger commodément dans un même catalogue un groupe de fichiers ayant des points communs pour pouvoir y avoir accès plus facilement. Les catalogues sont organisés de manière à ce que chacun d'entre eux soit contenu dans un autre (le catalogue père); le chemin d'accès des catalogues revient finalement au catalogue principal, la racine, que l'on appelle aussi le catalogue

Volume. Le catalogue principal est celui que l'on crée et que l'on nomme quand le disque est formaté. Nous analyserons plus tard et en détail la struture hiérarchisée des catalogues.

5.0.9 Périphérique disque /RAM

L'Apple II GS, l'Apple IIc, et l'Apple IIe (doté d'une carte 80 colonnes étendue) possède une mémoire auxiliaire de 64 Ko en supplément des 64 Ko de mémoire principale utilisée normalement pour le rangement des programmes. ProDos 8 utilise cet espace mémoire pour le stockage de fichiers de la même façon qu'il le fait sur un lecteur de disquettes ou un disque dur. Ceci s'appelle un Ram Disque. Les principales différences entre l'utilisation d'un Ram Disque et un lecteur de disques conventionnel sont que les opérations I/O se font beaucoup plus rapidement (il n'y aucune partie mécanique à déplacer) et que le contenu du RAM Disque disparait dès que l'alimentation électrique de l'ordinateur est coupée. Nous le verrons plus loin, mais il faut savoir que chaque volume disque du système a un nom, il s'agit du nom de volume. Le nom de volume pour le RAM Disque est /RAM.

5.0.10 Prolongement du BASIC.SYSTEM

Le BASIC.SYSTEM offre une méthode simple que vous pouvez utiliser pour lui ajouter des commandes additionnelles.

5.0.11 Séparation des pouvoirs

Contrairement au DOS 3.3, l'interpréteur de commandes bas niveau de ProDos 8, celui qui prend en charge toutes les opérations disque I/O fondamentales n'est pas intégré au sein de l'interpréteur BASIC.SYSTEM qui permet de communiquer avec un jeu de commandes disque "en anglais" lorsque l'on programme en BASIC Applesoft. Cela signifie que si vous désirez écrire un autre interpréteur de commandes, ou un programme 100% en langage assembleur, vous pouvez économiser environ 12 Ko de mémoire en le chargeant à la place de BASIC.SYSTEM.

5.0.12.1 *La commande RUN intelligente*

La commande - du BASIC.SYSTEM est une commande très pratique pour les gens qui n'aiment pas taper au clavier. Elle exécute soit un programme Applesoft (comme le fait un RUN), soit un fichier binaire

(BRUN), soit un fichier texte (EXEC) en déterminant automatiquement quel type de fichier a été spécifié puis en exécutant le travail nécessaire au lancement de ce fichier. Cette commande peut également exécuter des fichiers SYStem comme par exemple BASIC.SYSTEM. En gros, un programme SYStème est un programme indépendant écrit en langage assembleur, qui définit un environnement de programmation, ou un programme réalisant des fonctions spécifiques sans tenir compte de la présence d'un autre programme SYStème.

5.0.13 Des paramètres très utiles

Beaucoup des commandes du BASIC.SYSTEM supportent des paramètres très utiles permettant un contrôle plus grand sur la manière dont elles sont exécutées. Par exemple, vous pouvez utiliser le suffixe # (où# représente un numéro de ligne) avec la commande RUN du BASIC.SYSTEM pour charger un programme puis le lancer à partir de ce numéro de ligne. Vous pouvez utiliser aussi le suffixe E# (où# représente une adresse mémoire) pour spécifier une adresse de fin en utilisant une commande agissant sur un fichier BINaire (BLOAD et BSAVE). Vous pouvez aussi utiliser le suffixe Ttype avec BLOAD ou BSAVE pour travailler avec tous les types de fichiers autres que les fichiers BINaires. Le type de fichier est les trois caractères qui suivent le nom d'un fichier : BAS pour BASIC, BIN pour BINaire, TXT pour TeXT, etc... Un autre paramètre très utile est F#; lors de la lecture d'un fichier texte son utilisation permet de sauter un nombre spécifié d'enregistrements (un enregistrement est un groupe de caractères suivi par un <RETURN>.

5.0.14 Vitesse

ProDos 8 réalise les opérations disque I/O sur une disquette 5.25 à un débit d'environ 8 Ko par seconde. Cela est beaucoup plus rapide qu'en DOS 3.3 qui a un débit d'environ 1 Ko par seconde. De plus, BASIC.SYSTEM comprend une nouvelle version de la commande FRE qui permet de faire le ménage parmi les variables chaines du BASIC Applesoft beaucoup plus rapidement que son équivalent en DOS 3.3 (il faut plusieurs minutes au DOS 3.3 pour faire ce travail alors qu'il ne faut que quelques secondes à ProDos 8 pour réaliser le même travail).

5.0.15 Taille des fichiers et taille des volumes

ProDos 8 peut gérer des fichiers ayant une taille allant jusqu'à 16 Méga-octets, sur un volume pouvant avoir une taille jusqu'à 32 Méga-

octets. Les volumes DOS 3.3 ne peuvent pas excéder 400 Ko.

5.0.16 Comparaison de GS / OS et de ProDos 8

La différence fondamentale entre GS/OS et ProDos 8 est bien sûr que GS/OS ne fonctionne que sur l'Apple II GS. Ceci parce que GS/OS est écrit en langage assembleur 65C816, et qu'il utilise la Boite à outils (Toolbox) de l'Apple II GS comme par exemple le Memory Manager et le System Loader. Quoique la plupart des commandes GS/OS ont un équivalent sous ProDos 8, quelques commandes uniques font de GS/OS un environnement de programmation beaucoup plus riche.

1- Une application sous GS/OS peut appeler toutes les commandes GS/OS de n'importe quelle adresse mémoire à l'intérieur des 16 Méga-octets adressables par le 6C816. Une application sous ProDos 8 peut appeler les commandes ProDos 8 seulement à partir des premiers 64 Ko de mémoire.

2- Les applications sous GS/OS sont stockées dans des fichiers relogeables; cela signifie qu'elles peuvent être chargées et exécutées à toute adresse mémoire. Les applications ProDos 8 sont simplement des images binaires du code programme, et en général elles ne peuvent être exécutées qu'à partir d'une adresse mémoire spécifique. Il est bien entendu possible d'écrire des applications sous ProDos 8 relogeables, mais cela rend leur programmation particulièrement difficile et la plupart des programmeurs ne s'ennuient pas avec ce problème.

3- Les applications sous GS/OS utilisent le Memory Manager de l'Apple II GS pour s'assurer qu'elles n'utilisent pas des zones de mémoire déjà utilisées par d'autres programmes ou fichiers ressources du système. Les applications sous ProDos 8 sont responsables de leur gestion mémoire, les programmeurs doivent donc veiller aux zones mémoire utilisées par le ProDos 8.

4- GS/OS a 33 préfixes de pathname qui peuvent être référencés par un nom spécial abrégé comme par exemple 1/ ou 28/. ProDos 8 dispose seulement d'un préfixe de pathname que l'on appelle le préfixe par défaut.

5- GS/OS identifie les périphériques disque par un nom alors que ProDos 8 les identifie par un numéro de connecteur (slot) et de lecteur (drive).
6- GS/OS dispose d'une commande intégrée pour formater les disques (Format) et d'une commande permettant de cataloguer un volume (GetDirEntry). ProDos 8 ne dispose pas de ces commandes.

7- GS/OS a une commande permettant de déplacer des fichiers d'un catalogue à un autre (ChangePath). ProDos 8 ne dispose pas de cette commande.

8- Sous GS/OS, une application peut déterminer son propre nom avec la commande GetName. ProDos 8 ne dispose pas de commande similaire; une application peut déduire son nom en inspectant le buffer pathname.

9- GS/OS dispose d'une commande Quit améliorée qu'une application peut utiliser pour directement passer le contrôle au programme SYStème qui l'a appelée, ou passer le contrôle à tout programme système presque comme s'il s'agissait d'une sous-routine. La commande Quit de ProDos 8 peut seulement passer le contrôle à un sélecteur de programmes.

10- GS/OS peut créer et gérer des fichiers étendus, ProDos 8 ne le peut pas. Les fichiers étendus (parfois appelés fichiers ressource) sont constitués de deux parties logiques : une partie données et une partie ressource. La partie data contient généralement les données spécifiques à l'application, la partie ressource contient en général un groupe de données que l'on appelle ressources qui peuvent être des choses comme des icônes, des chaines de caractères, les messages d'erreurs, etc ...

11- GS/OS utilise des FST (File System Translator) pour permettre aux applications l'accès à des volumes disque n'utilisant pas le système d'exploitation ProDos, comme par exemple le système High Sierra pour le CD Rom ou le système HFS pour le Macintosh. GS/OS utilise aussi un FST pour accéder aux fichiers de type ProDos. Le ProDos 8 fonctionne uniquement avec des disques formatés sous ProDos.

12- GS/OS permet à une application d'avoir accès à des périphériques orientés caractères comme par exemple l'écran vidéo, le clavier, le modem, et l'imprimante en utilisant le même type de commandes que pour accéder aux fichiers disque. Sous ProDos 8, l'application doit utiliser des techniques tout à fait différentes pour accéder aux périphériques orientés caractères, chacun d'eux nécessitant la compréhension du fonctionnement du hardware en détail.

13- GS/OS accède aux disques plus rapidement que ProDos 8 car il utilise des techniques de disque cache et un codage assembleur 65C816 optimisé. Il peut aussi formater les disques avec un entrelacement de pistes plus bas (2:1 à la place de 4:1), améliorant ainsi la vitesse de transfert des données.

14- GS/OS permet d'ouvrir un nombre illimité de fichiers et de volumes; il n'impose pas de limites sur le nombre de périphériques par slot. ProDos 8 autorise seulement 8 fichiers ouverts simultanément, 14 volumes actifs, et deux périphériques par slots.

15- GS/OS utilisant des FST peut accéder à des volumes non ProDos pouvant avoir une taille jusqu'à 2048 GO (Giga-Octets), et peut gérer des fichiers ayant jusqu'à une longueur de 4096 Méga-octets. La taille des volumes ProDos ne peut pas excéder 32 Méga-octets, les fichiers ne peuvent pas dépasser 16 Méga-octets.

16- GS/OS n'est pas fourni avec un langage BASIC interprété comme BASIC.SYSTEM sous ProDos 8.

5.1 Volumes disques et organisation des fichiers

Dans cette partie, nous allons nous familiariser avec le concept de fichier et expliquer comment le système ProDos organise les fichiers sur la surface du disque. Il est nécessaire de connaitre les détails de ce système si vous voulez comprendre parfaitement le fonctionnement des commandes agissant sur les fichiers. GS/OS peut travailler avec des systèmes d'exploitation complètement différents du ProDos, mais la plupart des utilisateurs utilisent GS/OS avec des disques formatés sous ProDos.

Le concept de fichier est commun et fondamental à tous les systèmes d'exploitation. Un fichier est tout simplement un ensemble de données qui peut être un programme exécutable, une lettre à votre éditeur, une feuille de calcul pour un tableur, ou tout autre document qu'un programme peut gérer. La structure générale d'un fichier est définie par le système d'exploitation lui même; le système d'exploitation fournit également les diverses commandes nécessaires pour accéder aux fichiers : création, ouverture, lecture, écriture, fermeture, destruction, etc ...

5.1.1 Donner un nom aux fichiers

Lorsque vous sauvez pour la première fois un fichier sur disque, vous devez lui donner un nom qu'un programme pourra identifier par la suite. Un nom de fichier ProDos peut être constitué d'un maximum de 15 caractères. Il doit commencer par une lettre alphabétique (A à Z), mais les autres caractères peuvent être toute combinaison de lettres, de chiffres (0 à 9), et de points (.). Vous pouvez aussi utiliser les lettres minuscules, mais ProDos 8 et GS/OS les convertissent immédiatement en majuscules. Voici quelques exemples de nom de fichiers acceptés par ProDos :

MODELE.LETTRE

CONTRAT.3 CHAPITRE.QUATRE

Voici quelques exemples de noms de fichiers non valides :

5.MORCEAUX	Commence par un nombre
CHIEN ROUGE	Contient un espace
CECI&CELA	Contient un &
LE.DANUBE.DE.LA.PENSEE	Trop long

Une erreur souvent commise est l'utilisation de l'espace comme séparateur de mots (voir exemple 2 ci dessus). Cela est permis en DOS 3.3 mais pas en ProDos. Les points (.) et non les espaces doivent être utilisés pour séparer les mots dans un nom de fichier pour améliorer la lisibilité. Certains programmes, comme Appleworks, permettent à l'utilisateur d'entrer des espaces dans le nom des fichiers, mais ils convertissent automatiquement les espaces en points avant d'utiliser les commandes du système d'exploitation. GS/OS peut travailler avec des volumes disque qui ont été formatés avec d'autres systèmes d'exploitation, si le FST approprié se trouve sur le disque système. Les règles pour nommer les fichiers sont différentes pour ces systèmes d'exploitation. Par exemple, le système HFS du Macintosh permet de donner des noms d'une longueur maximum de 31 caractères: ces non peuvent contenir tout caractère ASCII imprimable sauf les deux points (:). Consultez les manuels de référence du système d'exploitation utilisé pour connaitre les règles en ce qui concerne les noms de fichiers.

5.1.2 Catalogues et Sous-Catalogues

Quand vous sauvegardez un fichier ProDos sur disque, vous pouvez le ranger dans n'importe lequel des catalogues qui a pu être créé sur ce disque. Ces catalogues sont similaires à des dossiers dans lesquels on peut classer des fiches, car on place dans des dossiers des fiches ayant un lien entre elles. En fait le terme dossier est employé en informatique comme un synonyme de catalogue, on emploie aussi le terme répertoire. Par exemple, il est possible de créer un catalogue pour y stocker tous les documents traitement de textes, et un autre catalogue pour y placer tous les programmes Applesoft. La possibilité de créer des catalogues séparés sur le même disque facilite l'organisation et le rangement d'un nombre important de fichiers.

Quand on formate un disque, seul un catalogue, le catalogue principal du volume, existe. On lui donne un nom lorsque l'on formate le disque. Il devient alors le nom de ce volume. Les règles pour nommer les catalogues sont identiques à celles qu'il faut observer pour nommer un fichier. Le catalogue principal d'un volume formaté à l'aide de ProDos peut contenir le nom d'un maximum de 51 fichiers. Le DOS 3.3 accepte 105 noms de fichiers.

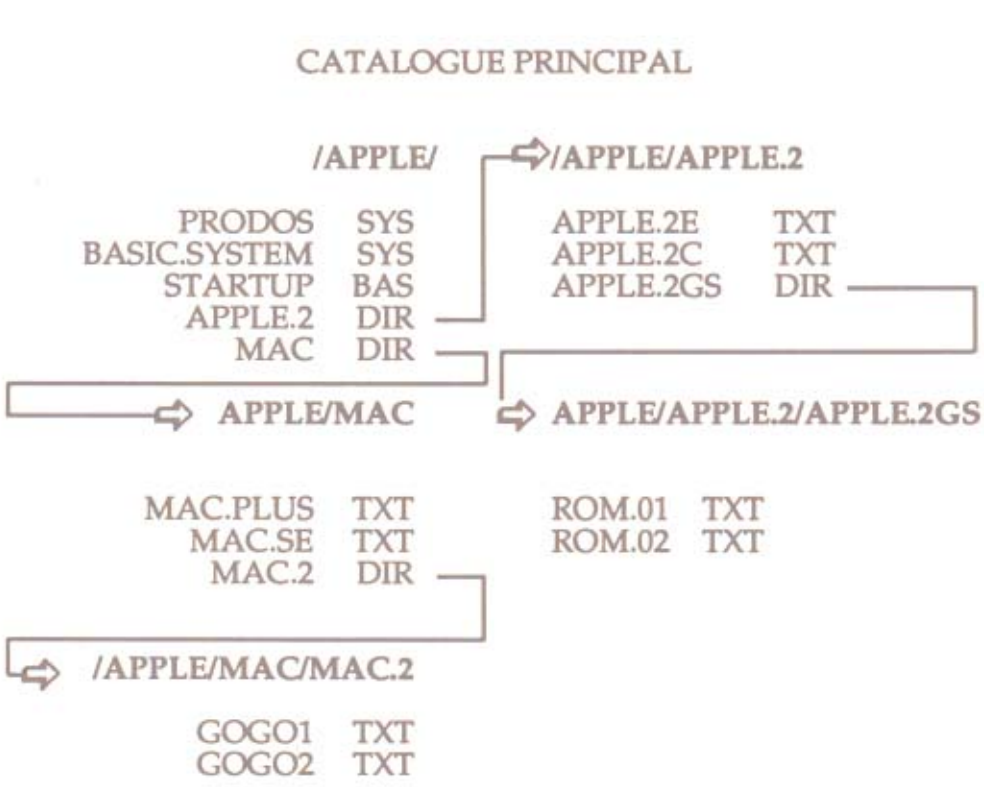
Il est possible de créer des catalogues supplémentaires (on les appelle alors des sous-catalogues) à partir du catalogue principal en utilisant la commande CREATE de GS/OS ou de ProDos. On peut même créer des sous-catalogues à l'intérieur d'autres sous-catalogues. Un sous-catalogue peut contenir les noms d'autant de fichiers que vous désirez, jusqu'à ce que votre disque soit plein. Ce système de catalogues imbriqués s'appelle une structure de catalogues hiérarchisée.

Tous les systèmes d'exploitation modernes utilisent une structure de catalogue hiérarchisée.

Pour indiquer le catalogue où un fichier doit être sauvé, vous devez ajouter à son nom un préfixe spécial afin de créer un identificateur unique appelé le PATHNAME ou Chemin d'Accès en français. Un Pathname est constitué des noms d'une série de catalogues commençant par le nom du catalogue principal et continuant avec les noms de tous les catalogues par lesquels il faut passer pour atteindre le catalogue final, suivie par le nom du fichier lui même. Chaque nom de catalogue est séparé du précédent par un délimitateur spécial, et ce délimitateur doit précédé le nom du catalogue principal.
Sous GS/OS, ce caractère délimitateur peut être soit le slash (/) ou les deux points (:). Sous ProDos 8, cela doit être un slash (/). Nous utiliserons le / comme délimitateur dans les explications à venir.

Les noms de catalogues dans un Pathname (chemin d'accès) doivent définir un chemin continu; cela signifie que chaque catalogue spécifié doit être contenu dans le catalogue précédent. Par exemple, supposez que nous ayons un disque appelé APPLE; le catalogue principal de ce disque a donc le nom de APPLE. Supposez que le volume APPLE ait deux sous-catalogues appelés APPLE.2 et MAC.

Le tableau suivant montre cette structure de catalogues.



Si vous voulez sauvegarder un fichier appelé MAC.PLUS dans le sous-catalogue MAC, vous devrez indiquer le Pathname suivant :

/APPLE/MAC/MAC.PLUS

Si vous aviez seulement indiqué le nom de fichier MAC.PLUS, ce fichier aurait été sauvegardé dans le catalogue courant, qui est en général le catalogue principal à moins que le catalogue courant ait été changé avec la commande SetPrefix que nous décrirons plus loin.

Sous GS/OS vous pouvez spécifier un nom de périphérique, à la place d'un nom de volume (le nom du volume est le même que le nom de catalogue principal), au moment où l'on forme le pathname. Les noms de périphériques commencent par un point (.) et peuvent avoir un nombre de caractères compris entre 2 et 31. .SCSI1, .DEV4, .APPLEDISK3.5A sont des exemples de noms de périphériques. Si le fichier MAC.PLUS de l'exemple précédent se trouve sur le périphérique appelé .SCSI1; on pourrait l'identifier avec le pathname suivant :

.SCSI1/MAC/MAC.PLUS

Cette technique ne peut pas être utilisée sous ProDos 8 car ProDos 8 n'utilise pas les noms de périphériques.

Comme indiqué plus haut, sous GS/OS, le délimitateur dans un Pathname peut être / ou : mais vous ne pouvez pas utiliser deux délimitateurs différents pour un seul pathname. GS/OS détermine le délimitateur en examinant le Pathname indiqué de gauche à droite jusqu'à ce qu'il trouve un / ou un :; le caractère trouvé servira de délimitateur.

Si le délimitateur (sous GS/OS) est : vous pouvez utiliser des / dans les noms de fichiers, ce qui est très important si vous désirez accéder à des fichiers ne se trouvant pas sur un volume ProDos (en utilisant un FST approprié). Par exemple, les fichiers Macintosh peuvent contenir des / . L'inverse n'est pas possible. En effet, si le délimitateur est un /, vous ne pouvez pas utiliser : dans un nom de fichier. Il est préférable dans une application sous GS/OS d'utiliser le : comme délimitateur dans les Pathnames.

5.1.3 Les Préfixes

Si la plupart des fichiers que vous allez utiliser se trouvent dans le même sous-catalogue, il devient rapidement ennuyeux de spécifier le même Pathname (chemin d'accès) de catalogues à chaque fois que l'on veut accéder à l'un de ces fichiers. Pour supprimer ce problème, GS/OS et ProDos 8 ont une commande SetPrefix que vous pouvez utiliser pour former une chaine de noms de catalogues qui s'ajoutera automatiquement à tout nom de fichier que vous spécifierez dans une commande. Cette chaine s'appelle le Préfixe par défaut et ne peut pas avoir une longueur supérieure à 64 caractères sous ProDos 8 ou de 8 Ko sous GS/OS. Si par exemple vous fixez le préfixe par défaut sur /APPLE/MAC/, vous pouvez faire référence à tout fichier de ce sous-catalogue en lui ajoutant simplement le nom de ce fichier (par exemple MAC.PLUS). Un nom qui est la continuation du préfixe par défaut peut aussi accéder à des niveaux de sous-catalogues plus bas; un tel nom s'appelle un Pathname partiel. Si, par exemple, le préfixe par défaut est /APPLE/MAC/, et si le sous-catalogue MAC contient un sous-catalogue appelé MAC.2 contenant lui meme un fichier appelé GOGO1, vous pouvez accéder à ce fichier en accolant au Pathname /APPLE/MAC/ le Pathname partiel MAC.2/GOGO1. Le Pathname partiel n'est pas précédé par un /.
Sous GS/OS (mais pas sous ProDos 8), on peut utiliser une forme raccourcie du Préfixe par défaut en le remplaçant par 0/. Dans notre exemple, cela signifie que 0/ est équivalent à /APPLE/MAC. Comme le montre le tableau suivant, GS/OS reconnait 32 préfixes différents auxquels on peut faire référence par un nombre suivi de /(de 0/à 31/). GS/OS reconnait aussi le préfixe de boot (le nom du volume à partir

duquel vous avez booté) en l'identifiant par */; on ne peut pas modifier */. 1/ et 9/ identifient le catalogue dans lequel l'application courante réside. 2/ identifie le catalogue qui contient les fichiers système. On peut modifier 1/, 2/, et 9/ avec la commande GS/OS SetPrefix. Vous pouvez utiliser les autres préfixes quand une application a besoin d'identifier un catalogue particulier en utilisant les préfixes abrégés de GS/OS.

Sous ProDos 8 les préfixes peuvent, au maximum, avoir une longueur de 64 caractères, en comptant le / qui doit les précéder. Les Pathnames partiels ne peuvent eux non plus dépasser 64 caractères. GS/OS a deux sortes de préfixes : les courts et les longs. Les préfixes courts (*/ , et 0/ à 7/) peuvent avoir au maximum une longueur de 64 caractères; les préfixes longs (8/ à 31/) peuvent avoir au maximum une longueur de 8192 caractères.

LES NUMEROS DE PREFIXE SOUS GS/OS

Numéro de PREFIXE	DESCRIPTION
*/	LE PREFIXE DE BOOT. C'EST LE NOM DU VOLUME DE BOOT.
0/	LE PREFIXE PAR DEFAUT. AUTOMATIQUEMENT AJOUTE.
1/	LE PREFIXE OU EST LOCALISEE L'APPLICATION.
2/	LE PREFIXE QUI POINTE SUR LA BIBLIOTHEQUE SYSTEME.
3/ à 8/	LAISSES AU CHOIX DE L'UTILISATEUR.
9/	IDENTIQUE A 1/.
10/ à /31	LAISSES AU CHOIX DE L'UTILISATEUR.

Une caractéristique intéressante de GS/OS et de ProDos 8 est qu'à chaque fois qu'une commande doit localiser un fichier décrit par un Pathname, elle effectue cette recherche sur tous les volumes disques connectés au système. Ceci est tout à fait différent du DOS 3.3 où l'on doit expliciter le numéro de slot et de lecteur pour accéder à un fichier (en utilisant les paramètres ,S# et ,D#). Pour des raisons de compati-

bilité avec le DOS 3.3, on peut aussi utiliser ces paramètres avec le BASIC.SYSTEM. De toute façon, BASIC.SYSTEM utilisera automatiquement le nom du volume disque indiqué par les paramètres de numéro de slot et de lecteur pour créer le Pathname.

Les avantages apportés dans l'utilisation des sous-catalogues sont souvent peu évidents aux utilisateurs de lecteurs de disquettes, mais tout à fait clairs aux utilisateurs de disques durs. Sur un disque dur, on peut ranger des centaines de fichiers. Si tous les fichiers étaient placés dans un catalogue unique, il faudrait attendre un temps important pour localiser un fichier en particulier lors du catalogue, de plus il y a de fortes chances pour que vous ne le trouviez pas parmi tous les autres fichiers. Heureusement que ProDos permet une organisation hiérarchisée des catalogues permettant ainsi de ranger les fichiers de même intéret dans un même sous-catalogue pour y accéder plus facilement.

5.1.4 Les concepts fondamentaux pour la manipulation d'un fichier

GS/OS et ProDos 8 disposent d'un interpréteur de commandes qui peut comprendre une variété de commandes permettant de manipuler un fichier.

OPEN	Ouverture d'un fichier pour les opérations I/O.
READ	Lire des données à partir d'un fichier.
Write	Ecrire des données dans un fichier.
Close	Fermeture d'un fichier pour les opérations I/O.

Quatre commandes similaires sont également disponibles sous Applesoft quand on utilise l'interpréteur BASIC.SYSTEM dans l'environnement ProDos 8.

5.1.5 Ouverture d'un fichier

Avant d'utiliser un fichier, on doit l'ouvrir. On réalise cette action en utilisant la commande Open suivie du nom du fichier que l'on désire ouvrir. Le système d'exploitation ouvre un fichier d'abord en le localisant sur le volume disque, puis en lui attribuant un buffer spécial en mémoire. Une partie de ce buffer contient l'information qui indique au système d'exploitation à quel endroit du volume disque sont localisées les données de ce fichier; une autre partie de ce buffer contient la portion du fichier à laquelle on vient d'accéder. A chaque fois que l'on demande une opération I/O pour un fichier, le système

d'exploitation détermine si la portion du fichier à laquelle on veut accéder se trouve dans ce buffer. Si c'est le cas, le système d'exploitation n'a pas besoin d'accéder à ce fichier sur le volume disque. Il range simplement les données dans le buffer (dans le cas d'une opération d'écriture), ou lit les données à partir de ce buffer (dans le cas d'une opération de lecture). Ainsi, les opérations portant sur un fichier s'exécutent beaucoup plus rapidement que dans le cas où on utiliserait des techniques ne mettant pas en oeuvre de buffer en mémoire.

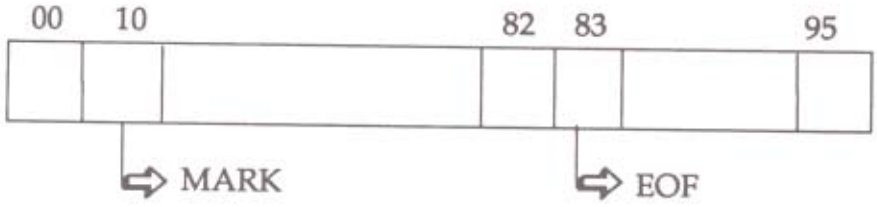
ProDos 8 peut ouvrir un fichier à l'un des 16 différents niveaux de fichiers système (numérotés de 0 à 15); GS/OS reconnait 256 niveaux différents de fichiers système (numérotés de 0 à 255). Sous ProDos 8, une application peut spécifier un niveau de fichier système en stockant le numéro de niveau à une adresse mémoire particulière ($BF94) juste avant d'ouvrir le fichier. Sous GS/OS, l'application doit utiliser la commande SetLevel. Le fichier système par défaut a le niveau 0. L'avantage d'avoir différents niveaux de fichiers permet d'écrire facilement des programmes "superviseurs". Ce type de programmes ouvrent leurs propres fichiers de travail, passent le contrôle à des programmes utilisateur, et reprennent le contrôle quand les programmes utilisateur ont terminés. Si un programme "superviseur"a un niveau supérieur à un programme utilisateur, ses fichiers de travail ne pourront pas être fermés par inadvertance par le programme utilisateur même si ce programme tente de fermer tous les fichiers ouverts. (sauf bien sûr si le programme utilisateur ne respecte pas la procédure normale en décrémentant le niveau de fichier).

5.1.6 Lecture et Ecriture d'un fichier

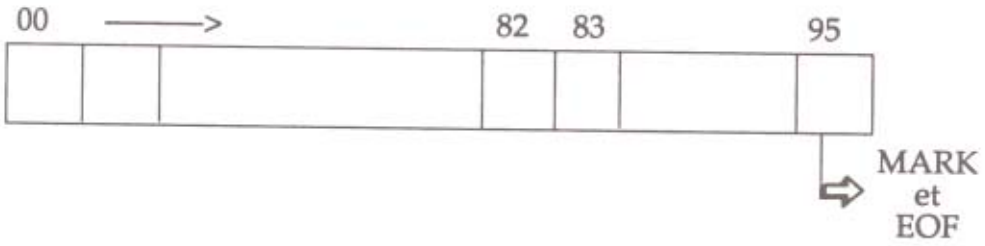
Quand le système d'exploitation ouvre un fichier, il initialise deux pointeurs internes importants qu'il utilise pour garder trace de la taille de ce fichier et de la dernière position dans ce fichier référencée par une application. Ces pointeurs sont appelés EOF et MARK.

1) MARK & EOF après l'ouverture d'un fichier de 83 octets

2) MARK & EOF aprés la lecture de 10 octets dans ce fichier



3) MARK ET EOF APRES L'ECRITURE DE 12 OCTETS A LA FIN DU FICHIER.



Note : EOF est automatiquement étendu.

EOF (End Of File = Fin De Fichier) est le pointeur qui indique la fin du fichier; il pointe toujours un octet supplémentaire après le dernier octet réel du fichier. Si vous tentez de faire une lecture de données dans ce fichier après cet emplacement, une erreur se produira (l'erreur "fin de données"). EOF se modifie seulement si une application écrit des données à la fin de ce fichier; quand cela arrive, EOF est automatiquement incrémenté du nombre d'octets approprié, et si nécessaire, le système d'exploitation attribue des blocks supplémentaires sur le disque pour ce fichier. GS/OS et ProDos 8 disposent d'une commande SetEOF permettant d'attribuer à EOF une valeur spécifique.

MARK est un pointeur qui donne la position courante dans le fichier; il indique toujours la position à laquelle la prochaine opération de lecture ou d'écriture prendra place. Il est égal à 0 (le début du fichier) quand on ouvre un fichier pour la première fois, et il est automatiquement incrémenté à chaque fois que des données sont lues et écrites dans ce fichier. Par exemple, si MARK pointe sur 10 (c'est à dire sur

le 11 ème octet du fichier), et que vous lisez ou écrivez 14 octets supplémentaires d'information, MARK se positionnera alors à 24.

Il est aussi possible de donner une valeur explicite au pointeur MARK permettant ainsi d'accéder à toutes les positions dans un fichier. Cela signifie qu'il est possible d'accéder à un enregistrement d'un fichier contenant des longueurs d'enregistrement fixes très rapidement, car il n'est pas nécessaire de lire tous les enregistrements précédents.

5.1.7 Fermeture d'un fichier

Une fois que l'on a terminé de travailler avec un fichier, il faut le fermer. Cela permet de s'assurer que les données se trouvant dans le buffer fichier, et pas encore rangées sur disque, vont être sauvegardées sur le disque lui même. La fermeture d'un fichier met également à jour dans le catalogue la taille de ce fichier.

Il n'est pas nécessaire de fermer un fichier immédiatement après avoir fini de travailler avec lui. On peut attendre jusqu'à la fin du programme pour procéder à cette fermeture. Il est quand même préférable de procéder à cette fermeture le plus rapidement possible pour réduire le risque de perte de données dans le cas d'un plantage du système ou d'une coupure électrique. Le fait de fermer les fichiers inactifs libère de la mémoire.

5.1.8 GS/OS et la technique du disque cache

Pour accélérer les opérations disque comme celles décrites plus haut, GS/OS effectue un cache des blocs disque. Le cache disque est une zone de mémoire où GS/OS sauvegarde une copie des blocs disque quand il les a lu une fois à partir du disque. GS/OS place aussi dans ce cache des copies des blocks qu'il écrit sur disque. Une fois qu'un bloc est dans le cache, GS/OS peut rapidement le récupérer de la mémoire à chaque fois qu'il a besoin de relire ce bloc disque. GS/OS n'a alors pas besoin de refaire un accès disque (relativement lent) pour relire ce bloc.
L'utilisateur peut régler la taille de ce cache disque avec l'accessoire de bureau (NDA) Disk Cache. (Dans le système 5.0 ce réglage est incorporé dans le NDA plus général Control Panel qui permet de configurer tout le système.) Une application peut régler la taille de ce cache en utilisant la commande GS/OS ResetCache après avoir sauvegardé cette nouvelle taille du cache dans la Ram alimentée (BRAM) en utilisant pour cela la fonction WriteBParam. Plus la taille du cache est importante, plus GS/OS est performant, mais l'espace mémoire utilisable par les applications s'en trouve réduit en proportion.

Dans la plupart des cas, la taille du cache n'est pas suffisamment importante pour contenir tous les blocs que GS/OS voudrait y placer. Quand la mémoire cache est saturée, GS/OS enlève de la mémoire cache le bloc le moins récemment utilisé pour libérer de la place au prochain bloc.

Les commandes GS/OS Read et Write vous permettent de spécifier des blocs disque spécifiques qui pourront ou non etre placés dans cette mémoire cache.

5.1.9. Organisation des fichiers sous ProDos

Les systèmes d'exploitation utilisent différentes méthodes pour organiser les fichiers sur disque, et pour garder trace de quelles parties du disque ont été utilisées pour la sauvegarde des données, de telle manière que les fichiers peuvent facilement et efficacement être créés, effacés, etc... Dans ce passage, nous allons examiner les points suivants :

- La structure d'un volume disque formaté sous ProDos
- La structure d'une bit map d'un volume ProDos
- La structure des catalogues et des sous catalogues sous ProDos
- La structure d'une entrée dans un catalogue ProDos
- Le processus d'index utilisé par ProDos pour localiser les fichiers

ProDos utilise la même méthode générale pour organiser les fichiers quel que soit le volume disque utilisé; il s'agit de toute façon de volumes disque divisés en blocs : lecteur Apple 5.25, lecteur Apple 3.5, HD20SC, volume /RAM, etc ... Des différences spéciques peuvent apparaitre car la capacité de stockage de ces différents volumes est variable. De plus, les tailles de deux structures très importantes, le catalogue principal, et la bit map (table d'occupation) d'un volume peuvent être différentes.

5.1.10 Formatage du volume disque

Avant de pouvoir utiliser une disquette (ou tout autre support magnétique) avec GS/OS ou ProDos 8, elle doit être formatée pour que GS/OS ou ProDos 8 la reconnaisse. Vous pouvez formater un volume disque avec un Filer (ProSel, Copy II+, etc ...), ou avec les utilitaires système qui se trouvent sur la disquette master ProDos 8, ou encore avec le Finder de GS/OS. GS/OS dispose d'une commande Format utilisable par les applications pour formater un volume

disque.

La méthode utilisée pour formater un disque dépend de la nature de ce périphérique disque. Quand on formate une disquette 5.25, 35 pistes sont créées sur le disque (numérotées de 0 à 34), chacune d'entre elles contenant 4096 octets d'information. Ces pistes sont arrangées en anneaux concentriques tout autour du trou central de la disquette; la piste 0 se trouvant à l'extérieur, et la piste 34 à l'intérieur. Le système d'exploitation peut accéder à chacune de ces pistes en déplaçant une tête de lecture/écriture sur la piste choisie. Ceci est réalisé en utilisant des emplacements I/O particuliers qui activent un moteur pas à pas contrôlant le déplacement de la tête de lecture/écriture.

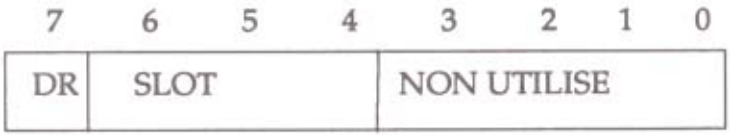
Chacune de ces 35 pistes est subdivisée en 16 unités plus petites appelées secteurs. Un secteur est la plus petite unité de données qui peut être lue ou écrite en même temps. Les secteurs qui constituent une piste sont numérotés de 0 à 15; chacun d'entre eux pouvant contenir 256 octets d'information. Si vous savez faire une multiplication, vous vous apercevrez qu'une disquette 5.25 peut contenir 560 secteurs d'information (140 Ko).

C'est la dernière fois que vous entendrez parler de secteurs car ProDos utilise des blocs de 512 octets comme unité de base dans la manipulation des fichiers; chaque bloc est donc constitué de deux secteurs. Une disquette 5.25 est donc constituée de 280 de ces blocs (numérotés de 0 à 279). Heureusement, il est rarement nécessaire de connaitre la localisation de ces blocs sur disque; le système d'exploitation se charge de cette gestion.

5.1.11 Volumes disque et Lecteurs de disque

Une disquette formatée qui est en ligne (placée dans un lecteur et prête à être accédée) est souvent appelée un volume disque. Les volumes ProDos ont des noms qui suivent les mêmes règles de baptème que les fichiers, mais sont précédés d'un / pour les distinguer des noms de fichiers.

Les lecteurs de disque eux-mêmes ont des identificateurs uniques. ProDos 8 donne un numéro d'unité à chaque périphérique disque connecté au système. La valeur de ce numéro d'unité est formée par le numéro de slot de la carte contrôleur de ce lecteur, et du numéro du lecteur.



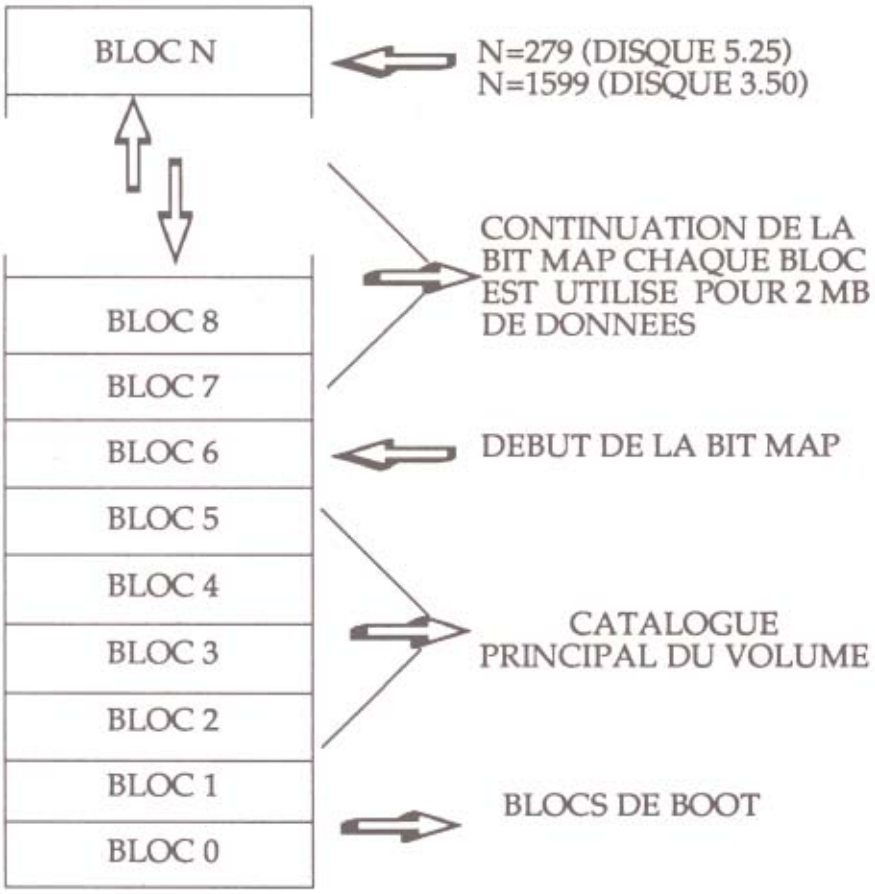
SLOT peut en fait être le numéro réel ou logique d'un slot si le système contient des périphériques disque comme par exemple un RAM disque. Par exemple, le numéro d'unité d'un volume /RAM connecté en slot 3, Drive 2 sur un IIe, IIc, ou II GS est $B0 soit 1 011 0000.

DR indique le numéro de DRive (lecteur) : 0 pour le lecteur 1 et 1 pour le lecteur 2. Plus de deux lecteurs peuvent être connectés au port 5 du SmartPort. Dans ce cas, ProDos 8 assigne logiquement les deux prochains lecteurs en Slot 2, Drive 1 et en Slot 2, Drive 2. ProDos 8 ignore tous les lecteurs connectés au SmartPort après le quatrième.

GS/OS donne des numéros de référence uniques aux périphériques disque (et aux périphériques caractères) qu'il trouve connecté au système. Ces nombres sont des entiers consécutifs commençant par 1. Il leur donne aussi un nom de périphérique; par exemple .APPLEDISK3.5A, .SCSI1, etc ... ; ces noms peuvent être constitués de 2 à 31 caractères de long. GS/OS n'utilise pas le procédé du numéro d'unité utilisé par ProDos 8. Voir plus loin pour des explications détaillées concernant les périphériques disque et les conventions sur leur nom.

5.1.12 Utilisation des blocs sur un Volume disque

Nous allons maintenant examiner la méthode utilisée par ProDos pour gérer les fichiers sur disque. Notre discussion inclu une analyse des structures des catalogues renfermant l'information sur les fichiers, de la bit map qui renferme l'information concernant l'utilisation des blocs du volume disque, et des blocs d'index qui contiennent les emplacements des blocs de données utilisés par chacun des fichiers du volume. Comme nous l'avons dit plus haut, un total de 280 blocs contenant 140 Ko de données sont disponibles sur une disquette 5.25 formatée sous ProDos. Si un programme standard de formatage est utilisé, 7 de ces blocs (0 - 6) ne sont pas disponibles pour l'utilisation pour le stockage des fichiers car ProDos se les réserve. La figure suivante montre l'utilisation des blocs sur des disquettes 5.25 et 3.5 venant d'être formatées



Chaque bloc contient 512 octets.
La capacité totale de stockage est de 280 blocs (140 Ko) pour un disk 5.25.
La capacité totale de stockage est de 1600 blocs (800 Ko) pour un disk 3.50.

Les blocs 0 et 1 contiennent un court programme en langage machine que le firmware de la carte contrôleur de disque charge en mémoire et exécute à chaque fois qu'il boote un disque. Ce programme est appelé programme de boot; il localise, charge et exécute un fichier SYStème spécial appelé ProDos s'il le trouve sur le disque. Un fichier SYStème a un type de fichier ayant pour code $FF, et le mnémonique SYS. PRODOS est le programme qui va installer et activer le système d'exploitation.

Les blocs 2 à 5 sont les blocs qui contiennent le catalogue principal du volume disque. Nous décrirons plus loin la structure de ce catalogue.

Le bloc 6 est le premier bloc contenant la bit map pour un volume disque. Dans la bit map, chaque bit indique si le block auquel il correspond est libre ou occupé. ProDos réserve un bloc bit map pour chaque 2 MB (4096 blocs) d'espace de stockage.

Les blocs au delà du bloc de bit map (ou des blocs), soit un total de 273 pour une disquette 5.25 ou 1593 pour une disquette 3.50, sont libres et peuvent être utilisés pour le stockage des fichiers sur disque.

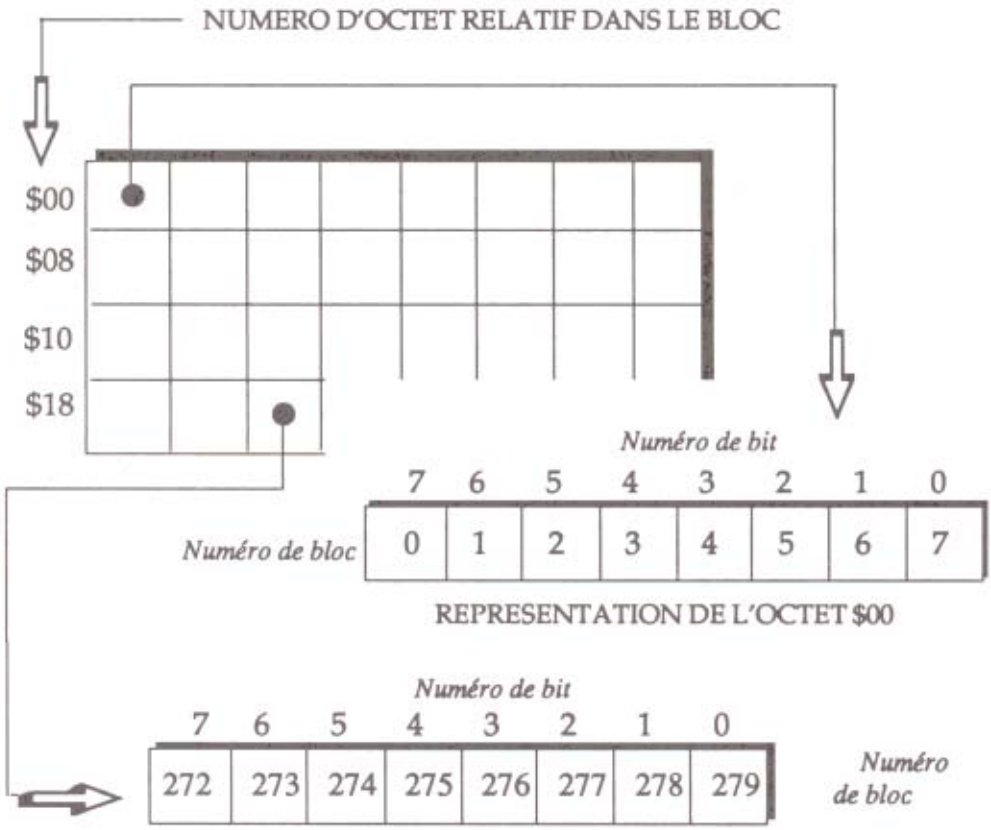
5.1.13 La bit map d'un volume (Table d'occupation)

Le système d'exploitation accède à la bit map d'un volume quand il a besoin de déterminer l'état de chaque bloc sur le disque. Il lit la bit map à chaque fois qu'il attribue un nouvel espace disque à un fichier, de telle manière qu'il peut rapidement localiser les blocs libres de ce disque. Il écrit dans la bit map pour réserver des blocs à un fichier; cela arrive quand un fichier déjà existant augmente en taille, ou libère des blocs; cela arrive quand un fichier diminue de taille ou quand il est effacé.

Les routines standards de formatage utilisent le bloc 6 comme premier bloc bit map d'un volume. Le bloc 6 est seulement un emplacement conventionnel pour placer le début de la bit map; il est possible de ranger la bit map n'importe où ailleurs sur un bloc libre du disk. Par exemple, la bit map d'un volume /RAM est en bloc 3. Comme indiqué plus loin, le numéro de bloc du premier bloc de bit map d'un volume apparait dans l'entête du catalogue principal décrivant les caractéristiques du volume disque.

Sur une disquette 5.25, seuls les 35 premiers octets (280 bits) du bloc de bit map du volume sont utilisés; chacun des bits de chaque octet correspond à un numéro unique de bloc. Un bloc de bit map peut donc faire référence à des volumes constitués d'un maximum de 4096 blocs. Pour des volumes plus importants, comme les disques durs, une continuation de la bit map peut être trouvée sur les blocs suivants

directement celui utilisé en premier. Par exemple, le vieux disque dur APPLE PROFILE (9728 blocs) nécessite 3 blocs pour constituer sa bit map. Le programme standard de formatage stocke la première partie de la bit map sur le bloc 6 et la continue sur les blocs 7 et 8. Le système d'exploitation détermine la taille de la bit map d'un volume en examinant deux octets dans l'entête du catalogue du volume indiquant la taille de ce volume.



REPRESENTATION DE L'OCTET $22

CHAQUE OCTET DANS LA BIT MAP DU VOLUME DEFINIT L'ETAT DE HUIT BLOCS CONTIGUS. LE BIT CORRESPONDANT A UN NUMERO DE BLOC DONNE PEUT ETRE CALCULE EN DIVISANT D'ABORD LE NUMERO DE BLOC PAR 8; LA PARTIE ENTIERE DU RESULTAT DONNE LE NUMERO DE L'OCTET CORRESPONDANT. POUR OBTENIR LE NUMERO DU BIT DANS CET OCTET, OTEZ LE RESTE DE CETTE DIVISION DE 7.
NUMERO D'OCTET = INT (NUMERO DE BLOC/ 8)
NUMERO DE BIT = 7 - ((NUMERO DE BLOC) - (8 * NUMERO D'OCTET))

la figure ci-dessus montre la structure d'une bit map pour des disquettes 5.25. Comme vous pouvez le voir, les bits de chacun des octets de cette bit map reflètent l'état de 8 blocs consécutifs; le bit 0 correspond au bloc ayant le numéro le plus haut, le bit 7correspond au bloc ayant le numéro le plus bas. Si le bit correspondant à un bloc particulier est égal à 0, alors ce bloc est occupé. S'il vaut 1, il est libre.

Exemple de Bit Map

```
00    00 3F FF FF FF FF FF FF FF FF FF FF
0C    FF FF FF FF FF FF FF FF FF FF FF FF
18    FF FF FF FF FF FF FF FF FF FF FF 00
24    00 00 00 00 00 00 00 00 00 00 00 00
30    00 00 00 00 00 00 00 00 00 00 00 00
3C    00 00 00 00 00 00 00 00 00 00 00 00
```

Le première entrée dans la table (octet $00) a pour valeur $00 ce qui correspond à 0000 0000 en binaire; cela signifie que les blocs $00 à $07 sont occupés. La deuxième entrée dans la table (octet $01) a pour valeur $3F ce qui correspond à 0011 1111 en binaire; cela signifie que les blocs $08 à $09 sont occupés (valeur binaire 0), et que les blocs $0A à $0F sont libres (valeur binaire 1). Les autres blocs de la bit map ($10 à $117) sont libres car leur valeur binaire est 1. Dans cet exemple nous avons donc un total de dix blocs occupés sur le volume disque. Ici la bit map s'arrête à partir de l'octet $23; le système d'exploitation connait la taille de la bit map car il la retrouve dans l'entête du catalogue principal.

5.1.14 Catalogues et Sous-Catalogues d'un Volume

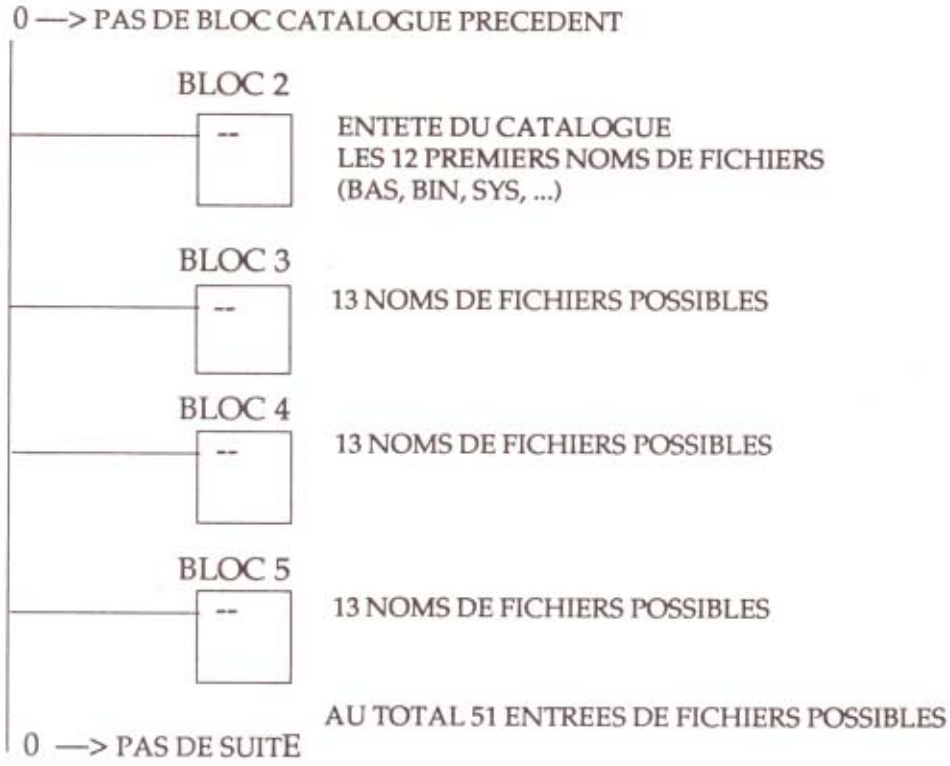
Un catalogue est un enchevêtrement compliqué de données que ProDos utilise pour conserver les informations importantes concernant chaque fichier sauvegardé sur le volume. Cela comprend le nom du fichier, son type, sa taille, sa date de création, l'emplacement sur le volume des données de ce fichier, etc ... Sans ces informations, il ne serait pas possible de gérer efficacement de nombreux fichiers sur un volume.

Comme nous l'avons dit, ProDos permet de créer de multiples catalogues sur un volume. A l'exception du catalogue principal (on accède à tous les autres catalogues en faisant référence au catalogue principal), ces catalogues peuvent occuper sur le volume tout l'espace nécessaire car ProDos les traite comme des fichiers standards. Le catalogue principal débute toujours à partir du bloc 2; si vous utilisez

un programme de formatage standard, ou les commandes Format de GS/OS et EraseDisk, le catalogue principal occupera aussi les blocs 3, 4 et 5.
Un catalogue ProDos est un exemple d'une structure de données à double lien. Les liens sont en fait des paires de pointeurs sur 2 octets stockés au début de chaque bloc de catalogue. Un de ces pointeurs (Octets $00-$01) contient le numéro du bloc de catalogue précédent dans la chaine, ou zéro s'il n'y a pas de bloc précédent. L'autre pointeur (Octet $02-$03) contient le numéro du prochain bloc catalogue, ou zéro s'il n'y a pas de bloc catalogue à la suite. Cette technique permet de créer des catalogues de toute taille.

Chaque bloc utilisé par un catalogue peut contenir 13 entrées de fichier; chaque entrée de fichier est constituée de 39 octets. Cela signifie que le catalogue principal (blocs $02-$05) peut contenir un total de 52 entrées, dont l'une est l'entrée pour le nom de Volume lui même. Voilà pourquoi vous obtenez parfois le sympathique message d'erreur "Disque Plein" alors que votre disque dur de 100 MO ne contient qu'une poignée de fichiers; vous avez simplement oublié de sauvegarder vos précieuses données ailleurs que dans le catalogue principal qui ne peut contenir que 51 entrées ...
Quand ProDos doit trouver un fichier particulier, il lit en premier le bloc 2 du volume; il s'agit du bloc clé du volume, celui auquel le système d'exploitation accède le plus souvent (en conséquence le plus fragile). Si ce fichier n'est pas trouvé dans ce bloc, le bloc de catalogue suivant est lu, et ainsi de suite jusqu'au bloc 5.

5.1.14 Le Catalogue Principal d'un Volume ProDos



STRUCTURE D'UN BLOC CATALOGUE

N D'OCTET DANS LE BLOC CATALOGUE	SIGNIFICATION
$000-$001	NUMERO DE BLOC DU BLOC CATALOGUE PRECEDENT (OCTET DE POIDS FAIBLE EN PREMIER) EGAL à 0 S'IL S'AGIT DU PREMIER BLOC CATALOGUE
$002-$003	NUMERO DE BLOC DU BLOC CATALOGUE SUIVANT (OCTET DE POIDS FAIBLE EN PREMIER) EGAL à 0 S'IL S'AGIT DU DERNIER BLOC CATALOGUE
$004-$02A	ENTREE AU CATALOGUE DU FICHIER NUMERO 1 OU, S'IL S'AGIT DU PREMIER BLOC CATALOGUE (BLOC CLE),LES OCTETS $00 ET $01 SONT EGALS à 0 ET ON TROUVE ICI L'ENTETE DU CATALOGUE (DIRECTORY HEADER)
$02B-$051	ENTREE AU CATALOGUE DU FICHIER NUMERO 2
$052-$078	ENTREE AU CATALOGUE DU FICHIER NUMERO 3
$079-$09F	ENTREE AU CATALOGUE DU FICHIER NUMERO 4
$0A0-$0C6	ENTREE AU CATALOGUE DU FICHIER NUMERO 5
$0C7-$0ED	ENTREE AU CATALOGUE DU FICHIER NUMERO 6
$0EE-$114	ENTREE AU CATALOGUE DU FICHIER NUMERO 7
$115-$13B	ENTREE AU CATALOGUE DU FICHIER NUMERO 8
$13C-$162	ENTREE AU CATALOGUE DU FICHIER NUMERO 9
$163-$189	ENTREE AU CATALOGUE DU FICHIER NUMERO 10
$18A-$1B0	ENTREE AU CATALOGUE DU FICHIER NUMERO 11
$1B1-$1D7	ENTREE AU CATALOGUE DU FICHIER NUMERO 12
$1D8-$1FE	ENTREE AU CATALOGUE DU FICHIER NUMERO 13
$1FF	NON UTILISE

5.1.15 L'Entête du Catalogue (Directory Header)

Le premier bloc qu'un catalogue (ou qu'un sous-catalogue) utilise est le bloc clé, et il a une structure un peu différente des autres blocs catalogue. La différence réside dans les premiers 39 octets qui normalement décrivent l'entrée au catalogue du fichier numéro 1; dans ce cas ces octets servent à décrirent le catalogue lui même; cette entrée est alors appelée entête du catalogue. Le tableau qui suit montre la signification de ces 39 octets constituant l'entête du catalogue.

ENTETE D'UN CATALOGUE

Numéro de l'Octet dans le Bloc Clé	SIGNIFICATION
$04	4 BITS PD FORT: TYPE D'ENREGISTREMENT $F = CATALOGUE VOLUME $E = SOUS CATALOGUE 4 BITS PD FAIBLE: LONGUEUR DU NOM DU VOLUME IL SE TROUVE DANS LE CHAMP SUIVANT
$05-$13	NOM DU VOLUME: 15 OCTETS (ASCII POSITIF) LA LONGUEUR DE CE NOM EST DANS L'OCTET $04 LE RESTE DE CET ENREGISTREMENT EST IGNORE / N'EST PAS UTILISE ICI
$14-$1B	RESERVE POUR UNE UTILISATION FUTURE EN GENERAL REMPLI AVEC DES 00
$1C-1D	DATE DE CREATION DU CATALOGUE (FORMATAGE) YYYYYYYMMMMDDDD (Y=ANNEE, M=MOIS, D=JOUR) FORME BINAIRE
$1E-$1F	MINUTE ET HEURE DE CREATION DU CATALOGUE 000HHHHH00MMMMMM(H=HEURE, M=MINUTE) FORME BINAIRE
$20	VERSION DE PRODOS QUI A FORMATE LE VOLUME
$21	VERSION MINIMUM DE PRODOS POUVANT UTILISER CE VOLUME
$22	CODE D'ACCES PRODOS (VOIR PLUS LOIN) $80 LE VOLUME PEUT ETRE REFORMATE $40 LE VOLUME PEUT ETRE RENOMME $20 LE CATALOGUE A ETE MODIFIE DEPUIS LE DERNIER BACKUP $02 AUTORISATION D'ECRITURE SUR LE VOLUME

	$01 AUTORISATION DE LECTURE SUR LE VOLUME
$23	LE NOMBRE D'OCTETS UTILISES PAR CHAQUE ENTREE AU CATALOGUE ($27)
$24	LE NOMBRE D'ENTREE SUR CHAQUE BLOC DE CATALOGUE ($0D).L'ENTETE DU CATALOGUE EST CONSIDERE COMME UNE ENTREE
$25-$26	NOMBRE D'ENTREES ACTIVES DANS LE CATALOGUE SANS COMPTER L'ENTETE DU CATALOGUE. UNE ENTREE ACTIVE DECRIT UN FICHIER OU UN SOUS CATALOGUE NON EFFACE.
$27-$28	POINTEUR SUR LE PREMIER BLOC CONTENANT LA BIT MAP (EN GENERAL LE BLOC 6) DANS UN SOUS-CATALOGUE C'EST LE POINTEUR INDIQUANT LE BLOC DEFINISSANT L'ENTREE DE CE SOUS-CATALOGUE
$29-$2A	TAILLE EN BLOCS DU VOLUME ($0118) POUR UN DISK 5.25 (280)

5.1.15 Entrée au Catalogue

Toutes les entrées au catalogue, autres que l'entête du Catalogue, représentent soit des fichiers de données standards (binaires, texte, programme Basic Applesoft, etc...), soit des sous-catalogues. Les formats de ces différentes entrées sont dans l'essentiel identiques et sont décrits ici.

DESCRIPTIF D'UNE ENTREE D'UN FICHIER AU CATALOGUE

Numéro Relatif de L'OCTET dans L'Enregistrement	SIGNIFICATION
$00	4 BITS DE PD FORT: TYPE D'ENREGISTREMENT $0 = ENTREE EFFACEE - ENTREE RE-DISPONIBLE $1 = FICHIER "SEEDLING" (1 SEUL BLOC DE DATA) $2 = FICHIER "SAPLING" (2 à 256 BLOCS DATA) $3 = FICHIER "TREE" (257 à 32768 BLOCS DATA) $4 = ZONE PASCAL $5 = FICHIER ETENDU $D = SOUS-CATALOGUE $E = RESERVE POUR L'ENTREE D'UN SOUS-CATALOGUE $F = RESERVE POUR L'ENTETE DU CATALOGUE PRINC. 4 BITS DE PD FAIBLE: LONG. DU NOM DE FICHIER QUAND LE FICHIER EST EFFACE CET ENREGISTREMENT EST EGAL A $0
$01-$0F	NOM DU FICHIER (ASCII POSITIF)
$10	TYPE DE FICHIER $00 UNK PAS DE TYPE DE FICHIER $01 BAD FICHIER BLOC S DEFECTUEUX $02 PCD FICHIER CODE PASCAL $03 PTX FICHIER TEXTE PASCAL $04 TXT FICHIER TEXTE (ASCII POSITIF) $05 PDA FICHIER DE DATAS PASCAL $06 BIN FICHIER BINAIRE (IMAGE BIN. 8 BITS) $07 FNT SOS (APPLE 3) FICHIER FONT $08 FOT FICHIER FOTO SOS (APPLE 3) $09 BA3 PROGRAMME BUSINESS BASIC $0A DA3 FICHIER DATAS BUSINESS BASIC $0B WPF FICHIER TRAITEMENT DE TEXTE $0C SOS FICHIER SYSTEM SOS (APPLE 3) $0F DIR FICHIER SOUS-CATALOGUE $10 RPD FICHIER DATAS RPS $11 RPI FICHIER INDEX RPS $12 AFD FICHIER DISCARD APPLEFILE $13 AFM FICHIER MODEL APPLEFILE $14 AFR FICHIER FORMAT DE RAPPORT APPLEFILE $15 SCL FICHIER SCREEN LIBRARY $19 ADB FICHIER BASE DE DONNEES APPLEWORKS $1A AWP FICH. TRAITEMENT DE TEXTE APPLEWORKS

Numéro Relatif de L'OCTET dans L'Enregistrement	SIGNIFICATION
	$1B ASP FICHIER TABLEUR APPLEWORKS
	$AB GSB FICHIER PROGRAMME GS BASIC
	$AC TDF FICHIER DEFINITION TOOLBOX GS BASIC
	$AD BDF FICHIER DATAS GS BASIC
	$B0 SRC FICHIER SOURCE APW
	$B1 OBJ FICHIER OBJET APW
	$B2 LIB FICHIER BIBLIOTHEQUE APW
	$B3 S16 FICHIER SYSTEM GS/OS
	$B4 RTL BIBLIOTHEQUE RUN-TIME APW
	$B5 EXE FICHIER EXECUTABLE APW
	$B6 PIF FICHIER INIT PERMANENT GS/OS
	$B7 TIF FICHIER INIT TEMPORAIRE GS/OS
	$B8 NDA FICHIER NEW DESK ACCESSORY
	$B9 CDA FICHIER CLASSIC DESK ACCESSORY
	$BA TOL FICHIER OUTIL GS/OS
	$BB DRV FICHIER DRIVER GS/OS
	$BC GLF FICHIER LOAD GS/OS
	$BD FST FICHIER FILE SYSTEM TRANSLATOR GS/OS
	$C0 PNT FICHIER IMAGE SHGR COMPRESSE
	$C1 PIC FICHIER IMAGE SHGR
	$C8 FON FONT GS/OS
	$C9 FND FICHIER DATAS FINDER
	$CA ICN FICHIER ICONE
	$CB AIF FICHIER AUDIO INTERCHANGE FORMAT
	$EE R16 FICHIER OBJET RELOGEABLE EDASM 816
	$EF PAS PARTITION PASCAL SUR UN VOLUME
	$F0 CMD FICHIER COMMANDE BASIC.SYSTEM
	$F1
	.
	. TYPES DE FICHIERS A DEFINIR
	.
	$F8
	$F9 O.S SYSTEME D'EXPLOITATION GS/OS
	$FA INT FICHIER PROGRAMME BASIC INTEGER
	$FB IVR FICHIER DE VARIABLES BASIC INTEGER
	$FC BAS FICHIER PROGRAMME BASIC APPLESOFT
	$FD VAR FICHIER DE VARIABLES BASIC APPLESOFT
	$FE REL FICHIER RELOGEABLE CODE EDASM
	$FF SYS FICHIER SYSTEME PRODOS 8
$11-$12	POINTEUR SUR LE BLOC CLE

Numéro Relatif de L'OCTET dans L'Enregistrement	SIGNIFICATION
	NUMERO DE BLOC DU BLOC CLE D'UN FICHIER DANS LE CAS D'UN FICHIER "SEEDLING" C'EST LE NUMERO DE BLOC DU SEUL BLOC DE DATAS DANS LE CAS D'UN FICHIER "SAPLING" C'EST LE NUMERO DE BLOC DU BLOC D'INDEX POUR LES FICHIERS "TREE" C'EST LE NUMERO DE BLOC DU BLOC D'INDEX MASTER POUR UN SOUS-CATALOGUE C'EST LE NUMERO DE BLOC DE SON PREMIER BLOC
$13-$14	TAILLE DU FICHIER EN NOMBRE DE BLOCS ON Y INCLUT LES BLOCS D'INDEX ET LES BLOCS DE DATAS SI LE FICHIER EST UN SOUS CATALOGUE C'EST LE NOMBRE DE BLOCS DE CE SOUS CATALOGUE
$15-$17	EOF (END OF FILE) - TAILLE DU FICHIER EN OCTETS (OCTETS DE POIDS FAIBLE EN TETE)
$18-$19	DATE DE CREATION DU FICHIER YYYYYYYMMMMDDDDD CHAQUE LETTRE EST UN ELEMENT BINAIRE (Y=ANNEE, M=MOIS, D=JOUR)
$1A-$1B	HEURE DE CREATION DU FICHIER 000HHHHH00MMMMMM (HEURES/MINUTES)
$1C	VERSION PRODOS A LA CREATION DU FICHIER
$1D	VERSION MINIMUM DE PRODOS POUR LE FICHIER
$1E	CODE D'ACCES POUR CE FICHIER

	$80 : LE FICHIER PEUT ETRE DETRUIT $40 : LE FICHIER PEUT ETRE RENOMME $20 : FICHIER MODIFIE DEPUIS LE DERNIER BACKUP $02 : AUTORISATION D'ECRITURE DANS LE FICHIER $01 : AUTORISATION DE LECTURE DU FICHIER EN GENERAL LE CODE D'ACCES D'UN FICHIER NON VERROUILLE EST $C3 ($80 + $40 + $02 + $01) - UN FICHIER VERROUILLE A UN CODE D'ACCES DE $01
$1F-$20	TYPE DE FICHIER AUXILIAIRE SA SIGNIFICATION DEPEND DU TYPE DE FICHIER

TXT	:	LONGUEUR DU FICHIER TEXTE
BIN	:	ADRESSE DE CHARGEMENT D'UNE IMAGE BINAIRE
BAS	:	ADRESSE DE CHARGEMENT D'UN PROGRAMME BASIC ($801)
VAR	:	ADRESSE DE CHARGEMENT D'UN FICHIER DE VARIABLES
SYS	:	ADRESSE DE CHARGEMENT D'UN PROG. SYSTEME ($2000)

$21-$24	DATE/HEURE DE DERNIERE MODIFICATION DU FICHIER MEME FORMAT QUE POUR LA CREATION
$25-$26	NUMERO DE BLOC DU BLOC CLE DU CATALOGUE CONTENANT L'ENTREE DU FICHIER

5.1.16 Code d'accès Fichier

Pour chaque entrée de fichier au catalogue, l'octet relatif $1E permet de définir les opérations possibles pour ce fichier. Le bit 0 de l'octet identifie la lecture, le bit 1 l'écriture, le bit 6 la possibilité de renommer le fichier, et le bit 7 la possibilité de détruire ce fichier. Si la valeur d'un bit est à 1, ProDos autorise l'opération associée à ce bit.

Le bit 2 indique si le fichier doit être consideré comme visible ou invisible. Si ce bit est à 1, les sous-routines de catalogue d'un disque ignorent ce fichier.

Le bit 5 indique si ce fichier a été modifié depuis sa dernière copie. C'est bien entendu au programme de copie de mettre ce bit à 0 quand il réalise une copie de ce fichier.

Les bits 3 et 4 ne sont pas utilisés et doivent toujours être à 0.

Si le bit correspondant est à 1:

D = Autorisation de destruction
r = Autorisation de renommer
B = Backup nécessaire
I = Invisibilité du fichier
W = Autorisation d'écriture
R = Autorisation de lecture

Les commandes BASIC.SYSTEM LOCK et UNLOCK affectent l'état du code d'accès fichier : LOCK interdit l'écriture, la fonction Rename et la fonction Destroy; UNLOCK les autorise à nouveau.

Il n'y a pas de commandes BASIC.SYSTEM pour modifier individuellement chacun de ces bits, par contre nous verrons plus loin que l'on peut le faire avec la commande SetFileInfo de ProDos 8 ou de GS/OS.

Si un bit est à 1, la fonction attribuée à ce bit est autorisée; s'il vaut 0 la fonction n'est pas autorisée. Les bits 4 et 3 (bits réservés) doivent toujours être à 0.

Si les bits D, r, et W sont tous à 1, on dit que le fichier n'est pas verrouillé; s'ils sont tous les trois à 0, alors ce fichier est verrouillé. Toutes autres combinaisons signifient que ce fichier à des limitations restrictives d'accès.

Le bit d'invisibilité concerne les routines de catalogue d'un volume qui gèrent les fichiers cachés appelés aussi fichiers invisibles. Si ce bit est à 1, la routine de catalogue ne doit pas faire apparaitre ce fichier dans le listing.

ProDos 8 et GS/OS mettent automatiquement à 1 le bit de Backup (B) dès qu'il écrivent une data dans un fichier. Cela donne la possibilité d'écrire des programmes de copie qui ne recopient que les fichiers ayant été modifiés. C'est aux programmes de copie en question de remettre le bit B à 0 dès que la copie du fichier a été faite.

5.1.17 Organisation des Datas d'un Fichier : Structures de fichier

Le travail principal de ProDos et de GS/OS est de connaitre les numéros des blocs qui constituent un fichier. Le système d'exploitation à une technique d'index pour retrouver ces blocs. En général (pour les fichiers ayant un nombre de blocs de datas compris entre 2 et 256), le pointeur sur le Bloc clé dans l'entrée du fichier au catalogue (octets relatifs $11 et $12) pointe sur un bloc d'index contenant une

liste ordonnée des numéros de tous les blocs que le fichier utilise pour ranger ses datas. L'avantage principal de cette technique d'index, c'est qu'un fichier peut occuper sur un volume une série de blocs qui ne sont pas nécessairement consécutifs (par exemple le système d'exploitation Pascal oblige l'utilisation de blocs consécutifs pour le rangement des datas). Cela signifie que l'on ne perd pas d'espace sur le volume disque. L'inconvénient est que les opérations disque I/O sont plus lentes car il est nécessaire de positionner la tête de lecture/écriture sur tous les blocs dispersés sur le volume d'un fichier fragmenté. Pour défragmenter un fichier il existe l'utilitaire Beach Comber de Glen Bredon (sur un disque dur cela peut faire gagner énormément de temps).

Il éxiste 3 types de structures de fichier dépendantes de la taille du fichier auquel le système doit accéder :

- Fichier "Seedling" : 1 à 512 octets (1 Bloc de Datas)
- Fichier "Sapling" : 513 à 131072 octets (128 Ko - maximum 256 blocs)
- Fichier "Tree" : 131073 à 16777215 octets (16 Mb-1 32768 blocs maxi)

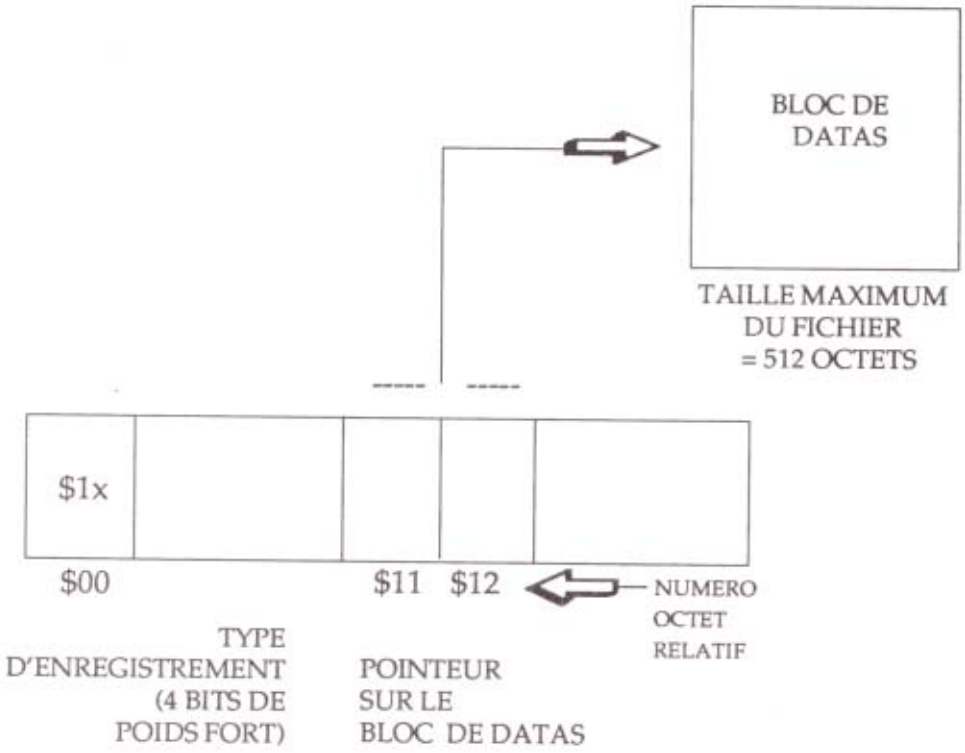
On peut déterminer la technique d'index utilisée par un fichier (autre qu'un sous-catalogue) en examinant les 4 bits de poids fort du code de type d'enregistrement stockés dans l'octet relatif $00 dans une entrée de fichier au catalogue. Ce nombre est égal à $1 pour un fichier "Seedling", $2 pour un fichier "Sapling", et $3 pour un fichier "Tree". Si ce nombre est égal à $0 le fichier a été effacé. Les fichiers sous-catalogues utilisent les codes $D, $E, ou $F; $D identifie une entrée au catalogue pour un fichier sous-catalogue, $E un sous-catalogue, et $F le catalogue du volume. Le code $4 identifie une partition Pascal et le code $5 identifie un fichier de type étendu.

Le pointeur clé d'un fichier (octets relatifs $11 et $12) pointe sur un bloc d'index (aussi appelé Bloc clé). Nous allons maintenant examiner comment le système d'exploitation utilise le bloc d'index pour chacun de ces trois types de fichier.

Fichier "Seedling" - fichier de 1 à 512 octets (1 bloc). Ce type de fichier utilise un bloc unique pour ses datas; c'est le numéro de bloc indiqué par le pointeur clé (octets relatifs $11 et $12).

Par exemple, créons le programme Basic Applesoft suivant:

```
10      PRINT CHR$(4); «OPEN TXTFILE,L64»
20      FOR I = 0 TO 2
30      PRINT CHR$(4); «WRITE TXTFILE,R»;I
40      PRINT «RECORD»;I
50      NEXT I
60      PRINT CHR$(4); «CLOSE TXTFILE»
70      END
```

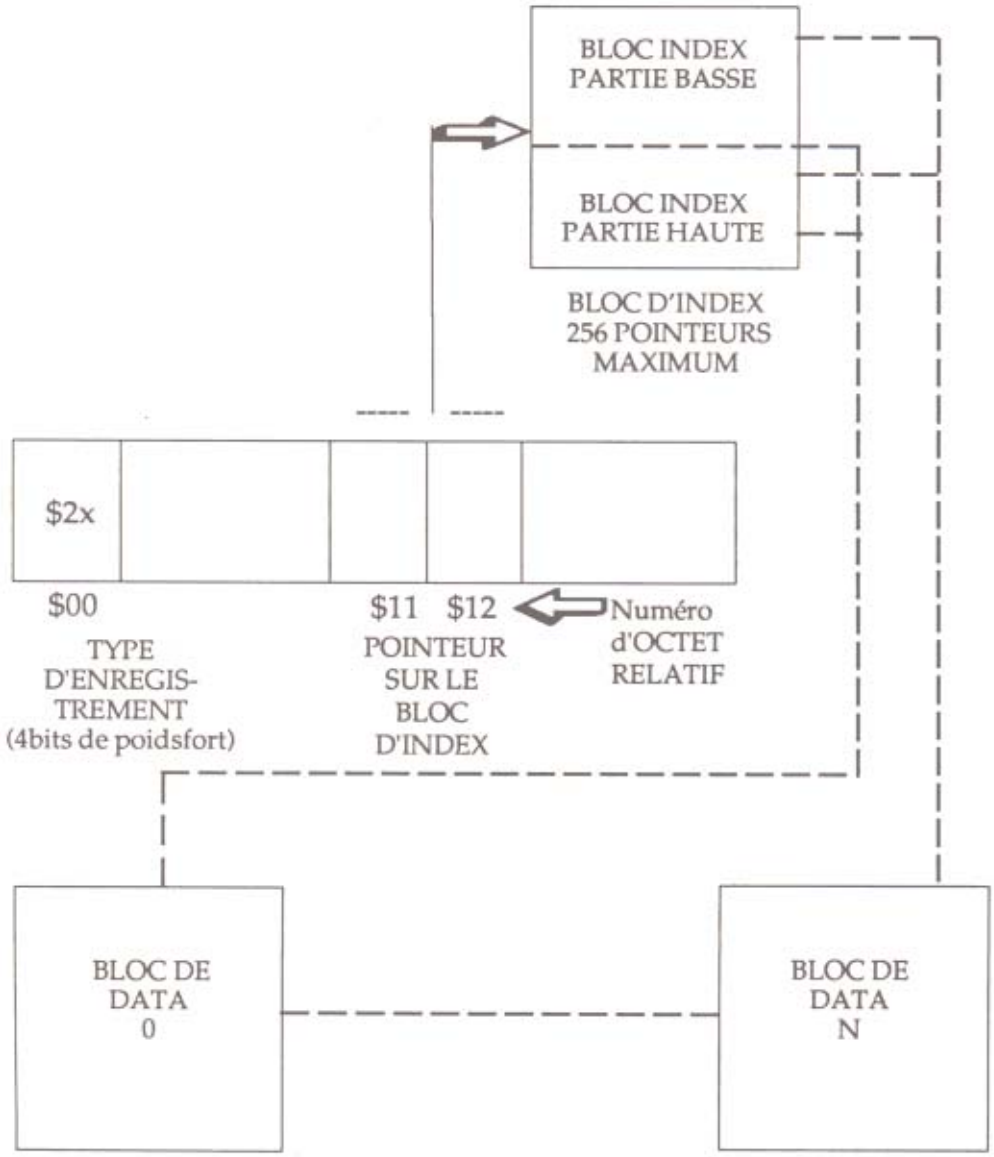


Ce petit programme crée un fichier texte appelé «TXTFILE» avec des enregistrements d'une longueur de 64 octets. Il écrit ensuite dans ce fichier 3 enregistrements contenant les chaines de caractères "RECORD0", "RECORD1", et "RECORD2". La taille totale de ce fichier est de 3 fois 64 octets soit 192 octets. Ce nombre étant inférieur à 512 octets, ce fichier sera sauvegardé sous forme de fichier "Seedling".

Fichier "Sapling" - Le pointeur clé d'un fichier "Sapling" pointe sur un numéro de bloc qu'on appelle le bloc d'index. Ce bloc d'index contient une liste ordonnée de numéro de blocs utilisés pour le rangement des datas de ce fichier. Deux octets sont nécessaires pour

stocker chaque numéro de bloc. La partie basse du numéro de bloc est toujours stockée dans la première partie de ce bloc d'index; la partie haute du numéro de bloc est stockée dans la deuxième partie du bloc d'index. La taille maximum d'un fichier "Sapling" est de 128 Ko; il ne peut pas être plus important car un bloc d'index ne peut référérencer au maximum que 256 blocs.

Reprenons notre exemple de tout à l'heure, et modifions la ligne 20 par :




```
20        FOR I = 0 TO 100
```

Relançons le programme par l'instruction RUN. Le fichier créé contient maintenant 101 enregistrements de 64 octets chacun; la taille totale du fichier "TXTFILE" est donc de 6464 octets. Quand le neuvième enregistrement est écrit (RECORD8), le système d'exploitation (ProDos ou GS/OS) se rend compte que le bloc "Seedling" original est plein. Le système d'exploitation va alors créer ce que l'on appelle un bloc d'index. Ce bloc contient tous les numéros de blocs de chaque blocs de données pour le fichier "TXTFILE" dans l'ordre dans lequel le système d'exploitation va y accéder. En utilisant un bloc d'index, le système d'exploitation peut accéder à un fichier dans un ordre séquentiel même si ses blocs de datas ne sont pas physiquement contigus (l'un après l'autre sur le disque).

Ainsi, dans notre exemple, un nouveau bloc est attribué pour le bloc d'index (par exemple le bloc $A; on suppose que les datas du fichier de type "Seedling" étaient stockées en bloc $08). Les blocs $B à $16 sont utilisés pour la sauvegarde des datas suivantes du fichier "TXTFILE". Ce sont ces numéros de blocs qui sont placés dans le bloc d'Index ($A). Bien entendu, l'entrée au catalogue pour le fichier "TXTFILE" est mise à jour pour que le bloc clé pointe sur le bloc d'Index (octets relatifs $11-$12); le fichier est maintenant identifié comme étant "Sapling" (4 bits de poids fort de l'octetrelatif $00). Dans le bloc d'Index, les numéros de blocs pointés sont stockés en deux parties : le poids faible des numéros de blocs sont dans la première partie du bloc d'index (de l'octet relatif $000 à $0FF); le poids fort de ces mêmes numéros sont dans la deuxième partie du bloc d'index (de l'octet relatif $100 à $1FF).

BLOC $A

PARTIE BASSE BLOC INDEX

```
00C 080B0C0D0E0F101112131415
018 1600000000000000000000000000
024 0000000000000000000000000000
030 0000000000000000000000000000
03C 0000000000000000000000000000
048 0000000000000000000000000000
054 0000000000000000000000000000
060 0000000000000000000000000000
06C 0000000000000000000000000000
078 0000000000000000000000000000
084 0000000000000000000000000000
090 0000000000000000000000000000
09C 0000000000000000000000000000
0A8 0000000000000000000000000000
0B4 0000000000000000000000000000
0C0 0000000000000000000000000000
0CC 0000000000000000000000000000
0D8 0000000000000000000000000000
0E4 0000000000000000000000000000
0F0 0000000000000000000000000000
0FC 0000000
```

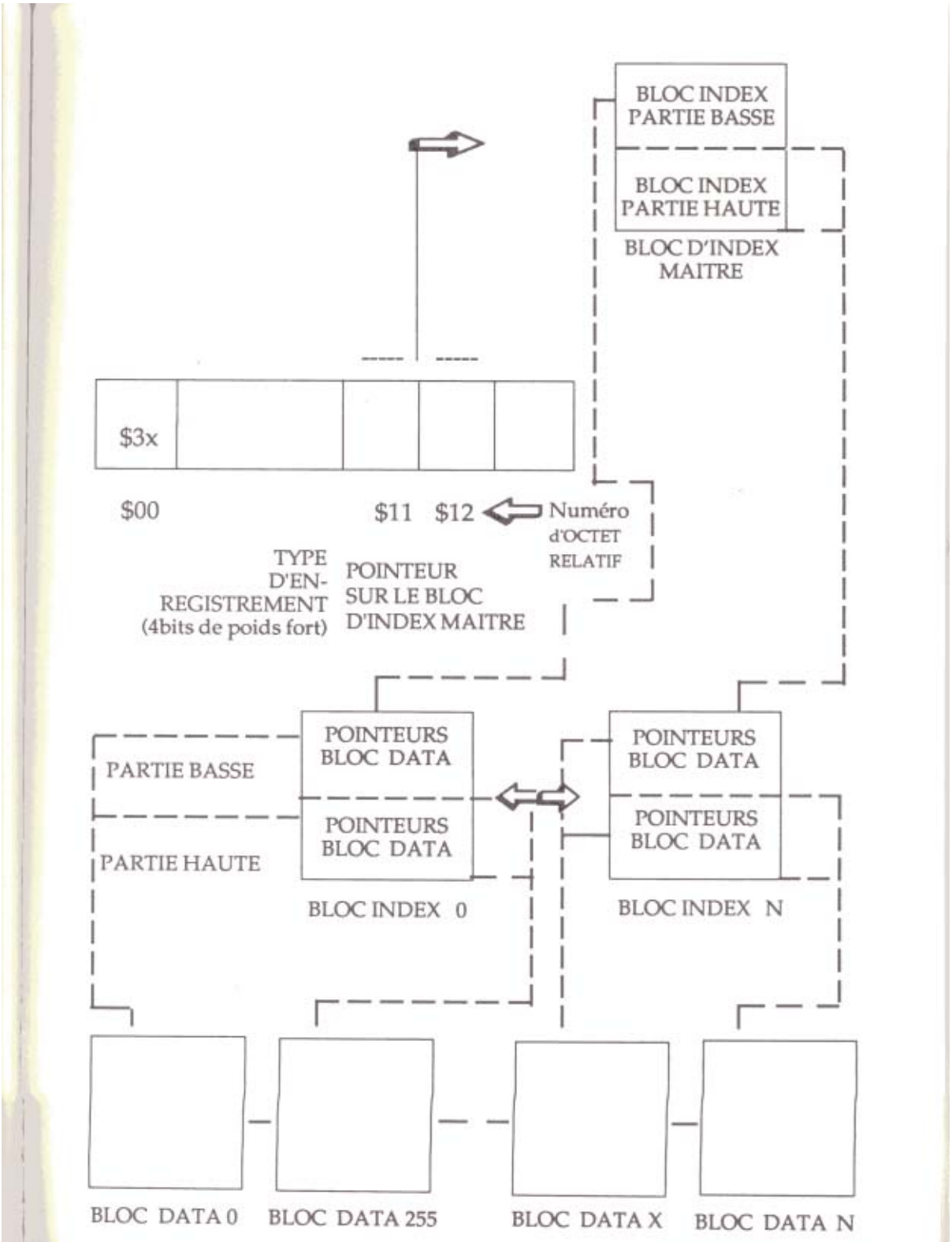
1ER BLOC DE DATAS DU FICHIER :
$0008
POIDS FAIBLE $08
POIDS FORT $00

DERNIER BLOC DE DATAS :
$0016
POIDS FAIBLE $16
POIDS FORT $0

PARTIE HAUTE BLOC INDEX

```
000 0000000000000000000000000000
10C 0000000000000000000000000000
118 0000000000000000000000000000
124 0000000000000000000000000000
130 0000000000000000000000000000
13C 0000000000000000000000000000
148 0000000000000000000000000000
154 0000000000000000000000000000
160 0000000000000000000000000000
16C 0000000000000000000000000000
178 0000000000000000000000000000
184 0000000000000000000000000000
190 0000000000000000000000000000
19C 0000000000000000000000000000
1A8 0000000000000000000000000000
1B4 0000000000000000000000000000
1C0 0000000000000000000000000000
1CC 0000000000000000000000000000
1D8 0000000000000000000000000000
1E4 0000000000000000000000000000
1F0 0000000000000000000000000000
1FC 00000000
```

Dans un bloc d'index, pour trouver un bloc de datas, il suffit de prendre l'octet N pour le poids faible du numéro de bloc, et l'octet relatif N + 256 pour le poids fort de ce numéro de bloc.



Fichier "Tree" - Pour un fichier de type "Tree", le Pointeur Clé pointe sur le numéro de bloc d'un bloc d'index maitre, qui contient une liste ordonnée de tous les numéros de blocs d'index. La taille maximum d'un fichier "Tree" est de 16 MO.

Supposons maintenant que nous modifions encore une fois notre programme Basic pour que 2144 enregistrements soient crées (ligne 20 du programme). Cela va donner une taille de fichier de 137216 octets, bien plus que ce que peut décrire un bloc d'index unique. Le système d'exploitation doit donc "promouvoir" notre fichier "TXTFILE" au niveau de hiérarchie suivant, c'est à dire un fichier de type "Tree". Un fichier "Tree" consiste en un bloc d'index maitre unique référencé dans l'Entrée au Catalogue par le Pointeur Clé. Ce bloc d'Index maitre contient les numéros de blocs des blocs d'index. Ces blocs d'index décrivent comme auparavant les numéros des blocs de datas qui constituent notre fichier. Un bloc d'index maitre peut décrire 256 bloc d'index, chacun d'entre eux pouvant décrire 256

BLOC $010A - BLOC D'INDEX MAITRE

```
000     0A0B00000000000000000000  ..........   DEUX BLOCS D'INDEX
00C     00000000000000000000000000 ..........  $000A - $010B
018     00000000000000000000000000 ..........
024     00000000000000000000000000 ..........
030     00000000000000000000000000 ..........
03C     00000000000000000000000000 ..........
048     00000000000000000000000000 ..........
                    |
                    |
blocs de datas.
100     00010000000000000000000000 ..........
10C     00000000000000000000000000 ..........
118     00000000000000000000000000 ..........
124     00000000000000000000000000 ..........
130     00000000000000000000000000 ..........
13C     00000000000000000000000000 ..........
148     00000000000000000000000000 ..........
                    |
                    |
1FC     00000000
```


BLOC $000A - BLOC D'INDEX 0

```
000     080B0C0D0E0F101112131415
00C     161718191A1B1C1D1E1F2021
018     2223242526272829 2A2B2C2D
024     2E2F30313233343536373839
030     3A3B3C3D3E3F404142434445
03C     464748494A4B4C4D4E4F5051
048     52535455565758595A5B5C5D

100     000000000000000000000000
10C     000000000000000000000000
118     000000000000000000000000
124     000000000000000000000000
130     000000000000000000000000
13C     000000000000000000000000
148     000000000000000000000000

                .
                .
                .
1FC   00000000
                .
                .
```

BLOC $010B - BLOC D'INDEX 1

```
000   0C0D0E0F1011121314151617
00C   000000000000000000000000
018   000000000000000000000000
024   000000000000000000000000
030   000000000000000000000000
03C   000000000000000000000000
048   000000000000000000000000

100   010101010101010101010101
10C   000000000000000000000000
118   000000000000000000000000
124   000000000000000000000000
130   000000000000000000000000
13C   000000000000000000000000
148   000000000000000000000000

                .
                .
                .
1FC   00000000
                .
                .
```

BLOC $0008 - BLOC DATA 0

```
                .
000     5245434F5244300D00000000   RECORD0.....
00C     000000000000000000000000   ............
018     000000000000000000000000   ............
024     000000000000000000000000   ............
030     000000000000000000000000   ............
03C     000000005245434F5244310D   ....RECORD1.
048     000000000000000000000000   ............
```

BLOC $0117 - BLOC DATA 267

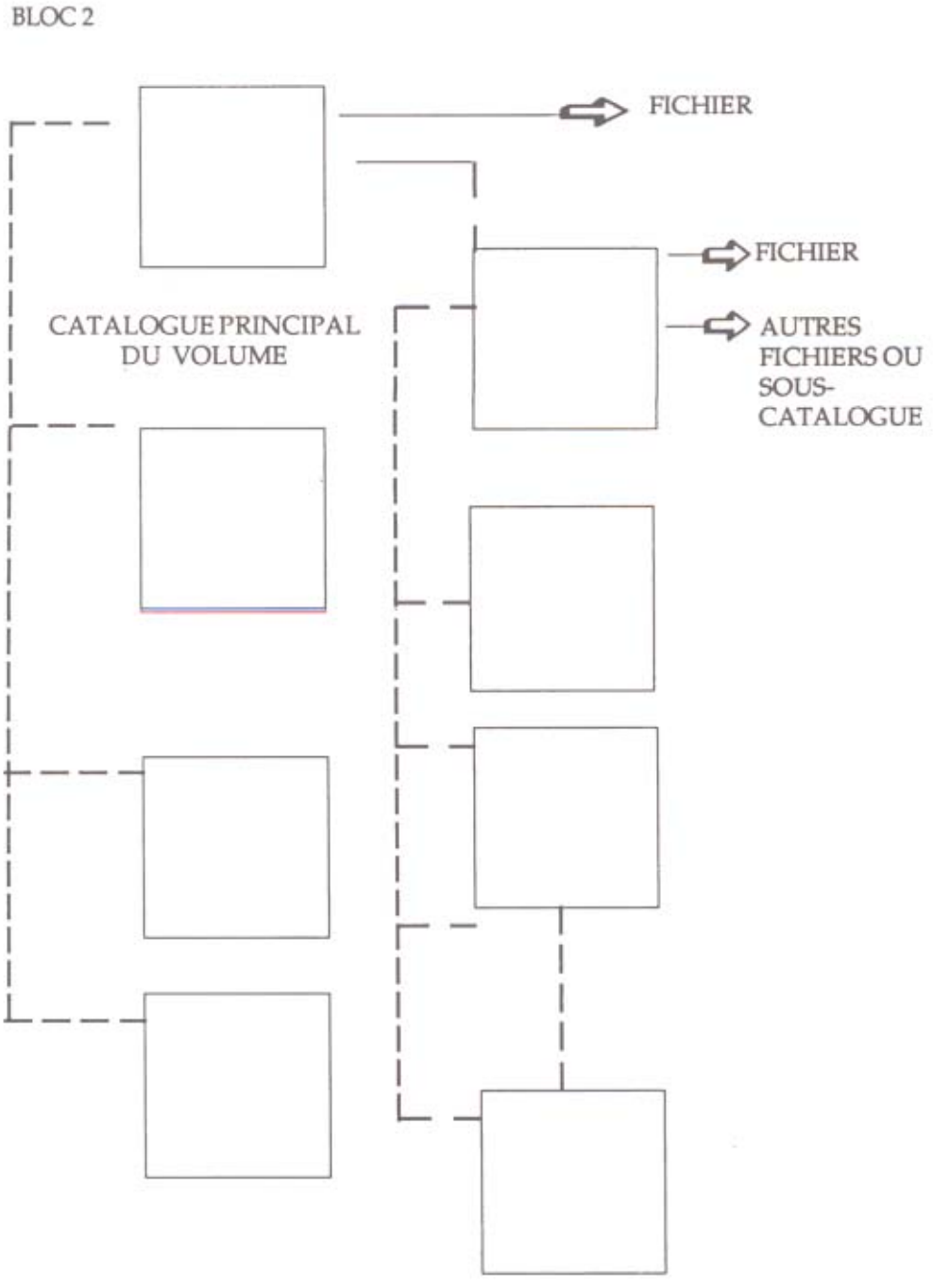
```
000     5245434F5244323133360D00   RECORD2136..
00C     000000000000000000000000   ............
018     000000000000000000000000   ............
024     000000000000000000000000   ............
030     000000000000000000000000   ............
03C     000000005245434F52443231   ....RECORD21
048     33370D0000000000000000     37..........

                .
                .
```


Il faut remarquer ici que le bloc d'index original du fichier "Sapling" (bloc $A) est devenu le premier Bloc d'Index du fichier "Tree". Au moment du changement de type de fichier, le bloc d'Index maitre a été attribué en premier ($10A), puis le second Bloc d'Index ($10B), et enfin le premier bloc de data du second bloc d'index ($10C). Le dernier bloc utilisé pour notre fichier "TXTFILE" est pour les enregistrements "RECORD2136" à "RECORD2143" (pour un total de 2144 enregistrements).

5.1.18 Les Sous-Catalogues ou fichiers de type DIR

Nous avons vu que le catalogue principal d'un volume ProDos ou GS/OS ne peut pas contenir plus de 51 entrées. Sans l'utilisation de sous-catalogues cette limite ne pourrait pas être dépassée. Il faut considérer un sous-catalogue comme une extension du catalogue principal du volume. Dans le cas le plus simple, un sous-catalogue est crée et est considéré comme une entrée placée dans le catalogue principal du volume. Le sous-catalogue a une structure très similaire au catalogue principal : il a une entête placée au début, ses blocs sont doublement référencés par des pointeurs sur les 4 premiers octets de chaque bloc qui le constitue, et il peut contenir des entrées pour d'autres fichiers (ainsi que des sous-catalogues). Contrairement au catalogue principal du volume, il peut avoir une longueur quelconque (au moment de la création il est constitué par un bloc unique auquel on ajoute d'autres blocs quand la taille du sous-catalogue augmente), son entête a une structure légérement différente que l'entête du catalogue principal du volume, il peut être situé n'importe ou sur le Volume, et enfin ses blocs ne sont pas nécessairement contigus.

Voici un diagramme d'une structure classique de sous-catalogues :



On peut voir qu'avec un seul sous-catalogue on peut créer autant de fichiers que l'on a de blocs disponibles sur le Volume. En principe, il est préférable de créer une arborescence de sous-catalogues, chaque sous-catalogue renfermant un certain genre de fichier; fichiers tableur, dessins, programmes personnels, etc ...

On peut considérer ces sous-catalogues comme de mini volumes à l'intérieur d'un volume d'une taille beaucoup plus importante.

Voici la description de la structure de l'entête d'un sous catalogue; les 4 premiers octets de chaque bloc de sous-catalogue pointent pour les deux premiers octets (octets $00-$01) sur le bloc précédent de ce sous-catalogue, les octets $02-$03 pointent sur le bloc suivant de ce sous-catalogue.

ENTETE D'UN SOUS-CATALOGUE

Numéro de l'octet dans le Bloc Clé	SIGNIFICATION
$04	4 BITS PD FORT: TYPE D'ENREGISTREMENT $F = CATALOGUE VOLUME $E = SOUS CATALOGUE 4 BITS PD FAIBLE: LONGUEUR DU NOM DU VOLUME IL SE TROUVE DANS LE CHAMP SUIVANT
$05-$13	NOM DU SOUS-CATALOGUE: 15 OCTETS (ASCII POSITIF) LA LONGUEUR DE CE NOM EST DANS L'OCTET $04 LE RESTE DE CET ENREGISTREMENT EST IGNORE / N'EST PAS UTILISE ICI
$14	DOIT CONTENIR $75
$15-$1B	RESERVE POUR UNE UTILISATION FUTURE EN GENERAL REMPLI AVEC DES 00
$1C-1D	DATE DE CREATION DU CATALOGUE (FORMATAGE) YYYYYYYMMMMDDDD (Y=ANNEE, M=MOIS, D=JOUR) FORME BINAIRE
$1E-$1F	MINUTE ET HEURE DE CREATION DU CATALOGUE 000HHHHH00MMMMMM (H=HEURE, M=MINUTE) FORME BINAIRE

suite entête d'un sous-catalogue

Numéro de l'octet dans le Bloc Clé	SIGNIFICATION
$20	VERSION DE PRODOS QUI A CREEE CE SOUS CATALOGUE
$21	VERSION MINIMUM DE PRODOS POUVANT UTILISER CE SOUS CATALOGUE
$22	CODE D'ACCES PRODOS (VOIR PLUS LOIN) $80 LE SOUS-CATALOGUE PEUT ETRE REFORMATE $40 LE SOUS-CATALOGUE PEUT ETRE RENOMME $20 LE SOUS-CATALOGUE A ETE MODIFIE DEPUIS LE DERNIER BACKUP $02 AUTORISATION D'ECRITURE SUR LE SOUS-CATALOGUE $01 AUTORISATION DE LECTURE SUR LE SOUS-CATALOGUE
$23	LE NOMBRE D'OCTETS UTILISES PAR CHAQUE ENTREE DANS LE SOUS-CATALOGUE ($27)
$24	LE NOMBRE D'ENTREE SUR CHAQUE BLOC DE SOUS-CATALOGUE ($0D). L'ENTETE DU SOUS-CATALOGUE EST CONSIDERE COMME UNE ENTREE
$25-$26	NOMBRE D'ENTREES ACTIVES DANS LE SOUS-CATALOGUE SANS COMPTER L'ENTETE DU SOUS-CATALOGUE UNE ENTREE ACTIVE DECRIT UN FICHIER OU UN SOUS CATALOGUE NON EFFACE.
$27-$28	DANS UN SOUS CATALOGUE C'EST LE POINTEUR INDIQUANT LE BLOC DEFINISSANT L'ENTREE DE CE SOUS-CATALOGUE DANS LE VOLUME PRINCIPAL OU UN SOUS-CATALOGUE
$29	NUMERO DE L'ENTREE FICHIER DANS LE NUMERO DE BLOC POINTE PAR LES OCTETS $27-$28 CE NOMBRE EST COMPRIS ENTRE $01 ET $0D
$2A	LONGUEUR DES ENTREES DANS LE CATALOGUE PERE EN GENERAL $27 (NOMBRE D'OCTETS)

5.1.19 Fichiers Etendus

GS/OS (mais pas ProDos 8) peut créer des fichiers de type étendu. Ces fichiers ont un code pour le type d'enregistrement de $5. Un fichier étendu comprend deux segments de données, le segment de ressource et le segment de données. Le segment de données contient en général les données spécifiques de l'application elle même. Le segment de ressource contient en général les structures de données définissables; ces structures de données définissent des éléments comme par exemple les définitions de menu, les boites de dialogue, etc ...

Le bloc clé pour un fichier étendu n'en est pas vraiment un; c'est simplement une extension de l'entrée du fichier au catalogue. La première moitié du bloc contient les informations du segment de données; la deuxième moitié contient les informations relatives au segment de ressource. GS/OS utilise seulement les huit premiers octets de chaque moitié de bloc. Voici la signification de chacun de ces octets :

$00	CODE DU TYPE D'ENREGISTREMENT POUR LE SEGMENT
$01-$02	NUMERO DU BLOC CLE POUR LE SEGMENT
$03-$04	TAILLE DU SEGMENT (NOMBRE DE BLOCS)
$05-$07	TAILLE DU SEGMENT (NOMBRE D'OCTETS)

Le code du type d'enregistrement pour le segment est soit $01 (Seedling), $02 (Sapling), ou $03 (Tree). Le bloc clé pour le segment d'un fichier étendu (octets $01-$02) est organisé de la même façon que un fichier de type classique.

5.1.20 Les fichiers Sparse

Il est possible de créer des fichiers qui ne sont pas séquentiels. Cela signifie qu'ils sont constitués d'une série d'enregistrements (dont on peut choisir la taille); on peut lire ou écrire sur n'importe le quel de ces enregistrements sans avoir à se positionner sur les précédents. Cela signifie que l'on peut écrire dans un enregistrement même si celui ci est séparé de l'enregistrement écrit précédemment par un grand nombre d'enregistrements vides (remplis de 00). Bien entendu, le système d'exploitation de sauvegarde pas les enregistrements vides car cela prendrait énormément de place sur le volume. En fait, dans le bloc d'index il place des $0000 pour indiquer les blocs de datas qui ne sont pas utilisés; il le seront quand ces enregistrements seront écrits.

Ce genre de fichier est appelé Sparse; il n'occupe pas sur le Volume autant de place que sa taille l'indique.

Créons un fichier Sparse avec le programme en Basic Applesoft qui suit. On suppose que la longueur de chaque enregistrement est de 128 octets, et que l'on écrit seulement dans les enregistrements 2 et 64

```
10          F$= «RANDOM»
30          PRINT CHR$ (4); «OPEN»; F$;»,L128"
40          PRINT CHR$ (4); «WRITE»; F$;»,R2"
50          PRINT «RECORD 2»
60          PRINT CHR$ (4); «WRITE»; F$;»,R64"
70          PRINT «RECORD 64»
80          PRINT CHR$(4); «CLOSE»
```

Le bloc d'index pour le fichier créé par ce programme est le suivant :

```
0000  8C 00 00 00 00 00 00 00   Cela indique que les Blos 0 et 16 de
0008  00  00 00 00 00 00 00 00   datas sont rangés en Bloc $008C et $008E
0010  8E 00 00 00 00 00 00 00   sur le Volume
                .
                .
                .

0100            00 00 00 00 00 00 00 00
0108            00 00 00 00 00 00 00 00
0110            00 00 00 00 00 00 00 00

                .
                .

01F8            00 00 00 00 00 00 00 00
```

BLOC DE DATAS 0 (BLOC $008C)

```
0000            00 00 00 00 00 00 00 00
                .
                .
                .

0100            52 45 43 4F 52 44 20 32        RECORD 2
0108            0D
0109            00 00 00 00 00 00 00 00
                .
                .
                .

01F8            00 00 00 00 00 00 00 00
```


L'enregistrement 2 (RECORD 2) est stocké à la position $100 (2 X 128) dans le fichier; cela correspond à la position $100 dans le premier bloc attribué au fichier (entrée 0 du bloc d'index pointant sur le bloc $008C)

BLOC DE DATAS 16 (BLOC $008E)

0000	52 45 43 4F 52 44 20 36	RECORD 6
0008	34	4
0009	0D	
000A	00 00 00 00 00 00 00 00	
	.	
	.	
	.	
01F8	00 00 00 00 00 00 00 00	

L'enregistrement 64 (RECORD 64) commence à la position $2000 dans le fichier (64 X 128); cela correspond à la position $0000 de la 16 ème entrée du bloc d'Index qui pointe sur le bloc de datas $008E. Les 15 blocs inutilisés entre ces deux enregistrements sont indiqués par des $0000 dans les entrées du bloc d'index.

Ainsi, même si logiquement ce fichier a une longueur de 17 blocs, le système d'exploitation en utilise seulement 3 pour le sauvegarder sur le volume (1 pour le bloc d'index, et 2 pour les blocs de datas).

5.2 Les commandes de GS/OS et de ProDos 8

GS/OS et ProDos 8 ont tous les deux un interpréteur de commandes de bas niveau.

L'interpréteur de commandes de ProDos 8 s'appelle le M.L.I. (Machine Language Interface). Le MLI reconnait 26 commandes.

GS/OS reconnait 47 commandes.

On peut appeler ces différentes commandes à partir d'un programme en langage machine à l'aide d'une technique générale utilisant des protocoles d'appel définis par Apple.

Les protocoles pour ProDos 8 et GS/OS sont de structures similaires mais pas identiques.

Dans cette partie, nous allons examiner de très près les commandes de GS/OS et de ProDos 8, et voir comment on les utilise dans les programmes en Langage machine.

5.2.1 Utilisation des commandes MLI de ProDos 8

Il est très facile d'éxécuter une commande MLI à partir de ProDos 8. Une séquence typique d'appel ressemble à quelque chose comme cela :

```
.
.
.

—> Placer les valeurs dans la table des paramètres
avant d'appeler la commande MLI.

.
.
.

JSR   $BF00      POINT D'ENTREE MLI
DFB   CMDNUM     NUMERO DE COMMANDE MLI
DA    PARMTBL    ADRESSE DE LA TABLE DE PARAMETRES
BCS   ERROR      RETENUE = 1 EN CAS D'ERREUR

.
.
.
```

```
        —> Suite du programme

        .
        .
        .

        RTS

ERROR   —> Routine de traitement d'erreurs

        .
        .
        .

        RTS

PARMTBL DFB  NPARMS   NOMBRE DE PARAMETRES DANS LA TABLE
                      DES PARAMETRES
```

On place dans cette table (PARMTBL) les paramètres dans l'ordre attendu par la commande MLI.

L'instruction clé est le JSR $BF00.

$BF00 est l'adresse du point d'entrée du MLI en mémoire principale. L'interpréteur MLI détermine à quelle commande l'application a fait appel et passe alors le contrôle à la sous routine ProDos 8 correspondante pour traiter cette commande.

Dès que le MLI prend le contrôle, il modifie 4 variables importantes dans la page globale ProDos 8 : MLIACTV ($BF9B), CMDADR ($BF9C-$BF9D), SAVEX ($BF9E), et SAVEY ($BF9F).

Le MLI modifie le bit 7 de MLIACT; il passe de 0 à 1 afin qu'une sous routine de gestion d'interruption puisse déterminer si une condition d'interruption survient pendant le déroulement d'une opération MLI.

Le MLI sauvegarde ensuite la valeurs des registres d'index X et Y dans SAVEX et SAVEY.

Enfin, le MLI range l'adresse de l'instruction suivant immédiatement les 3 octets après l'instruction JSR $BF00 en CMDADR. Le contrôle passera à cette adresse après l'exécution de la commande MLI.

Le MLI détermine quelle commande il doit traiter en examinant la valeur rangée dans l'octet qui suit immédiatement l'instruction JSR $BF00. Cette valeur est un code unique identifiant un numéro de commande.

Si le MLI rencontre un numéro de commande inconnu, une erreur système survient.

Les deux octets suivants le numéro de commande contiennent l'adresse (poids faible - poids fort) de la table des paramètres que la commande MLI va utiliser.

Cette table débute par un octet qui indique le nombre de paramètres contenus dans cette table. Le reste de la table contient des données que le MLI va utiliser pour exécuter la commande.

Après l'exécution de la commande MLI, cette table de paramètres contient les résultats retournés par le MLI.

Nous décrirons plus loin les contenus de la table des paramètres pour chacune des commandes MLI.

Les paramètres qu'une application passe au MLI sont de deux types: pointeurs ou valeurs.

• Un pointeur est une valeur sur deux octets qui pointe sur l'adresse d'une structure de données, par exemple une table de datas (poids faible - poids fort).

• Une valeur sur 1, 2, ou 3 octets est une quantité numérique ayant une signification binaire (octet de poids faible en premier).

Les paramètres retournés par le MLI sont appelés résultats.

Un résultat est en général une quantité numérique sur 1, 2, ou 3 octets (octet de poids faible en premier), mais il peut s'agir aussi d'un pointeur sur 2 octets; tout dépend de la commande MLI appelée.

Si le nombre au début de la table de paramètres ne correspond pas au nombre de paramètres attendus par la commande MLI, une erreur système survient. Autrement, le MLI exécute la commande.

Pendant l'exécution d'une commande, une erreur critique peut survenir.

Les erreurs critiques sont très rares et surviennent seulement

quand des zones de datas ProDos ont été effacées par un programme, ou si une interruption est requise et qu'il n'y a pas de programme d'interruption pour la gérer.

Avec ce genre d'erreur, on est obligé de rebooter tout le système.

Dans ce cas, le MLI exécute l'instruction JSR $BF0C.

La routine en $BF0C (SYSDEATH) fait apparaitre le message suivant :

INSERT SYSTEM DISK AND RESTART - ERR xx

où xx est un code d'erreur héxadécimal ayant 4 valeurs possibles :

$01 = Erreur Interruption
$0A = Bloc Contrôle volume endommagé
$0B = Bloc Contrôle Fichier endommagé
$0C = Bloc d'Allocation endommagé

Il s'agit de structures de datas internes que ProDos 8 utilise pour manipuler les volumes disque et pour ouvrir les fichiers.

Le MLI, une fois la commande traitée récupère les valeurs des registres X et Y (l'application n'a donc pas besoin de les sauvegarder). En cas d'erreur, le MLI exécute l'instruction $BF09. Cette sous routine (SYSERR) range un code d'erreur en $BF0F (SERR).

Finalement, le contrôle est passé à l'instruction qui suit immédiatement le pointeur sur la Table des Paramètres (BCS ERROR dans notre exemple).

Cette adresse a été rangé par le MLI en $BF9C - $BF9D juste au début du traitement.

5.2.2 Utilisation des commandes GS/OS

La procédure générale d'appel d'une commande GS/OS est similaire à l'appel d'une commande MLI ProDos 8.

```
JSL     $E100A8         Point d'entrée GS/OS
DA      COMMANDNUM      Numéro de Commande GS/OS
ADRL    PARMTABLE       Adresse de la Table des Paramètres
BCS     ERROR           Retenue = 1 en cas d'Erreur
```


$E100A8 est l'adresse du point d'entrée de l'interpréteur de commandes de GS/OS. Vous pouvez appeler ce point d'entrée soit en mode natif, soit en mode émulation (voir la partie consacrée au micro-processeur).

Immédiatement après l'instruction JSL $E100A8, il y a le Mot contenant le numéro d'indentification de la commande GS/OS que vous voulez utiliser.

A la suite du numéro de commande, on trouve l'adresse longue (4 octets; octets de poids faible en premier) de la Table des Paramètres nécessaires au fonctionnement de la commande et au rangement des résultats éventuels.

Les paramètres peuvent être des valeurs numériques sur 1 ou 2 Mots (un Mot est constitué de 2 octets), ou des pointeurs longs (4 octets); ils sont rangés dans l'ordre poids faible - poids fort.

La structure exacte de la Table des Paramètres varie d'une commande à une autre, mais elle débute toujours par un nombre indiquant le nombre de paramètres dans la table; ce paramètre s'appelle PCOUNT.
Lorsqu'il a terminé avec le traitement d'une commande, GS/OS ajoute 6 à l'adresse de retour placée dans la pile par l'instruction JSL puis revient avec une instruction RTL.
Cela permet de passer le contrôle au code se trouvant après le pointeur sur la Table des Paramètres (BCS ERROR dans notre exemple).

Au retour, le contenu de tous les registres reste inchangé sauf l'Accumulateur (qui contient un code d'erreur), le Program Counter (PC), et le registre d'état P (les indicateurs m, x, D, I et e sont inchangés, N et V sont indéfinis; C et Z indiquent s'il y a eu une erreur).

A partir de là, on peut vérifier l'état de C pour déterminer s'il y a une erreur :

Si C = 0, il n'y a pas d'erreur
Si C = 1, il y a eu une erreur

On peut aussi examiner l'état de Z. En cas d'erreur, Z = 0.

Le code d'erreur indiquant la nature de cette erreur se trouve dans l'Accumulateur.

S'il n'y a pas d'erreur, l'Accumulateur contient $00.

5.2.3 Les erreurs sous GS/OS et ProDos 8

Une erreur qui n'est pas une erreur critique est appelée une erreur système.

Ces erreurs peuvent avoir diverses origines :

Un pathname non valide, une tentative d'écriture sur un disque protégé, une ouverture d'un fichier inexistant, etc...

Si aucune erreur n'est survenue pendant l'exécution d'une commande, A = 0, C = 0, et Z = 1.

En cas d'erreur, l'Accumulateur contient le code de cette erreur, C = 1 et Z = 0.

Cela signifie que l'on peut utiliser un BCS ou un BNE pour se brancher sur une sous routine de traitement des erreurs.
On doit toujours vérifier si une erreur est survenue après le traitement d'une commande ProDos 8 ou GS/OS. Si on ne le fait pas, on obtient dans la plupart des cas un programme qui ne fonctionne pas toujours parfaitement. Pensez par exemple aux conséquences d'une commande d'écriture dans un fichier qui n'a pas pu etre ouvert parce qu'il n'existe pas.

Voici la liste des codes d'erreur :

$00 Pas d'erreur.

$01 Numéro de commande non valide.

$04 Nombre de Paramètres spécifié dans la Table des Paramètres non valide.

$07 Une commande GS/OS a été demandé par une routine d'interruption.

$10 Un Device spécifié n'a pu être trouvé. GS/OS donne cette erreur après que la commande GetDevNum n'ait pu localiser un Device.

$11 Le numéro de référence du Device n'est pas valide. GS/OS donne cette erreur si le numéro de Device n'est pas dans la liste des Devices actifs.

$22 Paramètre invalide pour un Driver GS/OS.

$23 Le Driver "CONSOLE DRIVER" n'est pas ouvert.

$25 La Table des vecteurs d'Interruption de ProDos 8 est pleine.

$27 Une erreur I/O sur disque 5.25" est survenue empêchant le transfert correct des données. Le volume disque est sans aucun doute abimé. On obtient aussi cette erreur s'il n'y a pas de disque dans le lecteur.

$28 Le Device disque spécifié n'est pas présent. Cette erreur arrive quand on tente d'accéder à un second lecteur quand il n'y a qu'un lecteur de connecté.

$2B Une opération d'écriture a échoué parce que le Volume disque est protégé contre l'écriture.

$2E Une commande a échoué parce que le Volume disque contenant un fichier ouvert a été enlevé de son lecteur.

$2F Le Device spécifié n'est pas en ligne. Cette erreur arrive s'il n'y a pas de disque dans le lecteur 3.5".

$40 La syntaxe du pathname n'est pas valide.

$42 Tentative d'ouverture d'un 9ème fichier. ProDos 8 n'autorise l'ouverture que de 8 fichiers au maximum.

$43 Le numéro de référence d'un fichier n'est pas valide. Cela arrive quand un mauvais numéro de référence est spécifié lors de l'ouverture d'un fichier ou lorsqu'un mauvais numéro de référence est spécifié lors de la fermeture d'un fichier.

$44 Le Pathname spécifié n'a pas été trouvé. Cela signifie que l'un des noms de sous-catalogue, dans un autre pathname valide, n'existe pas.

$45 Le Volume spécifié n'a pas été trouvé. Cela arrive quand par exemple on change le disque du lecteur.

$46 Le fichier spécifié n'a pas été trouvé. Cela signifie que le nom de fichier dans un Pathname correct n'existe pas.

$47 Le nom de fichier spécifié existe déjà. Cette erreur arrive quand on tente de renommer ou de créer un fichier, et que le nom utilisé existe déjà.

$48 Le Volume disque est plein. Il n'y a plus de blocs disponibles sur le Volume disque pour la sauvegarde des données.

$49 Le Catalogue principal du Volume est plein. Seulement 51 fichiers peuvent être rangés dans le Catalogue principal.

$4A Le format du fichier spécifié est inconnu ou incompatible avec la version du système d'exploitation utilisée.

$4B Le type de fichier n'est pas valide ou n'est pas reconnu.

$4C La fin du fichier (EOF) a été atteinte durant une opération de lecture.

$4D La valeur MARK spécifiée est supérieure à EOF.

$4E Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier contrôle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.

$4F La taille du buffer de classe 1 de sortie de GS/OS est trop petite.

$50 Commande non valide parce que le fichier est ouvert. Les commandes Open, Rename, et Destroy ne s'appliquent qu'aux fichiers fermés.

$51 Le nombre de fichiers au catalogue est différent du nombre de fichiers indiqué dans l'entête du catalogue.

$52 Volume Disque non ProDos. La structure de catalogue est inconsistante avec ProDos.

$53 Paramètre non valide. Un paramètre est invalide quand il est hors des limites fixées par le système d'exploitation.

$54 Plus de mémoire disponible.

$55 Cette erreur survient si 8 fichiers sur 8 Devices différents ont été ouverts, et que la commande ONLINE est demandée sur un Device n'ayant pas de fichiers ouverts.

$56 L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map de

ProDos 8, ou parce que le buffer ne débute par sur un saut de page mémoire.

$57 Volume disque en ligne ayant le même nom de catalogue principal.

$58 Le Device spécifié n'est pas organisé en blocs.

$59 Le niveau de priorité passée par la commande GS/OS SetLevel est hors des limites acceptées par le système.

$5A La Bit Map du Volume indique qu'un bloc ayant un numéro supérieur au nombre de blocs disponibles sur le Volume est déclaré libre. La Bit Map du volume a été endommagée.

$5B Changement illégal de Pathname. Cette erreur survient si les Pathnames spécifiés dans la commande ChangePath de GS/OS font référence à 2 Volumes différents. On ne peut déplacer des fichiers seulement entre les catalogues d'un même volume.

$5C Le fichier spécifié n'est pas un fichier système exécutable. GS/OS donne cette erreur si on tente d'utiliser la commande Quit pour passer le contrôle à un fichier qui n'est pas un fichier système GS/OS (S16 ou $B3; EXE ou $B5), ou un fichier système ProDos 8 (SYS ou $FF).

$5D Le système d'exploitation spécifié n'est pas disponible. GS/OS retourne cette erreur quand on tente d'exécuter un programme système ProDos 8 quand le fichier /SYSTEM/P8 ne se trouve pas sur le volume système.

$5E Le Volume /RAM ne peut pas être supprimé.

$5F La pile des adresses de retour des Quit est saturée. GS/OS retourne cette erreur quand on tente de pousser un autre ID d'un programme sur cette pile quand celle-ci est pleine.

$61 Fin de Catalogue. Cette erreur ne peut être retournée que par la commande GS/OS GetDirEntry.

$62 Numéro de classe non valide.

$64 ID invalide pour un fichier Système.

$65 Opération FST invalide.

5.2.4 Buffers de sortie de classe 0 et de classe 1

Même si un pointeur sur une chaine ou sur un buffer est placé dans une Table de Paramètres, ce n'est jamais ProDos 8 ou GS/OS qui a placé ce pointeur. Le système d'exploitation se contente de retourner des données dans le buffer pointé par ce pointeur.

•Sous ProDos 8, c'est l'application qui doit s'allouer un buffer d'une taille convenable et qui doit passer le pointeur sur l'adresse de début de ce buffer dans la Table des Paramètres.

Si vous ne créez pas un buffer d'une taille suffisante, les datas qui suivent ce buffer seront écrasées. Un buffer de ce genre est appelé un buffer de sortie de classe 0.

•GS/OS utilise des buffers de sortie de classe 1 pour éviter l'écrasement de datas pour le cas ou le buffer de sortie n'aurait pas une taille suffisante. Un buffer de classe 1 débute par un mot (2 octets) indiquant le nombre d'octets de ce buffer (en y incluant ce mot).

Quand on appelle une commande qui utilise un buffer de sortie de classe 1, GS/OS vérifie le mot de longueur placé en début du buffer pour voir si la taille du buffer est suffisamment importante. Si cette taille est trop petite, la commande retourne un code d'erreur $4F (buffer trop petit) et retourne la taille du buffer nécessaire dans le mot suivant le mot de longueur du buffer. Si la taille du buffer est suffisante, la commande envoi les datas dans le buffer juste après le mot contenant la longueur du buffer.

5.2.4 Les commandes GS/OS - ProDos 8 par ordre alphabétique

ALLOC_INTERRUPT

GS/OS : n'existe pas

ProDos 8 : $40

Fonction : Placer l'adresse d'une sous-routine de gestion d'interruption dans la table des vecteurs d'interruption de ProDos 8. La table des vecteurs d'interruption peut contenir jusqu'à 4 sous-routines.
Sous GS/OS on utilise la commande BindInt.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (2)
+1	int_num	O	Numéro de référence du Handler d'interruption
+2 à +3	int_code	I	Pointeur sur le Handler d'interruption

Description des paramètres :

num_parms : Nombre de paramètres dans la Table des Paramètres ProDos 8 (toujours 2).

int_num : C'est le numéro de référence que ProDos 8 attribue à la sous-routine de gestion d'interruption. Il faut utiliser ce nombre quand onutilise la commande DEALLOC_INTERRUPT pour retirer cette sous-routine.

int_code : Pointeur sur la sous-routine de prise en chargede l'interruption. ProDos 8 passe le contrôle à cette sous-routine quand une interruption est demandée. La sous-routine de prise en charge de l'interruption doit débuter par une instruction CLD.

Il est important d'installer la sous-routine de prise en charge de l'interruption avant de permettre au périphérique de générer une interruption. Dans le cas contraire, le système plantera si une interruption est demandée avant l'installation de la sous-routine destinée à la prendre en charge.

Codes d'erreur possibles :

$25 : La Table des Vecteurs d'Interruption est pleine. La solution consiste à enlever une sous-routine de prise en charge d'interruption en utilisant la commande DEALLOC_INTERRUPT, puis de ré-essayer.
Exemple de programme :

Voici comment installer une sous-routine ProDos 8 de prise en charge d'interruption qui a été chargée en mémoire à l'adresse $300.

```
JSR   MLI
DFB   $40          ; ALLOC_INTERRUPT
DA    PARMTBL      ;ADRESSE DE LA TABLE DES PARAMETRES
BCS   ERROR        ;BRANCHEMENT EN CAS D'ERREUR
RTS
```

```
PARMTBL  DFB  2        ;NOMBRE DE PARAMETRES
         DS   1        ;LE NUMERO DE REFERENCE REVIENDRA
                       ;ICI
         DA   $300     ;ADRESSE DE LA SOUS-ROUTINE D'INTER-
                       ;RUPTION
```

Votre application devra sauvegarder int_num pour pouvoir utiliser par la suite la commande DEALLOC_INTERRUPT.

Begin Session

GS/OS : $201D

Fonction : Indique à GS/OS de différer toutes les opérations d'écriture qui portent sur la mise à jour de la Bit Map et des blocs de catalogue.
Il n'y a pas de commande équivalente en ProDos 8.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (0)

Description des paramètres :

pcount : C'est le nombre de paramètres dans la Table des paramètrès. Cette valeur est toujours 0.

Codes d'erreur possibles :

aucun

Commentaires : Les sessions d'écriture différée sont utiles quand une application doit transférer rapidement un groupe de fichiers d'un disque sur un autre. Si on n'utilise pas une session d'écriture différée, les opérations de copie seront ralenties parce que la tête de Lecture / Ecriture doit balayer toute la surface du disque pour accéder à la Bit Map et aux blocs de Catalogue avant chaque transfert de fichier. A la fin d'une opération de copie de ce type, il faut utiliser la commande EndSession pour écrire sur le disque les blocs restants. A chaque commande BeginSession doit correspondre une commande EndSession.

ProDos 8 : n'existe pas

BindInt

GS/OS : $2031

Fonction : Permet d'attribuer une sous-routine de gestion d'interruption sous GS/OS vers une source particulière d'interruption. Sous ProDos 8, on utilise la commande ALLOC_INTERRUPT.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (3)
+2 à +3	int_num	O	Numéro de référence de l'interruption
+4 à +5	vrn	I	Numéro de référence du vecteur
+6 à +9	int_code	I	Pointeur sur le Handler d'interruption

Description des paramètres :

pcount : C'est le nombre de paramètres dans la Table de Paramètres GS/OS. La valeur est toujours 3.

int_num : C'est le numéro de référence que GS/OS attribue à la sous-routine de gestion d'interruption. On utilise ce numéro de référence quand on veut enlever la sous-routine avec la commande UnBindInt.

vrn : C'est le numéro de référence qui identifie le type d'interruption système auquel est attribué le Handler d'interruption.

$0008	:	Appletalk (SCC)
$0009	:	Ports Série (SCC)
$000A	:	Scan Line
$000B	:	Fin de Waveform (Ensoniq)
$000C	:	VBL (Vertical Blanking Signal)
$000D	:	Souris (Mouvement ou click)
$000E	:	Timer 1/4 de seconde
$000F	:	Clavier
$0010	:	Octet de réponse ADB pret
$0011	:	SRQ (ADB Service Request)
$0012	:	Accessoire de bureau demandé
$0013	:	Interruption Buffer clavier
$0014	:	Interruption Micro Clavier
$0015	:	Timer 1 seconde
$0016	:	VGC (Video Graphics Controller) (Externe)
$0017	:	Autre source d'interruption

SCC = Serial Communications Controller
ADB = Apple Desktop Bus

int_code : Pointeur sur le début de la routine de gestion d'interruption.

Codes d'erreur possibles :

$25 : La table des vecteurs d'interruption est remplie. Utilisez la commande UnbindInt puis recommencez.

autres codes possibles : $04, $07, $53.

ProDos 8 : n'existe pas

ChangePath

GS/OS : $2004

Fonction : Permet de renommer un fichier ou un Volume disque ou de déplacer un fichier d'un sous-catalogue à un autre sur le même Volume Disque. Vous pouvez modifier le path de tout fichier fermé dont le bit de code d'accès est à 1. Sous ProDos 8, utilisez la commande RENAME pour renommer un fichier ou un Volume Disque. Il n'y a pas de commande ProDos 8 pour déplacer un fichier d'un sous-catalogue à un autre.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de Paramètres (2)
+2 à +5	pathname	I	Pointeur sur le Pathname
+6 à +9	new_pathname	I	Pointeur sur le niveau Pathname

Description des paramètres :

pcount : Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 2.

pathname : Pointeur de classe 1 pointant sur le Pathname courant du fichier qui doit être modifié. Si ce Pathname n'est pas précédé par un séparateur (/ ou :), le système d'exploitation ajoute automatiquement le préfixe par défaut (le préfixe 0/) pour créer le pathname complet.

new_pathname :Pointeur de classe 1 pointant sur le nouveau Pathname du fichier qui va etre modifié. Si ce Pathname n'est pas précédé par un séparateur (/ ou :), le système d'exploitation ajoute automatiquement le préfixe par défaut (le préfixe 0/) pour créer le pathname complet.

Codes d'erreur possibles :

$2B : Le Volume Disque est protégé en écriture.

$40 : Le Pathname contient des caractères non valides, ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).

$44 : Un Catalogue dans un Pathname n'a pas été trouvé.

$45 : Le Catalogue principal n'a pas été trouvé.

$46 : Le fichier n'a pas été trouvé.

$47 : Le nouveau Pathname indiqué existe déjà.

$4E : Le système ne peut pas accéder au fichier. Son code d'accès ne lui permet pas d'etre renommé. Utilisez la commande SetFileInfo pour modifier ce code d'accès.

$50 : Le fichier est ouvert. Cette commande ne fonctionne qu'avec des fichiers fermés.

$5B : Les deux Pathnames indiquent des Volumes différents.

autres codes d'erreur possibles : $07, $27, $4A, $4B, $52, $57, $58.

Exemple de programme :

Supposez que vous vouliez déplacer un fichier appelé ACCESSOIRE d'un sous-catalogue appelé CDA.NDA du disque de Boot vers le Catalogue DESK.ACCS du disk de boot.

```
            JSL     $E100A8         ;Appel GS/OS
            DA      $2004           ;Commande ChangePath
            ADRL    CP_Parms        ;Adresse Table des Paramètres
            BCS     Error           ;En cas d'erreur

CP_Parms    DA      $2              ;Nombre de Paramètres
```

```
        ADRL  NomCourant
        ADRL  NouveauNom

NomCourant  ASC   «*:CDA.NDA:ACCESSOIRE»
NouveauNom  ASC   «*:SYSTEM:DESK.ACCS : ACCESSOIRE»
```

Quand les deux Pathnames spécifiés décrivent le même fichier dans le même sous-catalogue, la commande ChangePath est équivalente à la commande ProDos 8 RENAME.

ProDos 8 : n'existe pas

ClearBackup

GS/OS : $200B

Fonction : Met à 0 le bit Backup dans le code d'accès d'un fichier.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de Paramètres (1)
+2 à +5	pathname	I	Pointeur sur le Pathname

Description des Paramètres :

pcount :Nombre de paramètres dans la Table des Paramètres GS/OS; toujours 1.

pathname :Pointeur GS/OS de classe 1 pointant sur le Pathname courant du fichier à utiliser. Si ce Pathname n'est pas précédé par un séparateur (/ ou :), le système d'exploitation ajoute automatiquement le préfixe par défaut (le préfixe 0/) pour créer le pathname complet.

Codes d'erreur possibles :

$40 : Le Pathname contient des caractères non valides, ou un Path name complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).

$44 : Un Catalogue dans le Pathname n'a pas été trouvé.

$45 : Le Catalogue principal n'a pas été trouvé.

$46 : Le fichier n'a pas été trouvé.

autres codes d'erreur possibles : $07, $4A, $52, $58.

Exemple de programme :

Un programme de copie de fichier doit seulement copier les fichiers qui ont été modifié depuis la dernière opération de copie. Le programme de copie doit vérifier le bit de Backup de chacun des fichiers pour déterminer s'ils doivent être copiés ou non; la copie se fait si le bit de Backup est à 1. GS/OS et ProDos 8 mettent automatiquement ce bit à 1 après chaque opération d'écriture concernant un fichier. Une fois la copie terminée, le programme de copie doit remettre à 0 le bit de Backup en appelant la commande ClearBackup.

```
            JSL     $E100A8         ;Appel GS/OS
            DA      $200B           ;Code de la commande Clear
                                    ;Backup
            ADRL    ParmTbl         ;Adresse de la Table de Para
                                    ;mètres
            BCS     Error           ;En cas d'erreur
ParmTbl     DA      $1              ;Nombre de Paramètres
            ADRL    Pathname        ;Adresse du Pathname
Pathname    ASC     «/DISK/NEW.FILE» ;Fichier cible
```

ProDos 8 : n'existe pas

Close

GS/OS : $2014

Fonction : Permet de fermer un fichier qui a été ouvert. Le système d'exploitation écrit le contenu du buffer attribué à ce fichier sur le Volume et met à jour l'Entrée au Catalogue de ce fichier. Ensuite, le système d'exploitation libère la mémoire utilisée par le buffer de ce fichier.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcour	I	Nombre de paramètres
+2 à +3	ref_nu	I	Numéro de Référence du fichier

CLOSE

ProDos 8 : $CC
Fonction : même chose que pour GS/OS

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (1)
+1	ref_num	I	Numéro de Référence du fichier

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres (1).

ref_num : Numéro de Référence que le système d'exploitation a attribué au fichier quand celui-ci a été ouvert.

Si on fixe ref_num à 0, tous les fichiers ouverts ayant le même niveau de priorité que le fichier système, ou ayant un niveau de priorité supérieur au fichier système sont fermés. Sous ProDos 8, pour fixer la valeur du niveau de priorité d'un fichier, on stocke cetie valeur à l'adresse LEVEL ($BF94). Sous GS/OS, on utilise la commande SetLevel.

Codes d'erreur possibles :

$2B : Le Volume est protégé en écriture.

$43 : Le Numéro de Référence du fichier n'est pas valide. Vous utilisez très certainement le numéro de Référence d'un fichier ayant déjà été fermé.

autres codes d'erreur possibles : $04, $07, $27, $5A.

Exemple de programme :

Pour fermer tous les fichiers ouverts ayant un niveau de priorité supérieur ou égal à 1, on utilise la commande SetLevel et la commande Close avec ref_num = 0. Voici comment procéder sous GS/OS.

```
                JSL     $E100A8
                DA      $201A       ;Commande SetLevel1
                ADRL    ParmTbl1
                BCS     Error       ;En cas d'erreur
                JSL     $E100A8
                DA      $2014       ;Commande Close
                ADRL    ParmTbl2
                BCS     Error       ;En cas d'erreur

ParmTbl1        DA      $1          ;Nombre de Paramètres
                DA      $1          ;Nouveau Niveau de priorité
                                    ;pour le fichier

ParmTbl2        DA      $1          ;Nombre de Paramètres
                DA      $0          ;Numéro de Référence
                                    ;= 0, Fermer tous les fichiers
```

Create

GS/OS : $2001

Fonction : Utilisée pour créer un nouveau fichier. Le système d'exploitation réalise cette fonction en plaçant une nouvelle Entrée dans le Catalogue spécifié.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (7)
+2 à +5	pathname	I	Pointeur sur l'adresse du Pathname
+6 à +7	access	I	Code d'Accès
+8 à +9	file_type	I	Code du Type de Fichier
+10 à +13	aux_type	I	Code du Type Auxiliaire de Fichier
+14 à +15	storage_type	I	Code du Type d'Enregistrement
+16 à +19	eof	I	Taille prévue du segment de données
+20 à +23	resource_eof	I	Taille prévue du segment de ressources

Create

ProDos 8 : $C0

Fonction : même chose que GS/OS
Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (7)
+1 à +2	pathname	I	Pointeur sur l'Adresse du Pathname
+3	access	I	Code d'Accès
+4	file_type	I	Code du Type de Fichier
+5 à +6	aux_type	I	Code du Type Auxiliaire de Fichier
+7	storage_type	I	Code du Type d'Enregistrement
+8 à +9	create_date	I	Date de Création
+10 à +11	create_time	I	Heure de Création

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres (7).

pathname : Un pointeur sur un buffer de classe 0 (ProDos 8) ou de classe 1 (GS/OS) sur l'adresse du Pathname du fichier qui va être crée. Si le Pathname spécifié n'est pas précédé par un séparateur (/ pour ProDos 8; / ou : pour GS/OS), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut (sous GS/OS c'est le préfixe 0/) pour créer le Pathname entier.

access :Voir la partie consacrée à l'Organisation des Fichiers sur Disque pour une explication détaillée du Code d'Accès.

file_type : Code indiquant le Type du Fichier à créer. Ces codes ont déjà été explicités.

aux_type : Code indiquant le Type Auxiliaire du Fichier à créer. La signification de ce code dépend du Type de Fichier. Pour les fichiers SYS, BIN, BAS, et VAR, le Type Auxiliaire de Fichier contient l'adresse de chargement par défaut de ce fichier; pour les fichiers TXT, ce code est la longueur de chaque enregistrement du fichier texte.

storage_type : Ce code indique la façon dont le système d'exploitation a sauvegardé le fichier sur le volume disque;

c'est le Type d'Enregistrement du fichier :

$00-$03	Fichier standard
$05	Fichier de type étendu
$0D	Fichier Sous-Catalogue

On ne peut pas changer ce code une fois que le fichier est créé.
create_date : Date de création du fichier (Année, mois, jour). Si ces octets sont à 0 la Date par défaut est utilisée.
create_time : Heure de création (Heure, Minute). Si ces octets sont à 0 l'Heure par défaut est utilisée.

eof : Si le fichier crée est un fichier standard, ce champ indique la taille prévue pour ce fichier (en nombre d'octets). Le système d'exploitation prévoit un nombre suffisant de blocs pour sauvegarder ce fichier.

Si le fichier crée est un fichier de type étendu, ce champ indique la taille prévue du segment de datas (en nombre d'octets).

Si le fichier crée est un fichier Sous-Catalogue, ce champ indique le nombre d'Entrées prévues dans ce Sous-Catalogue.

resource_eof : Si le fichier crée est de type étendu, ce champ indique la taille prévue du segment de ressource (en nombre d'octets).

Codes d'erreur possibles :

$2B : Le Volume Disque est protégé en écriture.

$40 : Le Pathname contient des caractères non valides, ou un Path name complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).

$44 : Un Catalogue dans un Pathname n'a pas été trouvé.

$45 : Le Catalogue principal n'a pas été trouvé.

$47 : Le nouveau Pathname indiqué existe déjà.

$48 : Le Volume Disque est plein.

$49 : Le Catalogue principal est plein. Seuls 51 fichiers peuvent être sauvegardés dans le catalogue principal.

$4B : Code du Type d'Enregistrement non valide.

autres codes d'erreur possibles : $04, $07, $10, $27, $52, $53, $58.
Exemple de programme :

Voici une petite routine sous GS/OS permettant de créer un fichier Texte (TXT); le nom du fichier TXT est ALLIANCE; le Pathname complet est : GS:ALLIANCE.

```
        JSL   $E100A8
        DA    $2001        ;Commande Create
        Adrl  ParmTbl      ;Table des Paramètres
        BCS   Error        ;En cas d'erreur

ParmTbl DA    $5           ;Nombre de Paramètres
        ADRL  Pathname     ;Adresse du Pathname du fichier à créer
        DA    $E3          ;Code d'Accès standard (non verrouillé)
        DA    $04          ;Type de Fichier = $04 = Fichier TXT
        ADRL  $0           ;Type Auxiliaire (0 = séquentiel)
        DA    $01          ;Type d'Enregistrement = 1 (standard)

Pathname  ASC «:GS:ALLIANCE»
```

DControl

GS/OS : $202E

Fonction : Permet de passer les commandes à un Device GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (5)
+2 à +3	dev_num	I	Numéro de Référence du Device
+4 à +5	control_code	I	Code de contrôle demandé
+6 à +9	control_list	I	Pointeur sur la liste de Contrôle
+10 à +13	request_count	I	Taille de la liste de Contrôle
+14 à +17	transfer_count	O	Nombre d'octets transférés

Description des Paramètres :

pcount : Nombre de paramètres.

dev_num :Numéro de Référence du Device.

control_code : C'est un code qui indique l'opération de contrôle à réaliser :

$0000	reset device
$0001	formatage disque device
$0002	éjection disque device
$0003	réglages paramètres de configuration
$0004	réglage mode attente / pas d'attente
$0005	réglage des options de formatage
$0006	attribution de la partition
$0007	armement du signal
$0008	dé-armement du signal
$0009	réglage carte de partition
$000A-$7FFF	réservé
$8000-$FFFF	opérations spécifiques sur le Device

control_list : Pointeur sur un buffer contenant des données supplémentaires dont GS/OS peut avoir besoin pour réaliser une opération de contrôle.

request_count : Taille du buffer de la Liste de Contrôle.

transfer_count : Nombre d'octets retournés dans le buffer de la Liste de Contrôle. Retournés par le système d'exploitation.

Codes d'erreur possibles :

$11 : Le numéro de Référence du Device n'est pas valide.

$53 : Le paramètre est hors des limites admises.

autre code d'erreur possible : $07

Exemple de programme :

La seule commande de contrôle que vous serez très vraisemblablement amené à utiliser est la commande d'éjection d'un disque.

```
JSR         $E100A8
            DA      $202E       ;DControl
            Adrl    ParmTbl     ;Adresse Table des Paramètres
            BCS     ERROR       ;En cas d'erreur
ParmTbl     DA      $5          ;Nombre de paramètres
            DA      $2          ;Numéro de Device
            DA      $2          ;Code de Contrôle (2 = Eject)
            Adrl    Ctrl_List   ;Adresse de la Liste de Contrôle
            Adrl $0
            DS      $4

Ctrl_List   DS      $4          ;Liste de Contrôle vide
```

On peut déterminer si un disque est éjectable en utilisant la commande DInfo et en examinant le bit 2 du mot des caractéristiques; si le bit est à 1, le disque est éjectable. Pour avoir des informations complètes sur les commandes de contrôle, consultez GS/OS Reference Volume 2.

ProDos 8 : n'existe pas

DEALLOC_INTERRUPT

GS/OS : n'existe pas

ProDos 8 : $41

Fonction : Permet d'enlever l'adresse d'une sous-routine de gestion d'interruption dans la table des vecteurs d'interruption de ProDos 8. Sous GS/OS, on utilise la commande UnBindInt.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (1)
+1	int_num	I	Numéro de référence du Handler d'interruption

Description des Paramètres :

num_parms : Nombre de paramètres

int_num : Numéro d'identification du Handler d'interruption. ProDos 8 attribue ce numéro lorsque le Handler est installé avec la commande ALLOC_INTERRUPT.

Important : Ne pas enlever la sous-routine de gestion d'interruption avant que l'application est indiquée au périphérique d'arrêter de générer des interruptions.

Codes d'erreur possibles :

$53 : Le paramètre int_num n'est pas valide. Il faut utiliser le numéro que la commande ALLOC_INTERRUPT a retourné quand le Handler d'Interruption a été installé.

autre erreur possible : $04

Exemple de programme :

Voici comment enlever de la Table des Vecteurs d'interruption une sous-routine de gestion d'interruption dont le numéro de Référence est 1.

```
            JSL  MLI
            DFB  $41          ;DEALLOC_INTERRUPT
            DA   PARMTBL      ;Adresse de la Table des Paramètres
            BCS  ERROR        ;En cas d'erreur
            RTS
PARMTBL     DFB  $1           ;Nombre de paramètres
            DFB  $1           ;Numéro de code de l'Interruption
```

Destroy

GS/OS : $2002

Fonction : Permet d'enlever un fichier d'un volume disque. Quand on détruit un fichier, le système d'exploitation libère tous les blocs que le fichier utilisait et met à zéro l'octet de longueur dans l'entrée catalogue qui correspond à ce fichier. On ne peut détruire les fichiers dont le bit de code d'accès de destruction est à 1; les fichiers sous-catalogues doivent être vides avant de pouvoir être détruits.

Table de Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +5	pathname	I	Pointeur sur l'Adresse du Pathname

DESTROY

ProDos 8 : $C1

Fonction : même chose que pour GS/OS

Table de Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	num_parms	I	Nombre de paramètres (1)
+2 à +5	pathname	I	Pointeur sur l'Adresse du Pathname

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des paramètres.

pathname : Un pointeur de classe 0 (ProDos 8) ou de classe 1 (GS/OS) sur l'adresse du Pathname du fichier qui va être détruit. Si le Pathname spécifié n'est pas précédé par un séparateur (/ pour ProDos 8; / ou : pour GS/OS), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut (sous GS/OS c'est le préfixe 0/) pour créer le Pathname entier.

Si le Pathname décrit un fichier étendu, les deux segments sont détruits.

Codes d'erreur possibles :

$2B : Le Volume Disque est protégé en écriture.

$40 : Le Pathname contient des caractères non valides, ou un Path name complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).

$44 : Un Catalogue dans un Pathname n'a pas été trouvé.

$45 : Le Catalogue principal n'a pas été trouvé.

$46 : Le fichier n'a pas été trouvé.

$4E : Le système ne peut pas accéder au fichier.

$50 : Le fichier est ouvert; on ne peut détruire que les fichiers fermés. autres codes d'erreur possibles : $04, $07, $10, $27, $4A, $52, $58. Exemple de programme :

Prenons le cas dans lequel le Préfixe 0/ est /DEMO/NUCLEUS. Pour détruire un fichier ayant le pathname entier /DEMO/NUCLEUS/PROG, on peut utiliser la sous-routine GS/OS suivante.

```
            JSL     $E100A8
            DA      $2002       ;Destroy
            Adrl    ParmTbl     ;Table des Paramètres
            BCS     Error       ;En cas d'erreur
            RTS
ParmTbl     DA      $1          ;Nombre de paramètres
            Adrl    Pathname    ;Adresse du Pathname

Pathname    ASC«PROG»
```

Un fichier de type étendu ne peut être détruit que par la commande Destroy de GS/OS.

DInfo

GS/OS : $202C

Fonction : Permet d'avoir des informations sur le Device connecté au système.

Table de Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (10)
+2 à +3	dev_num	I	Numéro de Référence du Device
+4 à +7	dev_name	O	Pointeur sur le nom du Device
+8 à +9	characteristic	O	Caractéristiques du Device
+10 à +13	total_blokcs	O	Capacité du volume en nombre de Blocs
+14 à +15	slot_num	O	Numéro de Slot du Device
+16 à +17	unit_num	O	Numéro d'Unité du Device
+18 à +19	version	O	Numéro de version du driver duDevice
+20 à +21	device_ID_num	O	Numéro ID du Device
+22 à +23	head_link	O	Premier Device en relation
+24 à +25	forward_link	O	Prochain Device en relation

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres. La valeur minimum est 2; la valeur maximum est 10.
dev_num : Numéro de Référence du Device.

dev_name : Pointeur de classe 1 sur un buffer de sortie dans lequel GS/OS a retourné le nom du Device. Un nom de Device peut avoir jusqu'à 31 caractères de longueur; il faut donc fixer la taille de ce buffer à 35 octets.

characteristics : Chacun des bits de ce mot donne les caractéristiques du Device.

bit 15 : 1 = le Device est un Ramdisk ou un Romdisk
bit 14 : 1 = le driver du Device a été généré
bit 13 : réservé
bit 12 : 1 = le Device est actif
bit 11 : réservé
bit 10 : réservé
bit 9 : vitesse du Device (poids fort)
bit 8 : vitesse du Device (poids faible)
bit 7 : 1 = Device organisé en Blocks
bit 6 : 1 = Ecriture autorisée
bit 5 : 1 = Lecture autorisée
bit 4 : réservé
bit 3 : 1 = Formatage autorisée
bit 2 : 1 = Device ejectable
bit 1 : réservé
bit 0 : réservé

Les bits 9 et 8 indiquent la vitesse à laquelle le Device peut fonctionner :

00 : Device 1 MHz
01 : Device 2.6 MHz
10 : Device > 2.6 MHz
11 : La vitesse ne joue pas sur ce Device

total_blocks : Pour un Device organisé en blocs, c'est la capacité du volume en nombre de blocs. Pour un Device Character, ce champ reste à 0.

slot_num : Numéro de Slot du Device.

unit_num : Numéro d'unité du Device utilisé par le SmartPort.

version : Numéro de version du driver du Device

bits 15-12 : Numéro de version principale
bits 11- 8 : Premier numéro de version accessoire
bits 7- 4 : Deuxième numéro de version accessoire
bits 3- 0 : Type de version
$0 =version finale
$A =version alpha
$B =version beta
$E =version expérimentale
$F =version finale non achevée

Par exemple, une version beta 2.12 serait codée par le mot $212B.

device_ID_num : numéro de code qui identifie un type de Device :

$0000 : lecteur de disques 5.25
$0001 : disque dur Profile (5 Mo)
$0002 : disque dur Profile (10 Mo)
$0003 : lecteur de disques 3.5
$0004 : device SCSI générique
$0005 : disque dur SCSI
$0006 : lecteur de bandes SCSI
$0007 : lecteur CD-Rom SCSI
$0008 : imprimante SCSI
$0009 : modem série
$000A : console
$000B : imprimante série
$000C : LaserWriter série
$000D : LaserWriter AppleTalk
$000E : Ram disque

$000F	:	Rom disque
$0010	:	fichier serveur
$0011	:	telephone IBX
$0012	:	device ADB
$0013	:	disque dur générique
$0014	:	lecteur de disques générique
$0015	:	lecteur de bandes générique
$0016	:	device Caractère générique
$0017	:	lecteur de disques de type MFM
$0018	:	device générique réseau AppleTalk
$0019	:	device SCSI à accès séquentiel
$001A	:	scanner SCSI
$001B	:	scanner non SCSI
$001C	:	LaserWriter SCSI
$001D	:	driver AppleTalk principal
$001E	:	driver fichier de service Appletalk
$001F	:	driver AppleTalk RPM

head_link : Numéro de device indiquant quelle est la première entrée dans une liste de numéros de device. Les devices de la liste sont ceux qui pour un volume ont une partition distincte. Si head_link = 0, il n'y a pas de lien.

forward_link : Numéro de device indiquant quelle est la prochaine entrée dans une liste de numéros de device. Les devices de la liste sont ceux qui pour un volume ont une partition distincte. Si forward_link = 0, il n'y a pas de lien.

Codes d'erreur possibles :

$11 : Numéro de Référence du Device non valide.

autre codes d'erreur possible : $07

Exemple de programme :

On peut utiliser la commande DInfo pour connaitre tous les noms de Devices connectés au système. Pour cela, il faut faire une série d'appels à DInfo en incrémentant à chaque appel dev_num de 1, jusqu'à ce que DInfo retourne un code d'erreur de $11 (Numéro de Référence du Device non valide). Le premier dev_num à passer à DInfo doit être 1 puisque c'est le numéro de Device que GS/OS attribue au premier Device qu'il trouve au moment du chargement. Il ne faut pas oublier que le nombre de Devices actifs connectés au système peut changer pendant l'exécution du programme. Par exemple, dans un réseau, des Volumes peuvent se connecter ou se

déconnecter à tout moment. Ainsi, quand on veut constituer une liste de Devices connectés au système, il est préférable de constituer cette liste à chaque fois que l'on en a besoin plutôt que de la constituer une fois pour toute au tout début du programme.

Voici une partie du code pour réaliser ce travail :

```
            LDA   #1
            STA   DevNum

Loop        JSL   $E100A8
            DA    $202C       ;DInfo
            Adrl  ParmTbl     ;Table de Paramètres
            BCS   Exit        ;En cas d'erreur on sort
            .
            .     Affichage
            .

            BRA   Loop

Exit        RTS
ParmTbl     DA    10          ;Nombre de Paramètres
DevNum      DA    $1          ;Numéro de Device
            Adrl  Dev_Space   ;Pointeur sur buffer nom du Device
Dev_Space   DA    35          ;Taille du buffer
Dev_Name    DS    33          ;Nom stocké ici
```

ProDos 8 : n'existe pas

DRead

GS/OS : $202F

Fonction : Permet de réaliser des opérations de lecture de bas niveau sur un Device GS/OS. Sous ProDos 8, on utilise la commande READ_BLOCK.

Table de Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (6)
+2 à +3	dev_num	I	Numéro de Référence du Device
+4 à +7	buffer	O	Buffer des datas
+8 à +11	request_count	I	Nombre d'octets à lire
+12 à +15	starting_block	I	Premier numéro de bloc à lire
+16 à +17	blokc_size	I	Nombre d'octets par bloc
+18 à +21	transfer_count	O	Nombre d'octets lus

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres.

dev_num : Numéro de Référence du Device.

buffer : Pointeur vers un buffer de sortie de classe 0 dans lequel les datas vont être lues.

request_count : Nombre d'octets à lire.

starting_block : Pour un device organisé en blocs, c'est le numéro de bloc où va commencer la lecture. Pour les autres devices, ce champ n'est pas utilisé.

block_size : Taille d'un bloc en nombre d'octets.

transfer_count : Nombre d'octets qui ont été lus sur le Device.

Codes d'erreur possibles :

$11 : Le Numéro de Référence du Device n'est pas valide.

$53 : Paramètre hors des limites autorisées par GS/OS.

autre code d'erreur possible : $07

Exemple de Programme :

DRead est surtout utilisée pour lire le contenu des blocs d'un volume disque. Voici une sous-routine GS/OS permettant de lire les blocs 6 et 7 sur un volume disque organisé en blocs de 512 octets.

```
            JSL     $E100A8
            DA      $202F           ;DRead
            Adrl    ParmTbl
            RTS
ParmTbl     DA      6               ;Nombre de paramètres
            DA      2               ;Numéro de Device
            Adrl    Buffer
            Adrl    1024            ;Lecture de 1024 octets
            Adrl    100             ;En commençant à partir du
                                     Bloc 100
            DA      512             ;512 octets par bloc
            DS      4               ;Résultat tranfer_count

Buffer      DS      1024
```

ProDos 8 : n'existe pas

DStatus

GS/OS : $202D

Fonction : Permet de déterminer les caractéristiques d'un Device GS/OS.

Table de Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (5)
+2 à +3	dev_num	I	Numéro de Référence du Device
+4 à +5	status_code	I	Code de Contrôle
+6 à +9	status_list	O	Pointeur sur la Liste de Contrôle
+10 à +13	request_count	I	Taille de la Liste de Controle
+14 à +17	transfer_count	O	Nombre d'octets transférés

Description des Paramètres :

pcount : Nombre de paramètres dans la table des paramètres.

dev_num : Numéro de référence du device.

status_code : Code indiquant quelle caractéristique est demandée.

$0000	:	caractéristiques du Device
$0001	:	paramètres de configuration
$0002	:	caractéristique attente / pas d'attente
$0003	:	options de formatage
$0004	:	caractéristiques de la partition
$0005-$7FFF	:	réservé
$8000-$FFFF	:	caractéristiques d'appels spécifiques

status_list : Pointeur de classe 0 sur un buffer contenant les caractéristques que la commande retourne.

request_count : Nombre d'octets de caractéristiques à retourner dans la Liste de Contrôle.

transfer_count : Nombre d'octets retournés dans la Liste de Contrôle.

Codes d'erreur possibles :

$11 : Numéro de Référence du Device non valide.

$53 : Paramètre hors des limites autorisées par GS/OS.

autre code d'erreur possible : $07

ProDos 8 : n'existe pas

DWrite

GS/OS : $2030

Fonction : Permet de réaliser des opérations d'écriture de bas niveau sur un Device GS/OS.

Table de Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (6)
+2 à +3	dev_num	I	Numéro de Référence du Device
+4 à +7	buffer	O	Buffer des datas
+8 à +11	request_count	I	Nombre d'octets à écrire
+12 à +15	starting_block	I	Premier numéro de bloc à écrire
+16 à +17	block_size	I	Nombre d'octets par bloc
+18 à +21	transfer_count	O	Nombre d'octets écrits

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres.

dev_num : Numéro de Référence du Device.

buffer : Pointeur vers un buffer contenant les datas qui vont être ecrites.

request_count : Nombre d'octets à écrire.

starting_block: Pour un Device organisé en blocs, c'est le numéro de bloc où va commencer l'écriture. Pour les autres Devices, ce champ n'est pas utilisé.

block_size : Taille d'un bloc en nombre d'octets.

transfer_count : Nombre d'octets qui ont été écrits sur le Device.

Codes d'erreur possibles :

$11 : Le Numéro de Référence du Device n'est pas valide.

$53 : Paramètre hors des limites autorisées par GS/OS.
autre code d'erreur possible : $07

ProDos 8 : n'existe pas

EndSession

GS/OS : $201E

Fonction : Permet de réaliser toutes les opérations d'écriture de blocs disque qui n'ont pas été faites parce qu'une commande BeginSession est active.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (0)

Description des paramètres :

pcount : C'est le nombre de paramètres dans la Table des paramètres. Cette valeur est toujours 0.

Codes d'erreur possible :

aucun

ProDos 8 : n'existe pas

EraseDisk

GS/OS : $2025

Fonction : Permet d'écrire sur un disque le programme de boot, la Bit Map, et la structure du Catalogue principal vide. Cette commande efface donc le volume disque. Contrairement à la commande Format, EraseDisk n'initialise pas le disque; cela signifie qu'on ne peut l'utiliser qu'avec des disques ayant déjà été formatés.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (4)
+2 à +5	dev_name	I	Pointeur sur le nom du Device
+6 à +9	vol_name	I	Pointeur sur le nom du Volume
+10 à +11	file_sys_id	O	Code ID pour le fichier système utilisé
+12 à +13	requested_fsys	I	Code ID pour le fichier système utilisé

Description des Paramètres :

pcount : Nombre de paramètres utilisés dans la Table des Paramètres GS/OS. La valeur minimum est 3; la valeur maximum est 4.

dev_name : Pointeur de classe 1 sur l'adresse de la chaine décrivant le nom du Device à utiliser.

vol_name : Pointeur de classe 1 sur l'adresse de la chaine décrivant le nom du Volume à utiliser. Ce nom doit être précédé par un /.

file_sys_id : Si le champ requested_fsys est à 0, GS/OS affiche une boite de dialogue qui permet à l'utilisateur de choisir le fichier système utilisé sur le volume disque. Au retour, le champ file_sys_id indique quel type de fichier système a été sélectionné.

$01	:	ProDos / SOS
$02	:	Dos 3.3
$03	:	Dos 3.2 / 3.1
$04	:	Apple II Pascal
$05	:	Macintosh MFS
$06	:	Macintosh HFS
$07	:	Macintosh XL (Lisa)
$08	:	Apple CP/M
$09	:	non utilisé
$0A	:	MS-DOS
$0B	:	High Sierra (CD-ROM)
$0C	:	ISO 9660 (CD-ROM)

Si GS/OS retourne 0 dans ce champ, cela signifie que l'utilisateur a annulé l'opération.

requested_fsys : Ce champ contient le code ID du fichier système à écrire sur le Volume Disque. Ce code est le même que pour file_sys_id. Si ce champ est à 0, GS/OS affiche une boite de dialogue permettant de choisir le type de fichier système ; GS/OS retourne le code ID choisi dans le champ file_sys_id.

Codes d'erreur possibles :

$10 : Le nom de Device indiqué n'existe pas.

$40 : Le nom de Volume indiqué contient des caractères non valides, ou ne commence par un séparateur valide (/ ou :).

$5D : Le fichier système indiqué n'est pas reconnu.
autres codes d'erreur possibles : $07, $11, $27.

Exemple de Programme :

Supposez que nous voulions effacer la disquette placée dans un device appelé .APPLEDISK3.5A; et que nous allons lui donner comme nom : BLANK.

```
        JSL  $E100A8
             DA  $2025          ;EraseDisk
             Adrl ParmTbl
             RTS

ParmTbl      DA  4              ;Nombre de paramètres
             Adrl DevName       ;Pointeur sur le nom de Device
             Adrl VolName       ;Pointeur sur le nom de Volume
             DS  2              ;file_sys_id
             DA  0              ;0 = l'utilisateur peut choisir

DevName ASC «.APPLEDISK3.5A»
VolName ASC «:BLANK»
```

ProDos 8 : n'existe pas

ExpandPath

GS/OS : $202E

Fonction : Permet de convertir un nom de fichier, un pathname partiel, ou un pathname complet, en un pathname complet utilisant le : comme séparateur.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de Paramètres (3)
+2 à +5	input_path	I	Pathname devant etre converti
+6 à +9	output_path	O	Pointeur sur le pathname converti
+10 à +11	flags	I	Indicateur de convertion en Majuscules

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres GS/OS. La valeur minimum est 2 ;la valeur maximum est 3.

input_path : Pointeur sur un buffer de classe 1 contenant la chaine décrivant le pathname à convertir.

output_path : Pointeur sur un buffer de classe 1 ou GS/OS va retourner le pathname converti.

flags : Le Bit 15 de cet indicateur montre si les caractères en Minusculesdoivent être convertis en Majuscules.

bit 15 : 1 = convertir en caractères majuscules
0 = ne pas convertir

bits 14 - 0 : doivent toujours être à 0

Codes d'erreur possibles :

$40 : La syntaxe du Pathname n'est pas valide.

$4F : Le Buffer de sortie de classe 1 est trop petit pour contenir le résultat.

ProDos 8 : n'existe pas

Flush

GS/OS : $2015

Fonction : Force le système d'exploitation à écrire le contenu du buffer fichier sur le Volume Disque; le fichier n'est pas fermé.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +3	ref_num	I	Numéro de Référence du fichier

FLUSH

ProDos 8 : $CD

Fonction : même chose pour GS/OS

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (1)
+1	ref_num	I	Numéro de Référence du fichier

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres.

ref_num : Numéro de Référence attribué au fichier quand celui-ci a été ouvert.

Codes d'erreur possibles :

$2B : Le disque est protégé en écriture.

$43 : Le Numéro de Référence du fichier n'est pas valide. Vous devez utiliser un Numéro de Référence d'un fichier qui a déjà été fermé.

autres codes d'erreur possibles : $04, $07, $27, $48

Format

GS/OS : $2024

Fonction : Permet de formatter un disque, d'y écrire le programme de boot, la Bit Map, et une structure de Catalogue vide pour le système d'exploitation indiqué.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (4)
+2 à +5	dev_name	I	Pointeur sur le nom du Device
+6 à +9	vol_name	I	Pointeur sur le nom du Volume
+10 à +11	file_sys_id	O	Code ID pour le fichier système utilisé
+12 à +13	requested_fsys	I	Code ID pour le fichier système utilisé

Description des Paramètres :

pcount : Nombre de paramètres utilisés dans la Table des Paramètres GS/OS. La valeur minimum est 3; la valeur maximum est 4.

dev_name : Pointeur de classe 1 sur l'adresse de la chaine décrivant le nom du Device à utiliser.

vol_name : Pointeur de classe 1 sur l'adresse de la chaine décrivant le nom du Volume à utiliser. Ce nom doit être précédé par un /.

file_sys_id : Si le champ requested_fsys est à 0, GS/OS affiche une boite de dialogue qui permet à l'utilisateur de choisir le fichier système utilisé sur le Volume Disque. Au retour, le champ file_sys_id indique quel type de fichier système a été sélectionné.

$01 : ProDos / SOS
$02 : Dos 3.3
$03 : Dos 3.2 / 3.1
$04 : Apple II Pascal
$05 : Macintosh MFS
$06 : Macintosh HFS
$07 : Macintosh XL (Lisa)
$08 : Apple CP/M
$09 : non utilisé
$0A: MS-DOS
$0B : High Sierra (CD-ROM)
$0C : ISO 9660 (CD-ROM)

Si GS/OS retourne 0 dans ce champ, cela signifie que l'utilisateur a annulé l'opération.

requested_fsys : Ce champ contient le code ID du fichier système à écrire sur le Volume Disque. Ce code est le même que pour file_sys_id. Si ce champ est à 0, GS/OS affiche une boite de dialogue permettant de choisir le type de fichier système. ; GS/OS retourne

le code ID choisi dans le champ file_sys_id.

Codes d'erreur possibles :

$10 : Le nom de Device indiqué n'existe pas.

$40 : Le nom de Volume indiqué contient des caractères non valides, ou ne commence par un séparateur valide (/ ou :).

$5D : Le fichier système indiqué n'est pas reconnu.

autres codes d'erreur possibles : $07, $11, $27.

ProDos 8 : n'existe pas

FSTSpecific

GS/OS : $2033

Fonction : Permet de réaliser des opérations propres à un FST.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (3)
+2 à +3	file_sys_id	I	Code ID du fichier système
+4 à +5	command_num	I	Numéro de commande du FST
+6 à +7/9	command_parm	I/O	Paramètre de la commande ou Résultat

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres.

file_sys_id : Ce champ indique le fichier système que le FST implémente :

$01	:	ProDos / SOS
$02	:	Dos 3.3
$03	:	Dos 3.2 / 3.1
$04	:	Apple II Pascal
$05	:	Macintosh MFS
$06	:	Macintosh HFS
$07	:	Macintosh XL (Lisa)
$08	:	Apple CP/M
$09	:	non utilisé
$0A	:	MS-DOS
$0B	:	High Sierra (CD-ROM)
$0C	:	ISO 9660 (CD-ROM)

command_num : Ce champ contient un numéro de commande spécifique au FST.

command_parm : Ceci peut être une entrée ou un champ de résultat; cela dépend de command_num. Sa signification dépend du FST avec lequel vous communiquez.

Code d'erreur possible :

$53 : Paramètre non valide

autres codes d'erreur possibles : $04, $54

Remarque : Cette commande permet de communiquer avec les FST. La nature de ces opérations varie d'un FST à un autre.

ProDos 8 : n'existe pas

GetBootVol

GS/OS : $2028

Fonction : Permet de déterminer le nom du Volume Disque à partir duquel GS/OS a été lancé.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +5	vol_name	O	Pointeur sur le Nom du Volume

Note : Le nom de Volume que GetBootVol retourne est le meme nom que GS/OS attribue au préfixe */ lors du boot.

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres.

vol_name : Pointeur sur un buffer de classe 1 ou GS/OS retourne le nom du Volume Disque; ce nom est précédé et suivi par le séparateur :
Le buffer doit avoir 35 octets de longueur.

Code d'erreur possible : $07

Exemple de Programme :

```
            JSL   $E100A8
            DA    $2028       ;GetBootVol
            Adrl  ParmTbl
            RTS
ParmTbl     DA    $1          ;Nombre de paramètres
            Adrl  BootSpace   ;Pointeur sur le buffer de sortie
BootSpace   DA    35
BootName    DS    33          ;Emplacement pour le nom
```

En sortie, le nom de Volume est rangé à BootName, il est précédé par un mot indiquant la longueur du nom de Volume.

ProDos 8 : n'existe pas

GET_BUF

GS/OS : n'existe pas

ProDos 8 : $D3

Fonction :Permet de déterminer l'adresse de départ d'un buffer de 1024 octets utilisé par un fichier ouvert.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (2)
+1	ref_nums	I	Numéro de Référence du fichier
+2 à +3	io_buffer	O	Pointeur sur le buffer

Description des paramètres :

num_parms : Nombre de paramètres.

ref_num : Numéro de Référence attribué au fichier quand celui-ci a été ouvert.

io_buffer : Pointeur sur un buffer de 1024 octets utilisé par le fichier ouvert. L'octet de poids faible de ce pointeur est toujours $00, car un buffer commence toujours sur un saut de page mémoire.

Codes d'erreur possibles :

$43 : Le Numéro de Référence du fichier n'est pas valide. Vous devez utiliser le Numéro de Référence d'un fichier déjà fermé.

autre code d'erreur possible : $04

Exemple de Programme :

Vous pouvez utiliser ce programme pour déterminer l'adresse d'un buffer utilisé par un fichier ayant le Numéro de Référence 2. Après l'éxécution de la commande GET_BUF, l'adresse du buffer se trouve en BUFFPTR.

```
        JSR   MLI
        DFB   $D3          ;GET_BUF
        DA    PARMTBL      ;Adresse de la Table des
                           ;Paramètres
        BCS   Error        ;En cas d'erreur
        RTS

PARMTBL DFB   2            ;Nombre de paramètres
        DFB   2            ;Numéro de Référence du fichier
BUFFPTR DS    2            ;L'adresse du Buffer est
                           ;retournée ici
```

GetDevNumber

GS/OS : $2020

Fonction : Permet de déterminer le numéro de référence d'un device en indiquant un nom de device ou un nom de volume.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres
+2 à +5	dev_name	I	Pointeur sur un nom de Device ou de Volume
+6 à +7	dev_num	O	Numéro de Référence du Device

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres. La valeur minimum est 1; la valeur maximum est 2.

dev_name : Pointeur de classe 1 pointant sur l'adresse du buffer contenant le nom du Device ou le nom du Volume. Un nom de Volume doit être précédé par / ou :

dev_num : Numéro de Référence du Device retourné par la commande.

Codes d'erreur possibles :

$10 : Le nom de Device indiqué n'existe pas.

$40 : Le nom de Volume indiqué contient des caractères non valides

ou ne commence pas avec un séparateur valide (/ ou :).

$45 : Le Volume indiqué ne peut etre trouvé.

autres codes d'erreur possibles : $07, $11.

Exemple de Programme :

Voici un exemple de programme pour déterminer le Numéro de Référence d'un Device contenant un disque appelé /PIMP :

```
            JSL     $E100A8
            DA      $2020          ;GetDevNumber
            Adrl    ParmTbl
            RTS

ParmTbl     DA  2                  ;Nombre de paramètres
            Adrl VolName
            DS  2                  ;Numéro de Référence du
                                   ;Device retourné ici

VolName ASC «/PIMP»
```

ProDos 8 : n'existe pas

GetDirEntry

GS/OS : $201C

Fonction : Permet de lire les caractéristiques d'un fichier du Catalogue. GS/OS retourne les informations contenues au catalogue pour un fichier particulier.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (17)
+2 à +3	ref_num	I	Numéro de Référence du fichier
+4 à +5	flags	O	Indicateur pour fichier étendu
+6 à +7	base	I	Code de Base
+8 à +9	displacement	I	Code de Déplacement
+10 à +13	name_buffer	I	Pointeur sur le Nom du fichier
+14 à +15	entry_num	O	Numéro absolu d'Entrée au Catalogue

+16 à +17	file_type	O	Type de fichier
+18 à +21	eof	O	Taille du fichier
+22 à +25	block_count	O	Nombre de blocs utilisés par le fichier
+26 à +33	create_td	O	Heure et Date de création
+34 à +41	modify_td	O	Heure et Date de modification
+42 à +43	access	O	Code d'Accès
+44 à +47	aux_type	O	Type Auxiliaire de fichier
+48 à +49	file_sys_id	O	Code ID du système d'exploitation
+50 à +53	option_list	O	Pointeur sur la Liste Option
+54 à +57	res_eof	O	Taille du segment Ressource
+58 à +61	res_block_count	O	Nb de blocs utilisés par le segment Ressource

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres. La valeur minimum est 5.

ref_num : Numéro de Référence attribué au fichier quand celui ci a été ouvert.

flags : Le Bit 15 de ce mot indique si le fichier est de fichier étendu (bit 15 = 1), ou non (bit 15 = 0).

base : Ce code indique à GS/OS comment calculer le numéro de la prochaine entrée au catalogue à lire. Si base = 0, le déplacement est une entrée au catalogue absolue; si base = 1, GS/OS ajoute le déplacement au numéro courant de l'Entrée pour calculer le numéro suivant d'Entrée; si base = 2, GS/OS soustrait le déplacement au numéro courant de l'Entrée pour calculer le numéro suivantd'entrée. Notez que GS/OS fixe le numéro courant d'entrée à 0 quand il ouvre un fichier pour la première fois, et qu'il le met à jour à chaque fois que l'application appelle GetDirEntry.

displacement : Si base = 0, cela représente le numéro absolu de l'entrée au catalogue à retourner. Autrement, il représente le déplacement sur la prochaine entrée au catalogue à retourner, qui peut être positive ou negative selon la valeur de la base.
Notez que si base et displacement sont tous les deux à 0, GS/OS retourne dans entry_num le nombre total d'Entrées actives dans le sous-catalogue.
Pour lire dans le catalogue une Entrée après l'autre, fixez base et displacement à 1 et appelez GetDirEntry jusqu'à obtenir l'erreur $61 (fin de catalogue).

name_buffer : Pointeur sur un buffer de sortie de classe 1 dans lequel GS/OS va ranger le nom de fichier qu'il a trouvé dans l'entrée au catalogue. Sous les système ProDos et GS/OS la taille du

buffer doit être de 19 octets (15 pour le nom, 2 pour le mot de longueur, et 2 pour le mot indiquant la taille du buffer). GetDirEntry pouvant être utilisée pour lire les catalogues d'autres systèmes d'exploitation utilisant des noms de fichiers plus longs, il faudra ajuster en conséquence la taille de ce buffer.Si le buffer de sortie est trop petit, GetDirEntry retourne la longueur nécessaire au nom de fichier.

entry_num : Numéro absolu de l'Entrée au catalogue de l'Entrée courante.

file_type : Code indiquant le type de fichier.

eof : Valeur contenant la position EOF courante. Cette valeur est égale à la taille du fichier en octets. Si le fichier est de type étendu, ce champ ne s'applique qu'au segment de datas.

block_count : Ce champ contient le nombre total de blocs utilisés par le fichier pour le rangement de ses datas et de ses blocs d'index. Si le fichier est de type étendu, ce champ ne s'applique qu'au segment de datas.

create_td : Heure et Date de création. Les 8 octets sont rangés dans l'ordre suivant :

secondes	
minutes	
heure	
année	Année - 1990
jour	jour du mois - 1
mois	0 = Janvier, 1 = Février, etc ...
non utilisé	
jour de la semaine	1 = Dimanche, 2 = Lundi, etc ...

modify_td : Heure et Date de la dernière modification. Le format est le même que pour create_td.

access : Code d'accès au fichier. Voir partie consacrée à l'organisation sur disque.

aux_type : Type Auxiliaire de fichier.

file_sys_id : Code d'identification du fichier système.

$01 : ProDos / SOS
$02 : Dos 3.3
$03 : Dos 3.2 / 3.1

$04 : Apple II Pascal
$05 : Macintosh MFS
$06 : Macintosh HFS
$07 : Macintosh XL (Lisa)
$08 : Apple CP/M
$09 : non utilisé
$0A : MS-DOS
$0B : High Sierra (CD-ROM)
$0C : ISO 9660 (CD-ROM)

Toutes les autres valeurs sont réservées.

option_list : Pointeur sur un buffer de sortie de classe 1 dans lequel GS/OS retourne des informations spécifiques utilisées par le FST accédant au fichier.

res_eof : Valeur contenant la position EOF courante du segment de ressource pour un fichier de type étendu. Cette valeur est égale à la taille du segment de ressource en octets.

res_block_count : Ce champ contient le nombre total de blocs utilisés par le segment ressource d'un fichier de type étendu pour les blocs de datas et les blocs d'index.

Codes d'erreur possibles :

$4F : Le buffer est trop petit pour contenir le nom du fichier.

$61 : Fin de catalogue. Après avoir obtenu cette erreur, fermez le fichier sous-catalogue que vous avez ouvert avant d'appeler GetDirEntry.

autres codes d'erreur possibles : $07, $27, $43, $4A, $4B, $52, $53, $58

Exemple de Programme :

Voici une sous-routine GS/OS qui permet d'afficher tous les noms de fichiers se trouvant dans un sous-catalogue donné; on appelle continuellement la commande GetDirEntry. En entrée, le pointeur sur le pathname du sous-catalogue doit etre dans les registres A (mot fort) et X (mot faible).

```
Catalog  Début

         STX     Name_Ptr      ;Pointeur sur le pointeur
         STA     Name_Ptr + 2
```

```
            JSL     $E100A8
            DA      $2010           ;Commande Open
            Adrl    ParmTbl1

            LDA     ref_num
            STA     ref_num1
            STA     ref_num2

Lecture     JSL     $E100A8
            DA      $201C           ;GetDirEntry
            Adrl    ParmTbl2
            BCS     EXIT
            .
            .
            .
            .
      AFFICHAGE
            .
            .
            .
            .
            BRA     Lecture

Exit        JSL     $E100A8
            DA      $2014           ;Close
            Adrl    ParmTbl3
            RTS

ParmTbl1    DA      2               ;Nombre de paramètres pour
                                    ;OPEN
ref_num     DS      2               ;Numéro de Référence
Name_Ptr    DS      4               ;Pointeur sur le pathname

Parmtbl2    DA      5               ;Nombre de paramètres pour
                                    ;GetDirEntry
ref_num2    DS      2               ;Numéro de Référence
            DS      2               ;Flags
            DA      1               ;Base = incrémente
            DA      1               ;Displacement = +1
            Adrl    NameBuff        ;Pointeur sur le buffer conte
                                    ;nant le nom

ParmTbl3    DA      1               ;Nombre de paramètres pour
                                    ;Close
```

```
ref_num1 DS       2

NameBuffDA        19          ;Taille du buffer
         DS       2           ;Longueur
         DS       15          ;Nom de fichier

         END
```

ProDos 8 : n'existe pas

GetEOF

GS/OS : $2019

Fonction : Permet de déterminer dans un fichier ouvert la valeur courante du pointeur EOF. Cette valeur représente la taille du fichier.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (2)
+2 à +3	ref_num	I	Numéro de Référence du fichier
+4 à +7	eof	R	Position EOF

GET_EOF

ProDos 8 : $D3

Fonction : même chose que pour GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (2)
+1	ref_num	I	Numéro de Référence du fichier
+2 à +4	eof	O	Position EOF

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres.

ref_num : Numéro de Référence attribué au fichier quand celui-ci a été ouvert.

eof : Valeur représentant la position courante EOF. Cette valeur est égale à la taille du fichier en nombre d'octets.

Code d'erreur possible :

$43 : Le numéro de Référence du fichier n'est pas valide. Vous devez très certainement utiliser un numéro d'un fichier qui a été fermé.

autre codes d'erreur possibles : $04, $07

Exemple de Programme :

```
            JSL     $E100A8
            DA      $2019       ;GetEof
            Adrl    ParmTbl
            BCS     Error       ;En cas d'erreur
            RTS

ParmTbl     DA      2           ;Nombre de paramètres
            DA      1           ;Numéro de Référence du fi
                                ;chier
Position    DS      4           ;Position EOF courante
```

Ce programme permet de connaitre la taille d'un fichier ayant 1 comme Numéro de Référence.

GetFileInfo

GS/OS : $2006

Fonction : Permet de retrouver l'information relative à un fichier stockée dans l'Entrée au catalogue de ce fichier. Cela comprend le code d'Accès, le code du Type de fichier, le code du Type Auxiliaire de fichier, le code du Type d'Enregistrement, le nombre de blocs utilisés par le fichier, et la Date et l'Heure de création et de dernière modification de ce fichier.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (12)
+2 à +5	pathname	I	Pointeur sur le pathname
+6 à +7	access	O	Code d'Accès
+8 à +9	file_type	O	Code Type de fichier
+10 à +13	aux_type	O	Code Type Auxiliaire de fichier
+14 à +15	storage_type	O	Code Type d'Enregistrement
+16 à +23	create_td	O	Heure et Date de Création
+24 à +31	modify_td	O	Heure et Date de dernière Modification
+32 à +35	option_list	O	Pointeur sur la Liste Option
+36 à +39	eof	O	Taille du fichier
+40 à +43	blocks_used	O	Nombre de blocks utilisés par le fichier
+44 à +47	resource_eof	O	Taille du segment de ressource
+48 à +51	resource_blocks	O	Nombre de blocs du segment de ressource

GET_FILE_INFO

ProDos 8 : $C4

Fonction : même chose que pour GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (10)
+1 à +2	pathname	I	Pointeur sur le Pathname
+3	access	O	Code d'Accès
+4	file_type	O	Code Type de fichier
+5 à +6	aux_type	O	Code Type Auxiliaire de fichier
+7	storage_type	O	Code Type d'Enregistrement
+8 à +9	blocks_used	O	Nombre de blocs utilisés par le fichier
+10 à +11	modify_date	O	Date de dernière Modification
+12 à +13	modify_time	O	Heure de dernière Modification
+14 à +15	create_date	O	Date de Création
+16 à +17	create_time	O	Heure de Création

Note : Quand un Pathname pointe sur le nom d'un Volume plutot que sur un nom de fichier, la taille du Volume (en blocs) est retournée dans le champ aux type; le nombre de blocs utilisés par tous les fichiers de ce volume est retourné dans le champ blocks_used.

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres. Sous GS/OS, la valeur minimum est 1.

pathname : Pointeur de classe 0 (ProDos 8) ou de classe 1 (GS/OS) pointant sur l'adresse de la chaine décrivant le Pathname du fichier à utiliser. Si le Pathname spécifié n'est pas précédé par un séparateur (/ pour ProDos 8; / ou : pour GS/OS), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut (sous GS/OS c'est le préfixe 0/) pour créer le Pathname entier.

access : Code d'Accès au fichier. Voir explications dans la partie consacrée à l'organisation des fichiers sur un Volume.
file_type : Code du Type de fichier.

aux_type : Code du Type auxiliaire de fichier.

storage_type : Code décrivant l'organisation physique du fichier sur disque :

$01	:	fichier "seedling"
$02	:	fichier "sapling"
$03	:	fichier "tree"
$04	:	partition Pascal
$05	:	fichier de type étendu
$0D	:	fichier sous-catalogue
$0F	:	fichier catalogue du Volume

blocks_used : Ce champ contient le nombre total de blocs utilisés par le fichier pour le rangement des datas et des blocs d'index. Si le fichier est de type étendu, c'est seulement le nombre de blocs utilisés par le segment de datas. Ce champ n'est pas défini sous GS/OS dans un fichier sous-catalogue.

modify_date : Ce champ, sous ProDos 8 contient la date (heure, mois, année) de dernière modification du fichier. Le format est celui utilisé par ProDos 8 pour le codage de la date.

modify_time : Ce champ, sous ProDos 8 contient l'heure (heure, minute) de dernière modification du fichier. Le format est celui utilisé par ProDos 8 pour le codage de l'heure.

create_date : Ce champ, sous ProDos 8 contient la date (heure, mois, année) de création du fichier. Le format est celui utilisé par ProDos 8 pour le codage de la date.

create_time : Ce champ, sous ProDos 8 contient l'heure (heure, minute) de création du fichier. Le format est celui utilisé par ProDos 8 pour le codage de l'heure.

create_td : Heure et date de création du fichier sous GS/OS. Ces 8 octets représentent les paramètres suivants :

secondes	
minutes	
heure	
année	Année - 1990
jour	jour du mois - 1
mois	0 = Janvier, 1 = Février, etc ...
non utilisé	
jour de la semaine	1 = Dimanche, 2 = Lundi, etc ...

modify_td : Heure et date de dernière modification du fichier sous GS/OS. Ces 8 octets sont rangés dans le même ordre que dans le champ create_td.

option_list : Pointeur sur un buffer de sortie de classe 1 ou GS/OS retourne des informations spécifiques utilisées par le FST pour accéder au fichier.

eof : Taille du fichier en nombre d'octets. Si le fichier est de type étendu, cette valeur ne concerne que le segment de datas. Ce champ n'a pas de signification pour un fichier sous-catalogue.

resource_eof : Si le fichier est de type étendu, cette valeur est la taille du segment de ressource.

resource_blocks : Si le fichier est de type étendu, cette valeur est le nombre de blocs utilisés par le segment de ressource.

Codes d'erreur possibles :

$40 : Le Pathname contient des caractères non valides, ou un Path name complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).

$44 : Un Catalogue dans un Pathname n'a pas été trouvé.

$45 : Le Catalogue principal n'a pas été trouvé.

$46 : Le fichier n'a pas été trouvé.

autres codes d'erreur possibles : $04, $07, $27, $4A, $4B, $52, $53, $58.

Exemple de Programme :

Voir l'exemple donné pour la commande SetFileInfo.

GetFSTInfo

GS/OS : $202B

Fonction : Permet d'obtenir des informations générales sur les caractéristiques d'un FST.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (8)
+2 à +3	FST_num	I	Numéro de Référence du FST
+4 à +5	file_sys_id	O	ID du fichier système
+6 à +9	FST_name	O	Pointeur sur le nom du FST
+10 à +11	version	O	Numéro de version du FST
+12 à +13	attributes	O	Attributs du FST
+14 à +15	block_size	O	Taille en Blocs
+16 à +19	max_vol_size	O	Taille du Volume
+20 à +23	max_file_size	O	Taille du Fichier

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres. La valeur minimum est 1.

FST_num : Numéro de Référence du FST. GS/OS attribue des numéros de référence consécutifs aux différents FST qu'il trouve dans le système.

file_sys_id : Code d'identification pour le fichier système reconnu par le FST :

$01	:	ProDos / SOS
$02	:	Dos 3.3
$03	:	Dos 3.2 / 3.1
$04	:	Apple II Pascal
$05	:	Macintosh MFS
$06	:	Macintosh HFS
$07	:	Macintosh XL (Lisa)
$08	:	Apple CP/M
$09	:	non utilisé
$0A	:	MS-DOS
$0B	:	High Sierra (CD-ROM)
$0C	:	ISO 9660 (CD-ROM)

FST_name : Pointeur sur un buffer de sortie de classe 1, ou GS/OS retourne le nom du FST.

version	:	Numéro de version du FST :
bit 15	:	1 =version prototype 0 =version finale
bits 14- 8	:	Numéro de version principale
bit 7- 0	:	Numéro de version secondaire

attributes : Attributs du FST :

bit15 : 1 = les noms de fichiers doivent etre en Majuscules
bit14 : 1 = FST Caractères; 0 = FST blocks
bit12 : 1 = les noms de fichiers doivent être en ASCII positif

block_size : Taille en octets du nombre de blocks que le FST peut gérer.

max_vol_size : Taille maximum en nombre de blocs des Volumes Disque que le FST peut gérer.

max_file_size : Taille maximum en nombre d'octets du nombre de fichiers que le FST peut gérer.

Codes d'erreur possibles :

$53 : Paramètres hors des limites admises par GS/OS.

autre code d'erreur possible : $07

Commentaires : GS/OS ne donne pas un moyen facile pour déterminer le nombre de FST actifs. Pour avoir des informations sur tous les FST, appelez continuellement GetFSTInfo jusqu'à obtenir une erreur $53.

ProDos 8 : n'existe pas

GetLevel

GS/OS : $201B

Fonction : Permet de déterminer la valeur du niveau d'un fichier système.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +3	level	O	Niveau du fichier systeme

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres.

level : Valeur du Niveau du fichier système. Les valeurs retounées peuvent être comprises entre $0000 et $00FF.

Codes d'erreur possible : $07

Exemple de Programme :

Voici une sous-routine sous GS/OS permettant de retourner le Niveau du fichier système :

```
            JSL  $E100A8
            DA   $201B      ; GetLevel
            Adrl ParmTbl
            RTS

ParmTbl     DA   1          ; Nombre de paramètre
level       DS   2          ; Niveau du fichier Système retourné ici
```

ProDos 8 : n'existe pas

GetMark

GS/OS : $2017

Fonction : Permet de déterminer la valeur courante du pointeur MARK d'un fichier ouvert.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (2)
+2 à +3	ref_num	I	Numéro de Référence du fichier
+4 à +7	position	O	Position courante du Pointeur MARK

GET_MARK

ProDos 8 : $CF

Fonction : même chose que pour GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (2)
+1	ref_num	I	Numéro de Référence du fichier
+2 à +4	position	O	Position courante du Pointeur MARK

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres.

ref_num : Numéro de Référence attribué au fichier quand celui-ci a été ouvert.

position : Position courante de MARK en octets.

Code d'erreur possible :

$43 : Le Numéro de Référence du fichier n'est pas valide.
autre code d'erreur possible : $04, $07.

GetName

GS/OS : $2027

Fonction : Permet d'obtenir le nom de l'application en cours d'exécution.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +5	data_buffer	O	Pointeur sur le nom de l'application

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres.

data_buffer : Pointeur sur un buffer de classe 1 dans lequel GS/OS place le nom de l'application courante. Le nom est une chaine ASCII précédée par un mot de longueur. Ce buffer doit avoir 35 octets de longueur pour pouvoir y placer le nom de l'application.

Codes d'erreur possibles : $07, $4F

Exemple de Programme :

Une application en exécution doit parfois déterminer son propre nom.

```
            JSL   $E100A8
            DA    $2027          ; GetName
            Adrl  ParmTbl
            RTS

ParmTbl     DA    2              ; Nombre de paramètres
            Adrl  NameSpace      ; Pointeur sur le buffer

NameSpaceDA   35                 ; Taille du buffer
Name        DS    33             ; Espace pour ranger le nom
```

ProDos 8 : n'existe pas

GetPrefix

GS/OS : $200A

Fonction : Permet de déterminer le nom d' un Préfixe GS/OS (0/ à 31/).

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (2)
+2 à +3	prefix_num	I	Numéro du Préfixe (0 à 31)
+4 à +7	prefix	O	Pointeur sur le Préfixe

GET_PREFIX

ProDos 8 : $C7

Fonction : Permet de déterminer le nom du Préfixe par défaut.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (1)
+1 à +2	prefix	O	Pointeur sur le Préfixe

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres.

prefix_num : Numéro de Préfixe GS/OS (0 à 31). C'est un nombre binaire et pas une valeur ASCII suivie d'un /.

prefix :Pointeur sur un buffer de sortie de classe 0 (ProDos 8) ou de classe 1 (GS/OS) dans lequel le système d'exploitation va retourner le Préfixe.
Sous ProDos 8, ce buffer doit avoir 67 octets de longueur car un nom de Préfixe peut etre constitué de 64 caractères de long; plus l'octet de longueur, plus les deux délimitateurs /.
Code d'erreur possible :

$56 : L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map de ProDos 8, ou parce que le buffer ne débute pas sur un saut de

page mémoire.

autres codes d'erreur possibles : $04, $07, $4F, $53.

Exemple de Programme :

Cette sous routine GS/OS va lire le Préfixe 7/ et le range dans un buffer débutant en PathName précédé par un mot de longueur.

```
            JSL     $E100A8
            DA      $200A           ;GetPrefix
            Adrl    ParmTbl
            BCS     Error           ;En cas d'erreur
            RTS
ParmTbl     DA      2               ;Nombre de paramètres
            DA      7               ;Numéro du Préfixe (7)
            Adrl    PathBuff

PathBuff    DA      69              ;Taille du buffer
PathName    DS      67
```

Notez que si le Préfixe 7/ n'a pas été défini avec la commande SetPrefix, le mot de longueur du Préfixe retourné par GetPrefix est alors égal à 0.

GetSysPrefs

GS/OS : $200F

Fonction: Permet de déterminer les Préférences système.

Table des Paramètres

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +3	preference	I	Préférences système

Description des Paramètres :

pcount : Nombre de Paramètres dans la Table des Paramètres.

preferences : Mot indiquant les Préférences système :

bit 151 = afficher la boite de dialogue Changement de Volume
0 = ne pas afficher cette boite de dialogue

Commentaires :

Les commandes GS/OS qui utilisent des Pathnames comme paramètres d'entrée, affichent normalement une boite de dialogue demandant à l'utilisateur d'insèrer le Volume Disque demandé si celui ci n'est pas en ligne. Si l'application est capable de gérer les erreurs "Volume non trouvé", il faut utiliser SetSysPrefs pour mettre à 0 le bit 15 du mot de Préférences système.

ProDos 8 : n'existe pas

GET_TIME

GS/OS : n'existe pas

ProDos 8 : $82

Fonction : Permet de lire et de placer la date et l'heure dans la Page globale ProDos 8 aux adresses DATE ($BF90 - $BF91) et TIME ($BF92 - $BF93).

Tables des Paramètres :

Il n'y a pas de Table des Paramètres, la routine appelante doit pointer sur une adresse fictive.

Exemple de Programme :

Quand on utilise cette commande, la date courante (année, mois, jour) et l'heure courante (heure, minute) sont rangées dans la Page globale ProDos 8 aux adresses DATE ($BF90 - $BF91) et TIME ($BF92 - $BF93), dans le format utilisé par ProDos 8. Il faut bien entendu posséder une carte horloge compatible ProDos.

```
JSR  MLI
DFB $82        ; GET_TIME
DA  $0000      ; Table des Paramètres fictive
RTS
```


GetVersion

GS/OS : $202A

Fonction : Permet de déterminer le numéro de version de GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +3	version	O	Numéro de version de GS/OS

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres.

version : Numéro de version courant de GS/OS. L'octet de poids fort contient le numéro de version secondaire; l'octet de poids faible contient le numéro de version principale. Par exemple, la version 2.1 serait codée par $0201. Le bit 7 de l'octet de poids fort s'il est à 1 indique que la version de GS/OS est un prototype (version beta).

Code d'erreur possible : $07

ProDos 8 : n'existe pas

NewLine

GS/OS : $2011

Fonction : Permet d'activer ou de désactiver le mode de lecture "newline". Quand le mode de lecture "newline" est activé, toutes les opérations de lecture successives se terminent à chaque fois qu'un caractère déterminé est lu; c'est le caractère "newline". Quand le mode de lecture "newline" est désactivé, toutes les opérations de lecture se terminent quand la position EOF (fin de fichier) est atteinte, ou quand le nombre indiqué de caractères a été lu.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (4)
+2 à +3	ref_num	I	Numéro de Référence du fichier
+4 à +5	enable_mask	I	Masque d'activation "newline"
+6 à +7	num_chars	I	Nombre de caractères dans la table
+8 à +11	newline_table	I	Pointeur sur la table

NEWLINE

ProDos 8 : $C9

Fonction : meme chose que pour GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (3)
+1	ref_num	I	Numéro de Référence du fichier
+2	enable_mask	I	Masque d'activation "newline"
+3	newline_char	I	Caractère "newline"

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres.

ref_num : Numéro de Référence que le système d'exploitation a attribué au fichier quand celui-ci a été ouvert.

enable_mask : Une opération mathématique AND est faite avec cette valeur et chaque octet lu consécutivement dans le fichier. Si le résultat de l'opération AND est identique à newline_char (sous ProDos 8), ou à l'un des caractères setrouvant dans la table newline_table (sous GS/OS), l'opération de lecture s'arrête; autrement, la lecture continue normalement.
Exception : si enable_mask est égal à 0, le mode de lecture "newline" est désactivé, et les opérations de lecture ne sont pas affectées.

num_chars : Nombre de caractères se trouvant dans la table "newline" des caractères. Si enable_mask est différent de 0, num_chars ne peut pas valoir 0.

newline_table : Pointeur sur une table des caractères "newline". Chaque caractère occupe un octet dans la table qui peut etre constitué au maximum de 256 octets de long.

newline_char : Valeur du caractère new_line. Les opérations de lecture s'arretent automatiquement si l'opération AND de enable_mask et du caractère venant d'être lu est égale à newline_char.

Code d'erreur possible :

$43 : Le Numéro de Référence du fichier n'est pas valide.

autres codes d'erreur possibles : $04, $07.

Exemple de Programme :

En général, on utilise la commande NewLine lorsque l'on veut lire une ligne à la fois dans un fichier texte. Chaque ligne dans un fichier texte est terminée par le code ASCII $0D qui est le code du retour charriot. Pour cela, il suffit de mettre enable_mask égal à $FF et newline_char à $0D avant d'exécuter la commande NewLine. Certaines applications travaillent en ASCII négatif et utiliseront le code ASCII $8D pour le retour charriot. Si vous voulez terminer une opération de lecture pour $0D ou $8D, utilisez un newline_char de $0D et un enable_mask de $7F.

```
            JSL     $E100A8
            DA      $2011       ;NewLine
            Adrl    ParmTbl
            BCS     Error       ;En cas d'erreur
            RTL

ParmTbl     DA      4           ;Nombre de paramètres
            DA      1           ;Numéro de Référence du
                                ;fichier = 1
            DA      $7F         ;enable_mask
            DA      1           ;Nb de caractères new_line
                                ;dans la table
            Adrl    NL_Table    ;Pointeur sur la Table
                                ;New_Line

NL_Table    DA      $0D         ;Retour charriot
```

Null

GS/OS : $200D

Fonction : Permet d'exécuter des événements en attente dans la liste d'attente GS/OS et du Scheduler.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètre (0)

ProDos 8 : n'existe pas

ON_LINE

GS/OS : n'existe pas

ProDos 8 : $C5

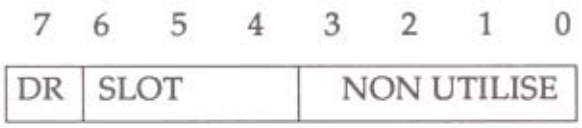
Fonction : Permet de déterminer le nom de Volume d'un disque spécifique, ou tous les noms de tous les Volumes ProDos 8 actifs.
Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (2)
+1	unit_num	I	Numéro d'Unité
+2 à +3	data_buffer	I	Pointeur sur le buffer de datas

Description des Paramètres :

num_parms : Nombre de paramètres dans la Table des Paramètres.

unit_num : Numéro de slot et de drive auquel on veut accéder.



SLOT peut en fait être le numéro réel ou logique d'un slot si le système contient des périphériques disque comme par exemple un RAM disque. Par exemple, le numéro d'unité d'un volume /RAM

connecté en slot 3, Drive 2 sur un IIe, IIc, II GS est $B0 soit 1 011 0000.

DR indique le numéro de DRive (lecteur) : 0 pour le lecteur 1 et 1 pour le lecteur 2. Plus de deux lecteurs peuvent etre connectés au port 5 du SmartPort. Dans ce cas, ProDos 8 assigne logiquement les deux prochains lecteurs en Slot 2, Drive 1 et en Slot 2, Drive 2. ProDos 8 ignore tous les lecteurs connectés au SmartPort après le quatrième.

Exception : Si unit_num est égal à 0, les noms de Volume de tous les drives sont retournés.

data_buffer : Pointeur sur un buffer contenant le nom de Volume du Drive spécifié. Si unit_number = 0, les noms de Volume de tous les drives sont retournés. Chaque nom de Volume est constitué d'un champ de 16 octets.
Le premier octet de chaque champ de 16 octets contient le numéro de Drive et de slot de ce Volume disque puis la longueur du nom de ce Volume. Le format utilisé est le suivant :

7	6	5	4	3	2	1	0
DR	SLOT			LONGUEUR DU NOM			

DR et SLOT ont le même format que unit_num. On trouve la longueur du nom de Volume dans les 4 bits de poids faible. Les 15 octets suivants contiennent le nom de Volume; ce nom n'est pas précédé par un /.

Si unit_num est égal à 0, l'enregistrement après le dernier champ de 16 octets commence par $00. Vous devez réserver un buffer de 256 octets si vous appelez ON_LINE avec unit_num = 0.

Codes d'erreur possibles :

$27 : Le disque est illisible. Il est très certainement endommagé. Cette erreur peut aussi survenir si la porte du lecteur est ouverte, ou s'il n'y a pas de disquette dans le lecteur.

$28 : Pas de périphériques de connectés. ProDos 8 retourne cette erreur quand on essaie d'accéder à un second lecteur 5.25 qui n'existe pas.

$2E : Un disque a été enlevé de son lecteur, alors qu'un fichier était encore ouvert.

$2F : Le périphérique n'est pas prêt. Il n'y a pas de disque dans le lecteur 3.5.

$52 : Le disque dans le lecteur indiqué par unit_num n'est pas un disque formaté sous ProDos.

$56 : L'adresse du buffer pour le Pathname n'est pas valide; elle est déclarée occupée dans la Bit Map système de ProDos 8.

autres codes d'erreur possibles : $04, $55.

ON_LINE traite les erreurs d'une manière différente des autres commandes MLI. En cas d'erreur, "LONGUEUR DU NOM" prend la valeur 0, et le code d'erreur est stockée dans le deuxième octet de l'enregistrement de 16 octets correspondant. Le code d'erreur n'est pas rangé dans l'accumulateur (A), l'indicateur de carry (C) n'est pas mis à 1.

Exemple de Programme :

Ce programme lit le nom de Volume d'un disque placé en Slot 6, Drive 2.

```
        JSR   MLI
        DFB   $C5        ;ON_LINE
        DA    PARMTBL    ;Adresse Table des Paramètres
        BCS   ERROR      ;Branchement en cas d'erreur
        RTS

PARMTBL DFB   2          ;Nombre de paramètres
        DFB   $E0        ;unit_num = slot 6, drive 2
        DA    BUFFER     ;Pointeur sur le buffer du path
                         ;name

BUFFER  DS    1          ;slot / drive (bits 7-4) et Longueur
                         ;du nom de Volume (bits 3-0)
        DS    15         ;Nom du Volume (ASCII)
```

Si par exemple, le nom de Volume est ASM.FILES, l'octet stocké en BUFFER est $E9, et les octets stockés à partir de BUFFER + 1 sont 41 53 4D 2E 46 49 4C 45 53.

Ce sont les codes ASCII des caractères de ASM.FILES.

Open

GS/OS : $2010

Fonction : Permet de préparer un fichier pour des opérations de lecture et d'écriture à venir. Quand on ouvre un fichier, le pointeur MARK pointe sur le début du fichier (MARK = 0), et le niveau du fichier est égal au niveau du fichier système. Sous GS/OS, la commande renvoie aussi tous les attributs du fichier.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (15)
+2 à +3	ref_num	O	Numéro de Référence du fichier
+4 à +7	pathname	I	Pointeur sur le Pathname
+8 à +9	request_access	I	Type d'accès demandé
+10 à +11	resource_num	I	Type de Segment
+12 à +13	access	O	Code d'Accès
+14 à +15	file_type	O	Type de Fichier
+16 à +19	aux_type	O	Type Auxiliaire de Fichier
+20 à +21	storage_type	O	Type d'Enregistrement
+22 à +29	create_td	O	Heure et Date de Création
+30 à +37	modify_td	O	Heure et Date de Modification
+38 à +41	option_list	O	Pointeur sur la liste Option
+42 à +45	eof	O	Taille du fichier
+46 à +49	blocks_used	O	Blocs utilisés par le fichier
+50 à +53	resource_eof	O	Taille du segment de Ressource
+54 à +57	resource_blocks	O	Blocs utilisés par le segment Ressource

OPEN

ProDOS 8 : $C8

Fonction : même chose que pour GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (3)
+1 à +2	pathname	I	Pointeur sur le Pathname
+3 à +4	io_buffer	O	Pointeur sur le buffer I/O
+5	ref_num	O	Numéro de Référence du fichier

Important : On ne peut ouvrir qu'un fichier qui est fermé, mais si un fichier est ouvert, et que son code d'accès d'autorisation d'écriture n'est pas à 1 (c'est à dire que l'on ne peut pas y écrire), ce fichier peut etre ouvert plus d'une fois.

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres. Sous GS/OS, la valeur minimum est 2.
ref_num : Numéro de Référence attribué au fichier par le système d'exploitation. De très nombreuses commandes utilisent ce Numéro de Référence pour identifier le fichier. Le niveau du fichier est le même que pour le fichier système.

pathname : Pointeur de classe 0 (ProDos 8) ou de classe 1 (GS/OS) pointant sur l'adresse de la chaine décrivant le Pathname du fichier à utiliser. Si le Pathname spécifié n'est pas précédé par un séparateur (/pour ProDos 8; / ou : pour GS/OS), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut (sous GS/OS c'est le préfixe 0/) pour créer le Pathname entier.

request_access : Ce mot décrit le mode d'accès

bit 1 1 = écriture demandée
bit 0 1 = lecture demandée

Par exemple, on ne peut pas demander une écriture sur un lecteur de CD-ROM.

Si ce mot est égal à $0000, l'accès demandé est le même que celui permis par access_code.

io_buffer : Pointeur sur un buffer de 1024 octets. L'octet de poids faible de ce pointeur doit etre à $00; cela signifie que ce buffer doit débuter sur un saut de page.
La première moitié de ce buffer contient une copie des blocs de datas auxquels on vient d'accéder; la deuxième moitié contient les blocs d'index.

resource_num : Si le fichier est de type étendu, ce mot indique à GS/OS quel segment il faut ouvrir :

$0000	ouvrir le segment de datas
$0001	ouvrir le segment de ressource

Note : Le reste des paramètres dans la Liste des Paramètres GS/OS

est identique à celle de la commande GetFileInfo.

Codes d'erreur possibles :

$40 : Le Pathname contient des caractères non valides, ou un Path name complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).

$42 : Tentative d'ouverture d'un 9eme fichier. Sous ProDos 8, on ne peut ouvrir que 8 fichiers simultanément.

$44 : Un Catalogue dans un Pathname n'a pas été trouvé.

$45 : Le Catalogue principal n'a pas été trouvé.

$46 : Le fichier n'a pas été trouvé.

$50 : Le fichier est ouvert. Cette commande ne fonctionne qu'avec des fichiers fermés.

$56 : L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map de ProDos 8, ou parce que le buffer ne débute pas sur un saut de page mémoire.

autres codes d'erreur possibles : $04, $07, $27, $4A, $4B, $52.

Exemple de Programmes :

La sous-routine suivante ouvre un fichier appelé SESAME qui réside dans un sous catalogue ayant pour préfixe 0/.

```
            JSL  $E100A8
            DA   $2010          ; Open
            Adrl ParmTbl
            BCS  Error          ; En cas d'erreur
            RTS

ParmTbl     DA   2              ; 2 paramètres
            DS   2              ; ref_num est retourné ici
            Adrl Pathname       ; Pointeur sur le Pathname

Pathname ASC «SESAME»           ; Nom du fichier
```

GS/OS retourne un code d'erreur de $46 si vous tentez d'ouvrir un fichier qui n'existe pas. Une fois que le fichier est ouvert, vous devez récupérer le Numéro de Référence du fichier pour pouvoir l'utiliser avec les autres commandes.

OSShutdown

GS/OS : $2003

Fonction : Permet de déconnecter GS/OS avant de rebooter ou d'éteindre le système.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +5	shutdown_flag	I	Pointeur sur le Pathname suivant

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres.

shutdown_flag : Les deux bits de poids faible de cet indicateur contrôle le mécanisme de déconnection de GS/OS :

bit 0 : 1 = GS/OS est déconnecté et le système est rebooté

0 = GS/OS est déconnecté et l'utilisateur a le choix entre rebooter ou éteindre le système.

bit 1 : 1 = Le Ram Disque est laissé intact

0 = Le Ram Disque est initialisé

Commentaires : Quand GS/OS est déconnecté, il écrit tous les blocs qui se trouvent dans la mémoire cache, il ferme tous les NDA ... Cette commande doit seulement etre utilisée par les sélecteurs de programmes comme le Finder ou ProSel 16, et pas par les applications.

ProDos 8 : n'existe pas

Quit
GS/OS : $2029

Fonction : Permet de quitter l'application courante. Le controle est repassé au sélecteur de programme.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (2)
+2 à +5	pathname	I	Pointeur sur le prochain Pathname
+6 à +7	flags	I	Indicateurs Retour / Reboot

QUIT

ProDos 8 : $65

Fonction : Même chose que pour GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (4)
+1	quit_type	I	Type du Quit
+2 à +3	pathname	I	Pointeur sur le prochain Pathname
+4	réservé	I	Zone réservée
+5 à +6	réservé	I	Zone réservée

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres. Sous GS/OS le nombre de paramètres minimum est 0.

pathname : Pointeur sur un buffer de classe 0 (ProDos 8) ou de classe 1 (GS/OS) contenant le pathname du prochain fichier système à lancer. Le type de fichier doit etre $FF (ProDos 8) ou $B3 !GS/OS). Le pathname ne peut pas résider en page 2 car la commande Quit utilise cette zone mémoire. Sous ProDos 8, ce champ doit etre à 0 si quit_type est $00.
quit_type : Code du type de Quit pour ProDos 8. Il y a deux types de Quit sous ProDos 8; $00 Quit standard, $EE Quit vers un programme système. Le code $EE peut seulement etre utilisé si le fichier système a été lancé depuis GS/OS.

flags :Indicateurs du type de Quit; seuls les bits 15 et 14 sont significatifs. Si le bit 15 est à 1, le UserID du programme est placé dans la pile de Quit, afin que le programme puisse être relancé. Si le bit 14 est à 1, le programme est relancé à partir de la mémoire.

Codes d'erreur possibles :

$46 : Le fichier n'a pas été trouvé.

$5C : Le fichier indiqué par le Pathname n'est pas un programme exécutable. Ce Pathname doit etre un programme système ProDos 8 (type de fichier $FF) ou un programme système GS/OS (type de fichier $B3).

$5D : Le Pathname indique un programme système ProDos 8, mais le fichier P8 (qui contient le système d'exploitation ProDos 8) n'est pas présent dans le sous-catalogue /SYSTEM du disque de boot de GS/OS.

$5F : La pile des adresses de retour des Quit est saturée. GS/OS retourne cette erreur quand on tente de pousser un autre ID d'un programme sur cette pile quand celle ci est pleine.

autres codes d'erreur possibles : $04, $07, $40, $5E.

Read

GS/OS : $2012

Fonction : Permet de lire dans un fichier ouvert des octets à partir de la position courante du pointeur MARK. Après l'opération de lecture, le système d'exploitation incrémente MARK du nombre d'octets qui ont été lu dans le fichier. L'opération de lecture se termine quand le nombre indiqué d'octets a été transféré, quand un caractère "newline" est rencontré, ou quand la fin du fichier est atteinte.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (5)
+2 à +3	ref_num	I	Numéro de Référence du fichier
+4 à +7	data_buffer	I	Pointeur sur le début du buffer de data
+8 à +11	request_count	I	Nombre d'octets à lire
+12 à +15	transfer_count	O	Nombre d'octets lus
+16 à +17	cache_priority	I	Niveau de priorité du cache

READ

ProDos 8 : $CA

Fonction : même chose que pour GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (4)
+1	ref_num	I	Numéro de Référence du fichier
+2 à +3	data_buffer	I	Pointeur sur le début du buffer de data
+4 à +5	request_count	I	Nombre d'octets à lire
+6 à +7	tranfer_count	O	Nombre d'octets lus

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres. Sous GS/OS, la valeur minimum est 2.

ref_num : Numéro de Référence attribué au fichier par le système d'exploitation quand celui-ci a été ouvert.

data_buffer : Pointeur sur le début du buffer dans lequel les datas du fichier vont etre placées. La taille du buffer doit etre égal à request_count.

request_count : Nombre de caractères à lire dans le fichier. Ils sont lus dans le fichier et placés dans le buffer pointé par data_buffer.

transfer_count : Nombre de caractères qui ont été lus dans le fichier. Il est en général égal à request_count, mais peut etre inférieur si le système d'exploitation atteint la fin du fichier, ou si le mode de lecture est actif et qu'un caractère "newline" est lu (voir la

commande NewLine).

cache_priority : Ce code indique comment GS/OS va utiliser le cache pour y placer les blocks lors de la lecture du fichier.

$0000 : pas de cache pour ce fichier
$0001 : le cache est utilisé pour ce fichier

Codes d'erreur possibles :

$43 : Le numéro de référence d'un fichier n'est pas valide.

$4C : La fin du fichier (EOF) a été atteinte durant une opération de lecture.

$4E : Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier contrôle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.

$56 : L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map de ProDos 8, ou parce que le buffer ne débute par sur un saut de page mémoire.

autres codes d'erreur possibles : $04, $07, $27.

Exemple de Programme :

La sous-routine suivante lit $1000 octets dans un fichier ayant comme Numéro de Référence 1 à l'adresse mémoire Buffer.

```
            JSL     $E100A8
            DA      $2012       ;Read
            Adrl    ParmTbl
            BCS     Error       ;En cas d'erreur
            RTS
ParmTbl     DA      4           ;Nombre de paramètres
            DA      1           ;Numéro de Référence du
                                ;fichier
            Adrl    Buffer      ;Pointeur sur le buffer de
                                ;datas
            Adrl    $1000       ;Nombre d'octets à lire
TransCnt    DS      4           ;Nombre d'octets lus

Buffer      DS      $1000       ;Buffer pour les datas
```


Après chaque appel à cette sous-routine, vous devez examiner le nombre sur 4 octets stocké en TransCnt, pour déterminer le nombre d'octets qui ont été lus. Ce nombre peut etre inférieur à $1000 si GS/OS atteint la fin du fichier (EOF), ou s'il rencontre un caractère "newline"
Si la commande Read renvoie le code d'erreur $4C (Fin de fichier), il n'y a plus d'octets à lire, et on peut donc fermer le fichier.

READ_BLOCK

GS/OS : n'existe pas

ProDos 8 : $80

Fonction : Permet de transférer un block de datas (512 octets) d'un périphérique disque vers un buffer en mémoire. Sous GS/OS, on utilise la commande DRead.

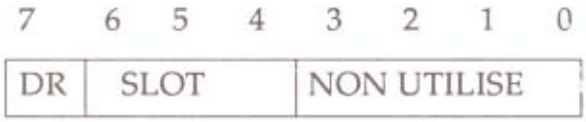
Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (3)
+1	unit_num	I	Unit number
+2 à +3	data_buffer	O	Pointeur sur le buffer de datas
+4 à +5	block_num	I	Numéro du bloc à lire

Description des Paramètres :

num_parms : Nombre de paramètres dans la Table des Paramètres.

unit_num : Numéro de slot et de drive auquel on veut accéder.



SLOT peut en fait être le numéro réel ou logique d'un slot si le système contient des périphériques disque comme par exemple un RAM disque. Par exemple, le numéro d'unité d'un volume /RAM connecté en slot 3, Drive 2 sur un IIe, IIc, II GS est $B0 soit 1 011 0000.

DR indique le numéro de DRive (lecteur) : 0 pour le lecteur 1 et 1 pour le lecteur 2. Plus de deux lecteurs peuvent être connectés au port 5 du SmartPort. Dans ce cas, ProDos 8 assigne logiquement les

deux prochains lecteurs en Slot 2, Drive 1 et en Slot 2, Drive 2. ProDos 8 ignore tous les lecteurs connectés au SmartPort après le quatrième.

data_buffer : Pointeur sur le début d'un block mémoire de 512 octets qui contient le bloc lu par READ_BLOCK.

block_num : Numéro du bloc à lire. Les valeurs autorisées dépendent du périphérique disque.

0 - 279 pour les lecteurs 5.25
0 - 1599 pour les lecteurs 3.50
0 - 127 pour les volumes /Ram

Vous pouvez déterminer la taille d'un périphérique en utilisant la commande GET_FILE_INFO.

Codes d'erreur possibles :

$27 : Le disque est illisible. Il est très certainement endommagé. Cette erreur peut aussi survenir si la porte du lecteur est ouverte, ou s'il n'y a pas de disquette dans le lecteur.

$28 : Pas de périphériques de connectés. ProDos 8 retourne cette erreur quand on essaie d'accéder à un second lecteur 5.25 qui n'existe pas.

autre codes d'erreur possibles : $04, $07, $11, $2F, $53, $56.

RENAME

GS/OS : n'existe pas

ProDos 8 : C2

Fonction : Permet de modifier le nom d'un fichier.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (2)
+1 à +2	curr_name	I	Pointeur sur le Pathname courant
+3 à +4	new_name	I	Pointeur sur le nouveau Pathname

Description des Paramètres :

num_parms : Nombre de paramètres dans la Table des Paramètres.

curr_name : Pointeur vers un buffer de classe 0 contenant le pathname du fichier à renommer.

new_name : Pointeur vers un buffer de classe 0 contenant le nouveau pathname pour le fichier. Le fichier doit se trouver dans le meme sous-catalogue.

Codes d'erreur possibles :

$2B : Une opération d'écriture a échoué parce que le volume disque est protégé contre l'écriture.

$40 : La syntaxe du pathname n'est pas valide.

$44 : Le Pathname spécifié n'a pas été trouvé. Cela signifie que l'un des noms de sous-catalogue, dans un autre pathname valide, n'existe pas.

$45 : Le Volume spécifié n'a pas été trouvé. Cela arrive quand par exemple on change le disque du lecteur.

$46 : Le fichier n'a pas été trouvé.

$47 : Le nouveau nom de fichier existe déjà.

$4E : Impossible d'accéder au fichier.

$50 : Le fichier est ouvert. On ne peut renommer que les fichiers fermés.

autres codes d'erreur possibles : $04, $27, $4A.

Exemple de Programme :

Voici une sous-routine permettant de changer le nom d'un fichier appelé FILE.1 dans un sous-catalogue /PROG; le nouveau nom de fichier sera FILE.2 dans le meme sous-catalogue.

```
            JSR   MLI
            DFB   $C2
            DA    PARMTBL
            BCS   ERROR
            RTS
PARMTBL     DFB   2           ; Nombre de paramètres
            DA    PATH1       ; Pointeur sur le pathname actuel
            DA    PATH2       ; Pointeur sur le nouveau nom

PATH1       ASC «/PROG/FILE.1»

PATH2       ASC «/PROG/FILE.2»
```

ResetCache

GS/OS : $2026

Fonction : Permet de forcer la taille du cache utilisé par GS/OS; cette taille est stockée dans la Bram.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (0)

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres.
ProDos 8 : n'existe pas
SessionStatus

GS/OS : $201F

Fonction : Permet de déterminer si une session d'écriture est différée; c'est à dire si une commande BeginSession est active.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +3	status	O	Code de retour

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres. La valeur minimum est 0.

status : Ce code indique si une session d'écriture est différée.

$0000 : session d'écriture différée active
$0001 : session d'écriture différée non active

ProDos 8 : n'existe pas

SET_BUF

GS/OS : n'existe pas

ProDos 8 : $D2

Fonction : Permet de déplacer un buffer d'un fichier de saposition courante vers une autre zone de 1024 octets en mémoire.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (2)
+1	ref_num	I	Numéro de Référence du fichier
+2 à +3	io_buffer	I	Pointeur sur le buffer

Description des Paramètres :

num_parms : Nombre de paramètres dans la Table des Paramètres.

ref_num : Numéro de Référence attribué au fichier par le système d'exploitation.

io_buffer : Pointeur sur un buffer de 1024 octets dans lequel le buffer courant va etre transféré.

Codes d'erreur possibles :

$43 : Le Numéro de Référence du fichier n'est pas valide.

$56 : L'adresse du buffer pour le pathname n'est pas valide.

autre code d'erreur possible : $04.

Exemple de Programme :

Le programme suivant déplace le buffer d'un fichier ayant pour Numéro de Référence 1 de sa position courante à l'adresse $2000. Vous devez vous assurer que la zone $2000-$23FF n'est pas utilisée.

```
            JSR     MLI
            DFB     $D2
            DA      PARMTBL
            BCS     ERROR
            RTS

PARMTBL     DFB     2           ; Nombre de paramètres
            DFB     1           ; Numéro de Référence du fichier
            DA      $2000       ; Pointeur sur le nouveau buffer
```

SetEOF

GS/OS : $2018

Fonction : Permet de modifier le pointeur EOF d'un fichier ouvert. Si on diminue EOF, tous les blocks de datas se trouvant après la nouvelle valeur EOF sont libérés. Si on augmente EOF, le système d'exploitation n'attribue pas de nouveaux blocks tant que d'autres datas ne sont pas écrites dans ce fichier. Si la nouvelle valeur EOF est inférieure au pointeur MARK, ce dernier prend la valeur de EOF. On peut modifier la valeur EOF de tout fichier dont le bit d'accès d'écriture est à 1.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (3)
+2 à +3	ref_num	I	Numéro de Référence du fichier
+4 à +5	base	I	Code pour déterminer la nouvelle valeur EOF
+6 à +9	displacement	I	Nouvelle valeur EOF

SET_EOF

ProDos 8 : $D0

Fonction : même chose que pour GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (2)
+1	ref_num	I	Numéro de Référence pour le fichier
+2 à +4	eof	I	Nouvelle valeur EOF

Description des paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres.

ref_num : Numéro de Référence attribué au fichier par le système d'exploitation.

base : Ce code indique à GS/OS comment déterminer la nouvelle valeur de EOF.

$0000 : nouveau EOF = displacement
$0001 : nouveau EOF = ancien EOF + displacement
$0002 : nouveau EOF = MARK + displacement
$0003 : nouveau EOF = MARK - displacement

displacement : GS/OS utilise cette valeur en conjonction avec base pour déterminer la nouvelle valeur du pointeur EOF.

eof : Nouvelle valeur du pointeur EOF.

Codes d'erreur possibles :

$2B : Le disque est protégé contre l'écriture.

$43 : Le Numéro de Référence du fichier n'est pas valide.

$4D : La valeur MARK spécifiée est supérieure à EOF.

$4E : Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier controle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.
autres codes d'erreur possibles : $04, $07, $27, $4E.

SetFileInfo

GS/OS : $2005

Fonction : Permet de modifier l'information concernant un fichier dans l'Entrée au catalogue.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (12)
+2 à +5	pathname	I	Pointeur sur le Pathname
+6 à +7	access	I	Code d'Accès
+8 à +9	file_type	I	Type de fichier
+10 à +13	aux_type	I	Type Auxiliaire de fichier
+14 à +15	non utilisé	I	
+16 à +23	create_dt	I	Date et Heure de Création
+24 à +31	modify_dt	I	Date et Heure de Modification
+32 à +35	option_list	I	Pointeur sur la liste Option
+36 à +39	non utilisé	I	
+40 à +43	non utilisé	I	
+44 à +47	non utilisé	I	
+48 à +51	non utilisé	I	

SET_FILE_INFO

ProDos 8 : $C3

Fonction : même chose que pour GS/OS.
Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (10)
+1 à +2	pathname	I	Pointeur sur le Pathname
+3	access	I	Code d'Accès
+4	file_type	I	Type de fichier
+5 à +6	aux_type	I	Type Auxiliaire de fichier
+7	non utilisé	I	
+8 à +9	non utilisé	I	
+10 à +11	modify_date	I	Date de Modification
+12 à +13	modify_time	I	Heure de Modification

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres. Sous GS/OS, la valeur minimum est de 2.

pathname : Pointeur de classe 0 (ProDos 8) ou de classe 1 (GS/OS) pointant sur l'adresse de la chaine décrivant le Pathname du fichier à utiliser. Si le Pathname spécifié n'est pas précédé par un séparateur (/ pour ProDos 8; / ou : pour GS/OS), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut (sous GS/OS c'est le préfixe 0/) pour créer le Pathname entier.

access : Code d'Accès au fichier. Voir explications dans la partie consacrée à l'organisation des fichiers sur un Volume.

file_type : Code du Type de fichier.

aux_type : Code du Type auxiliaire de fichier.

non utilisé : Ces octets ne sont pas utilisés. Ils permettent de conserver une symétrie avec la Table des Paramètres de la commande GET_FILE_INFO.

modify_date : Ce champ, sous ProDos 8 contient la date (heure, mois, année) de dernière modification du fichier. Le format est celui utilisé par ProDos 8 pour le codage de la date.

modify_time : Ce champ, sous ProDos 8 contient l'heure (heure, minute) de dernière modification du fichier. Le format est celui utilisé par ProDos 8 pour le codage de l'heure.

create_date : Ce champ, sous ProDos 8 contient la date (heure, mois, année) de création du fichier. Le format est celui utilisé par ProDos 8 pour le codage de la date.

create_time : Ce champ, sous ProDos 8 contient l'heure (heure, minute) de création du fichier. Le format est celui utilisé par ProDos 8 pour le codage de l'heure.

create_td : Heure et date de création du fichier sous GS/OS. Ces 8 octets représentent les paramètres suivants :

secondes	
minutes	
heure	
année	Année - 1990
jour	jour du mois - 1
mois	0 = Janvier, 1 = Février, etc ...
non utilisé	
jour de la semaine	1 = Dimanche, 2 = Lundi, etc ...

modify_td : Heure et date de dernière modification du fichier sous GS/OS. Ces 8 octets sont rangés dans le meme ordre que dans le champ create_td.

option_list : Pointeur sur un buffer de sortie de classe 1 ou GS/OS retourne des informations spécifiques utilisées par le FST pour accéder au fichier.

Note : Les paramètres non utilisés doivent etre mis à 0.

Codes d'erreur possibles :

$2B : Le disque est protégé contre l'écriture.

$40 : Le Pathname contient des caractères non valides, ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).

$44 : Un Catalogue dans un Pathname n'a pas été trouvé.

$45 : Le Catalogue principal n'a pas été trouvé.

$46 : Le fichier n'a pas été trouvé.

$4E : Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier controle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.

autres codes d'erreur possibles : $04, $07, $27, $4A, $4B, $52, $53, $58.

SetLevel

GS/OS : $201A

Permet de fixer le niveau du fichier système.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +3	level	I	Nouveau niveau du fichier système

Description des Paramètres :

pcount : Nombre de Paramètres dans la Table des Paramètres.

level : Valeur du niveau du fichier système. La valeur peut etre comprise entre $0000 et $00FF.

Code d'erreur possible :

$59 : Niveau de fichier non valide.

autre code d'erreur possible : $07

Exemple de Programme :

Voici comment mettre le niveau du fichier système à la valeur 2.

```
            JSL  E100A8
            DA   $201A
            Adrl ParmTbl
            RTS

ParmTbl     DA   2          ; Nombre de paramètres
            DA   2          ; Nouveau niveau du fichier système
```

Le niveau de fichier système joue sur les commandes Open, Close, et Flush.

ProDos 8 : n'existe pas

SetMark

GS/OS : $2016

Fonction : Permet de modifier la valeur du pointeur MARK dans un fichier ouvert. On peut utiliser cette commande pour se positionner n'importe où dans le fichier; les opérations de lecture et d'écriture prennent effet à partir de cette position.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (3)
+2 à +3	ref_num	I	Numéro de Référence du fichier
+4 à +5	base	I	Code pour déterminer MARK
+6 à +9	displacement	I	Nouvelle valeur de MARK

SET_MARK

ProDos 8 : $CE

Fonction : même chose que pour GS/OS

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (2)
+1	ref_num	I	Numéro de Référence du fichier
+2 à +4	position	I	Nouvelle valeur de MARK

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres.

ref_num : Numéro de Référence du fichier attribué par la système d'exploitation.

base : Ce code indique à GS/OS comment déterminer la nouvelle valeur de MARK.

$0000 : nouveau MARK = displacement
$0001 : nouveau MARK = EOF - displacement
$0002 : nouveau MARK = ancien MARK + displacement
$0003 : nouveau MARK = ancien MARK - displacement

position : Nouvelle valeur pour MARK. Ce pointeur ne doit pas etre supérieur à EOF.

displacement : GS/OS utilise cette valeur en conjonction avec base pour déterminer la nouvelle valeur du pointeur MARK.

Codes d'erreur possibles :

$43 : Le Numéro de Référence du fichier n'est pas valide.

$4D : Le pointeur MARK est supérieur à EOF.

autres codes d'erreur possibles : $04, $07, $27.

SetPrefix

GS/OS : $2009

Fonction : Permet de fixer le Préfixe par défaut. Quand on appelle une commande, le système d'exploitation convertit le Pathname demandé en y ajoutant le Préfixe par défaut pour obtenir le Pathname complet.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (2)
+2 à +3	prefix_num	I	Numéro de Préfixe (0 à 31)
+4 à +7	prefix	I	Pointeur sur le Préfixe

SET_PREFIX

ProDos 8 : $C6

Fonction : même chose que pour GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (1)
+1 à +2	prefix	I	Pointeur sur le Préfixe

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres.

prefix_num : Numéro de Préfixe GS/OS (0 à 31).

prefix :Pointeur de classe 0 (ProDos 8) ou de classe 1 (GS/OS) pointant sur l'adresse de la chaine décrivant le Préfixe à utiliser.

Codes d'erreur possibles :

$40 : La syntaxe du pathname n'est pas valide.

$44 : Le Pathname spécifié n'a pas été trouvé.
$45 : Le Volume spécifié n'a pas été trouvé.

$46 : Fichier non trouvé.

$4B : Le type de fichier n'est pas valide ou n'est pas reconnu.

autres codes d'erreur possibles : $04, $07, $27, $53.

SetSysPrefs

GS/OS : $200C

Fonction : Permet de régler les Préférences système.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +3	preference	I	Préférences système

Description des Paramètres :

pcount : Nombre de Paramètres dans la Table des Paramètres.

preferences : Mot indiquant les Préférences système :

bit 15 1 = afficher la boite de dialogue Changement de Volume
0 = ne pas afficher cette boite de dialogue

Commentaires :
Les commandes GS/OS qui utilisent des Pathnames comme paramètres d'entrée, affichent normalement une boite de dialogue demandant à l'utilisateur d'insèrer le Volume Disque demandé si celui ci n'est pas en ligne. Si l'application est capable de gérer les erreurs "Volume non trouvé", il faut utiliser SetSysPrefs pour mettre à 0 le bit 15 du mot de Préférences système.

ProDos 8 : n'existe pas

UnbindInt

GS/OS : $2032

Fonction : Permet d'enlever une sous-routine de gestion d'interruption.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (1)
+2 à +3	int_num	I	Numéro de Référence de l'interruption

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres.

int_num : Numéro de Référence que GS/OS a attribué à la sous-routine de gestion d'interruption.

Code d'erreur possible :

$53 : Le Numéro de Référence int_num n'est pas valide.
autres codes d'erreur possibles: $04, $07.

ProDos 8 : n'existe pas

Volume

GS/OS : $2008

Fonction : Retourne les caractéristiques du disque volume.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (6)
+2 à +5	dev_name	I	Pointeur sur le nom de Device
+6 à +9	vol_name	O	Pointeur sur le nom de Volume
+10 à +13	total_blocks	O	Taille du volume en Blocs
+14 à +17	free_blocks	O	Nombre de Blocks inutilisés
+18 à +19	file_sys_id	O	Code ID du système d'exploitation
+20 à +21	block_size	O	Nombre d'octets par Blocs

Description des Paramètres :

pcount : Nombre de paramètres dans la Table des Paramètres. La valeur minimum est 2.

dev_name : Pointeur sur un buffer de classe 1 contenant le nom du Device.

vol_name : Pointeur sur un buffer de classe 1 ou GS/OS retourne le nom du volume disque. Ce buffer doit avoir 35 octets.

total_blocks : Nombre total de blocs sur le Volume Disque.

free_blocks : Nombre de blocs inutilisés sur le Volume Disque. Pour le format FST cette valeur est toujours à 0.
file_sys_id : Code d'identification du système d'exploitation utilisé sur le volume disque.

$00 : réservé
$01 : ProDos / SOS
$02 : Dos 3.3
$03 : Dos 3.2 / 3.1
$04 : Apple II Pascal
$05 : Macintosh MFS
$06 : Macintosh HFS
$07 : Macintosh XL (Lisa)
$08 : Apple CP/M
$09 : non utilisé
$0A : MS-DOS
$0B : High Sierra (CD-ROM)
$0C : ISO 9660 (CD-ROM)

block_size : Taille d'un bloc sur disque en nombre d'octets.

Codes d'erreur possibles :

$10 : Le nom de Device indiqué n'existe pas.

$27 : Une erreur I/O sur disque 5.25" est survenue empechant le transfert correct des données. Le Volume disque est sans aucun doute abimé. On obtient aussi cette erreur s'il n'y a pas de disque dans le lecteur.

$28 : Pas de Device connecté.

$2F : Le Device spécifié n'est pas en ligne. Cette erreur arrive s'il n'y a pas de disque dans le lecteur 3.5".

autre codes d'erreur possibles : $07, $11, $2E, $40, $45, $4A, $52, $55, $57, $58.

ProDos 8 : n'existe pas

Write

GS/OS : $2013

Fonction : Permet d'écrire des datas dans un fichier ouvert. L'écriture débute à la position courante de MARK. Le pointeur MARK est incrémenté au fur et à mesure de l'écriture.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0 à +1	pcount	I	Nombre de paramètres (5)
+2 à +3	ref_num	I	Numéro de Référence du fichier
+4 à +7	data_buffer	I	Pointeur sur le buffer de datas
+8 à +11	request_count	I	Nombre d'octets à écrire
+12 à +15	transfer_count	O	Nombre d'octets écrits
+16 à +17	cache_priority	I	Niveau de priorité du cache

WRITE

ProDos 8 : $CB

Fonction : même chose que pour GS/OS.

Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (4)
+1	ref_num	I	Numéro de Référence du fichier
+2 à +3	data_buffer	I	Pointeur sur le début du buffer de datas
+4 à +5	request_count	I	Nombre d'octets à écrire
+6 à +7	transfer_count	O	Nombre d'octets écrits

Description des Paramètres :

pcount / num_parms : Nombre de paramètres dans la Table des Paramètres. Sous GS/OS la valeur minimum est 4.

ref_num : Numéro de Référence du fichier attribué par le système d'exploitation.

data_buffer : Pointeur sur le début du bloc mémoire contenant les datas à écrire sur le disque.

request_count : Nombre de caractères à écrire sur disque.

transfer_count : Nombre de caractères écrits sur disque.

cache_priority : Ce code indique comment GS/OS va utiliser le cache pour y placer les blocs lors de l'écriture du fichier.
$0000 : pas de cache pour ce fichier
$0001 : le cache est utilisé pour ce fichier

Codes d'erreur possibles :

$2B : Le disque est protégé contre l'écriture.
$43 : Le Numéro de Référence du fichier n'est pas valide.

$48 : Le Volume est plein.

$4E : Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier controle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.

$56 : L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map de ProDos 8, ou parce que le buffer ne débute par sur un saut de page mémoire.

autres codes d'erreur possibles : $04, $07, $27.

Exemple de Programme :

Cette sous-routine GS/OS écrit 256 octets dans un fichier dont le Numéro de Référence est 2; les datas sont stockées en Buffer.

```
JSL     $E100A8
DA      $2013
Adrl    ParmTbl
BCS     Error       ;En cas d'erreur
RTS
```

```
ParmTbl   DA      4         ;Nombre de Paramètres
          DA      2         ;Numéro de Référence du fichier
          Adrl    Buffer    ;Pointeur sur le buffer de datas
          Adrl    256       ;Nombre d'octets à écrire
TransCnt  DS      4         ;Nombre d'octets écrits par GS/OS

Buffer    DS      256       ;Buffer
```

WRITE_BLOCK

GS/OS : n'existe pas

ProDos 8 : $81

Fonction : Permet de transférer le contenu du buffer mémoire de 512 octets de la mémoire sur le Volume disque.

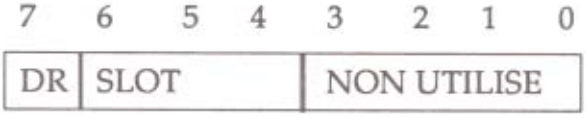
Table des Paramètres :

OFFSET	NOM	I/O	DESCRIPTION
+0	num_parms	I	Nombre de paramètres (3)
+1	unit_num	I	Numéro d'Unité
+2 à +3	data_buffer	I	Pointeur sur le buffer de datas
+4 à +5	block_num	I	Numéro du block à écrire

Description des Paramètres :

num_parms : Nombre de paramètres dans la Table des Paramètres.

unit_num : Numéro de slot et de drive auquel on veut accéder.



SLOT peut en fait etre le numéro réel ou logique d'un slot si le système contient des périphériques disque comme par exemple un RAM disque. Par exemple, le numéro d'unité d'un volume /RAM connecté en slot 3, Drive 2 sur un IIe, IIc, II GS est $B0 soit 1 011 0000.

DR indique le numéro de DRive (lecteur) : 0 pour le lecteur 1 et 1 pour le lecteur 2. Plus de deux lecteurs peuvent etre connectés au port 5 du SmartPort. Dans ce cas, ProDos 8 assigne logiquement les deux prochains lecteurs en Slot 2, Drive 1 et en Slot 2, Drive 2. ProDos 8 ignore tous les lecteurs connectés au SmartPort après le quatrième.

data_buffer : Pointeur sur le début d'un block mémoire de 512 octets qui contient le block à écrire par WRITE_BLOCK.
block_num : Numéro du bloc à écrire. Les valeurs autorisées dépendent du périphérique disque.

0 - 279 pour les lecteurs 5.25
0 - 1599 pour les lecteurs 3.50
0 - 127 pour les volumes /Ram

Vous pouvez déterminer la taille d'un périphérique en utilisant la commande GET_FILE_INFO.

Codes d'erreur possibles :

$27 : Le disque est illisible. Il est très certainement endommagé. Cette erreur peut aussi survenir si la porte du lecteur est ouverte, ou s'il n'y a pas de disquette dans le lecteur.

$28 : Pas de périphériques de connectés. ProDos 8 retourne cette erreur quand on essaie d'accéder à un second lecteur 5.25 qui n'existe pas.

$2B : Le disque est protégé contre l'écriture.

autre codes d'erreur possibles : $04, $07, $11, $2F, $53, $56.

VI

Son & Musique

Qu'ils s'agissent de programmes commerciaux (notamment dans les jeux) ou de ceux réalisés par des amateurs plus ou moins éclairés, vous avez pu vous rendre compte des capacités sonores et musicales de votre Apple IIgs. Je suis certain que vous n'avez pas été déçu par ses fantastiques possibilités dues en grande partie à son propre microprocesseur sonore : l'Ensoniq. Cependant il est curieux que peu de personnes se soient attaquées à la sonorisation, qui n'est pas aussi difficile qu'elle pourrait paraître. Une fois la méthode acquise, programmer le son sur Apple IIgs n'est plus que de la routine, sans pour autant devenir quelque chose de lassant.
Cependant, pour pouvoir profiter au maximum du potentiel sonore de votre Apple IIgs favori, vous ne devrez pas vous contenter du haut-parleur interne. Le mieux est de brancher le IIgs sur un amplificateur HIFI ou d'acheter spécialement des enceintes amplifiées. Et, si vous succombez à cet achat, pendant que vous y êtes achetez aussi une carte stéréo qui possède de nombreux avantages dont nous reparlerons.

6.1 Le son

Avant de vous lancer dans de fabuleuses musiques, vous devrez tout d'abord apprendre à programmer l'Ensoniq pour jouer des sons, puis à créer vous même vos propres sons. A ce moment là, vous pourrez déjà appréciez toutes les possibilités de bruitages.
Heureusement, pour vous aider, Apple a doté le IIgs du fameux Ensoniq 5503 Digital Oscillator Chip, que nous appellerons désor-

mais plus communément le DOC. Outre le fait d'être une puce utilisée dans plusieurs synthétiseurs, le DOC a l'avantage de pouvoir jouer plusieurs sons en même temps, sans pour autant prendre de temps machine. C'est à dire que le microprocesseur 65C816 reste entièrement disponible pour d'autres tâches, une fois les commandes exécutées.

6.1.1. Description du système

L'Ensoniq DOC dispose d'un très bon potentiel sonore puisqu'il possède 32 oscillateurs indépendants les uns des autres. Chacun des oscillateurs pouvant produire un son, ceci signifie que vous pouvez avoir jusqu'à 32 sons différents joués en même temps ! Bien entendu si vous écoutez autant de sons, vous risquez de ne plus rien entendre du tout pour cause de non harmonisation. Par ailleurs, chaque oscillateur peut fonctionner avec un volume réglable et indépendant et selon quatre modes distincts.
Il faut remarquer que tous les registres du Sound GLU et du DOC sont sur 8 bits, donc tous les exemples de routines sont à considérer avec l'accumulateur et les registres d'index XY sur 8 bits, sauf changements précisés.

6.1.1.1. *La Ram Son*

Le DOC possède sa propre ram de 64 K. Cependant cette ram est assez particulière, car d'une part elle n'est pas adressable par le microprocesseur 65C816 et d'autre part elle n'est pas exécutable, c'est à dire que même si vous y stockez des programmes, ceux-ci ne pourront au aucun cas fonctionner. Il est également important de savoir que lors d'un démarrage à chaud (de type Ctrl-Reset) cette Ram Son n'est pas effacée. Par contre elle est initialisée avec des valeurs $80 lors d'un démarrage à froid (de type Ctrl-Pomme -Reset ou lors de la mise sous tension de l'Apple). Etant donné que cette ram n'est pas adressable par le microprocesseur, il vous faudra passer par l'intermédiaire du Sound GLU et de ses commutateurs.
En ce qui concerne les valeurs à mettre en Ram Son, toute valeur de $01 à $FF est une constituante de son. Ainsi une onde sonore est une suite de valeurs qui sera ensuite convertie en un signal audible. La valeur $00 quant à elle est réservée pour indiquer si besoin la fin du son.
Comme vous pouvez vous en douter, la Ram Son contiendra des sons, mais ceux-ci ne peuvent pas avoir n'importe quelle taille, ainsi les longueurs possibles sont : $100 ; $200 ; $400 ; $800 ; $1000 ; $2000 ; $4000 ou $8000 octets par son. Comme vous pouvez le voir, il n'y a pas de taille de 64 K possible normalement ; cependant nous verrons que cela est tout de même réalisable avec une petite astuce. De plus les sons ne peuvent pas commencer n'importe où en Ram Son. Ainsi ils doivent

obligatoirement commencer en début de page mémoire de la Ram Son.
Voici un petit tableau indiquant où peut commencer un son selon sa taille :

taille	début du son en commencement de page de la Ram Son
$100	$00 ; $01 ; $02 ; $03 ; $04 ... $FB ;$FC ; $FD; $FE ; $FF
$200	$00 ; $02 ; $04 ; $06 ; $08 ... $F6 ;$F8 ; $FA; $FC ; $FE
$400	$00 ; $04 ; $08 ; $0C; $10 ... $EC ;$F0 ; $F4 ; $F8 ; $FC
$800	$00 ; $08 ; $10 ; $18 ; $20 ... $D8 ;$E0 ; $E8 ; $F0 ; $F8
$1000	$00 ; $10 ; $20 ; $30 ; $40 ... $B0 ;$C0 ; $D0; $E0 ; $F0
$2000	$00 ; $20 ; $40 ; $60 ; $80 ;$A0 ;$C0 ; $E0
$4000	$00 ; $40 ; $80 ; $C0
$8000	$00 ; $80

Par exemple un son de $4000 de long (16384 octets) pourra commencer en Ram Son aux adresses $0000 ; $4000 ; $8000 ; $C000. Mais il ne pourra en aucun cas commencer en $0003, ni $1000, ni $C500.
Ainsi vous remarquerez que plus un son est long, moins il y a de possibilités d'emplacements. Mais qu'arrive t'il si la taille d'un son ne correspond pas exactement aux longueurs standards ? Dans ce cas, il faut prendre la taille par excès et terminer le sons par des zéros.
Par exemple, si votre son occupe une taille de 13600 octets de long ($3520), on considèrera que sa taille est de 16384 octets ($4000) ; mais à partir du 13600 ème octet jusqu'au 16383 ème, vous devrez mettre des zéros. Evidemment les 2784 octets à $00 sont gaspillés, sauf si vous arrivez à y loger un autre petit son de 2048 octets. Ainsi dans cet exemple, les deux sons pourraient occuper la Ram Son comme suit :

$0000 : début du premier son de 13600 octets déclaré $4000 de long.
$351F : fin du premier son.
$3520 : début de la zone des $00 indiquant la fin du premier son.
$37FF : fin de la zone des $00.
$3800 : début du deuxième son de 2048 octets déclaré $800 de long.
$3FFF : fin du deuxième son.

Quoi qu'il en soit, la zone des $00 représente encore 736 octets perdus où l'on pourrait éventuellement installer un son de 512 octets. La mise en place organisée des sons peut vraiment devenir un art ! Aussi est-il préférable de connaitre exactement la taille de tous les sons que l'on désire utiliser avant de commencer la programmation.
Malgré sa petite taille et sa relative mauvaise organisation, la Ram Son

reste souvent suffisante pour la plupart des bruitages ou des musiques, pour peu que l'on choisisse bien ses sons.

6.1.1.2 Les commutateurs de contrôles (Sound GLU)

Le DOC possède ses propres registres de commandes permettant de le programmer complètement. En tout il y a 227 registres. Mais pour pouvoir accéder à ces registres, il vous faut passer par les quatres commutateurs du Sound GLU (General Logic Unit). Ce circuit est donc un intermédiaire entre le l'Ensoniq DOC et le microprocesseur 65C816. Voici les quatre registres de commande de L'Ensoniq DOC. A noter que les commutateurs peuvent également être accédés en adressage long (exemple : $E1C03C ou $E0C03C), il n'est cependant pas conseillé de les accéder par un adressage indexé ou autre. L'accès à la Ram Son se fait également via ces registres.

Registres GLU	Adresse	Type
Registre de contrôle du son	$C03C	lecture/écriture
Registre de donnée	$C03D	lecture/écriture
Registre pointeur d'adresse, octet bas	$C03E	lecture/écriture
Registre pointeur d'adresse, octet haut	$C03F	lecture/écriture

6.1.1.2/1 *Le commutateur de contrôle du son $C03C*

Ce commutateur contrôle d'une part le volume général du haut-parleur interne, mais contient également quelques informations relatives aux accès du DOC. Voici la description de ses bits :

```
7 6 5 4 3 2 1 0
! ! ! ! !_!_!_!___ Volume.
! ! ! !_____________ Réservé, ne pas modifier.
! ! !_______________ Auto incrémentation du pointeur d'adresse.
! !_________________ Accès au DOC ou à la Ram Son.
!___________________ DOC occupé ?
```


Bit	Valeur	Description
7	1	Quand ce bit est à 1, le DOC est occupé. Attendez jusqu'à ce qu'il soit libre.
	0	Quand ce bit est à 0, le DOC est libre.
6	1	Tous les accès se font en direction de la Ram Son.
	0	Tous les accès se font en direction des registres du DOC.
5	1	Auto-incrémentation des registres pointeurs activés.
	0	Auto-incrémentation inactivée.
4	-	Réservé, ne pas modifier.
3-0	$0-$F	Contrôle du volume : $0 faible, $F fort.

Le test du bit 7 de ce registre n'est normalement pas indispensable, car L'Ensoniq est assez rapide et rarement utilisé en rendement maximum. Ce bit est à lecture seulement, mais si vous écrivez dedans, rien ne se produira.
En ce qui concerne les bits 5 et 6, si vous voulez accéder aux registres du DOC, je vous conseille de les mettre tous deux à zéro. Par contre, lors d'un transfert de sons de la mémoire principale vers la Ram Son, les mettre tous deux à un.
Le bit 4 est réservé.
Enfin, les bits 0 à 3 contiennent donc le volume général du haut-parleur interne du GS, et de la prise jack située à l'arrière. Cependant, il faut noter que ce volume n'a aucun effet dans le cas d'une carte stéréo, car les connecteurs reliés directement au DOC (sur la fiche interne à 7 pattes, de type connecteur Molex) ont un volume indépendant de celui du petit haut-parleur.
Si vous tenez à mettre comme volume général celui du réglage du tableau de bord, sachez que la mémoire $E100CA contient dans ses bits 0 à 3 le volume.

Exemple d'accès aux registres du DOC :

```
LDAL $E100CA      ; Lecture du volume du tableau de bord
AND  #$0F         ; On ne garde que les bits 0 à 3.
STA  $C03C        ; Les bits 5 et 6 étant forcés à zéro, on
                  ; accède aux registres du DOC.
```

Exemple d'accès à la Ram Son :

```
LDAL $E100CA      ; Volume...
AND  #%00001111   ; On ne garde que les bits 0 à 3.
ORA  #%01100000   ; On force à 1 les bits 5 et 6.
STA  $C03C        ; Donc on accède à la Ram Son.
```

6.1.1.2/2 *Le commutateur de transfert des données $C03D*

Qu'ils s'agissent d'accès au DOC ou à la Ram Son, vous devrez passer des données, et c'est justement le rôle du commutateur $C03D. Cependant ce commutateur ne fonctionne pas de la même façon selon que vous faites une lecture ou une écriture.
Dans le cas d'une écriture, il suffit simplement d'un STA $C03D.

Exemple :

```
LDA  #$xx         ; xx étant une valeur hexa quelconque
STA  $C03D        ; que l'on transfère dans un registre du DOC ou
                  ; dans une mémoire de la Ram Son.
```

Mais lors d'une lecture, le comportement de $C03D est différent, ainsi si vous voulez lire une donnée, le premier octet ne doit pas être pris en compte si vous venez de programmer les commutateurs de pointage ($C03E ou $C03F). Nous verrons des exemples dans le paragraphe suivant.
De plus dans le cas d'une lecture d'un registre du DOC, le problème est un peu plus complexe, car vous serez amené à modifier très souvent le commutateur $C03E, je vous conseille donc très vivement d'effectuer systèmatiquement deux lectures pour lire une valeur !

6.1.1.2/3 *Les commutateurs de pointages $C03E et $C03F*

Les deux commutateurs $C03E et $C03F sont utilisés pour indiquer où seront transférées les données. Dans le cas d'un accès au DOC, seul le commutateur $C03E est utilisé. Il suffit alors d'écrire dans le commutateur quel registre du DOC vous voulez accéder. Il est inutile de s'occuper du commutateur $C03F, lors d'un accès au DOC puisque

sa valeur n'est pas prise en compte.

Exemple d'accès à un registre du DOC :

```
LDAL $E100CA      ; Tout d'abord, indiquer
AND  #$0F         ; que l'on veut accéder
STA  $C03C        ; aux registres du DOC.
LDA  #$xx         ; xx étant un nombre hexadécimal de $00
                  ; à $E2 inclus, indiquant le numéro du
                  ; registre du DOC.
STA  $C03E        ; Que l'on stocke dans $C03E.
```

A ce stade là, le registre xx du DOC est prêt pour être lu ou écrit.

Voici également une routine de transfert de données entre la mémoire et le DOC :

```
*=========================================
* Ecriture des 224 premiers registres du DOC
*=========================================
          XC              ; Directive d'assemblage.
          XC              ;

TAMPON  =  $2000          ; Adresse du tampon. Ou ailleurs.

          ORG  $1000      ; Ou une autre adresse du banc $00.

          CLC             ; Mode natif
          XCE
          SEP  $30        ; A et XY sur 8 bits

          LDAL $E100CA    ; Lecture du volume du tableau de bord.
          AND  #$0F       ; Mise à zéro des bits 4 à 7 pour choisir
          STA  $C03C      ; le mode accès DOC et sans auto-
                          ; incrémentation.

          LDX  #$00       ; On veut écrire depuis le début du tam
                          ; pon.
ENCORE    STX  $C03E      ; $C03E contient le numéro du registre à
                          ; écrire.
          LDA  TAMPON,X   ; Lecture d'une donnée du tampon (ici
                          ; $2000).
          STA  $C03D      ; Transfert de la donnée vers le DOC.
          INX             ; Valeur suivante.
```

```
            CPX  #$E0       ; Jusqu'à ce que l'on ait transféré 224 va
                            ; leurs.
            BCC  ENCORE

            RTS             ; Fin de routine.
```

Cette routine permet de programmer les registres du DOC en un seul transfert, mais n'est vraiment utile que pour faire des essais. Vous remarquerez que seuls les 224 premiers registres du DOC sont concernés, les trois derniers registres étant spéciaux.

De la même façon, voici la routine inverse qui permet de lire les registres du DOC :

```
*=======================================
* Lecture des 224 premiers registres du DOC
*=======================================
            XC
            XC

TAMPON   =  $2000       ; Adresse du tampon. Ou ailleur.

            ORG $1000   ; Ou ailleurs...

            CLC
            XCE
            SEP $30     ; A et XY sur 8 bits.

            LDAL $E100CA
            AND  #$0F
            STA  $C03C

            LDX  #$00       ; A partir du premier registre.
ENCORE      STX  $C03E
            LDA  $C03D      ; Attention, il faut faire deux
            LDA  $C03D      ; lectures pour ne pas avoir de problème !
            STA TAMPON,X    ; Stockage de la valeur lue dans le
                            ; tampon.
            INX             ; Valeur suivante...
            CPX  #$E0       ; 224 valeurs lues ?
            BCC  ENCORE     ; Si oui alors on a fini.

            RTS             ; Sortie du sous-programme.
```


Par contre, lors d'un accès à la Ram Son, l'emploi des commutateurs $C03E et $C03F constituent ensemble l'adresse 16 bits où doit être transférée la donnée. Ainsi $C03E contient la partie basse de l'adresse tandis que $C03F contient la partie haute.

De plus, ces deux commutateurs ont la possibilité d'être auto-incrémentés à chaque fois qu'une donnée est lue ou écrite par $C03D. Ceci est particulièrement utile lorsque vous voulez transférer des données en Ram Son.

Voici l'exemple d'une routine de transfert de 64 K de sons du banc $03 vers la Ram Son. La plupart du temps, c'est une routine de ce genre qui est utilisée pour installer d'un coup tous les sons. Bien entendu, il faut préalablement charger soit même les sons voulus dans le banc $03.

```
*=======================================
* Ecriture en Ram Son de 64 K de données
*=======================================
          XC                  ; Directive d'assemblage
          XC                  ;

MEMOIRE  =  $030000           ; Banc de stockage des sons. Ou un
                              ; autre.
          ORG $1000           ; Ou ailleurs..

          CLC
          XCE
          REP $30             ; Registres XY sur 16 bits
          SEP $20             ; et A sur 8 bits.

          LDAL $E100CA        ; On choisi d'accéder à la
          AND #%00001111      ; Ram Son avec auto-incrémentation
          ORA #%01100000      ; des pointeurs...
          STA $C03C

          STZ $C03E           ; Mise à zéro des
          STZ $C03F           ; deux pointeurs.
          LDX #$0000          ; On veut lire depuis le début du banc.
ENCORE LDAL MEMOIRE,X         ; Lecture d'une valeur du banc (ici le
                              ; banc $03).
          STA $C03D           ; Transfert en Ram Son. A noter qu'en
                              ; même temps les pointeurs sont
                              ; augmentés d'une unité, donc il est
                              ; inutile de s'occuper des pointeurs.
          INX                 ; Valeur suivante.

          BNE ENCORE          ; Jusqu'à ce que les 64 K soit transfé
```

```
                        ; rés.
        RTS             ; Sortie du sous-programme.
```

De même voici une routine de lecture de 64 K depuis la Ram Son vers le banc $03. Il est plus rare de lire la Ram Son que d'y écrire, mais cela peut parfois servir et la routine est légèrement différente de celle d'écriture.

```
*================================
* Lecture des 64 K de la Ram Son
*================================
        XC              ; Directive d'assemblage...
        XC

MEMOIRE =  $030000      ; Banc de stockage des sons. Ou un autre.

        ORG $1000       ; Ou ailleurs...

        CLC
        XCE
        REP $30
        SEP $20         ; A sur 8 bits et XY sur 16 bits.

        LDAL $E100CA
        AND #%00001111
        ORA #%01100000
        STA $C03C       ; Accès Ram Son avec auto-incrémenta
                        ; tion.

        STZ $C03E       ; Mise à zéro des deux
        STZ $C03F       ; pointeurs.
        LDA $C03D       ; Attention ! Ici on fait une permière lec-
                        ; ture pour rien, comme indiqué. Mais sur
                        ; tout, on l'a fait après la déclaration des
                        ; deux pointeurs et  non avant !
        LDX #$0000
ENCORE  LDA $C03D       ; Lecture d'une valeur et incrémentation
                        ; automatique des pointeurs.
        STAL MEMOIRE,X  ; Stockage...
        INX             ; Valeur suivante...
        BNE ENCORE

        RTS             ; Fin de la routine.
```


6.1.1.3 Les registres du DOC

La programmation des 32 oscillateurs du DOC se fait par l'intermédiaire de ses 227 registres. Trois de ces registres concernent le DOC en général, mais les autres sont divisés en sept séries de 32 registres. Chaque oscillateur peut produire un et un seul son à la fois, mais Apple recommande de ne pas utiliser les oscillateurs 31 et 32, car ils sont réservés. Cependant, si vous n'utilisez pas les outils sonores (Sound Tools), vous pourrez les programmer sans aucun problème. Vous pouvez donc avoir jusqu'à 32 sons différents à la fois.
Pour effectivement produire un son, il faut choisir un oscillateur libre, et lui indiquer dans ses registres les éléments suivants :

- La fréquence à laquelle le son sera joué.
- Le volume de l'oscillateur.
- Le début du son en Ram Son.
- La longueur du son.
- Le mode de fonctionnement de l'oscillateur et son ordre de mise en action.

Dans les exemples des paragraphes suivants, il faudra considérer que le registre de contrôle du son ($C03C) est programmé pour accéder aux registres du DOC. Je vous rappelle les instructions qui permettent d'accéder au DOC :

```
CLC
XCE
SEP  $30              ; A et XY sur 8 bits.
LDAL $E100CA          ; Lecture du volume général du tableau
                      ; de bord.
AND  #$0F             ; Mise à zéro des bits 5 et 6 pour accéder au
                      ; DOC
STA  $C03C            ; sans auto-incrémentation.
```

6.1.1.3/1 *Les registres de fréquence (bas et haut) $00-$3F*

Ce qui va déterminer la hauteur du son produit est la fréquence à laquelle le son est joué. A basse fréquence, le son sera plutôt grave alors qu'à fréquence élevée il aura tendance à être aigu. Cependant, la fréquence du son que l'on indique à l'oscillateur ne correspond pas à la fréquence réelle (en hertz) à laquelle le son est entendu. La fréquence de programmation correspond à la vitesse de lecture du son qui se trouve en Ram Son. En effet, l'oscillateur en lui même ne produit pas de son, mais il converti chaque valeur en une tension qui a pour effet de faire plus ou moins vibrer la membrane du haut-parleur. Ainsi le fait de convertir à la suite différentes valeurs va produire un son. Bien entendu, la réalité est plus compliquée que cela,

mais nous allons nous limiter à la programmation du son, sans entrer dans les concepts scientifiques.
La fréquence du son doit être donnée sur 16 bits, ce qui permet d'avoir 65536 fréquences différentes possibles. Evidemment, à partir d'une certaine fréquence élevée, les différences ne sont plus tellement audible. Cependant il ne semble pas possible d'atteindre les ultra-sons.
Les registres $00 à $1F contiennent la valeur basse de la fréquence de chacun des 32 oscillateurs tandis que les registres $20 à $3F la valeur haute de la fréquence.
Un petit détail : si vous programmez un oscillateur avec une fréquence très basse ($0001 par exemple), le son sera lu très lentement et sera à peine audible ; par contre, si vous choisissez une fréquence nulle ($0000), le son ne sera pas lu du tout, mais l'oscillateur ne sera pas arreté. Ceci peut être utilisé pour interrompre momentanément un son, qui ensuite pourra continuer à être joué depuis l'endroit où il avait été coupé.

Exemple : pour mettre la fréquence $156 (342) dans l'oscillateur n $07 :

```
LDA  #$07      ; Le registre n $07 est celui de fréquence (bas)
STA  $C03E     ; de l'oscillateur n $07.
LDA  #$56      ; Partie basse de la fréquence voulue.
STA  $C03D     ; On stocke par l'intermédiaire du registre data.
LDA  #$27      ; Le registre n $27 est quant à lui celui de
STA  $C03E     ; fréquence (haut) de l'oscillateur n $07.
LDA  #$01      ; Partie haute de la fréquence.
STA  $C03D     ; Stockage dans le DOC.
```

Il est également possible de lire si besoin les registres de fréquence :

```
LDA  #$07      ; Fréquence (bas) de l'oscillateur n $07.
STA  $C03E     ; Que l'on indique dans le pointeur bas.
LDA  $C03D     ; Deux lectures sont nécessaires pour lire
LDA  $C03D     ; une valeur valide, ne l'oubliez pas !
```

6.1.1.3/2 *Les registres de volume $40-$5F*

Il y a 256 volumes possibles; de plus, le fait que chaque oscillateur possède son propre volume devient très intéressant, surtout pour faire de la musique. Le volume est croissant : $00 équivaut à un son blanc, puisqu'il est inaudible et $FF est le volume maximum.

Exemple : mettre le volume de l'oscillateur n $00 au maximum :

```
LDA  #$40     ; Registre du volume de l'oscillateur n°$00.
STA  $C03E
LDA  #$FF     ; $FF étant le volume maximum.
STA  $C03D
```

6.1.1.3/3 *Les registres de données $60-$7F*

Ces registres sont à lecture seulement. Ils contiennent la dernière valeur de la Ram Son lue par tel ou tel oscillateur et leur emploi n'est pas d'une grande utilité. Cependant, nous verrons dans une autre partie un emploi détourné de ces registres.

Exemple : Quelle est la dernière valeur lue par l'oscillateur n $1F ?

```
LDA  #$7F
STA  $C03E
LDA  $C03D
LDA  $C03D    ; L'accumulateur contient cette dernière valeur.
```

6.1.1.3/4 *Les registres pointeurs en Ram Son $80-$9F*

Pour indiquer le début du son utilisé en Ram Son par tel ou tel oscillateur, il suffit d'écrire dans le registre correspondant la valeur de début de page. Reportez vous au paragraphe traitant de la Ram Son pour savoir où mettre les sons.

Exemple : Nous voulons indiquer l'emplacement du son qui sera joué par l'oscillateur n $10. Soit un son de $4000 octets de longs (16384 octets). Il est possible de l'installer en Ram Son aux adresses suivantes : $0000 ; $4000 ; $8000 ; $C000. Nous choisirons pour l'exemple l'emplacement en $C000 :

```
LDA  #$90     ; Pointeur Ram Son de l'oscillateur n $10.
STA  $C03E
LDA  #$C0     ; Page $C0 de la Ram Son (pour l'adresse $C000).
STA  $C03D
```


6.1.1.3/5 *Les registres de contrôle des oscillateurs $A0-$BF*

Ce sont les registres qui contiennent le plus d'informations. Voici la description des bits d'un de ces registres :

```
7 6 5 4 3 2 1 0
! ! ! ! ! ! ! !___ Arrêt/marche de l'oscillateur.
! ! ! ! ! ! !_!_____ Mode de fonctionnement de l'oscillateur.
! ! ! ! ! !_________ Indicateur de mise en activité d'interruption.
!_!_!_!_____________ Canal de sortie du son.
```

Bit	Valeur	Description
7-4	$0-$F	Détermine un des 16 canaux de sortie du son.
3	1	L'oscillateur produira une interruption à la fin du son.
	0	Aucune interruption ne sera produite.
2-1	0 0	Mode «Free-run». Le son est automatiquement répété.
	0 1	Mode «One-shot». Arrêt de l'oscillateur en fin de son.
	1 0	Mode «Sync». Synchronisation de deux oscillateurs.
	1 1	Mode «Swap». Alternation de deux oscillateurs.
0	1	L'oscillateur est en arrêt.
	0	Il est en activité.

Les bits 7 à 4 contiennent donc le numéro du canal de sortie du son joué par l'oscillateur en question, ce qui permet avec une carte stéréo d'entendre les sons sur les enceintes gauche ou droite. Pour l'instant seules les cartes stéréo existent, mais il serait tout à fait possible de fabriquer des cartes avec 4,6 ou 8 haut-parleurs branchés, permettant ainsi d'extraordinaires effets... Cependant, bien que l'on puisse programmer jusqu'à 16 canaux de sortie, seulement 8 sont réellement accessibles étant donné qu'il n'y a que trois des quatres broches «Channel address» qui sont effectivement reliées au connecteur J25.

Mais de toute façon, comme la carte stéréo est le standard, il suffira de commuter le bit 4, les trois autres restant à zéro. On obtient donc une sortie à droite si le bit 4 est à 0 et à gauche s'il est à 1.

Si l'oscillateur arrive à la fin d'un son et que le bit 3 de son registre de contrôle est à 1, il enverra un signal d'interruption au microprocesseur. Bien entendu ce signal est traitable et l'on peut même savoir quel oscillateur l'a envoyé.

Les modes de fonctionnement des oscillateurs déterminent la façon dont le son va être joué. Les bits 2 et 1 commutent les différents modes. Ainsi le mode «Free-run» permet de produire un son qui une fois fini se répétera indéfiniment depuis le début. A l'opposé le mode «One-shot» arrêtera l'oscillateur dès la fin du son. Les deux autres modes ont une utilisation plus particulière dont nous reparlerons dans le paragraphe spécialement consacré aux quatre modes.

Enfin, le bit 0 est l'indicateur de marche de l'oscillateur. S'il est à 0 cela signifie que l'oscillateur est en train de produire un son, tandis qu'à 1 il est arrêté et ne produit donc aucun son. Ainsi, il suffit de programmer ce bit pour mettre l'oscillateur en action, mais il faut également savoir que la commutation de ce bit peut se faire automatiquement selon les modes de fonctionnement. Ainsi avec le mode «One-shot» lorsque le son a été joué en entier, le DOC arrête automatiquement l'oscillateur et le bit 0 de son registre de contrôle est donc mis à 1.

Exemple : nous voulons que l'oscillateur n $05 joue un son unique, sur le haut-parleur de gauche, sans produire d'interruption :

1) Pour jouer sur le haut-parleur de gauche, il faut mettre $1 dans les bits 7 à 4 du registre $A5 (0001 en binaire).
2) Le bit 3 doit être à 0 puisqu'aucune interruption ne doit être produite.
3) Pour un son unique, il faut le mode «One-shot», donc les bits 2 et 1 doivent contenir 0 et 1.
4) Pour la mise en marche de l'oscillateur, il faut mettre 0 dans le bit 0.

Ainsi il faut mettre la valeur binaire 00010010 ($12) dans le registre $A5.

```
LDA  #$A5      ; Registre de contrôle de l'oscillateur n $05.
STA  $C03E
LDA  #$12      ; Son unique à gauche, sans interruption.
STA  $C03D
```

6.1.1.3/6 *Les registres de taille des sons $C0-$DF*

Il nous reste encore à indiquer la taille du son utilisée par l'oscillateur désiré. C'est entre autre le rôle des registres de la série $C0 dont voici la description :

```
7 6 5 4 3 2 1 0
! ! ! ! ! !_!_!___ Codage de la résolution.
! ! !_!_!_________ Codage de la taille du son.
! !_______________ Réservé, doit être à 0.
!_________________ Réservé.
```

Bit	Valeur	Description
7	-	Réservé.
6	0	Réservé à l'extension de la Ram Son. Il faut mettre 0
5-3		Codage de la taille des sons :
	000	256 octets ($0100)
	001	512 octets ($0200)
	010	1024 octets ($0400)
	011	2048 octets ($0800)
	100	4096 octets ($1000)
	101	8192 octets ($2000)
	110	16384 octets ($4000)
	111	32768 octets ($8000)
2-0	$0-$7	Résolution d'adressage.

Le bit 7 est réservé et le mettre à zéro conviendra sans problème.
Par contre, le bit 6 doit absolument être mis à zéro, car il est réservé pour une éventuelle extension de la Ram Son. Il se pourrait qu'un futur GS utilise 128 K de Ram Son et dans ce cas le bit 6 aurait le rôle de commuter entre les deux bancs. Donc, afin de ne pas avoir de mauvaise surprise de compatibilité avec les «prochains GS», restez d'office dans le premier banc en mettant ce bit à zéro.
La taille du son est donc codée avec les bits 5 à 3, pas de problème particulier si ce n'est de bien choisir la taille des sons et leurs emplacements.
La résolution d'adressage qui est codée avec les bits 2 à 0 est d'un usage peu évident. Elle détermine selon son codage et selon la taille

du son le nombre d'octet effectivement pris en compte lors de la lecture de la Ram Son par un oscillateur. Ainsi à fréquence et résolution égales un son de différente taille n'aura pas la même hauteur. Heureusement, le fait de choisir la résolution proportionnellement à la taille du son annule cet effet le plus souvent indésirable. Afin de vous aider dans le calcul des valeurs à mettre dans le registre de taille des sons, voici un petit tableau indiquant les bonnes valeurs selon les différentes tailles de sons :

taille du son	valeur à mettre dans le registre de la série $C0
256 $0100	$00 (%00000000)
512 $0200	$09 (%00001001)
1024 $0400	$12 (%00010010)
2048 $0800	$1B (%00011011)
4096 $1000	$24 (%00100100)
8192 $2000	$2D (%00101101)
16384 $4000	$36 (%00110110)
32768 $8000	$3F (%00111111)

Exemple : Soit un son de $4000 octets de long (16384 octets). Nous voulons l'indiquer pour l'oscillateur n $15.

```
LDA  #$D5     ; Registre de taille du son de l'oscillateur n $15.
STA  $C03E
LDA  #$36     ; Selon la tableau, c'est la valeur $36 qui
STA  $C03D    ; convient.
```

6.1.1.3./7 *Le registre des interruptions $E0*

Ce registre, comme les deux suivants ne concerne pas un oscillateur en particulier, mais le DOC en général. Le registre n $E0 s'occupe des interruptions sonores en indiquant tout d'abord si une interruption a été provoquée, et si tel est le cas lequel des 32 oscillateurs en est l'origine. A noter que ce registre est à lecture seulement.

Voici la description des différents bits :

```
7 6 5 4 3 2 1 0
! ! ! ! ! ! ! !___ Réservé.
! ! !_!_!_!_!_____Numéro de l'oscillateur ayant provoqué l'interruption.
! !_______________ Réservé.
!_________________ Indicateur d'interruption.
```

Bit	Valeur	Description
7	1	Aucun oscillateur n'a produit d'interruption.
	0	Un des 32 oscillateurs est à l'origine de l'interruption.
6	-	Réservé.
5-1	$0-$1F	Contient le numéro de l'oscillateur ayant produit l'interruption.
0	-	Réservé.

L'indicateur d'interruption (bit 7) n'a d'intérêt que si vous gérez vous même les interruptions. Dans ce cas, il permet de savoir que c'est le DOC qui a provoqué l'interruption et non autre chose (VGC, Horloge, souris, etc...). Bien entendu, si vous décidez de gérer vous même les interruptions, vous devrez faire une routine de sondage qui aura pour but de repérer la source de l'interruption. Mais le GS possède déjà un excellent système de gestion des interruptions employant différents vecteurs de branchement. Dans le cas du son, il se trouve en $E1002C. Par contre, il peut être très utile de savoir lequel des 32 oscillateurs a généré l'interruption, sauf s'il est prévu qu'un seul oscillateur puisse le faire. Donc, en cas d'interruption, les bits 5 à 1 du registre $E0 contiennent le numéro de l'oscillateur, qu'il faudra ensuite traiter.

Exemple : Extrait d'une routine de gestion d'interruption :

On supposera que le microprocesseur 65C816 vient de produire une interruption et qu'il se soit branché sur votre routine de gestion d'interruption. Cette routine à pour but de connaitre l'origine de l'interruption...

```
.               ; D'autres instructions auraient ici pour but de
.               ; savoir si c'est le VGC, la souris, où autre qui
.               ; vient de provoquer l'interruption...

LDA  #$E0       ; Etant donné qu'ici ni le VGC, ni la souris n'ont
STA  $C03E      ; produit d'interruption, peut être que c'est le
LDA  $C03D      ; DOC ? On va donc lire le registre $E0.
LDA  $C03D      ; Mais n'oubliez pas qu'il faut deux lectures !
```



```
        BPL INTERSON  ; Si l'accumulateur est positif (bit 7 à 0) alors
                      ; oui, c'est bien une interruption sonore. On va
                      ; se brancher à la gestion.

        .             ; Non, ce n'est pas le DOC... C'est donc autre
        .             ; chose, il faut continuer à sonder...

*_____________________________________
* Gestion de l'interruption sonore
*_____________________________________

INTERSON AND #%00111110 ; On ne garde que les bits 5 à 1.
         LSR            ; Un petit décalage à droite et l'accu
                        ; mulateur
                        ; contient maintenant le numéro de
                        ; l'oscillateur
                        ; qui a provoqué l'interruption.
        .               ; Suite de la routine...
        .
```

6.1.1.3/8 *Le registre de fonctionnement des oscillateurs $E1*

Normalement les 32 oscillateurs sont en fonction et chacun d'eux prend 1,2 micro seconde en temps machine, ce qui fait environ 38 micro secondes pour les 32. Ce temps est négligeable, mais si pour une raison ou une autre vous voudriez aller plus vite, il est possible de ne sélectionner qu'une partie des oscillateurs. Toutefois il faut au minimum qu'un oscillateur soit en fonction, et il n'est possible de les mettre en action que dans un ordre séquentiel. Pour choisir le nombre d'oscillateurs que vous voulez, il suffit de multiplier ce nombre par deux et de l'indiquer via le registre $E1.
Mais le fait d'avoir moins d'oscillateurs en même temps provoque donc une accélération du traitement, et les fréquences réelles des sons s'en trouvent également modifiées. Donc je recommande absolument de garder 32 oscillateurs actifs étant donné que le gain de temps n'en vaut pas la peine.
Les 32 oscillateurs sont d'office sélectionnés lors du démarrage de la machine ou après un Reset, mais il est plus prudent de l'indiquer avant de commencer une utilisation du son (il faut toujours penser qu'un jour cela pourrait ne plus être compatible). Voici donc les quelques instructions nécessaires :

```
LDA  #$E1      ; Registre $E1.
STA  $C03E
LDA  #$40      ; On veut 32 oscillateurs. Mais il faut multiplier
STA  $C03D     ; ce nombre par 2, d'où en hexa $40.
```

6.1.1.3/9 *Le registre de numérisation $E2*

L'Ensoniq est doté d'un système de numérisation qui permet d'acquérir des données de type analogique et de les convertir en nombres de 0 à 255. Bien entendu ce système est utilisé en priorité pour la numérisation du son depuis un magnétophone par exemple, mais rien n'empêche de l'utiliser pour d'autres genres de numérisations. Pour numériser des données, il suffit de relier deux fils au connecteur du DOC, l'un à la masse (broche n 2) et l'autre à l'entrée du convertisseur (broche n 1), le tout branché à la source. Cependant la tension maximale admise lors de l'acquisition des données est de 2,5 volts avec une impédance maximale de 3000 ohms. Il vous est vivement conseillé DE NE PAS DEPASSER CES VALEURS pour ne pas mettre l'Ensoniq, voire le GS en danger. De toute façon si vous décidez d'utiliser la numérisation en bricolant vous même, ce sera à vos risques et périls. De plus, si vous souhaitez numériser du son et travailler avec soin et facilité, mieux vaut vous acheter une carte d'extension stéréo-numérisante.
En ce qui concerne la routine de numérisation, sachez que le DOC prend 31 micro secondes pour convertir une valeur et qu'il vous faudra donc attendre suffisament de temps entre chaque cycle d'acquisition, sous peine de manquer des valeurs. Si vous souhaitez travailler sur des temps précis, je vous conseille de commuter la cadence du GS en 1 Mhz, ce qui facilitera le calcul des temps.
Dernier point : ce registre est à lecture seulement, mais vous vous en seriez douté !

Voici une routine exemple qui a pour but de numériser 64 K de donné.

```
*=======================================
* Exemple d'une routine de numérisation
*=======================================
            XC              ; Directive d'assemblage
            XC              ;

MEMOIRE  =  $030000  ; Banc de stockage des données. Ou un autre.

            ORG  $1000      ; Ou ailleurs...

            CLC             ; Mode natif.
            XCE
```



```
        REP  $30        ; XY sur 16 bits
        SEP  $20        ; et A sur 8 bits.

        LDA  #$80       ; Mise en vitesse 1 Mhz du GS en mettant
                        ; le bit 7
        TRB  $C036      ; du commutateur $C036 à zéro.

        LDAL $E100CA    ; Volume selon celui du beep.
        AND  #$0F
        STA  $C03C      ; Accès aux registres du DOC.

        LDA  #$E2       ; Choix du registre de numérisation
        STA  $C03E
        LDA  $C03D      ; Première lecture pour rien. Remarquez
                        ; qu'ici nous n'appliquons pas la règle de
                        ; deux lectures pour une valeur, étant
                        ; donné que nous ne changeons pas de
                        ; registre.


        LDX  #$0000     ; Début du banc.
BOUCLE  LDA  $C03D      ; Lecture d'une valeur venant d'être
                        ; convertie.
        BNE  ADMIS      ; Si c'est un $00, alors on ne l'admet
                        ; pas
        LDA  #$01       ; et on le remplace par un $01.
ADMIS   STAL MEMOIRE,X  ; Stockage.
        INX             ; Valeur suivante.
        BEQ  FINI       ; Si on est arrivé à 64 K de données,
                        ; alors fin.
        LDY  #$20       ; Boucle de délai à régler selon la
                        ; qualité de
DELAY   DEY             ; numérisation que l'on veut obtenir.
        BNE  DELAY
        BRA  BOUCLE     ; Retour pour un nouveau cycle.

FINI    RTS             ; Fin de la routine.
```

6.1.1.4 *Le connecteur du DOC*

A titre indicatif, voici les caractéristiques du connecteur 7 broches de type «Molex» qui se trouve placé sur la carte mère juste à droite du DOC et à gauche du connecteur d'extension mémoire. La patte n 1 se trouve vers le voyant vert du GS.

Signal	Patte	Maximum	Unité
Entrée du convertisseur A/D Impédance d'entrée	1 3000	2,5 Ohms	Volts «peak-to-peak»
Masse analogique (GND)	2	-	-
Sortie analogique Charge de sortie	3	-5 à +5 10000	Volts «peak-to-peak» Ohms (minimum)
Adresse du canal 0 Adresse du canal 1 «Canal strobe» Adresse du canal 2	4 5 6 7	1 1 1 1	Charge LS TTL Charge LS TTL Charge LS TTL Charge LS TTL

«Canal strobe» est au niveau bas lorque l'adresse du canal est valide. Le fait que seulement trois des quatre canaux de sortie du DOC soient reliés au connecteur ne permet que 8 combinaisons possibles. Donc, bien qu'il soit possible de programmer les oscillateurs sur 16 canaux, vous ne pourrez réellement en entendre que jusqu'à huit ! Mais de toute façon, la plupart des personnes se contenteront de la stéréo qui n'utilise que deux canaux.

6.1.2 Création des sons

Vous voilà presque capable de commencer les premiers essais, si ce n'est un petit détail : vous ne possèdez encore aucun son et il va falloir les créer. Il existe différentes possibilités en matière de création sonore. Cependant, n'oubliez pas que vous ne disposez que de 64 K de Ram Son et que la valeur $00 est réservée pour indiquer si besoin la fin d'un son. Voici donc trois méthodes pour constituer une bibliothèque de sons :

6.1.2.1 *Sons numérisés*

Le principe de la numérisation consiste à convertir en données numériques des informations de type analogiques. Dans le cas du son, une source sonore (micro, radio, lecteur CD, etc...) est reliée au GS et les impulsions sonores sont transformées en valeurs. Bien entendu pour réaliser ces convertions, il faut disposer du matériel nécessaire et il vous faudra acheter une carte d'extension, de numérisation. Cependant la plupart des cartes stéréo possèdent également la numérisation et votre investissement se révélera vite rentable. De plus un logiciel de numérisation est normalement fourni avec ces cartes.

Les sons ainsi obtenus ont une origine réelle mais leurs qualités à la restitution dépendent de plusieurs paramètres : le matériel utilisé bien sûr, mais aussi la fréquence d'échantillonnage. L'échantillonnage consiste à convertir en un certain temps un certain nombre de valeurs. Ainsi, si le son que vous souhaitez numériser à une durée réelle de 3 secondes, vous pouvez choisir par exemple de prendre des échantillons toute les 40 micro secondes ou toutes les 80 micro secondes. Dans le premier cas, le son aura un meilleur rendu que dans le deuxième cas, mais il sera également deux fois plus long en mémoire ! Hélas la numérisation est couteuse en place mémoire, et les 64 K de Ram Son peuvent parfois être un peu justes.
La plupart du temps, les sons numérisés sont joués en mode «One-shot» pour un seul bruit, ou en mode «Free-run» pour des répétitions. Mais, étant donné que la numérisation suppose l'achat de matériel, il me sera impossible de donner ici un exemple.

6.1.2.2 *Sons synthétisés*

La synthèse sonore est complètement différente par rapport à la numérisation :

- Les sons créés n'ont pas d'origine réelle.
- La taille des sons est très courte. La plupart du temps 256 octets.

Le principe de la synthèse consiste lui à créer une forme d'onde qui représentera une période et qui sera jouée en mode «Free-run», c'est à dire en continu. Toute forme d'onde est imaginable, mais certaines formes sont plus harmonieuses que d'autres et seul l'écoute finale pourra être un critère de choix.

Pour notre exemple, nous allons créer une onde de type sinusoidale à l'aide d'un petit programme en Basic :

```
10 FOX X=0 TO 255
20 Y=INT (128-127*SIN(X/40.6))
30 POKE 16384+X,Y
40 NEXT X
```

Ligne 10 : On veut créer une onde de 256 octets.
Ligne 20 : Il faut des valeurs entières. Le 128 est utilisé car il correspond au milieu de l'axe des ordonnées, si l'on imagine un repère représentant la courbe. La formule a été étudiée pour ne pas fournir de valeur nulle. Rappelez vous qu'un zéro indiquerait la fin du son, or nous voulons jouer le son en continu. Rappelez vous également que le Basic Applesoft calcule en radians.
Ligne 30 : L'onde sera stockée à partir de $4000 jusqu'à $40FF.

Puis nous allons transférer l'onde en Ram Son aux adresses $0000-$00FF.Ensuite, il faudra programmer les registres du DOC pour enfin pouvoir entendre quelque chose. Nous allons utiliser l'oscillateur n $00 pour notre exemple, avec une fréquence de $65B et son volume au maximum. Mais avant de commencer la programmation, voici un petit tableau récapitulatif des valeurs exactes à mettre dans les registres :

n	Valeur
$00	$5B (partie basse de la fréquence)
$20	$06 (partie haute de la fréquence)
$40	$FF (volume maximum)
$80	$00 (l'onde sonore débute en $0000 de la Ram Son)
$A0	$00 (canal de sortie à droite ; pas d'interrutpion ; mode «Free-run» ; oscillateur en action)
$C0	$00 (l'onde occupe 256 octets)

Il est judicieux de programmer les différents registres dans leur ordre, mais il faut absolument programmer le registre de contrôle en dernier, car c'est lui qui enclenchera le son. Par contre, s'il n'était pas déclaré en dernier, il y aurait certainement un problème, car le son serait joué avant que ses paramètres fussent indiqués.

Voici la routine en assembleur, qui jouera cette onde après l'avoir transféré en Ram Son :

```
*==================================
* Routine jouant un son de synthèse
*==================================
          XC              ; Directive d'assemblage
          XC              ;

ONDE   =  $4000           ; Adresse de l'onde.

          ORG  $1000      ; Ou ailleurs...

          CLC             ; Mode natif.
```



```
            XCE

*_____________________________
* Transfert en Ram Son de l'onde
*_____________________________
            REP  $30        ; Registres XY sur 16 bits
            SEP  $20        ; et A sur 8 bits.

            LDAL $E100CA    ; On choisi d'accèder à la
            AND  #%00001111 ; Ram Son avec auto-incrémentation
            ORA  #%01100000 ; des pointeurs...
            STA  $C03C

            STZ  $C03E      ; Mise à zéro des
            STZ  $C03F      ; deux pointeurs.
            LDX  #$0000     ; On veut lire depuis le début du son.
ENCORE      LDA  MEMOIRE,X  ; Lecture d'une valeur (ici depuis
                            ; $4000).
            STA  $C03D      ; Transfert en Ram Son. A noter qu'en
                            ; même temps les pointeurs sont augmen
                            ; tés d'une unité, donc il est inutile de s'oc
                            ; cuper des pointeurs.
            INX             ; Valeur suivante.
            CPX  #$100      ; A t'on déjà transféré les 256 octets ?
            BCC  ENCORE

*_____________________
* Programmation du DOC
*_____________________
            LDAL $E100CA ; Volume selon celui du beep.
            AND  #$0F
            STA  $C03C      ; Accès aux registres du DOC.

            LDA  #$00       ; Registre fréquence (bas) de l'oscillateur
                            ; n $00.
            STA  $C03E
            LDA  #$5B
            STA  $C03D
            LDA  #$20       ; Registre fréquence (haut).
            STA  $C03E
            LDA  #$06
            STA  $C03D
            LDA  #$40       ; Registre volume.
        STA  $C03E
        LDA  #$FF
        STA  $C03D
        LDA  #$80       ; Registre d'adresse en Ram Son.
```



```
STA  $C03E
LDA  #$00
STA  $C03D
LDA  #$C0      ; Registre de taille du son.
STA  $C03E
LDA  #$00
STA  $C03D
LDA  #$A0      ; Registre de contrôle de l'oscillateur.
STA  $C03E     ; Attention, il est toujours préfèrable d'écrire le
LDA  #$00      ; registre de controle en dernier, car c'est lui
STA  $C03D     ; qui va effectivement déclencher le son.

RTS            ; Sortie. Le son est en train d'être joué.
```

6.1.2.3 *Sons synthétisés utilisés comme des sons numérisés*

Ce type de son est un compromis entre les sons numériques et synthétiques. En effet, ils n'ont pas d'existance réelle tout comme les sons de synthèse, mais ils sont utilisées de la même façon que les sons numérisés. Les avantages sont d'une part la pureté des sons (contrairement à la numérisation qui produit toujours quelques parasites), d'autre part le fait de n'avoir besoin d'aucun appareil relatif à la numérisation. Cependant le désavantage est de ne pas pouvoir obtenir les sons voulus et ici encore il faudra faire beaucoup d'essais pour trouver des sons plaisants. De plus comme la taille de ce type de sons est voisine de celle des sons numérisés et qu'ils sont construits par calculs, un certain temps d'attente est nécessaire.

Pour notre exemple, nous allons créer un son de 16 K ($4000 octets) à l'aide d'un petit programme Basic. Attention, le programme prend environ six minutes pour tout calculer.

```
10 FOR X=0 TO 16383
20 Y=INT ((128+(121-(X/140))*SIN(.12*X)))
30 POKE 8192+X,Y
40 NEXT X
```

Ligne 20 : La formule est prévue pour donner un son dont le volume diminuera dans le temps. Bien entendu vous êtes libre de créer n'importe quel son à partir du moment où il ne comporte pas de valeur nulle.

Ligne 30 : Le son est stocké de $2000 à $5FFF.

Un petit «BSAVE SOUND,A$2000,L$4000» serait le bienvenu pour sauver le son sur un disque au format Prodos 8 afin de ne pas avoir à patienter six minutes à chaque fois.

Une fois le son créé et sauvé, tapez les commandes suivantes :

```
CALL -151              (passage dans le moniteur)
0<3/0.FFFFZ            (mise à zéro du banc $03)
3/0<0/2000.5FFFM       (transfert du son au début du banc $03)
```

Le programme exemple jouant un son numérisé serait le même que pour ce type de son. Ainsi nous voulons que l'oscillateur n $01 (pour changer) joue ce son une seule fois (mode «One-shot»), à fréquence $130, volume $C0, et sur le canal de gauche. Voici donc le petit tableau récapitulatif :

n°	Valeur
$01	$30 (partie basse de la fréquence)
$21	$01 (partie haute de la fréquence)
$41	$C0 (volume maximum)
$81	$00 (le son débute en $0000 de la Ram Son)
$A1	$12 (canal de sortie à gauche ; pas d'interruption ; mode «One-shot» ; oscillateur en action)
$C1	$36 (l'onde occupe 16384 octets). Référez vous au tableau du paragraphe 6.1.1.3.6 traitant des registres de taille des sons, pour trouver la bonne valeur à mettre

Puis vous pourrez exécuter le programme machine suivant :

```
*================================
* Routine jouant un son numérisé
*================================
            XC          ; Directive d'assemblage
            XC          ;

MEMOIRE  =  $030000     ; Banc de stockage des sons.

            ORG $1000   ; Ou ailleurs...

            CLC         ; Mode natif.
            XCE
```



```
*-------------------------------------
* Ecriture en Ram Son de 64 K de données
*-------------------------------------
          REP  $30      ; Registres XY sur 16 bits
          SEP  $20      ; et A sur 8 bits.

          LDAL $E100CA      ; On choisi d'accéder à la
          AND  #%00001111   ; Ram Son avec auto-incrémentation
          ORA  #%01100000   ; des pointeurs...
          STA  $C03C

          STZ  $C03E    ; Mise à zéro des
          STZ  $C03F    ; deux pointeurs.
          LDX  #$0000   ; On veut lire depuis le début du banc.
ENCORE    LDAL MEMOIRE,X ; Lecture d'une valeur du banc (ici le
                         ; banc $03).
          STA  $C03D    ; Transfert en Ram Son. A noter qu'en
                        ; même temps
                        ; les pointeurs sont augmentés d'une uni
                        ; té, donc il est inutile de s'occuper des
                        ; pointeurs.
          INX           ; Valeur suivante.
          BNE  ENCORE ; Jusqu'à ce que les 64 K soit transférés.

*-------------------
* Programmation du DOC
*-------------------
          LDAL $E100CA  ; Volume selon celui du beep.
          AND  #$0F
          STA  $C03C    ; Accès aux registres du DOC.

          LDA  #$01     ; Registre fréquence (bas) de l'oscillateur
                        ; n $01.
          STA  $C03E
          LDA  #$30
          STA  $C03D
          LDA  #$21     ; Registre fréquence (haut).
          STA  $C03E
          LDA  #$01
          STA  $C03D
          LDA  #$41     ; Registre volume.
          STA  $C03E
          LDA  #$C0
          STA  $C03D
          LDA  #$81     ; Registre d'adresse en Ram Son.
          STA  $C03E
          LDA  #$00
```

```
STA $C03D
LDA #$C1      ; Registre de taille du son.
STA $C03E
LDA #$36
STA $C03D
LDA #$A1      ; Registre de contrôle de l'oscillateur.
STA $C03E     ; Attention, il est toujours préférable d'écrire le
LDA #$12      ; registre de contrôle en dernier, car c'est lui
STA $C03D     ; qui va effectivement déclencher le son.

RTS           ; Sortie. Le son est en train d'être joué.
```

Comme vous pouvez le constater la programmation d'un son est en elle même extrêmement simple, à partir du moment où l'on connait les bases de fonctionnement de l'Ensoniq DOC.

6.1.3 Les 4 modes de fonctionnement des oscillateurs

Comme nous avons pu le voir les oscillateurs du DOC peuvent fonctionner selon quatre modes, choisis en programmant les bits 1 et 2 des registres de la série $A0. Certains de ces modes n'utilisent qu'un seul oscillateur, d'autres doivent être groupés par deux. Dans le cas où ils sont groupés par deux, il s'agit toujours d'un oscillateur pair couplé avec son voisin impair, par exemple le n $00 avec le n $01, le n $08 avec le n $09, le n $1E avec le n $1F, etc...

6.1.3.1 *Mode «One-shot»*

C'est le mode le plus simple, que nous avons déjà employé. Principalement utilisé pour des sons longs (de type son numérisé), il suffit d'un seul oscillateur pour qu'il fonctionne.
Le fait de programmer l'oscillateur avec le mode «One-shot» remet à zéro son accumulateur, c'est à dire que lorsque l'on va mettre en route l'oscillateur, le son sera joué obligatoirement depuis le début. L'oscillateur jouera une seule fois le son puis s'arrêtera de lui même à la fin du son.
Dernier détail sur le mode «One-shot» : il permet de jouer sans problème un son finissant par des zéros.

6.1.3.2 Mode «Free-run»

Egalement utilisé avec un seul oscillateur ce mode permet de jouer plusieurs fois le même son. Pour les sons de synthèses, généralement de courtes tailles (256 octets le plus souvent), c'est le mode par excellence.

Cependant, pour rejouer plusieurs fois le même son, il faut obligatoi-

rement que celui-ci ait une taille exacte (256, 512, 1024, 2048, 4096, 8192, 16384 ou 32768 octets) en ne contenant aucun zéro. Si tel était le cas, l'oscillateur s'arrêterait de lui même.

Contrairement au mode «One-shot», le mode «Free-run» ne remet pas l'accumulateur de l'oscillateur à zéro, ce qui signifie que si vous interrompez un son en train d'être joué en arrêtant l'oscillateur, puis que vous le remettiez en marche, le son reprendra la où il en était et non depuis le début.

6.1.3.3 Mode «Swap»

Ce mode qui fonctionne obligatoirement avec une paire d'oscillateurs offre des possibilités intéressantes. Le principe de fonctionnement est le suivant : un oscillateur programmé en mode «Swap» va jouer le son comme s'il était en mode «One-shot», puis va s'arrêter et mettre en action l'oscillateur voisin. Mais il y a différentes possibilités de programmer les oscillateurs pairs et impairs en combinaison avec le mode «Swap» :

Oscillateurs		Effets recherchés
Pair	Impair	
Swap	Swap	Possibilité de jouer les 64 K de la Ram Son ou un son finissant par des zéros, en émulation du mode Free-run
Swap	One-shot	Possibilité de jouer les 64 K de la Ram Son en émulation du mode «One-shot» ou faire jouer deux fois seulement le même son.
One-shot	Swap	Possibilité de faire jouer l'oscillateur pair en mode émulation «Free-run». Pour cela il suffit de déclarer l'oscillateur impair en mode «Swap» sans le mettre en marche ni programmer ses autres registres à part celui de contrôle.

Les autres combinaisons qui seraient possibles avec le mode «Swap» et les modes «Free-run» et «Sync» ne donnent rien d'intéressant.

Exemple de deux oscillateurs programmés en mode «Swap» et «Swap» afin de jouer un son finissant par des zéros. Le son sera joué par les oscillateurs n $06 et n $07 en alternance, avec une fréquence de $120 en volume maximum et sur le canal droit. Effectuez d'abord les opérations suivantes pour charger votre son :

```
BLOAD SOUND          (chargement du son en $2000)
CALL-151             (passage sous moniteur)
3/4000<0/2000.5FFFM  (transfert du son en $034000-$037FFF)
0<3/7500.7FFFM       (ici on met une suite de zéros pour que
                      notre son n'occupe plus que $3500 octets
                      de long)
```

Ainsi notre son à une taille réelle de 13568 octets. Cependant, d'après la structure de la Ram Son, il faut déclarer sa taille par excès, d'où une taille de $4000 octets et la valeur $36 à mettre dans les registres de la serie $C0. Voici maintenant le tableau récapitulatif des valeurs à mettre dans les registres des oscillateurs n $06 et n $07 :

n	Valeur
$06	$20 (partie basse de la fréquence de l'oscillateur pair)
$07	$20 (partie basse de la fréquence de l'oscillateur impair)
$26	$01 (partie haute de la fréquence de l'oscillateur pair)
$27	$01 (parite haute de la fréquence de l'oscillateur impair)
$46	$FF (volume de l'oscillateur pair)
$47	$FF (volume de l'oscillateur impair)
$86	$40 (le son commence en $4000 de la Ram Son)
$87	$40 (de même pour l'oscillateur impair)
$A6	$06 (canal de sortie à droite ; pas d'interruption ; mode «Swap» ; oscillateur en action)
$A7	$07 (canal de sortie à droite ; pas d'interruption ; mode «Swap» ; oscillateur à l'arrêt)
$C6	$36 (Notre son de 13568 octets de long est tout de même déclaré $4000, d'où la valeur $36 selon le tableau des tailles).
$C7	$36 (de même pour l'oscillateur impair)

Notez que l'oscillateur n $06 va être le premier à se mettre en action, tandis que le numéro $07 est au départ à l'arrêt. Par la suite, le n $06 passera à l'arret dès qu'il rencontrera un $00 dans la Ram Son, enclenchant immédiatement le n $07, et ainsi de suite...
Voici enfin le programme, avec la routine de transfert des sons, maintenant bien connue :

```
*=============================================
* Routine jouant un son finissant par des zéros (première méthode)
*=============================================
            XC              ; Directive d'assemblage
            XC              ;

MEMOIRE  =  $030000         ; Banc de stockage des sons.

            ORG $1000       ; Ou ailleurs...

            CLC             ; Mode natif.
            XCE

*_____________________________________________
* Ecriture en Ram Son de 64 K de données
*_____________________________________________
            REP  $30        ; Registres XY sur 16 bits
            SEP  $20        ; et A sur 8 bits.

            LDAL $E100CA        ; On choisi d'accèder à la
            AND  #%00001111     ; Ram Son avec auto-incrémentation
            ORA  #%01100000     ; des pointeurs...
            STA  $C03C

            STZ  $C03E      ; Mise à zéro des
            STZ  $C03F      ; deux pointeurs.
            LDX  #$0000     ; On veut lire depuis le début du banc.
ENCORE      LDAL MEMOIRE,X  ; Lecture d'une valeur du banc (ici le
                            ; banc $03).
            STA  $C03D      ; Transfert en Ram Son. A noter qu'en
                            ; même temps
                            ; les pointeurs sont augmentés d'une uni
                            ; té, donc il est inutile de s'occuper des
                            ; pointeurs.
            INX             ; Valeur suivante.
            BNE  ENCORE ; Jusqu'à ce que les 64 K soit transférés.
```



```
*_____________________
* Programmation du DOC
*_____________________
        LDAL $E100CA        ; Volume selon celui du beep.
        AND  #$0F
        STA  $C03C          ; Accès aux registres du DOC.

        LDA  #$06       ; Registre fréquence (bas) de l'oscillateur n $06.
        STA  $C03E
        LDA  #$20
        STA  $C03D
        INC  $C03E      ; Registre de l'oscillateur n $07.
        STA  $C03D      ; Notez que l'accumulateur contient toujours
                        ; $20.
        LDA  #$26       ; Registre fréquence (haut).
        STA  $C03E
        LDA  #$01
        STA  $C03D
        INC  $C03E
        STA  $C03D
        LDA  #$46       ; Registre volume.
        STA  $C03E
        LDA  #$FF
        STA  $C03D
        INC  $C03E
        STA  $C03D
        LDA  #$86       ; Registre d'adresse en Ram Son.
        STA  $C03E
        LDA  #$40
        STA  $C03D
        INC  $C03E
        STA  $C03D
        LDA  #$C6       ; Registre de taille du son.
        STA  $C03E
        LDA  #$36
        STA  $C03D
        INC  $C03E
        STA  $C03D
        LDA  #$A6       ; Registre de contrôle de l'oscillateur.
        STA  $C03E      ; Attention, il est toujours préférable d'écrire le
        LDA  #$06       ; registre de contrôle en dernier, car c'est lui
        STA  $C03D      ; qui va effectivement déclencher le son.
        INC  $C03E
        LDA  #$07       ; Remarquez qu'au début seul l'oscillateur pair
        STA  $C03D      ; va être en action.

        RTS             ; Sortie. Le son est en train d'être joué.
```


Cette technique pour jouer un son se terminant par des zéros est tout à fait valable, cependant elle à l'inconvénient de demander deux fois les mêmes valeurs pour les registres pair et impair. Ainsi cela peut être génant si vous avez besoin d'intervenir en court de traitement du son, car vous devrez le faire pour l'oscillateur pair, mais aussi pour l'oscillateur impair. Voici donc la seconde méthode, utilisant deux oscillateurs programmés en mode «One-shot» et «Swap». Nous garderons le même son et les mêmes paramètres. Le tableau des valeurs est le suivant :

n	Valeur
$06	$20 (partie basse de la fréquence de l'oscillateur pair)
$26	$01 (partie haute de la fréquence de l'oscillateur pair)
$46	$FF (volume de l'oscillateur pair)
$86	$40 (le son commence en $4000 de la Ram Son)
$A6	$02 (canal de sortie à droite ; pas d'interruption ; mode «One-shot» ; oscillateur en action)
$A7	$07 (canal de sortie à droite ; pas d'interruption ; mode «Swap» ; oscillateur à l'arrêt)
$C6	$36 (Notre son de 13568 octets de long est tout de même déclaré $4000, d'où la valeur $36 selon le tableau des tailles).

Notez que seul le fait que l'oscillateur impair soit en mode «Swap» arrêté est important. On se moque des autres paramètres de l'oscillateur impair. Voici maintenant la routine :

```
*================================================
* Routine jouant un son finissant par des zéros (seconde méthode)
*================================================
            XC              ; Directive d'assemblage
            XC              ;
MEMOIRE  =  $030000         ; Banc de stockage des sons.

            ORG $1000       ; Ou ailleurs...

            CLC             ; Mode natif.
            XCE
```



```
*-------------------------------------------
* Ecriture en Ram Son de 64 K de données
*-------------------------------------------
          REP  $30      ; Registres XY sur 16 bits
          SEP  $20      ; et A sur 8 bits.

          LDAL $E100CA       ; On choisi d'accéder à la
          AND  #%00001111    ; Ram Son avec auto-incrémentation
          ORA  #%01100000    ; des pointeurs...
          STA  $C03C

          STZ  $C03E         ; Mise à zéro des
          STZ  $C03F         ; deux pointeurs.
          LDX  #$0000        ; On veut lire depuis le début du banc.
ENCORE    LDAL MEMOIRE,X     ; Lecture d'une valeur du banc (ici le
                             ; banc $03).
          STA  $C03D         ; Transfert en Ram Son. A noter qu'en
                             ; même temps
                             ; les pointeurs sont augmentés d'une uni
                             ; té, donc il est inutile de s'occuper des
                             ; pointeurs.
          INX                ; Valeur suivante.
          BNE  ENCORE ; Jusqu'à ce que les 64 K soit transférés.


*-----------------------
* Programmation du DOC
*-----------------------
          LDAL $E100CA ; Volume selon celui du beep.
          AND  #$0F
          STA  $C03C     ; Accès aux registres du DOC.

          LDA  #$06      ; Registre fréquence (bas) de l'oscillateur
                         ; n $06.
          STA  $C03E
          LDA  #$20
          STA  $C03D
          LDA  #$26      ; Registre fréquence (haut).
          STA  $C03E
          LDA  #$01
          STA  $C03D
          LDA  #$46      ; Registre volume.
          STA  $C03E
          LDA  #$FF
          STA  $C03D
          LDA  #$86      ; Registre d'adresse en Ram Son.
          STA  $C03E
```



```
LDA  #$40
STA  $C03D
LDA  #$C6      ; Registre de taille du son.
STA  $C03E
LDA  #$36
STA  $C03D
LDA  #$A6      ; Registre de contrôle de l'oscillateur.
STA  $C03E     ; Attention, il est toujours préférable d'écrire le
LDA  #$02      ; registre de contrôle en dernier, car c'est lui
STA  $C03D     ; qui va effectivement déclencher le son.
INC  $C03E
LDA  #$07      ; Attention l'oscillateur impair doit absolument
STA  $C03D     ; être en mode «Swap» et à l'arret.

RTS            ; Sortie. Le son est en train d'être joué.
```

6.1.3.4 *Mode «Sync»*

Le mode «Sync» utilise aussi une paire d'oscillateurs, et généralement seul l'oscillateur impair sera programmé avec ce mode. L'effet provoqué est la synchronisation de l'oscillateur impair sur le pair, en produisant un meilleur niveau sonore. Il n'est pas besoin de programmer les autres registres de l'oscillateur impair, seul son registre de contrôle doit être en mode «Sync» et peut même être arrêté.
Il faut avouer que ce mode n'est pas d'une grande utilité, bien qu'il soit possible d'obtenir certains effets selon la façon de programmer les deux oscillateurs. Par exemple mettre les deux oscillateurs en mode «Sync», ou seulement le premier, ou encore choisir une fréquence différente pour chacun d'eux... Les possibilités sont trop nombreuses pour donner un exemple, et il vous sera facile de les tester vous même.

6.1.4 Les interruptions sonores

La facilité du traitement des interruptions du GS est telle qu'il serait dommage de s'en priver dans le cas du son. En effet chaque oscillateur du DOC a la possibilité d'envoyer un signal d'interruption s'il a été programmé pour. Le signal est déclenché dès la fin du son.
Si vous utilisez le système normal de gestion des interruptions, le vecteur de connexion du son se trouve en $E1002C et il vous suffit d'y mettre un JMP long vers votre routine de traitement. Par la suite votre routine devra déterminer, si nécessaire, lequel des 32 oscillateurs a produit l'interruption, en utilisant le registre $E0 du DOC (voir le paragraphe 6.1.1.3.7 pour l'utilisation de ce registre). Bien entendu, s'il n'y a qu'un seul oscillateur programmé pour produire des inter-

ruptions, il sera inutile de se servir du registre $E0.

Notre exemple aura pour but de changer la couleur du bord d'écran à chaque fois que le son sera joué.

```
BLOAD SOUND        (chargement du son en $2000)
CALL-151           (passage sous moniteur)
3/0<0/2000.5FFFM   (transfert du son en $030000-$033FFF)
```

Nous utiliserons l'oscillateur n $1F, programmé pour jouer le son en continu sur la canal gauche, avec une fréquence de $160, un volume de moitié et l'autorisation d'interruption. Voici le tableau des valeurs:

n	Valeur
$1F	$60 (partie basse de la fréquence)
$3F	$01 (partie haute de la fréquence)
$5F	$80 (volume de moitié)
$9F	$00 (le son commence en $0000 de la Ram Son)
$BF	$18 (canal de sortie à gauche ; interruptions autorisée ; mode «Free-run» ; oscillateur en action)
$DF	$36 (Notre son occupe $4000 octets de long).

```
*================================================
* Routine jouant un son produisant des interruptions
*================================================
            XC              ; Directive d'assemblage
            XC              ;
MEMOIRE  =  $030000         ; Banc de stockage des sons.

            ORG  $1000      ; Ou ailleurs...

            CLC             ; Mode natif.
            XCE

*________________________________________
* Ecriture en Ram Son de 64 K de données
*________________________________________
            REP  $30        ; Registres XY sur 16 bits
            SEP  $20        ; et A sur 8 bits.

            LDAL $E100CA        ; On choisi d'accéder à la
            AND  #%00001111     ; Ram Son avec auto-incrémentation
```



```
            ORA  #%01100000    ; des pointeurs...
            STA  $C03C

            STZ  $C03E         ; Mise à zéro des
            STZ  $C03F         ; deux pointeurs.
            LDX  #$0000        ; On veut lire depuis le début du banc.
ENCORE      LDAL MEMOIRE,X     ; Lecture d'une valeur du banc (ici le
                               ; banc $03).
            STA  $C03D         ; Transfert en Ram Son. A noter qu'en
                               ; même temps
                               ; les pointeurs sont augmentés d'une uni
                               ; té, donc il est inutile de s'occuper des
                               ; pointeurs.
            INX                ; Valeur suivante.
            BNE  ENCORE ; Jusqu'à ce que les 64 K soit transférés.


*_____________________________________
* Détournement du vecteur son
* vers notre routine de traitement
*_____________________________________
            LDA  #$5C          ; Code du JMP long.
            STAL $E1002C ; $E1002C est le vecteur son.
            LDA  #<INTER; Partie basse de l'adresse de notre routine
            STAL $E1002D ; de traitement des interruptions sonores.
            LDA  #>INTER; Partie haute de l'adresse.
            STAL $E1002E
            LDA  #$00          ; Nous sommes dans le banc $00.
            STAL $E1002F

*_____________________
* Programmation du DOC
*_____________________
            LDAL $E100CA ; Volume selon celui du beep.
            AND  #$0F
            STA  $C03C         ; Accès aux registres du DOC.

            LDA  #$1F          ; Registre fréquence (bas) de l'oscillateur
                               ; n $06.
            STA  $C03E
            LDA  #$60
            STA  $C03D
            LDA  #$3F          ; Registre fréquence (haut).
            STA  $C03E
            LDA  #$01
            STA  $C03D
            LDA  #$5F          ; Registre volume.
```



```
        STA  $C03E
        LDA  #$80
        STA  $C03D
        LDA  #$9F       ; Registre d'adresse en Ram Son.
        STA  $C03E
        LDA  #$00
        STA  $C03D
        LDA  #$DF       ; Registre de taille du son.
        STA  $C03E
        LDA  #$36
        STA  $C03D
        LDA  #$BF       ; Registre de contrôle de l'oscillateur.
        STA  $C03E      ; Attention, il est toujours préférable
        LDA  #$18       ; d'écrire le registre de contrôle en dernier,
        STA  $C03D      ; car c'est lui qui va effectivement déclen
                        ; cher le son.

        RTS             ; Sortie. Le son est en train d'être joué et le
                        ; bord d'écran change à chaque fin du son.

*-------------------------------
* Traitement de l'interruption
*-------------------------------
INTER   SEP  $30        ; A et XY sur 8 bits.
        INC  $C034      ; Change la couleur du bord d'écran.
        CLC             ; Fin de la
        RTL             ; routine d'interruption.
```

Cette routine, qui une fois lancée fonctionne automatiquement n'est qu'un simple exemple, mais vous pouvez déjà vous rendre compte des possibilités. Le fait de changer la couleur du bord d'écran n'est guère intéressant, mais vous remarquerez qu'en changeant la fréquence de l'oscillateur, vous changerez également la vitesse de variation de la couleur. Effectivement, cette routine n'est autre qu'un tempo, qui nous sera par la suite très utile pour la musique.

6.1.5 La stéréo

Si en dehors de l'extension mémoire il n'y avait qu'une seule carte à acheter, ce serait une carte stéréo ! En effet, le GS et ses capacités sonores le méritent bien et vous ne regretterez pas non plus d'y ajouter des enceintes. Ainsi une telle carte permet pour chaque oscillateur de choisir si le son sortira à droite ou à gauche. Du point de vue programmation, chaque oscillateur programmé avec un canal pair jouera à droite, tandis qu'avec un canal impair ce sera à gauche.

Cependant il est préférable de choisir le canal zéro pour la droite et le un pour la gauche.
Outre la stéréo cette extension offre d'autres avantages. Déjà, le fait de connecter des enceintes sur la sortie de la carte et non sur la sortie du GS évite certains parasites, car le son ne transite plus que par l'interface de l'Ensoniq. De part ce fait, le beep du contrôle G ne sera pas reproduit par les enceintes connectées à la carte stéréo. Ceci est très important car vous pourrez baisser le niveau du volume de beep pour ne pas être gené, sans pour autant perdre le volume des sons du GS, étant donné que le réglage du volume général par le tableau de bord ne concerne que le haut-parleur interne et la prise «jack» du GS.
La plupart des cartes stéréo comporte également un système de numérisation, et vous auriez tord de vous en priver. De plus, certaines cartes possèdent leur propre amplificateur réglable.
Mais si vous désirez donc acheter une carte stéréo, prévoyez également l'achat d'enceintes, auto amplifiées de préférence, à moins que vous ne souhaitiez brancher votre chaine HIFI via la carte.

6.1.6 Effets spéciaux et divers

Cette partie contient quelques exemples de ce que l'on peut faire avec le GS et le son, mais elle n'est pas exhaustive.

6.1.6.1 *Echo simple*

Le but pour produire un écho est de faire jouer un son une première fois à volume élevé, puis une seconde fois moins fort. Mais tout l'intérêt est de le faire de façon automatique, sans intervention d'un programme en cours de traitement. Pour cela, il suffit d'utiliser deux oscillateurs, le premier programmé en mode «Swap» en action, volume maximum et le second en mode «One-Shot» arrêté, volume plus faible. Bien entendu, il faut que la fréquence des deux oscillateurs soit la même, à moins que vous ne désiriez un autre effet. Vu la simplicité de la routine, il me parait inutile de vous la donner.

6.1.6.2 *Passage droite-gauche*

L'effet de passage d'un son d'un haut-parleur à l'autre pendant qu'il est en train d'etre joué est assez intéressant, mais il est nécessaire qu'un programme intervienne en cours de traitement, et vous ne pourrez donc rien faire d'autre pendant ce temps.
Le principe consiste à utiliser deux oscillateurs programmés en mode «One-shot» qui joueront le son en meme temps, l'un sur la droite à volume élevé, l'autre sur la gauche à volume nul. Mais pendant que

le son sera joué, une routine dimminuera le volume du premier oscillateur tout en augmentant celui du second. L'effet procure une agréable sensation, pour peu que les deux haut-parleurs soient bien disposés et à condition bien entendu de posséder une carte stéréo, sans laquelle il n'y aurait aucun effet.
Pour notre exemple, nous allons charger notre son comme d'habitude, et nous le ferons jouer à fréquence $120.

```
BLOAD SOUND        (chargement du son en $2000)
CALL-151           (passage sous moniteur)
3/0<0/2000.5FFFM   (transfert du son en $030000-$033FFF)
```

Nous utiliserons les oscillateurs n $00 et n $01, Voici le tableau des valeurs :

n	Valeur
$00	$20 (partie basse de la fréquence de l'oscillateur pair)
$01	$20 (partie basse de la fréquence de l'oscillateur impair)
$20	$01 (partie haute de la fréquence de l'oscillateur pair)
$21	$01 (parite haute de la fréquence de l'oscillateur impair)
$40	$FF (volume de l'oscillateur pair)
$41	$00 (volume de l'oscillateur impair)
$80	$00 (le son commence en $0000 de la Ram Son)
$81	$00 (de même pour l'oscillateur impair)
$A0	$02 (canal de sortie à droite ; pas d'interruption ; mode «One-shot» ; oscillateur en action)
$A1	$12 (canal de sortie à gauche ; pas d'interruption ; mode «One-shot» ; oscillateur en action)
$C0	$36 (Notre son occupe $4000 octets)
$C1	$36 (de même pour l'oscillateur impair)

```
*=========================================
* Routinue jouant un son de droite à gauche
*=========================================
          XC              ; Directive d'assemblage
          XC              ;
MEMOIRE =  $030000        ; Banc de stockage des sons.

          ORG  $1000      ; Ou ailleurs...

          CLC             ; Mode natif.
          XCE
```



```
*-------------------------------------------
* Ecriture en Ram Son de 64 K de données
*-------------------------------------------
          REP  $30      ; Registres XY sur 16 bits
          SEP  $20      ; et A sur 8 bits.

          LDAL $E100CA       ; On choisi d'accèder à la
          AND  #%00001111    ; Ram Son avec auto-incrémentation
          ORA  #%01100000    ; des pointeurs...
          STA  $C03C

          STZ  $C03E    ; Mise à zéro des
          STZ  $C03F    ; deux pointeurs.
          LDX  #$0000   ; On veut lire depuis le début du banc.
ENCORE    LDAL MEMOIRE,X ; Lecture d'une valeur du banc (ici le
                         ; banc $03).
          STA  $C03D    ; Transfert en Ram Son. A noter qu'en
                        ; même temps
                        ; les pointeurs sont augmentés d'une uni
                        ; té, donc il est inutile de s'occuper des
                        ; pointeurs.
          INX           ; Valeur suivante.
          BNE  ENCORE ; Jusqu'à ce que les 64 K soit transférés.


*-----------------------
* Programmation du DOC
*-----------------------
          LDAL $E100CA ; Volume selon celui du beep.
          AND  #$0F
          STA  $C03C    ; Accès aux registres du DOC.

          LDA  #$00     ; Registre fréquence (bas).
          STA  $C03E
          LDA  #$20
          STA  $C03D
          INC  $C03E
          STA  $C03D
          LDA  #$21     ; Registre fréquence (haut).
          STA  $C03E
          LDA  #$01
          STA  $C03D
          INC  $C03E
          STA  $C03D
          LDA  #$40     ; Registre volume.
          STA  $C03E
          LDA  #$FF
```



```
            STA  $C03D
            INC $C03E
            LDA  #$00
            STA  $C03D
            LDA  #$80       ; Registre d'adresse en Ram Son.
            STA  $C03E
            LDA  #$00
            STA  $C03D
            INC  $C03E
            STA  $C03D
            LDA  #$C0       ; Registre de taille du son.
            STA  $C03E
            LDA  #$36
            STA  $C03D
            INC  $C03E
            STA  $C03D
            LDA  #$A0       ; Registre de contrôle de l'oscillateur.
            STA  $C03E
            LDA  #$02
            STA  $C03D
            INC  $C03E
            LDA  #$12       ; A partir d'ici, le son commence à être
            STA  $C03D      ; joué, mais au début on ne l'entend qu'à
                            ; droite.

*_____________________
* Modification du volume
*_____________________
            LDY  #$FF       ; On va maintenant faire «glisser» le son
                            ; vers la gauche. Progressivement 255 fois.
BOUCLE      LDA  #$40       ; Accès au volume de droite.
            STA  $C03E
            LDA  $C03D      ; Deux lectures nécessaires.
            LDA  $C03D
            DEC             ; On diminue d'un.
            STA  $C03D
            LDA  #$41       ; Accès au volume de gauche.
            STA  $C03E
            LDA  $C03D
            LDA  $C03D
            INC             ; Mais cette fois on augmente d'un.
            STA  $C03D
            LDX  #$FF       ; Pour ne pas obtenir un changement trop
                            ; rapide,

BOUCLE2     JSR  ATTENTE    ; il est nécessaire d'avoir une routine de
                            ; timing.
```

```
        DEX             ; Timing qui dépendra de la longeur du
        BNE BOUCLE2     ; son et de la valeur de la fréquence.
        DEY
        BNE BOUCLE

        RTS             ; Fin de routine.

ATTENTE DS  18,$EA      ; Dans ce cas précis (longeur du son de
                        ; $4000 octets et fréquence de $120), il faut
                        ; 18 NOP (dont le code est $EA).
        RTS             ; Fin du sous-programme de timing.
```

Notez que le réglage à pour fonction de bien répartir l'effet de passage du volume de la droite vers la gauche, afin que le volume maximum arrive sur le haut-parleur de gauche exactement à la fin du son.

6.1.6.3 *Fréquences décalées*

L'effet appelé «Fréquences décalées» peut être parfois très surprenant lorsque l'on ne s'y attend pas. Le principe consiste à faire jouer un même son par plusieurs oscillateurs à la fois, mais pas avec la même fréquence. Par exemple, le premier oscillateur jouerait le son avec une fréquence de $120, le deuxième avec $121, le troisième avec $122, etc... Le mode de fonctionnement de tous les oscillateurs serait soit le «One-shot», soit le «Free-run». Quoi qu'il en soit l'effet est spectaculaire car au début les sons joués commencent en même temps, mais très vite ils se décalent les uns des autres. Plus vous utiliserez un nombre important d'oscillateurs, plus le résultat sera saisissant, mais veillez tout de même à ne pas saturer les enceintes en baissant le volume des oscillateurs le cas échéant.

6.1.6.4 *Ensoniq et le hasard*

Il arrive assez souvent que les programmeurs aient besoin de nombres aléatoires dans leurs applications, notamment s'il s'agit de jeux où le hasard tient une place importante. Il existe divers moyens d'obtenir des nombres tirés au hasard, et je vais vous proposer ici une méthode originale basée sur une utilisation détournée des registres du DOC de la série $60.
En effet, ces registres qui contiennent la dernière valeur lue par leur oscillateur correspondant (cf. paragraphe 6.1.1.3.3) n'ont pas d'utilisation pratique pour le son.
Mais imaginons maintenant qu'un oscillateur joue un son de 256 octets en continu, à volume nul. Il suffirait de lire son registre de donnée correspondant pour obtenir une des 256 valeurs contenues

dans l'onde de la Ram Son. Il suffirait donc de mettre des valeurs quelconques dans cette onde (à l'exclusion des zéros bien sûr) et de lire lorsque l'on en a besoin le registre de donnée pour obtenir un nombre aléatoire. Voici les bases d'un exemple :

- Programmer les 256 premiers octets de la Ram Son avec une suite de $01 et de $02.
- Programmer l'oscillateur n $00 pour jouer les 256 premiers octets de la Ram Son en mode «Free-run», fréquence $99 (choisir de préférence une fréquence impaire), volume $00 (car il n'y a aucun intérêt d'entendre ce son).
- Quand vous aurez besoin d'une valeur tirée au hasard (ici soit le $01, soit le $02), effectuer les instructions suivante :

```
LDA  #$60      ; Accès au registre de donnée.
STA  $C03E
LDA  $C03D     ; Deux lectures nécessaires.
LDA  $C03D     ; L'accumulateur contiendra une valeur aléa
               ; toire.
```

Les avantages d'une telle routine sont : la facilité, le fait qu'elle soit automatique, la possibilité de configurer exactement les nombres que vous souhaitez. En effet, vous pouvez très bien, dans notre exemple, mettre un plus fort pourcentage de $01 que de $02 afin de favoriser tel ou tel nombre. Et oui, vous pourrez vous même configurer les probabilités !
Seule restriction : il est nécessaire de lire le registre de donnée d'une façon irrégulière. Mais dans le cas d'un grand programme, il est rare d'avoir des cycles de temps réguliers.

6.2 La Musique

Si l'Ensoniq est capable de gérer automatiquement les sons, il n'en va pas de même pour la musique. Ainsi vous devrez vous même élaborer votre routine. L'idéal est de créer une routine de musique indépendante de votre programme, en utilisant les interruptions bien entendu. Comme cela il n'y aura pas de problème de décalage temporel dépendant de votre programme et, une fois lancée, la musique s'exécutera en prenant un minimum de temps machine. Mais de toute façon, faire de la musique sur GS n'est que l'aboutissement logique du son et la programmation reste d'un niveau simple.

6.2.1 La méthode des 64 K

Si pour une raison ou une autre vous avez besoin de tout votre temps machine ou de toute la RAM du GS et que vous désiriez tout de même

entendre une musique, voici une méthode très simple. Il s'agit de numériser 64 K de musique depuis un CD ou autre, puis de jouer les quelques secondes en continu de façon à ce que l'on ait l'impression d'entendre un morceau sans fin. Bien entendu, vous aurez tout intéret de choisir un passage bien rythmé et sans parole. Hélas, au bout de quelques instant d'écoute, même si le résultat est tout à fait acceptable, la musique n'en devient pas moins très lassante.
Donc, si vous choisissez cette solution, il vous suffira de programmer deux oscillateurs en mode «Swap», même fréquence pour les deux. Le premier oscillateur jouant les 32 premiers Kilos octets de la Ram Son, tandis que le second les 32 autres Kilos.

6.2.2 Musique et interruptions

Par contre, si vous cherchez une véritable solution pour jouer de la musique dans vos programmes, vous devrez utiliser les interruptions. Il est certain qu'il existe plusieurs méthodes possibles avec les interruptions, mais celle consistant à s'en servir comme tempo me parait la plus intéressante du point de vue facilité et rapidité d'exécution.
Ici, les sons représentants les instruments seront stockés en Ram Son, et les notes avec leurs durées seront inscrites quelque part en RAM normale. Les notes devront être converties en fréquences selon une table donnée, tandis que les durées seront en fait des délais d'attente entre chaque changement de note. Hélas il est très difficile de trouver les véritables fréquences correspondantes aux notes de la gamme, étant donné que les sons n'ont pas tous été numérisés à la même fréquence.
Pour fonctionner, la routine devra interrompre le programme en cours tout les x temps et effectuer les divers changement de notes et de durée. Bien entendu, le programmeur devra veiller à ce que sa routine de musique soit la plus rapide possible afin de ne pas trop perturber le programme principal.

Voici donc comme exemple une petite routine jouant une musique avec un seul son. La routine utilise alternativement l'oscillateur n $00 et n $01 pour éviter d'entendre la mise en action et l'arrêt des oscillateurs.
Pour exécuter cette routine, il suffit de charger notre son :

```
BLOAD SOUND        (chargement du son en $2000)
CALL-151           (passage sous moniteur)
3/0<0/2000.5FFFM   (transfert du son en $030000-$033FFF)
```

Puis il suffit de lancer la routine en $1000.


```
*================================
* Routine de musique
*================================
            XC              ; Directive d'assemblage.
            XC

MEMOIRE  =  $030000     ; Banc contentant les sons.
FREQBAS  =  $FE         ; Deux octets de la page zéro
FREQHAUT =  $FF         ; utilisés pour la sauvegarde des fréquen
                        ; ces.

            ORG $1000   ; Ou ailleurs...

            CLC         ; Mode natif.
            XCE

*--------------------------------------
* Ecriture en Ram Son de 64 K de données
*--------------------------------------
            REP $30     ; Registres XY sur 16 bits
            SEP $20     ; et A sur 8 bits.

            LDAL $E100CA        ; On choisi d'accèder à la
            AND  #%00001111     ; Ram Son avec auto-incrémentation
            ORA  #%01100000     ; des pointeurs...
            STA  $C03C


            STZ  $C03E      ; Mise à zéro des
            STZ  $C03F      ; deux pointeurs.
            LDX  #$0000     ; On veut lire depuis le début du banc.
ENCORE      LDAL MEMOIRE,X  ; Lecture d'une valeur du banc (ici le
                            ; banc $03).
            STA  $C03D      ; Transfert en Ram Son. A noter qu'en
                            ; même temps
                            ; les pointeurs sont augmentés d'une uni
                            ; té, donc il est inutile de s'occuper des
                            ; pointeurs.
            INX             ; Valeur suivante.
            BNE  ENCORE ; Jusqu'à ce que les 64 K soit transférés.


*--------------------------------
* Détournement du vecteur son
*--------------------------------
            SEP $30
            LDA  #$5C       ; $5C = opcode du JMP long.
```

```
                STAL $E1002C  ; Le vecteur d'interruption du son
                LDA  #<INTER; se trouve en $E1002C.
                STAL $E1002D
                LDA  #>INTER
                STAL $E1002E
                LDA  #$00
                STAL $E1002F

*_____________________________
* Mise en place du tempo
*_____________________________
                LDAL $E100CA
                AND  #$0F
                STA  $C03C      ; Accès aux registres du DOC.

                LDA  #$1F       ; On utilisera l'oscillateur n $1F
                STA  $C03E    ; pour le tempo, avec une fréquence de $100.
                LDA  #$00     ; En augmentant la fréquence, vous jouerez
                STA  $C03D    ; la musique plus rapidement.
                LDA  #$3F
                STA  $C03E
                LDA  #$01
                STA  $C03D

                LDA  #$5F     ; Le volume de cet oscillateur est à zéro étant
                STA  $C03E    ; donné que nous ne voulons pas entendre le
                LDA  #$00     ; son du tempo.
                STA  $C03D

                LDA  #$9F
                STA  $C03E
                LDA  #$00     ; Le «Son» joué commence en $0000 de la Ram
                STA  $C03D ; Son et n'a qu'une durée de 256 octets.
                LDA  #$BF
                STA  $C03E
                LDA  #$00
                STA  $C03D

                LDA  #$BF
                STA  $C03E
                LDA  #$08       ; Le mode utilisé est le «Free-run»
                STA  $C03D      ; avec interruption.

*_____________________________________________________
* Valeurs fixes pour les oscillateur n $00 et n $01
*_____________________________________________________
                LDA  #$40       ; Le volume des deux oscillateurs
```



```
            STA  $C03E      ; sera à $50.
            LDA  #$50
            STA  $C03D
            INC  $C03E
            STA  $C03D

            LDA  #$80       ; Le son utilisé commencera en $0000
            STA  $C03E      ; de la Ram Son.
            LDA  #$00
            STA  $C03D
            INC  $C03E
             STA  $C03D
            LDA  #$C0       ; Le son aura une taille de $4000 octets.
            STA  $C03E
            LDA  #$36
            STA  $C03D
            INC  $C03E
            STA  $C03D

*_____________________________
* Initialisation
*_____________________________
            REP  $30        ; A et XY sur 16 bits.
            STZ  NUMERO     ; Remise à zéro du compteur des notes
            STZ  DELAI      ; et du délai.

            RTS             ; Retour à l'appelant. A cet instant, la
                            ; musique est lancée et fonctionne auto
                            ; matiquement.

*_____________________________
* Routine d'interruption
*_____________________________
INTER       REP  $30        ; A chaque interruption provoquée par
                            ; l'oscillateur n $1F (tempo), le GS exécute
                            ; cette routine.

            LDA  DELAI      ; Si le délai est arrivé à zéro, alors c'est que
            BEQ  LIBRE      ; la note jouée est terminée.
            DEC  DELAI      ; Sinon elle est en train d'être jouée et on

            BRA  FIN        ; décrémente le délai, puis on sort.

LIBRE       LDX  NUMERO     ; X contient le numéro (*2) de la note à
                            ; jouer.
            LDA  MUSIQUE,X  ; A contient la note et sa durée.
```



```
CMP  #$FFFF   ; Si A = $FFFF alors on est en fin de
              ; morceau
BEQ  RESET    ; et on se branche à RESET pour recom
              ; mencer.
AND  #$00FF   ; On ne garde que la note.
ASL           ; Multiplication par 2 car les fréquences
TAY           ; sont codées sur deux octets. Puis trans
              ; fert dans Y.
 LDA  NOTE,Y  ; A contient maintenant la fréquence sur
              ; 16 bits.
SEP  $20      ; Mise de A sur 8 bits. Mais les 8 bits de
              ; poidfort de A restent cependant inchan
              ; gés.
STA  FREQBAS  ; Sauvegarde dans une mémoire de la
              ; fréquence (bas)
XBA           ; Les 8 bits de poids forts de A sont échan
              ; gés avec les 8 bits de poids faibles.

STA  FREQHAUT ; Sauvegarde de la fréquence (haut).

LDA  #$A0     ; Accès au registre de mode de l'oscilla
CLC           ; teur n $00 ou n $01 en fonction de celui
ADC  UNCOUP   ; qui est en train de jouer la note.
STA  $C03E

LDA  #$03     ; Mise à l'arret de cet oscillateur au cas où
STA  $C03D    ; il ne se serait pas encore arreté de lui
              ; même.

LDA  UNCOUP; Commutation de la variable UNCOUP
EOR  #$01     ; pour changer d'oscillateur.
STA  UNCOUP

LDA  #$00     ; Programmation de la fréquence de l'os
              ; cillateur

CLC
ADC UNCOUP
STA $C03E
LDA FREQBAS
```



```
            STA $C03D
            LDA #$20
            CLC
            ADC UNCOUP
            STA $C03E
            LDA FREQHAUT
            STA $C03D

            LDA #$A0          ;Mise en action de l'oscillateur avec
            CLC               ;le mode "One-Shot", canal de droite
            ADC UNCOUP
            STA $C03E
            LDA #$02
            STA $C03D

            REP $30           ; Passage de A sur 8 bits
            LDA MUSIQUE,X     ; Le registre X qui est toujours en 16
                              ; bits contient encore la valeur du
                              ; numéro de la note.
            XBA               ; On commute A, pour accéder à la
                              ; durée
            AND #$00FF        ;et on ne garde que la durée.
            STA DELAI         ; Stockage de la durée dans la varia
                              ; ble DELAI
            INX               ; Note suivante
            INX
            STX NUMERO        ; On sauve dans NUMERO
            BRA FIN           ; Et c'est fini
RESET       LDX #$0000        ; Mise à zéro de la variable
                              ; NUMERO
            STX NUMERO
FIN         CLC               ; CLC avant de sortir d'une routine
                              ; d'interruption
            RTL               ; Fin de routine d'interruption.

*
* Table et sauvegarde des données
*

NOTE        HEX 60008000A000C000 ;Table de convertion note
                                 ; -> fréquence
            HEX E000000120014001 ; les fréquences sont codés sur
                                 ; 16 bits
NUMERO HEX 0000       ; 2 octets réservés pour stocker le numéro
                      ; de la note. Les notes étant sur 2 octets, le
                      ; numéro est multiplé par 2.
```

```
UNCOUP   HEX 0000          ;  la variable UNCOUP est utilisée pour
                           ; déterminer si c'est l'oscillateurn $00 ou
                           ; n $01qui doit être utilisé.

MUSIQUE  HEX 0110011002200330 ; Table contenant le numéro de
                              ; musique

         HEX 0110052005200340 ; selon la structure suivante : un
         HEX 0610071006100510 ; octet pour la note, suivi d'un
         HEX 02200220FFFF     ; octet pour la durée. Une paire
                              ; $FFFF signifie la fin du mor
                              ; ceau.
```

Certes cet exemple ne joue de la musique que sur une seule voix, alors que le GS possède 32 oscillateurs. En tenant compte qu'il est préférable de prendre deux oscillateurs par instrument et que l'oscillateur n $1F est utilisé pour le tempo, cela nous fait 15 voix utilisables plus l'oscillateur n $1E de libre.

Création, mise en page, maquette, Impression
Sauvear Cacoab, Forgest
Imprimé en France
Dépôt légal, mars 1990

"Le livre français qui vous fera comprendre le IIGS"

par :

D.BÄR
D.DELAY
Y.DURANT
J.L. SCHMITT
E.ric WEYLAND

Dans ce livre de 464 pages, vous trouverez des informations techniques détaillées sur :

- le 65C816, ses registres, ses instructions, ses modes d'adressage ...
- les drives 5,25" & 3,5", lecture, écriture, déplacement du bras ...
- le SmartPort ...
- l'organisation de la mémoire, les softswiches, le shadowing, la BRAM ...
- le graphisme, les animations, les interruptions, la structure SCB ...
- le système d'exploitation GS/OS et Prodos, l'organisation des données sur disque ...
- le son, la Ram son, le GLU, le DOC, l'Ensoniq, la stéréo...

ISBN 2-9504508-0-6
Mars 1990, Edité par *Toolbox*
Dépôt légal, 3/90
Imprimé en France
Conception, mise en page, maquette, impression
Sauvear Cacoab, Forgest