MATÉRIELS

SYSTÈME ProDOS DE L'APPLE IIGS

Système ProDOS de l'Apple IIGS est un manuel de référence de haut niveau destiné au programmeur sur Apple IIGS . Il brosse un tableau complet de ce système d'exploitation et vous fournit toutes les informations utiles pour programmer.

Après avoir détaillé l'environnement matériel et logiciel de l'Apple IIGS, l'auteur présente les nouveaux protocoles relatifs aux différents devices (Character Device Driver, Block Device Driver, SmartPort, etc.), les vecteurs particuliers de ProDOS (démarrage a chaud, à froid, CTRL-Y, Reset, etc.), les différentes structures possibles et le boot d'une disquette ProDOS, les routines bootstrap et de chargement, les répertoires et sous-répertoires, la gestion des fichiers, les commandes du langage MLI...

PRIX : 285 FF

ISBN : 2-86595-436-6

285

EDITIONS P.S.I.
DIFFUSE PAR P.C.V. DIFFUSION
BP 86 - 77402 LAGNY-S/MARNE CEDEX – FRANCE

SYSTÈME ProDOS DE L'APPLE IIGS

MATÉRIELS

MATÉRIELS

SYSTÈME ProDOS DE L'APPLE IIGS

MARCEL COTTINI

PROGRAMMATION

- ProDOS 16
- Device driver, bootstrap

Connaissez-vous la collection Apple II chez P.S.I. ?

— 102 programmes pour Apple II — Jacques Deconchat
— Apple en famille — Jean-François Sehan
— Appleworks au travail — Alain Gargadennec et Jean-Michel Jégo
— Assembleur de l'Apple II — Nicole Bréaud-Pouliquen et Daniel-Jean David
— Clefs pour Apple IIc et IIe avec 65C02 — Nicole Bréaud-Pouliquen
— Exploitation d'enquêtes sur Apple et IBM-PC — Jean-François Grimmer
— Les ressources de l'Apple IIc et IIe — Nicole Bréaud-Pouliquen
— Programmation système de l'Apple II — Marcel Cottini
— Super jeux Apple — Jean-François Sehan

Apple IIgs

— Clefs pour Apple IIgs — Nicole Bréaud-Pouliquen
— La boîte à outils de l'Apple IIgs — Jean-Pierre Curcio

A paraître :

— L'assembleur de l'Apple IIgs — Nicole Bréaud-Pouliquen

Pour connaître les dernières nouveautés P.S.I.,
ou nous soumettre un problème technique,
nous mettons à votre disposition un service Minitel

Service Minitel
Sur le 3615 tapez OI puis PSI

© Editions P.S.I., une société du Groupe CEP
5 place du Colonel Fabien, 75491 Paris Cedex 10
1987
ISBN : 2-86595-436-6

SYSTÈME ProDOS DE L'APPLE IIGS

MARCEL COTTINI

ÉDITIONS P.S.I.

SOMMAIRE

REMERCIEMENTS

Je tiens à remercier tout particulièrement un certain nombre de personnes et de sociétés qui, par leur aide matérielle et leur dévouement spontané, ont rendu possible la rédaction de cet ouvrage.

La boutique XEROX STORE, représentée en la personne de Monsieur Flomont, a mis gracieusement à ma disposition tout un ensemble Apple *II*GS couleur, équipé d'une mémoire auxiliaire de 1 Mo et d'une imprimante ImageWriter.

Boutique XEROX STORE
82-84 en Fournirue
57000 Metz

La société GERB ELECTRONIQUE, représentée en la personne de Monsieur F. Crochet, qui a mis gracieusement à ma disposition un disque dur Mega Core d'une capacité de sauvegarde de 20 Mo.

Société GERB ELECTRONIQUE
Zone industrielle de Brais
44600 Saint-Nazaire

La société MEMSOFT, représentée en la personne de Monsieur Mouchon, qui a mis gracieusement à ma disposition les produits MemSoft disponibles sur Apple *II*GS .

Société MEMSOFT S.A.
1050 Route de la Mer
06410 Biot

Madeleine Hodé, auteur de Gribouille, traitement de texte dont on ne vantera jamais assez les nombreuses qualités, m'a spontanément offert son produit pour me permettre la rédaction du manuscrit. Il n'est plus besoin de présenter Gribouille, le traitement de texte convivial d'un rapport qualité/prix sans concurrence.

Société GRIBOUILLE S.A.R.L.
16 rue des Poules
67000 Strasbourg

La société THOT INFORMATIQUE, représentée en la personne de Monsieur J-Y Desmarres, qui a mis gracieusement à ma disposition une carte Speedisk d'une capacité mémoire de 1 Mo.

Société THOT INFORMATIQUE
B.P. 412 Cedex
49004 Angers

PRÉFACE

Dans le flot sans cesse croissant d'un marché de la micro-informatique, ô combien bouleversé, Apple Computer garde le cap sur la gamme des Apple *II*.

Dernier-né du standard Apple *II*, la couvée GS (Graphics & Sound) fait peau neuve. L'ensemble de la configuration se présente sous un aspect très agréable, intégrant un microprocesseur 16 bits, ainsi qu'un module nommé Mega *II*, 'chips' très particulier émulant un Apple *IIe*.

La nouvelle carrosserie nous dévoile un microprocesseur 16 bits de la famille des 65C.., le 65C816 de Western Digital, un C-MOS émulable avec le 6502 par adressage du registre d'état (bit e). C'est un processeur hybride, capable de travailler sur 8 ou 16 bits en passant d'un mode à un autre. Cette commutation ou émulation avec le 6502 se fait en positionnant la retenue et en exécutant une instruction machine (XCE). Dans cette version, 91 instructions de base sont disponibles.

Pour étendre le champ d'action, 24 modes d'adressage permettent au programmeur d'effectuer un choix parmi 255 codes machine. Le microprocesseur 65C816 est soutenu par une vitesse d'horloge déterminée en fonction du mode d'exécution en cours. Une fréquence de 2,8 Mhz pour le traitement de tout ce qui fait référence à la ROM résidente, une fréquence de 2,5 Mhz pour des applications exécutées en RAM et en mode native (65C816), et une fréquence d'horloge limitée à 1 Mhz pour des applications tournant en mode émulation (6502 ou 65C02). La capacité mémoire, dans la version de base, est fixée à 128 Ko de ROM et à 256 Ko de RAM; l'extension à un Mo ou plus, pourra être déterminée suivant les critères de programmation de chacun.

Système ProDOS16 de l'Apple *II*GS est un ouvrage destiné à donner au lecteur tous les moyens élémentaires pour une meilleure compréhension de certains mécanismes internes au système. Il est principalement orienté vers le système d'exploitation ProDOS16, système destiné à remplacer totalement le DOS 3.3. L'objectif premier d'un tel ouvrage est de familiariser le programmeur pratiquant l'Apple *II*GS et sous environnement ProDOS.

ProDOS8, système d'exploitation de la génération des processeurs 8 bits, est totalement compatible avec ProDOS 1.1.1. Il est destiné particulièrement aux applications programmées pour fonctionner avec les instructions des microprocesseurs 6502 et 65C02 (mnémoniques). Chargé en mémoire à partir de l'Apple *II*GS , ProDOS8 permet d'exécuter certains programmes de la bibliothèque Apple *II* tout en n'assurant qu'une compatibilité relative.

ProDOS16, système d'exploitation de la nouvelle génération, est destiné essentiellement aux applications de l'Apple *II*GS équipé du microprocesseur 65C816. Il traite en totalité les instructions du microprocesseur 65C816, qui est un vrai 16 bits, avec une possibilité d'adressage étendue à 24 bits (16 Mo). Ce système n'est pas destiné a priori à exécuter des programmes développés pour les processeurs 6502 et 65C02. Aucune compatibilité n'est possible entre ProDOS8 et ProDOS16.

Les deux versions de ProDOS (ProDOS8 et ProDOS16) diffèrent dans bien des domaines tout en gardant une certaine similitude dans l'exécution de leurs instructions, pendant le traitement d'une application en mode émulation. Cet ouvrage, parfaitement adapté au lecteur pratiquant l'Apple *II*GS , est le complément indispensable de la documentation en langue anglaise diffusée par Apple. Il permet par ailleurs d'acquérir une meilleure compréhension de la structure très particulière de ProDOS16.

MISE AU POINT ET NOTIONS FONDAMENTALES

Système ProDOS16 est un ouvrage destiné aussi bien à l'utilisateur averti, pratiquant l'Apple *II* en général et le GS en particulier, qu'au novice désireux de persévérer dans le domaine de la programmation. Complément indispensable (et en français) de l'imposante documentation de l'Apple Computer, ce livre est particulièrement recommandé à tous ceux qui désirent sortir des sentiers battus de la micro-informatique grand public. Par la suite, tous les efforts ont été principalement portés sur le développement du système d'exploitation ProDOS16 (fichiers PRODOS, P16 et LOADER), programmes système de base du GS.

Pour une meilleure compréhension du sujet, l'auteur a estimé indispensable de familiariser le lecteur avec certaines notions de base spécifiques à l'environnement du processeur 65C816 et du système ProDOS16. Certains termes et expressions techniques, ainsi qu'une grande partie du matériel, sont nouveaux, et liés directement ou indirectement à la technologie de pointe. Le but essentiel de ce chapitre est de doter l'utilisateur des moyens de base indispensables afin d'écarter toute équivoque par la suite. Que le lecteur se rassure, ces détails, souvent indispensables pour une totale assimilation, sont d'un niveau abordable par tout amateur de la programmation.

APPLE *II*GS ET NOTIONS NOUVELLES

Notion de microprocesseur

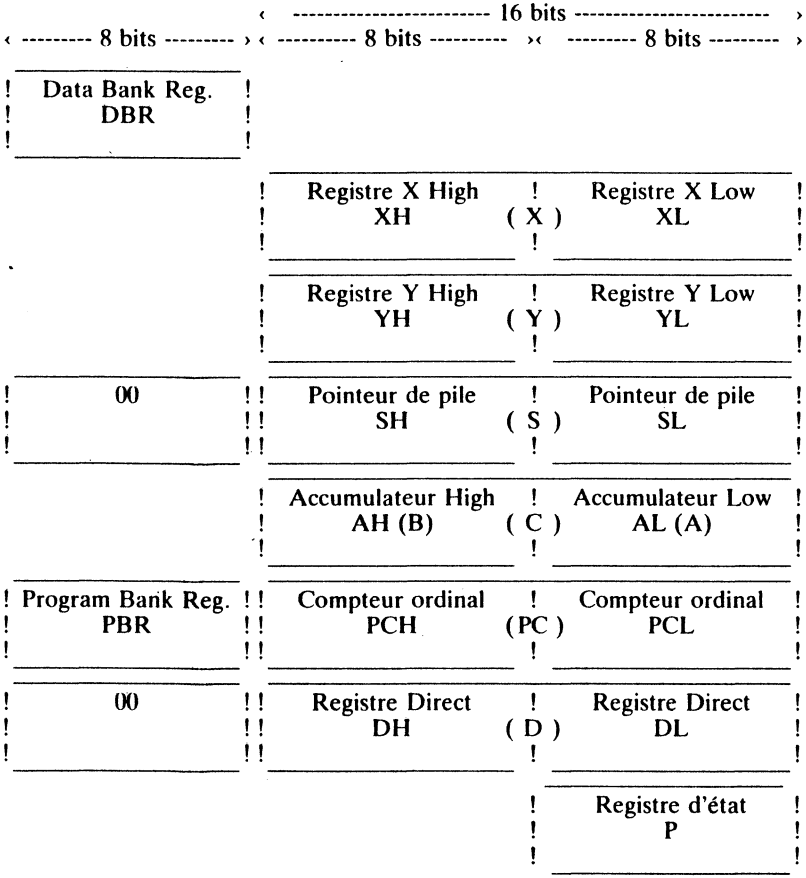
Le microprocesseur équipant le GS est un 16 bits de la famille des 65C... Il peut émuler le 65C02 équipant les versions Apple *II*c et *II*e+ (8 bits de données). Le bus d'adresses est de 24 bits, ce qui autorise un adressage de 16 Mo de mémoire globale (RAM et ROM).

En mode natif, deux cycles d'horloge du microprocesseur sont disponibles : si l'execution s'effectue au niveau de la ROM résidente, le 65C816 est alors cadencé à 2,8 Mhz; dans le cas d'un traitement en RAM, le cycle d'horloge est ramené à 2,5 Mhz. Cette différence de fréquence est imputable au processus de régénération de la RAM.

En mode émulation, le microprocesseur est cadencé à un cycle horloge de 1 Mhz, ce qui lui confère une similitude d'exécution avec le 65C02.

La fréquence d'horloge soutenant le microprocesseur est par ailleurs programmable au niveau du hardware du GS. Pour ce faire, il suffit de solliciter le tableau de bord (touches pomme-ouverte/Control/ESC), puis d'effectuer son choix dans le menu **System Speed** par l'intermédiaire du **Control Panel**. Deux options sont possibles : Fast et Normal. Dans le premier cas (Fast), le cycle d'horloge sera de 2,5 Mhz (ou 2,8 Mhz) et de 1 Mhz si votre choix s'est porté sur Normal.

Registres du microprocesseur 65C816



Détail du registre d'état du microprocesseur (P) – Status Register (P)

1 Toujours à 1 si E = 1.
B Break : remise à 0 de la pile après interruption si E = 1.
E Mode émulation : 0 = native
1 = émulation

1 B E
n v m x d i z c
7 6 5 4 3 2 1 0 (bits du registre P)

n Negative
Bit du signe = 1 si résultat est négatif

v Overflow
Débordement = 1 si le résultat est trop grand.

m Memory/Accumulator Select
Sélection de la mémoire ou de A = 1 si commuté en mode 8 bits
= 0 si commuté en mode 16 bits.

x Index Reg. Select
Sélection du registre d'index = 1 si commuté en 8 bits
= 0 si commuté en 16 bits.

d Decimal Mode
Mode décimal = 1 si commuté en mode décimal.
= 0 si commuté en mode binaire.

i IRQ Disable
Inhibition des interruptions = 1 si interruption inhibée.

z Zéro
Résultat nul = 1 si le résultat est nul.

c Carry
Retenue = 1 si le résultat est celui attendu.

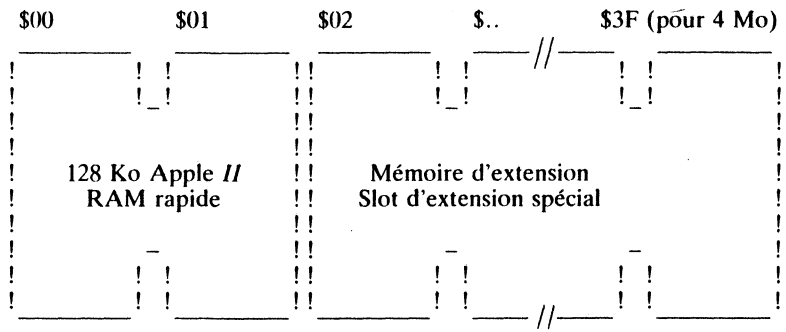
Notion de banc mémoire

Le microprocesseur 65C816, cœur de l'Apple *II*GS peut, par construction, gérer une mémoire globale de 16 Mo (RAM + ROM) suivant un principe établi par avance. Cet espace mémoire adressable est vu par le processeur comme des bancs mémoire, au nombre de 256, dont la taille pour chacun d'entre eux, est de 64 Ko. Par construction, le GS possède déjà 256 Ko de RAM (bancs $00-$01) et 128 Ko de ROM (bancs $FE-$FF). En mode émulation, cette structure correspond à celle d'un Apple *II*e+ possédant sur sa carte mère l'AppleSoft et le Moniteur résidents (ROM), une mémoire principale de 64 Ko (banc $00) et une mémoire auxiliaire de 64 Ko également (banc $01).

La mémoire du GS regroupe trois grandes zones distinctes : la RAM programme, la RAM système et la ROM résidente.

RAM PROGRAMME

La RAM programme, dotée de 'chips' à mémoire vive rapide, possède une taille de 128 Ko, et occupe les bancs $00 et $01 sur la carte mère. Cette mémoire est particulièrement destinée à des applications constituées de programmes usuels, ainsi qu'aux données (datas) et variables qui s'y rapportent. Cette zone mémoire assure une compatibilité relative avec les anciennes versions des Apple *II*, avec les appellations de **Main Memory** (64 Ko de mémoire principale) et **Aux Memory** (64 Ko de mémoire auxiliaire avec une carte d'extension 80 colonnes étendue par exemple).

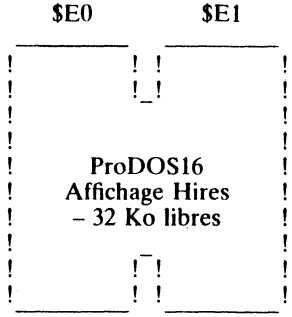


- les bancs $00 et $01, implantés par hardware, correspondent aux bancs Main et Aux de la structure Apple *II*.

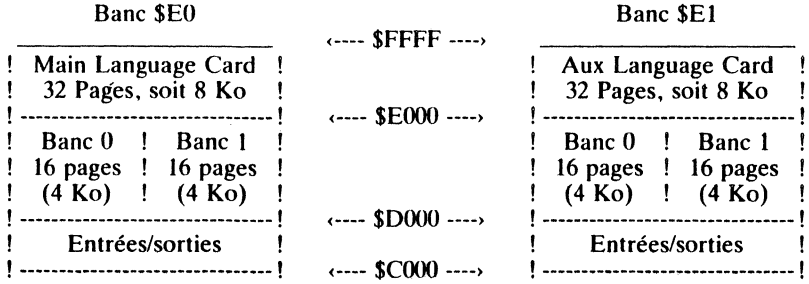
- les bancs $02 à $11, mémoire rapide et accessible par la carte d'extension Apple, permet d'augmenter la mémoire de base en portant celle-ci à 1256 Ko (1 Mo 256 Ko). Les bancs $02-$3F, extensibles à $7F, représentent l'espace mémoire total de la RAM d'extension (8 Mo au total).

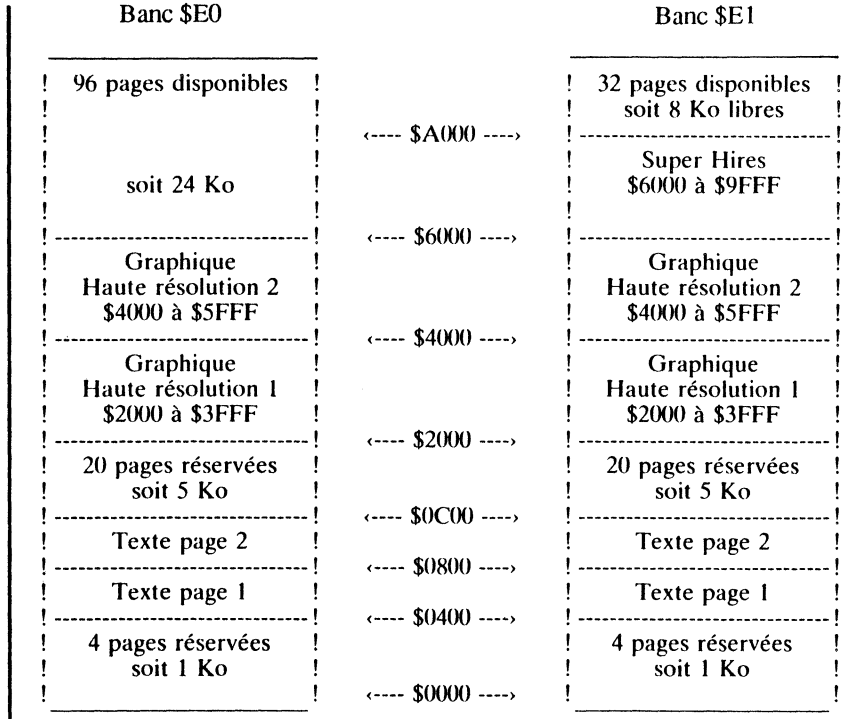
RAM SYSTEME

La RAM système, équipée de 'chips' à mémoire vive lente, possède une taille de 128 Ko, et occupe les bancs $EO et $E1 sur la carte mère. Elle supporte entre autres les pages graphiques, les zones d'entrées/sorties (Input/Output), de la place pour le système d'exploitation ProDOS16 et les RAM Tools s'y rapportant.



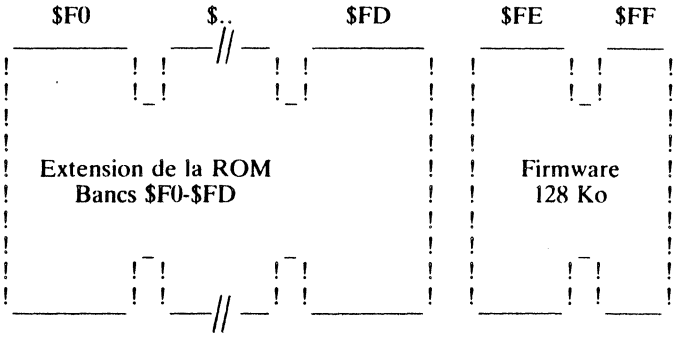
$E0 $E1/0400-07FF	Texte 40 & 80 colonnes – RAM Main et Aux
$E0/2000-3FFF	Graphique HGR page 1
$E0/4000-5FFF	Graphique HGR page 2
$E0/6000-BFFF	24 Ko disponibles
$E0/C000-CFFF	I/O toujours actives
$E1/2000-3FFF	Graphique HGR page 1
$E1/4000-5FFF	Graphique HGR page 2
$E1/6000-9FFF	Graphique Super Hires
$E1/A000-BFFF	8 Ko disponibles





ROM RESIDENTE

La ROM, ou mémoire morte, est dotée d'origine de 128 Ko. Les adresses des bancs $F0 à $FF, soit au total 1 Mo, sont réservées à de la mémoire morte. Les adresses des bancs $FE à $FF, soit un total de 128 Ko, sont disponibles par construction à de la ROM résidente, appelée communément firmware. Le choix de la zone d'implantation en mémoire a été dicté par un critère de compatibilité envers les Apple *II* de la génération des microprocesseurs 6502 et 65C02.
Les bancs $F0 à $FD sont disponibles pour des cartes d'extension, des cartes d'interface de ROM (ROM Cards), ou encore pour de la RAM sauvegardée.



Les 128 Ko de mémoire morte résidente (Firmware) se composent suivant le schéma suivant:

• Le Moniteur étendu ($FF/F800-FFFF) et l'interpréteur du Basic Apple-Soft ($FF/D000-F7FF).

• Une gestion de 'Old Desk Accessorie' permettant, même en mode émulation 65C02, l'accès à des outils de bureau.

• Le Moniteur intégré, composé d'un mini-assembleur/désassembleur, étend les capacités de l'ancien Moniteur (*IIe* par exemple). Il autorise entre autres:

- de manipuler l'espace mémoire;
- de modifier des octets dans les modes ASCII ou Hexa;
- de déplacer, remplir ou comparer des zones de la mémoire spécifiée;
- de rechercher des séquences particulières, définies au préalable, dans l'ensemble des 16 Mo disponibles (RAM et ROM);
- de visualiser et de modifier l'environnement d'exécution du programme (registre & flags);
- d'effectuer des opérations mathématiques élémentaires;
- de lancer l'exécution de programmes, etc

• La moitié des outils de la ToolBox, à savoir: Tool Locator, Memory Manager, Miscellaneous Tools, QuickDraw II, Desk Manager, Event Manager, Scheduler, Sound Tools, Apple Desktop Bus, Sane, Integer Math, Text Tools. L'autre partie des outils se charge en mémoire vive à partir d'un support disque par exemple, et sous la responsabilité exclusive de l'application qui les utilise.

Remarque

Les bancs $80 à $DF et $E2 à $EF libres de toute activité pour l'instant, sont néanmoins réservés à des applications futures.

Notion de circuits nouveaux

Un certain nombre de circuits nouveaux sont apparus sur le GS, ce qui explique en grande partie le degré d'intégration de l'unité centrale dans un volume aussi restreint.

Mega*II* — c'est un chips réduit à sa plus simple structure, représentant à lui seul toutes les fonctions d'un Apple *II*. Il est essentiellement destiné à assurer la compatibilité avec les Apple *IIe*$^+$ et *IIc* équipés du processeur 65C02.

FPI — c'est un contrôleur de mémoire et de vitesse d'horloge (FPI = Fast Processor Interface). Ce circuit assure le lien entre Mega *II*, soutenu par un circuit d'horloge de 1 Mhz d'une part, et le microprocesseur 65C816 à une fréquence de 2,8 Mhz d'autre part (2,5 Mhz pour la RAM). C'est encore lui qui gère et organise la mémoire d'extension limitée à 8 Mo, et gère le mode émulation du 65C02. Ce procédé est réalisé suivant le principe du Shadowing, qui consiste à porter l'image du contenu de la mémoire vive rapide, bancs $00-$01, vers certaines zones de la mémoire vive lente, bancs $E0-$E1. Il s'agit essentiellement des zones tampons d'affichage des modes texte et graphique.

IWM — interface de lecteurs de disques (IWM = Integrated Wozniak Machine). C'est un circuit issu de la technologie commune du Macintosh et de l'Apple *IIc*, il opère des contrôles complexes en ce qui concerne les tâches d'entrées/sorties (I/O) des accès au disque par des **Control** et **Data** bytes. C'est encore lui qui gère le Desktop Bus sur lequel se connectent sans distinction aucune, deux lecteurs 800 Ko Apple et l'Unidisk 140 Ko

Slot Maker — commutation interne/externe des entrées/sorties. Il gère entre autres le slot Data Bus, les deux autres étant le Desktop Bus (Mega *II*) et MDBUS.

Notion de slots et de ports

Le slot ou connecteur, est une structure hardware familière à ceux qui pratiquent l'Apple *II* depuis ses débuts. La particularité du slot consiste à permettre une extension des possibilités de l'unité centrale, et présente des similitudes, mais aussi des différences fondamentales avec la gamme ascendante.

7 slots ou connecteurs d'extension du bus 8 bits.

Une sortie haut-parleur, avec un son monocanal synthétisé (15 voies), dont les effets sont contrôlés par le processeur Ensoniq.

Deux ports séries aux normes RS422 – AppleTalk. Le maître d'œuvre du port intégré est un contrôleur Zilog du type SCC 8530 (2 voies, Serial Communication Chip). Le flux des informations pouvant être véhiculé sur un des bus série est fixé par les limites 50 et 19200 bauds. Sous AppleTalk, le transfert des données pourra atteindre 230400 bauds. Une imprimante compatible, reliée au port série, pourra éditer sur papier en tâche de fond (routine Firmware).

Deux sorties jeux: un connecteur interne, du type femelle 14 broches, permet le raccordement de manettes de jeux; un connecteur externe, du type femelle DB 9 broches, autorise le raccordement d'un joystick. Il est tout à fait possible de raccorder toute entrée/sortie véhiculant des signaux compatibles TTL.

Un connecteur de Block Device externe ou SmartPort, permet le raccordement des drives Unidisk, DuoDisk et AppleDisk. Tous ces périphériques d'entrées/sorties (I/O) sont chaînables, c'est-à-dire qu'ils permettent le raccordement de l'un à la suite de l'autre.

Sortie vidéo composite, permettant de relier un écran vidéo couleur aux normes RGB (rouge/vert/bleu).

Apple Desktop Bus. L'ABD est un contrôleur normalisé, autorisant le rac-

cordement de plusieurs interfaces du type Desktop (clavier, souris, table traçante, etc.).

Une horloge en temps réel, soutenue par une pile soudée sur la carte mère, permet d'avoir la date et l'heure courante, ainsi que la sauvegarde permanente d'une RAM de 256 octets (RAM non volatile).

Notion d'un système d'exploitation

Depuis sa venue sur le marché de l'Apple *IIc*, le système d'exploitation ProDOS est devenu le système de la gamme des Apple *II*. Avec une progression timide au départ, ProDOS gagne sans cesse du terrain sur ses concurrents potentiels. Dans la bousculade des versions dites nouvelles, bon nombre d'utilisateurs ne s'y retrouvent plus, effectuant très souvent la confusion la plus totale entre une version et une autre. Il se trouve que même le Basic-System, interface entre l'AppleSoft et le programme Basic en cours d'exécution, est très souvent pris pour le système d'exploitation. A noter que la page globale ou ProDOS Global Page, est l'interface entre le programme d'application et ProDOS MLI implanté en mémoire.

Tableau comparatif ProDOS8 et ProDOS16

Paramètre ou état	ProDOS8	ProDOS16
Mode d'exécution	Emulation 6502	Mode natif 65C816
Mémoire minimale	64 Ko	256 Ko
Mémoire maximale	128 Ko	16 Mo adressables
Organisation mémoire	Bit map Global Page	Memory Manager (ToolBox)
RAMDisk	Actif	Non actif
Pointeur adresse	2 octets	4 octets (3 utilisés)
Appel des commandes	JSR banc $00	JSL banc $E1
Suffixe fichier système	.SYS	.SYS16
Type de fichier système	$FF	$B3
Taille de la pile	256 bytes	Toute taille jusqu'à $B7FF
Adressage direct	$00/0000-00FF	N'importe où dans le banc $00
Adressage de la pile	$00/0100-01FF	$00/0800-BFFF
Taille accumulateur	8 bits (e = 1)	16 bits (m = 0)
Taille registre d'index	8 bits (e = 1)	16 bits (x = 0)
Mode CPU	Emulation (e = 1)	Native (e = 0)

SYSTEME D'EXPLOITATION ProDOS8

mode émulation

ProDOS8 est le système d'exploitation développé pour des applications spécifiques au matériel de la gamme des Apple *II*. Il est totalement compatible avec la version ProDOS 1.1.1. Matériel requis : tout Apple *II*+, *IIc*, *IIe* et *IIe*+ doté au minimum d'une mémoire de 64 Ko. Il exploite plus particulièrement les capacités de programmation et d'adressage des microprocesseurs 6502 et 65C02, ainsi que tous les programmes écrits en langage machine, abstraction faite de certaines références au DOS 3.3 (CALL, PEEK et POKE). Il supporte toutes les interruptions et périphériques normalement compatibles avec l'environnement Apple *II*.

SYSTEME D'EXPLOITATION PRODOS16

(mode natif)

ProDOS16 est le système d'exploitation développé pour exploiter toutes les capacités de programmation de l'Apple *IIGS* , permettant de tirer le meilleur parti des instructions spécifiques au microprocesseur 65C816. Il n'est pas compatible avec la gamme des Apple *II* antérieurs au GS. Un certain nombre de commandes MLI sont nouvelles, tandis que d'autres ont été simplement abandonnées. La version ProDOS16 actuellement disponible sur le marché n'est pas définitive, Apple Computer se réservant le droit d'y apporter toutes modifications utiles. Les versions antérieures à la version 2.0, dites versions bêta, ne sont que temporaires mais possèdent leurs points d'entrées aux appels MLI identiques. Ces vecteurs sont déterminés par convention, le développeur étant tenu de respecter les protocoles fixés par Apple Computer.

PROGRAMMES SYSTEME (FICHIERS PRODOS)

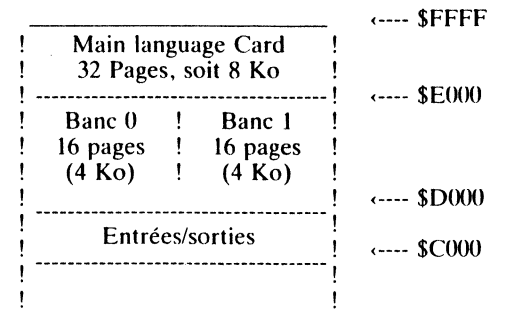
Un système d'exploitation est un ensemble de petits programme qui, associés les uns aux autres, forment un tout cohérent, capable de faire fonctionner les divers périphériques qui composent un ensemble informatique. C'est une suite d'instructions faisant partie des programmes système (Operating System), permettant entre autres de gérer un ou plusieurs périphériques de sauvegarde de masse à l'aide de commandes évoluées. Lors de ses différentes manipulations, l'opérateur n'a pas besoin de s'occuper du déplacement de la tête de lecture, ni de se soucier de la gestion et de l'écriture des divers fichiers sur le support. Ce rôle est confié au système d'exploitation. Ces définitions se rapportent plus particulièrement aux systèmes de la nouvelle génération, à savoir ProDOS8 et ProDOS16.

Depuis 1984, la société Apple Computer a bouleversé les habitudes de nombreux programmeurs, en lançant sur le marché un système plus performant qui, au départ, était destiné à un disque dur du type Profile. Très vite on s'est rendu compte que ce système offrait de nombreuses possibilités, ce qui permit au concepteur de l'adapter à l'environnement Apple. C'est ainsi qu'est né ProDOS, avec ses avantages très nombreux, mais aussi certaines restrictions.

Comme tout programme, le système d'exploitation occupe une place qui dépend de la richesse de ses possibilités. En règle générale, le concepteur essaie de réaliser un compromis entre les possibilités offertes par le matériel et le logiciel (hardware et software). Pour ProDOS, la mémoire vive se trouve amputée d'une zone mémoire directement liée à l'application en cours. ProDOS8 n'occupe pas les mêmes vecteurs de la mémoire que ProDOS16. De même, tout programme, exécuté en mode émulation et sous environnement ProDOS standard, simule un Apple *IIe* : les 12 Ko en haut de la mé-

moire sont attribués au fichier ProDOS par commutation alternée des bancs 0 et 1 des 16 Ko de la carte langage (voir Notion de commutateurs logiques). En présence d'un Apple *IIe*, cet espace est normalement occupé par la ROM AppleSoft et le Moniteur étendu. En réalité, le fichier ProDOS occupe 16 Ko, dont 4 Ko sont obtenus par la commutation de la Bank Switched Memory – bancs 0 et 1 (voir figure ci-dessous).

Organisation de la carte langage



Remarque

Les adresses $D000-$FFFF correspondent aux anciennes versions des Apple *II*, zone mémoire souvent appelée espace réservé à la carte langage. A partir des Apple *IIe* et *IIc*, cet espace mémoire se trouve affecté à la ROM AppleSoft ($D000-$F7FF) et au Moniteur étendu ($F800-$FFFF). Au niveau du GS, le concepteur a gardé cette notion de carte langage, assurant de ce fait une compatibilité relative entre les différents types de machines.

STRUCTURE DU SYSTEME PRODOS EN GENERAL

Le système d'exploitation est utilisé essentiellement pour la gestion de l'environnement d'une unité centrale en général et des fichiers de données en particulier.

Le DOS, Disk Operating System, comme bon nombre de produits firmware, subit de perpétuelles mutations dues essentiellement à l'évolution de la technologie, qu'elle soit du domaine matériel (hard) ou logiciel (soft). C'est ainsi qu'il est bien loin le temps du premier DOS 3 (juin 1978). DOS 3.3, dernière version en date, a été commercialisé au mois d'août 1980 et, comme ses prédécesseurs, subit la dure loi de l'évolution.

La technologie aidant, une nouvelle génération de DOS est apparue: ProDOS8 (Professional Disk Operating System). C'est plus qu'un système d'exploitation. Doté d'une structure professionnelle, le disque système se compose d'une part, d'un fichier ProDOS (Kernel), et, d'autre part, d'un fichier BASIC.SYSTEM (interface de la ROM AppleSoft).

Avec la venue sur le marché du GS, tout a été remis en question: hard et soft ont été totalement réactualisés par le concepteur. Un nouveau système d'exploitation a été mis au point, permettant d'exploiter au maximum les capacités du microprocesseur 65C816.

Dès à présent, deux configurations du système sont possibles, selon que l'on veuille tirer profit du mode natif (65C816) ou du mode émulation (65C02) de la machine. Dans le premier cas, une configuration ProDOS16 est requise, tandis qu'en mode émulation, uniquement ProDOS8 sera nécessaire. Des différences fondamentales existent entre ces deux structures de DOS, chacune d'elles étant mise en place par un fichier de chargement qui se nomme par convention ProDOS. Le fichier ProDOS16 est sauvegardé sous le nom de P16 tandis que le fichier ProDOS8 se retrouve sous le nom de P8.

ProDOS16 est le système d'exploitation de la troisième génération. Doté d'une conception nouvelle, il effectue une entrée en la matière comme gestionnaire simple tâche multi-utilisateur (AppleTalk). Les données sont traitées selon une structure hiérarchique et arborescente.

NOUVELLES STRUCTURES DU SYSTEME PRODOS

- bootStrap orienté vers ProDOS8 ou ProDOS16 suivant la présence de fichiers .SYS ou .SYS16 en premier lieu, dans l'ordre de la table des matières du disque;

- en mode natif et sous ProDOS16, pas de limitation de fichiers ouverts ni de volumes en lignes; fonctionne correctement entre 256 Ko et 4096 Mo;

- la taille d'un fichier est limitée à 16 Mo, celle d'un volume est étendue à 32 Mo;

- la commande QUIT exécute /VOLUME/SYSTEM/START (Finder) par l'intermédiaire d'un QUIT QUEUE qui stocke en attente les exécutions prochaines;

- jusqu'à 16 interruptions sont rendues possibles à travers un dispatcher qui gère les interruptions en cours (dont AppleTalk fait partie).

POURQUOI UN SYSTEME PRODOS16 ?

Pour tirer profit au maximum des possibilités d'adressage et d'exécution du microprocesseur 65C816.

Pour pouvoir être sollicité par un programme d'application et cela à partir de n'importe quel banc mémoire.

Pour permettre une compatibilité relative avec certaines applications tournant sur un Apple *II* équipé du processeur 65C02. A cet effet une mise au point s'impose, car le terme compatibilité, souvent employé à tort, désigne en fait la bibliothèque des programmes. Il est très clair que cette compatibilité n'est que relative, permettant entre autres le traitement d'un certain nombre de logiciels existants sur le marché, destinés dans un premier temps à la gamme des Apple *II* antérieurs au GS. D'ailleurs, quel serait l'intérêt d'acquérir un GS pour, en fin de compte, n'exécuter que des applications en mode émulation? Vous conviendrez avec moi qu'un tel investissement ne serait nullement justifié, point à la ligne.

Pour augmenter les capacités et la puissance du système ProDOS.

Pour doter le système de certaines améliorations technologiques tant du point de vue hard que soft, en particulier les dispositions ProDOS QUIT,

SYSTEM LOADER, MEMORY MANAGER qui permettent une homogénéisation des logiciels.

En présence du GS et de ProDOS16 booté en mémoire, un environnement très particulier s'offre au programmeur.

Tout un ensemble d'outils est alors disponible, en particulier Programmer's Workshop qui se présente comme un produit d'aide au développement d'applications. L'APW (Apple Programmer's Workshop) comporte un assembleur basé sur le Macro Assembleur ORCA/M, un C de Megamax, un Pascal transcrit par Tom Léonard et un Debugger qui consiste en une version améliorée de BugByter pour Apple *II*.

Notion de fichier système

Avec la pratique de l'Apple *II*GS , de nouveaux fichiers système ont fait leur apparition, en particulier les fichiers ProDOS, LOADER, P8 et P16. Dans les structures initiales, Apple *II*e, *II*e+ et *II*c, uniquement les fichiers PRODOS et BASIC.SYSTEM sont requis sur un disque système conventionnel.

SYSTEME D'EXPLOITATION CONVENTIONNEL

Dans la version mode émulation, compatible avec l'environnement des Apple *II* équipés d'un processeur 65C02, la dénomination ProDOS désigne un programme système et un programme interface. Dans ce cas particulier, la syntaxe ProDOS regroupe le fichier PRODOS et le fichier interface structuré selon l'application demandée, ce dernier étant alors affecté du suffixe .SYSTEM.

Fichier ProDOS

Ce type de fichier système se loge dans la carte langage, et regroupe tous les sous-programmes en langage machine (MLI). Il se trouve sauvegardé sur le disque sous la forme d'un programme du type .SYS, avec comme nom de fichier: ProDOS ou KERNEL. Ecrit en langage machine, il renferme tous les sous-programmes et routines MLI (Machine Language Interface). C'est le noyau du système (KERNEL), permettant de gérer tous les fichiers autres que ceux du type AppleSoft. Il est l'équivalent du File Manager et de RWTS du DOS 3.3; il occupe l'ensemble de la carte langage (à l'exception de la zone $D000-$D0FF du banc 1) ainsi que l'espace $BF00-$BFFF, qui sert à commuter les différentes parties hautes de la mémoire et à recevoir des appels de programmes ou de routines externes.

Fichier XXX.SYSTEM

Ce type de fichier interface, également du type .SYS, se retrouve toujours sous la forme XXX.SYSTEM et se loge au vecteur anciennement attribué au DOS 3.3. Ecrit en langage machine, il est l'interface entre un programme AppleSoft et l'interpréteur actif, permettant de passer la main à Basic pour exécuter des commandes adressées à l'interpréteur AppleSoft par exemple. Il se loge aux adresses $9600-$BEFF. Comme ProDOS, BASIC.SYSTEM est un fichier du type .SYS, uniquement bootable. BASIC.SYSTEM est sollicité en mémoire, à chaque fois qu'il s'agit de traiter des programmes du type AppleSoft. Lorsque celui-ci est chargé en mémoire centrale, HIMEM est fixé à l'adresse équivalente du DOS 3.3 ($9600).

Remarque

XXX.SYSTEM représente un format de fichier où les XXX désignent un nom de fichier pouvant être tout programme répondant à cette convention, par exemple un traitement de texte en langage machine. En règle générale, c'est BASIC.SYSTEM que vous rencontrerez pour la plupart des applications conventionnelles.

SYNTHESE DU SYSTEME D'EXPLOITATION PRODOS8

Ce système d'exploitation est totalement compatible avec la version ProDOS 1.1.1. C'est un système d'exploitation faisant référence aux applications liées à la conception du processeur 65C02, travaillant sur des mots de 8 bits de large, ou au mode émulation du GS. Un disque système, configuré avec ProDOS8, devra comporter au minimum les fichiers ProDOS (P8) et BASIC.SYSTEM, ce qui permettra de traiter des applications en langage Basic et binaires.

Caractéristiques de ProDOS8 en présence du GS

- ProDOS8 nécessite un minimum de 64 Ko de mémoire, et se trouve limité dans son adressage par la valeur $FFFF (65 535). En présence du GS, les applications lancées en mode émulation restent limitées à la taille de la mémoire vive et ne pourront en aucun cas exploiter la RAM au-delà de la valeur fixée par convention (64 Ko);

- uniquement les commandes MLI (Machine Language Interface) faisant référence aux applications 8 bits (65C02) sont reconnues;

- ProDOS8 exploite totalement les propriétés de la Global Page ($BF00-$BFFF), qui gère les paramètres, variables et pointeurs du système d'exploitation. La gestion et mise à jour de la bit map simplifiée, table d'occupation mémoire, s'effectue aux adresses $BF58-$BF6F);

- c'est encore la Global Page (banc $00) qui mémorise les variables système, la date et l'heure, le niveau du fichier ouvert (system level, adresse $BF94) et les vecteurs d'entrées/sorties (I/0 Buffer);

- ProDOS8 exploite uniquement la méthode simplifiée pour la gestion de ses périphériques et dans la définition des chemins de préfixe;

- ProDOS8 ne supporte pas les applications développées pour le processeur 65C816 utilisant des mots de 16 bits et des adresses codées sur 24 bits;

- ProDOS8 et ProDOS16 utilisent des mécanismes semblables dans la gestion, le traitement et la sauvegarde des fichiers. Ils possèdent de ce fait des points communs, permettant d'utiliser les mêmes périphériques de sauvegarde (Block Devices). C'est ainsi que chaque Volume (disque), chaque table des matières, qu'elle soit principale (Directory) ou auxiliaire (SubDirectory) pourront être traités indifféremment sous les systèmes ProDOS8 ou ProDOS16;

- ProDOS8, booté par le sous-programme de la ROM du contrôleur, se place progressivement à son adresse définitive. La routine Loader, module de chargement du système d'exploitation, est commune à tout système ProDOS. Elle débute à partir du secteur $00 de la piste $00, et occupe les

blocs $00 et $01 de tout disque système. Le Loader est mis en place au moment de la phase de formatage d'un disque;

- certaines commandes MLI, implémentées d'origine à ProDOS8, sont absentes sous ProDOS16, en particulier les commandes GET_TIME, SET_BUF, GET_BUF et ON_LINE;
- ProDOS8 ne reconnaît pas certains types de fichiers réservés exclusivement à ProDOS16 ($B0-$BF);
- à partir d'un environnement Apple *IIGS* , le boot du système peut être transmis indifféremment aux fichiers ProDOS8 (P8) ou ProDOS16 (P16), le processus étant directement lié à l'organisation du disque de démarrage;
- ProDOS8 autorise un maximum de 8 fichiers ouverts simultanément, une gestion de 14 devices en ligne à un moment donné, un protocole d'entrée/sortie (I/0) défini au périphérique du type Block Device, la déclaration des périphériques à partir d'une application, et la gestion du support de sauvegarde sous forme de Volume.

Notion de commutateur logique

Un commutateur logique permet de sélectionner ou de commuter une zone bien déterminée de l'espace mémoire. La plupart de ces commutateurs ont trois points d'entrée : l'un pour les connecter, l'autre pour les débrancher et le troisième pour lire leur état.

ProDOS 1.1.1, à l'inverse du BASIC.SYSTEM, se loge dans la carte langage et occupe ainsi 16 Ko en haut de la mémoire. Comme les adresses $D000 à $FFFF ne comptabilisent que 12 Ko, il faut bien que les 4 Ko manquants soient pris quelque part : c'est le banc 1 qui est utilisé conjointement avec le banc 0, par simple commutation alternée. L'action des commutateurs, permet entre autres, d'effectuer soit une lecture, soit une écriture, ou soit une lecture et une écriture de la carte langage.

Situation typique de la ROM

ROM : $D000-$FFFF (AppleSoft + Moniteur)
Banc 0 : $D000-$FFFF (ProDOS)
Banc 1 : $D000-$FFFF (à partir de $D100 routine Reboot)

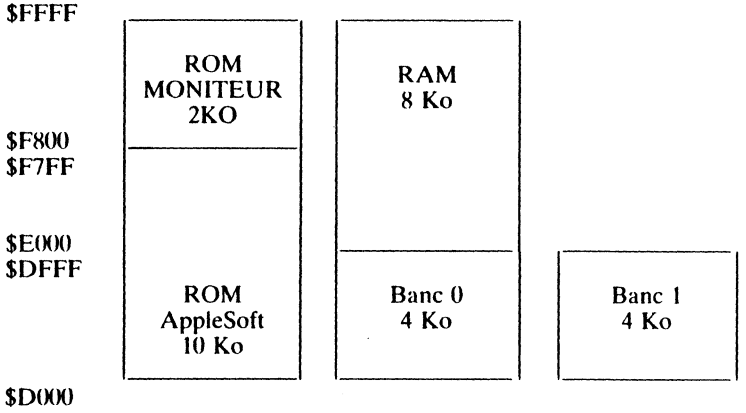
Activation des commutateurs

$D000-$DFFF Banc 0 avec 4 Ko
$D000-$DFFF Banc 1 avec 4 Ko
$E000-$FFFF Bancs 0 et 1 avec 8 Ko

Légende
R/W = Read/Write, lecture/écriture
R = Read, lecture
W = Write, écriture

Commutateurs	Banc 0	Banc 1	ROM
2 X LDA $C08B	R/W	R/W	
2 X LDA $C083			R
2 X LDA $C089	W	W	R
2 X LDA $C081			

Carte mémoire



Les 4 Ko de l'espace mémoire aux adresses $C000 à $CFFF sont réservés aux entrées/sorties : le programmeur ne pourra jamais adresser plus de 60 Ko des 64 Ko de la mémoire d'un Apple *II* standard. L'ensemble de la ROM est logé dans l'espace mémoire $D000-$FFFF et occupe ainsi 12 Ko placés en parallèle sur la RAM. L'espace mémoire de la ROM se compose de la ROM AppleSoft ($D000-$F7FF) et de la ROM Moniteur ($F800-$FFFF). Mais une zone a été plus particulièrement élaborée par le concepteur : ce sont les 4 Ko se situant aux adresses $D000-$DFFF et qui sont en fait 2 × 4 Ko placés en parallèle. Ces zones sont appelées bancs mémoire et sont au nombre de deux (bancs 0 et 1). Le microprocesseur, pièce maîtresse de l'ensemble, permet d'adresser cet espace mémoire par le jeu des commutateurs logiques. Ces commutateurs sont maîtrisés par des adresses de contrôle et permettent ainsi d'activer la ROM ou la RAM. Ces adresses sont mises en œuvre par un POKE ou un PEEK en langage Basic AppleSoft et par les instructions LDA, STA, BIT, CMP, etc., en langage machine.

ENVIRONNEMENT HARDWARE SOUS ProDOS16

PRESENTATION DE L'APPLE *II*GS

Introduction

Le roi est mort, vive le roi. Nul doute que la gamme des Apple *II* laisse une empreinte indélébile dans le monde de la micro-informatique grand public. Ce fut à l'époque de l'âge d'or, l'unité centrale grand public la plus vendue de par le monde. Le slogan lancé à cette époque reflétait bien l'esprit d'alors : un Apple dans chaque foyer.

La technologie aidant, les chercheurs actuels de Cupertino furent très vite confrontés à des réalités plus terre à terre : faire évoluer le produit existant tout en gardant une certaine compatibilité avec la bibliothèque des programmes en place. L'étude fut réalisée sous le projet Cortland et se concrétisa par la naissance de l'Apple *II*GS (Graphique & Son).

Caractéristiques essentielles de l'Apple *II*GS

Produit	Description du produit nouveau
65C816	Microprocesseur de la famille des 65C. C'est un vrai 16 bits. Doté d'un adressage sur 24 bits (bus d'adresse de 24 bits), il est compatible avec les instructions du processeur 6502.
Horloge	Ce circuit soutient les oscillations du CPU, permettant la sélection de deux fréquences (2,5 ou 2,8 Mhz et 1 Mhz). L'accès à ce choix est réalisé par l'intermédiaire du tableau de bord. Deux cycles de fréquences sont possibles : 1 Mhz, ce qui correspond au règlage normal du mode émulation (6502) et 2,8 Mhz ou 2,5 Mhz, ce qui est la vitesse rapide en mode natif (65C816).

Produit	Description du produit nouveau
Mémoire	La mémoire est dotée d'une part de 128 Ko de ROM (Read Only Memory ou mémoire morte), et, d'autre part, de 256 Ko de RAM, mémoire vive destinée à la programmation. Bancs $00-$01 : 128 Ko de mémoire rapide Bancs $E0-$E1 : 128 Ko de mémoire lente Bancs $FE-$FF : 128 Ko de mémoire morte
Memory Extend	Le bus d'adresse du microprocesseur 65C816 autorise un adressage sur 24 bits, ce qui correspond à 16 Mo d'emplacements mémoire (ROM & RAM). L'extension de la mémoire vive est rendue possible par l'adjonction d'une carte d'extension dans un emplacement spécial prévu sur la carte mère. Par construction, cette mémoire vive d'expansion est limitée à 8 Mo au total.
Clavier	Le clavier du type détachable, se raccorde au DeskTop Bus, permettant de raccorder en chaîne plusieurs périphériques du même type. Le clavier est muni de 78 touches alphanumériques diverses.
DeskTop Bus	L'Apple DeskTop Bus émule un port d'entrée/sortie (I/O), permettant de raccorder à un même bus d'I/O plusieurs périphériques d'entrées/sorties.
Vidéo RGB	L'interface vidéo est réalisée suivant les normes de la vidéo composite, c'est-à-dire en fournissant sur un bus de sortie les signaux RGB (RGB = Rouge, Vert et Bleu) et synchronisation.
Texte	L'interface 40 ou 80 colonnes, les modes texte et graphique et les différents modes couleurs (couleurs de bord et de fond) sont intégrés à la machine. Une palette de seize couleurs est possible.
Graphisme	Le graphisme a été particulièrement élaboré tout en maintenant une compatibilité avec le standard initial. Les modes Lo-Res, Hi-Res et Double Hi-Res du graphisme sont maintenus tout en ajoutant des modes nouveaux. Modes graphiques des Apple *IIe* Basse résolution: 40 × 40 pavés Haute résolution: 280 × 192 pixels Double haute résolution 1: 140 × 192 Double haute résolution 2: 560 × 192 Modes graphiques nouveaux du GS Super Hires 1: 320 × 200 pixels Super Hires 2: 640 × 200 pixels

Produit	Description du produit nouveau
Super Hi-Res	L'interface graphique nouvelle permet de développer du graphisme avec des caratéristiques proches du standard VGA (Video Gate Arrays) propre à la société IBM. Il est désormais possible d'exploiter deux modes Super Hi-res. La palette des couleurs est la suivante : – 16 couleurs par page parmi 4096 sans contrainte – 256 couleurs par page parmi 4096 avec des contraintes dues à la proximité verticale – 512 couleurs par page en modifiant les références des palettes pendant le balayage vidéo.
Interface DeskTop	Utilisation, par l'intermédiaire des outils de la ROM (ToolBox), de la page graphique Super Hi-res pour des applications exploitées à travers la souris (Icones, fenêtres et menus déroulants). La souris se relie en chaîne sur le bus d'entrée/sortie du clavier (DeskTop Bus).
Interface son	Les signaux diffusés par la sortie haut-parleur sont générés par un processeur Ensoniq Digital 15 voies. Le réglage du volume sonore se réalise électroniquement soit à l'aide du tableau de bord, soit à l'aide du logiciel faisant appel à l'application.
Tableau de bord	L'utilisateur du GS peut, par l'intermédiaire du tableau de bord (Control Panel), modifier certains paramètres stockés dans la RAM non volatile, soutenue par une pile soudée sur la carte mère. Les paramètres ajustables font référence à l'affichage vidéo, à la vitesse de travail du processeur (normale ou rapide), aux ports séries et aux slots destinés aux périphériques (devices).
Moniteur étendu	Le bus d'adresses se commute très facilement en 16 ou 24 bits, permettant un adressage soit en 16 ou 24 bits. Dans le premier cas, on dit que le GS émule un Apple *IIe* tandis que dans le second cas il travaille en mode natif. La ROM moniteur permet de désassembler des adresses sur 16 ou 24 bits. Elle reconnaît par ailleurs les instructions 8 (6502 mode) ou 16 bits (65C816 mode), améliore les temps de calcul (32 bits), structure sur 8 bits les signaux des connecteurs d'entrées/sorties (I/O) et intègre de nouvelles routines de gestion pour l'affichage vidéo (Output = sortie) et le clavier (Input = entrée).
AppleSoft	Le Basic AppleSoft intégré dans la ROM résidente, se trouve adapté pour l'affichage 80 colonnes et reconnaît désormais les caractères minuscules.

Produit	Description du produit nouveau
Montre	L'horloge en temps réel est soutenue par une pile soudée sur la carte mère et permet l'affichage en temps réel de l'heure et de la date.
Ports séries	Deux ports séries sont intégrés à la carte mère. Elaborés aux normes standard, ils permettent le raccordement d'un modem, d'une imprimante ou de la mise en service d'AppleTalk. L'utilisateur peut, par l'intermédiaire d'un connecteur d'extension, mettre en place sa propre carte série. Dans ce cas, il faudra effectuer un réajustement au niveau du tableau de bord, en signifiant à la machine la modification intervenue.
AppleTalk	Adapté au port d'entrée/sortie série intégré, il ne nécessite pas de carte interface supplémentaire. La sélection du port série se réalise au travers du tableau de bord.
Disque port	Un circuit intégré émule un port d'entrée/sortie pour des périphériques du type **Block Device**, lecteur de disque par exemple. Toute une gamme de possibilités est offerte à l'utilisateur grâce au tableau de bord (port intégré ou carte dans le connecteur, choix du BootSlot, mode Scan, etc.).
Slots d'extension	Sept slots sont disponibles pour étendre les capacités du GS et pour permettre une compatibilité relative avec l'Apple *IIe* au niveau du matériel (Hardware). Le slot auxiliaire (carte 80 colonnes étendue) n'est plus présent, le GS intégrant d'origine ce type d'interface.
I/O jeux	Un connecteur externe de 9 broches et un connecteur interne de 16 broches supportent tous les périphériques de jeux existants. Pour des applications nouvelles, il est conseillé d'utiliser le DeskTop Bus intégré.

Compatibilité du GS envers la gamme des Apple*II*

Apple *IIGS*	Equivalence	Commentaire
Codes du 6502	Tout Apple *II*	Le microprocesseur 65C816 émule un processeur 6502 et permet d'exécuter les programmes écrits pour les instructions qui leurs sont communes.
128 Ko de RAM	Apple *IIc* 128 Ko Apple *IIe*	Le GS possède une structure interne très proche de la gamme des Apple *II* – banc $00 = 64 Ko de mémoire principale (Main Memory) – banc $01 = 64 Ko de mémoire auxiliaire (Aux Memory) – similitude avec la carte langage – adresses d'entrées/sorties (I/O)
ROM AppleSoft	Tout Apple *II*	L'interpréteur Basic AppleSoft est résident en ROM. Il reconnaît maintenant les caractères minuscules et intègre les routines d'affichage 80 colonnes. Il se trouve dans le banc $FF aux adresses $D000-$F7FF.
ROM Monitor	Tout Apple *II*	Supporte les routines d'entrée/sortie (I/0) et des routines élaborées tel qu'un mini-assembleur par exemple.
Texte 40/80	Apple *IIc* 128 Ko Apple *IIe* + carte 80 colonnes	En mode émulation, affichage texte en noir et blanc. En mode natif, l'affichage texte est réalisé en couleur avec un écran RGB.
Lo-Res	Tout Apple *II*	Graphisme en mode basse résolution avec 40 x 40 pavés en couleur, 16 couleurs au choix.
Hi-Res	Tout Apple *II*	Graphisme haute résolution avec un affichage de 280 × 192 pixels, 6 couleurs au choix.
Double Hi-Res	Apple *IIc* 128 Ko Apple *IIe*	Graphisme double haute résolution avec un affichage de 560 × 192 pixels, 16 couleurs au choix.
Ports sériels	Apple *IIc*	Emulation de deux ports sériels aux normes RS-232, compatibles pour modem, imprimante ou autres cartes série.
Ports Disque	Apple *IIc*	Emule un port interface lecteur de disques à travers un circuit intégré nommé IWM. Ce type de port dit intelligent (SmartPort) reconnaît indifféremment les drives aux formats 5,25 (140 Ko) et 3,5 pouces (800 Ko).

Apple *IIGS*	Equivalence	Commentaire
Slots	Apple *II*, *II+* et *IIe*	Sept slots d'extension font partie intégrante de la carte mère, et permettent ainsi de mettre en place une carte d'entrée/sortie compatible Apple *II*.
Game I/O	Apple *IIc* et *IIe*	Deux connecteurs permettent le raccordement des manettes de jeux (9 broches externes et 16 broches internes).

Similitudes avec le Macintosh

En comparant avec le Macintosh, produit également développé par la société Apple computer, certaines similitudes au niveau du hardware sont frappantes. C'est ainsi que la célèbre ToolBox du Mac équipe d'origine le GS (128 Ko dans les bancs $FE-$FF), permettant d'exploiter toutes les qualités du DeskTop Bus Interface. Cette interface gère à travers le clavier et la souris, ou tout périphérique compatible, l'affichage des menus sur la page Super Hi-res en utilisant la pratique des icônes, des fenêtres et des menus déroulants.

- similitudes;
- utilisation des outils de ToolBox résidents en ROM;
- interface utilisateur Apple;
- gestion des menus déroulants (fenêtres, icônes, etc.);
- clavier détachable et souris en option chaînés sur le clavier;
- pavé numérique intégré d'origine;
- interface Zilog SCC pour les ports série.

STRUCTURE HARDWARE DE L'APPLE *IIGS*

Nouveaux circuits

Le GS bénéficie largement des nouveautés de la technologie de pointe, à savoir des circuits électroniques à très haute inégration. Le tableau suivant brosse le détail des différents circuits dits de type VLSI. Le microprocesseur, du type 65C816, ne répond pas à cette haute intégration.

Circuit	Description et fonction
Mega *II*	Circuit à très haute intégration, émule le fonctionnement de l'Apple *II*. Il intègre dans un faible encombrement tout le nécessaire de l'Apple *II*, interfaces d'entrées/sorties comprises.
SlotMaker	Gère et organise l'environnement nécessaire à l'adressage et au contrôle des signaux utilisés avec des applications faisant appel aux slots d'extension.
FPI	Fast Processor Interface. Gère et organise l'environnement nécessaire à l'adressage et aux cycles horloge au moment d'utiliser la mémoire rapide. Gère également la synchronisation des signaux du processeur et du circuit Mega *II*.
VGC	Video Graphics Controller. Gère la sortie vidéo en général et génère les signaux pour l'affichage Super Hi-res en particulier.
IWM	Integrated Woz Machine. C'est le contrôleur pour soutenir les lecteurs de disque aux formats 5,25 et 3,5 pouces.
Sound GLU	Sound General Logic Unit. Gère l'interface du bus de sortie son et de l'oscillateur digital (DOC = Digital Oscillator Chip).
DOC	Digital Oscillator Chip. C'est un générateur de sons simplifiés, exploité en association avec le GLU.
KeyGLU	Keyboard General Logic Unit. Gère et organise l'environnement nécessaire pour entamer le dialogue entre le système et le microprocesseur du clavier.
Keyboard Processor	Le microprocesseur (50740A) du clavier détachable constitue un support compatible à l'interface Apple DeskTop Bus qui permet de raccorder en chaîne un clavier, une souris ou tout périphérique compatible.

Microprocesseur 65C816

Le microprocesseur 65C816 réalise l'interface entre les différents circuits de la carte mère par l'intermédiaire de ses bus. C'est un C-MOS (Complementary Metal Oxide Silicon), un vrai 16 bits, de la famille des 65C... il émule parfaitement le fonctionnement d'un 6502.

Caractéristiques :

- accumulateur de 16 bits;
- registres d'index X et Y de 16 bits;
- page zéro relogeable dans n'importe quel banc mémoire;

- pile relogeable dans n'importe quel banc mémoire;
- bus d'adresses interne de 24 bits;
- banc de données adressable sur 8 bits;
- banc d'adresse du programme sur 8 bits;
- 11 nouveaux modes d'adressage disponibles;
- 36 nouvelles instructions, ce qui étend à 96 le nombre total des instructions disponibles (256 codes machine reconnus);
- instruction de déplacement rapide de blocs complets;
- émulation des microprocesseurs 6502 et 65C02;
- circuit d'horloge du processeur fixé à 2,8 Mhz, programmable au travers du tableau de bord (2,5 Mhz pour exécuter des programmes en RAM, 2,8 Mhz pour la ROM et 1 Mhz en mode émulation).

Disposition des adresses mémoire

Le bus d'adresses du microprocesseur 65C816, d'une taille de 24 bits, autorise un adressage de la mémoire jusqu'à 16 Mo. Cette mémoire adressable se répartit, par construction, en 8 Mo de mémoire vive (RAM) et 8 Mo de mémoire morte (ROM). L'espace mémoire affecté aux bancs $80 à $DF et $E2 à $EF n'est pas utilisé à ce jour, Apple Computer se réservant le droit de les utiliser pour des applications futures.

D'origine, le GS est doté de 128 Ko de ROM résidente et de 256 Ko de RAM, dont 128 Ko sont de structure lente et les autres 128 Ko à structure rapide. La mémoire lente, bancs $E0-$E1, est réservée pour traiter des applications spécifiques aux processeurs 6502 et 65C02. Elle est principalement exploitée par les zones tampons d'affichage du texte et du graphique. La mémoire vive rapide, bancs $00-$01, possède la faculté de projeter (Shadowing) l'image de son contenu sur certaines zones de la mémoire vive lente des bancs $E0-$E1. En mode émulation (6502) du GS, 128 Ko sont disponibles aux programmes, cet espace mémoire étant considéré comme de la mémoire principale (Main Memory) et de la mémoire auxiliaire (Aux Memory). Les routines de la boîte à outils de la ToolBox exploitent certaines zones mémoire des 128 Ko restants.

En mode natif, jusqu'à 176 Ko des 256 Ko sont disponibles pour des applications, le reste étant réservé aux tampons d'affichage vidéo et aux programmes système (ProDOS). Pour permettre une extension de la mémoire vive, un connecteur spécial est prévu d'origine sur la carte mère. Cette zone mémoire d'expansion est exploitée automatiquement en mode natif, ProDOS16 permettant la gestion via le Memory Manager résident dans la ToolBox (boîte à outils). Grande innovation sous ProDOS16, l'espace mémoire n'est plus géré par la System Bit-Map (adresses $BF58-$BF6F) à travers la Global Page de ProDOS, mais par le Memory Manager. Les adresses de l'espace mémoire d'expansion sont, par conception et construction, disposées dans une suite continue des adresses.

Les 256 Ko de la mémoire vive ne sont jamais disponibles en totalité, certaines adresses étant réservées au système et à la ROM résidente.

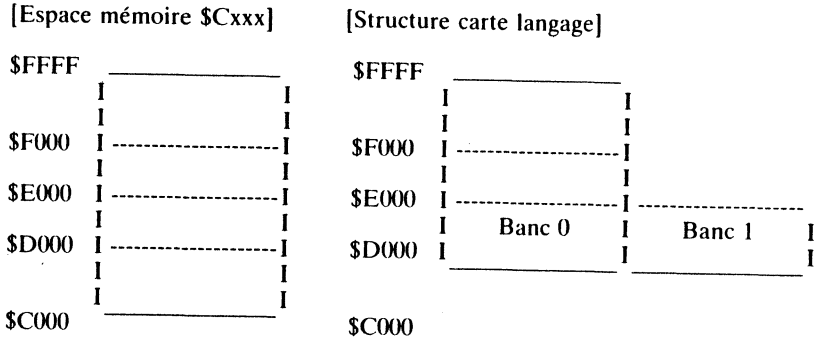
L'espace mémoire de la RAM disponible, et de façon continue, s'effectue par bancs de 64 Ko. Les bancs $00-$7F, soit 128 bancs de mémoire rapide, offrent 8 Mo de mémoire vive (128 bancs de 64 Ko). Les bancs $E0-$E1, soit 128 Ko de mémoire lente, sont en place pour garder une compatibilité relative avec la gamme des Apple *II*. Ils sont en principe réservés à des adresses d'entrées/sorties du système et aux tampons d'affichage vidéo.

L'espace mémoire alloué à la ROM s'étend des bancs $F0 à $FF, les bancs $F0 à $FD étant réservés à la ROM d'une carte d'extension et les bancs $FE-$FF à la ROM résidente (1 Mo au total).

Vecteurs d'entrées/sorties

Le système ProDOS exploite, à travers la RAM, des adresses d'entrées/sorties qui correspondent à de la mémoire morte mise en place par une interface dans un des slots d'extension. Ces slots, au nombre de sept (1 à 7) pour le GS, maintiennent une compatibilité envers la structure hardware dans la gamme des Apple *II* (Apple *II* et Apple *II+*). Le slot 0 a disparu sur le GS, des adresses réservées au système occupent maintenant cette zone mémoire.

Carte mémoire



Adresses d'entrées/sorties

• 8 octets sont réservés pour chaque carte d'extension, et ceci dans la zone mémoire réservée au tampon d'affichage en mode texte. Le contenu des octets réservés à chaque slot n'est pas affiché à l'écran, ni modifié par la routine COUT1 qui exploite cette mémoire de visualisation. Les cartes interface utilisent ces adresses pour mémoriser temporairement des données.

• 16 octets de l'espace mémoire des entrées/sorties sont utilisés comme tampon par chaque carte interface connectée dans un des slots de la carte mère (slots 1 à 7). Cès adresses se situent dans la zone d'adresses $C080 à $C0FF.

256 octets sont réservés à la ROM de chaque carte d'interface d'un slot en ligne. Ces adresses se situent dans la zone $C100 à $C7FF des adresses d'entrées/sorties.

2048 octets (2 Ko), destinés à l'expansion d'une ROM, sont mis en place par une carte interface connectée dans un des slots de la carte mère. Cette zone d'adresses se situe de $C800 à $CFFF.

COMPATIBILITE DES VECTEURS ENTREES/SORTIES

A part la série des Apple *II*c, chaque type de machine est doté par construction d'un certain nombre de slots d'extension. Cette structure permet d'étendre les capacités de l'ordinateur par le simple fait de lui connecter une carte interface compatible Apple. Les premiers Apple *II* étaient dotés de 8 slots d'extension, numérotés de 0 à 7, le slot 0 étant réservé à une carte langage. A partir de l'Apple *II*e, le slot 0 a totalement disparu, ne laissant en place que 7 slots, numérotés de 1 à 7. Les adresses de l'ancien slot 0 ont été affectées au système.

La nouvelle structure hardware du GS modifie certaines dispositions relatives aux cartes d'extension. En principe, et par définition, toute carte d'extension, destinée à prendre place dans un des slots, doit répondre aux critères suivants :

- la carte interface devra être reconnue par l'environnement ProDOS;
- un protocole Apple est défini pour chaque type de carte d'interface;
- chaque carte d'interface est dotée de ses propres circuits matériels (circuits hardware) et logiciel (ROM firmware);
- le périphérique en ligne, raccordé à la carte interface, devra être adapté à ce type de carte interface. Il communiquera par la suite avec l'environnement et sera reconnu comme tel;
- la carte interface n'est pas un simple artifice d'interface pour les entrées/sorties, mais accède au circuit horloge, aux différents signaux de contrôle, au bus de données et au bus d'adresses (uniquement les 16 bits les moins significatifs du bus d'adresses du 65C816;
- un seul et unique signal est commun aux 7 slots du GS (sauf le signal couleur, carry bit apparent au slot 7);
- une compatibilité relative est adoptée envers la gamme des Apple *II* existants;
- certains signaux de base sont différents par rapport aux structures initiales, à savoir les signaux d'inhibition et de synchronisation;
- certains slots occupent des emplacements réservés par définition et par construction : slot 1 réservé à l'imprimante, carte 80 colonnes dans le slot 3, slots 5 et 6 destinés aux Blocks Devices (lecteurs de disques par exemple);
- l'Apple *II*GS est doté d'un bus d'adresses de 24 bits. Pour garder une compatibilité avec la gamme initiale des Apple *II*, les slots d'extension ne traitent que des adresses sur 16 bits, le bus d'adresses n'utilisant dans ce cas que les 16 bits les moins significatifs;
- par défaut, les 64 Ko du banc $E0 sont utilisés par une carte d'extension y faisant référence pendant le mode émulation;
- en mode émulation, les applications chargées à partir d'un slot d'extension et exécutées dans les bancs $00-$01, sont projetées dans la mémoire lente des bancs $E0-$E1.

SLOT I/O – ESPACE MEMOIRE ALLOUE

Chaque slot d'extension possède par construction, seize adresses réservées dans la zone mémoire et exploitées par les interfaces connectées dans chaque slot. Ces interfaces permettent le plus souvent de dialoguer en entrée et en sortie avec l'unité centrale.

La zone mémoire réservée à chaque groupe de 16 adresses se définit de la façon suivante :

$C08x + s0

où,

x représente une valeur hexadécimale comprise entre 0 et F inclus;

s représente le numéro du slot considéré.

Exemple

Si nous considérons le slot 3, le début de la zone mémoire des 16 adresses réservées se situe à partir de : $C080 + $30, ce qui correspond à $C0B0. En ajoutant les 16 adresses réservées ($0F en hex) à la valeur obtenue précédemment, le résultat sera l'adresse de la fin de la zone réservée :

$C0B0 + $0F = $C0BF

Nous pouvons déclarer que les adresses $C0B0-$C0BF sont réservées à des données utilisées par l'interface mise en place dans le slot 3.

Adresses réservées aux slots (I/O Device select)

Slots	Adresses	Validée par
–	$C080-$C08F	Système (ancien slot 0)
1	$C090-$C09F	Slot 1 Device select
2	$C0A0-$C0AF	Slot 2 Device select
3	$C0B0-$C0BF	Slot 3 Device select
4	$C0C0-$C0CF	Slot 4 Device select
5	$C0D0-$C0DF	Slot 5 Device select
6	$C0E0-$C0EF	Slot 6 Device select
7	$C0F0-$C0FF	Slot 7 Device select

SLOT ROM – ESPACE MEMOIRE ALLOUE

Chaque slot possède par construction une zone mémoire réservée, d'une taille de 256 octets, destinée à y placer le contenu de la ROM (ou EPROM) de l'interface. Cette mémoire morte, résidente sur la carte d'extension, est programmée par le concepteur. C'est elle qui gère le bon fonctionnement du périphérique qui lui est raccordé par la suite.

La zone mémoire de chaque groupe de 256 octets réservés à de la ROM Firmware, se définit ainsi :

- l'adresse $Cs00 est utilisée comme vecteur de base, où s représente le numéro de slot dans lequel se trouve la carte interface.

Exemple

Si nous considérons à nouveau le slot 3, la zone mémoire des 256 octets réservés à de la ROM firmware se situe aux adresses $C300-$C3FF

Adresses réservées aux slots (I/O select)

Slots	Adresses	Validée par
–	$C000-$C0FF	Commutateurs logiques (ancien slot 0)
1	$C100-$C1FF	Slot 1 I/O select
2	$C200-$C2FF	Slot 2 I/O select
3	$C300-$C3FF	Slot 3 I/O select
4	$C400-$C4FF	Slot 4 I/O select
5	$C500-$C5FF	Slot 5 I/O select
6	$C600-$C6FF	Slot 6 I/O select
7	$C700-$C7FF	Slot 7 I/O select
–	$C800-$CFFF	Expansion de la ROM

En plus des zones mémoire attribuées à chaque slot de la carte mère, une carte périphérique a la possibilité d'utiliser un espace réservé à de la ROM d'expansion, ce qui augmente de 2 Ko la ROM disponible à un périphérique en ligne (adresses $C800-$CFFF). Cette extension de la ROM est souvent rendue nécessaire pour y loger des programmes plus longs, contenus dans les ROM ou EPROM. C'est la mémoire d'expansion pour périphérique. Si une carte interface désire utiliser cette mémoire d'expansion, elle doit être équipée d'un circuit qui autorise l'accès à la mémoire morte. C'est un composant électronique spécial qui effectue cette opération en deux temps :

- mise à 1 d'une bascule lorsque le signal I/O select sur la broche 1 du connecteur est actif (niveau bas);

- puis il autorise les dispositifs de mémoire morte d'expansion quand le signal I/O Strobe sur la broche 20 du connecteur (slot) devient actif (niveau bas). Le signal I/O Strobe d'un connecteur devient actif chaque fois que le microprocesseur traite une adresse de l'espace mémoire des 256 octets de mémoire morte attribuée à ce connecteur ($Cs00 à $CsFF).

SLOT RAM – ESPACE MEMOIRE ALLOUE

Comme il a été spécifié précédemment, le concepteur a doté le GS de zones mémoire réservées (adresses d'E/S), utilisées soit pour y stocker des paramètres (16 adresses pour chaque interface disponible : Device Select), soit pour y placer le sous-programme d'une ROM interface (I/O Select). Quelques adresses se trouvent encore éparpillées dans la zone d'affichage de la page texte, adresses $0400-$07FF. Chaque adresse de base + s correspond à de la mémoire vive accessible aux interfaces connectées en 's' (s = numéro du slot).

C'est ainsi que les adresses, normalement réservées à l'affichage vidéo, contiennent huit zones ou 'trous'. Chacun de ces 'trous' se compose de huit octets utilisés par les cartes interfaces connectées dans les slots d'extension. Chaque slot se réserve un octet dans chacun des huit 'trous' dans la zone tampon de visualisation.

Répartition des 'trous' réservés au slot RAM

$0478 à $047F où $0478 représente l'adresse de base
$04F8 à $04FF où $04F8 représente l'adresse de base
$0578 à $057F où $0578 représente l'adresse de base
$05F8 à $05FF où $05F8 représente l'adresse de base
$0678 à $067F où $0678 représente l'adresse de base
$06F8 à $06FF où $06F8 représente l'adresse de base
$0778 à $077F où $0778 représente l'adresse de base
$07F8 à $07FF où $07F8 représente l'adresse de base

Huit octets sont réservés par construction à chaque carte interface. Chaque 'trou' est constitué de huit octets. A chaque suite de huit octets (un 'trou') correspond un octet d'un slot occupé, l'octet 0 n'étant pas utilisé sous un environnement GS mais disponible au système. L'attribution des adresses se détermine de la façon suivante :

• La zone mémoire des adresses $0400-$07FF se divise en 8 blocs ou 'trous' mémoire : 1024/8 = 128 adresses. Chaque bloc de la mémoire de visualisation se compose de 128 adresses (0 à 127).

• Chaque série de 128 adresses se compose de 120 caractères de visualisation et de 8 octets réservés à chaque slot (l'octet du slot 0 n'est pas utilisé). Si nous considérons la première série des 128 adresses de visualisation, le détail est le suivant :

- 40 caractères de la ligne 0;
- 40 caractères de la ligne 8;
- 40 caractères de la ligne 16;
- 8 octets réservés aux variables des slots, le slot 0 étant assigné au système afin de garder une compatibilité avec la gamme des Apple *II*.

• On retranche à 128 le nombre 8 qui correspond aux sept slots + l'octet disponible au système, ce qui donne comme résultat :

128 – 8 = 120, ou $78 en hex.

• Chaque adresse de base des séries d'adresses de visualisation est déterminée en ajoutant la valeur $78 au résultat obtenu :

- première adresse de base : $0400 + $0078 = $0478 (adresses réservées aux slots : $0478-$047F)
- deuxième adresse de base : $0480 + $0078 = $04F8 (adresses réservées aux slots : $04F8-$04FF)
- troisième adresse de base : $0500 + $0078 = $0578 (adresses réservées aux slots ; $0578-$057F)
- etc.

• A chaque adresse de base s'ajoute la valeur du numéro du slot, ce qui détermine l'adresse réservée au périphérique connecté à l'interface mise dans le slot :

$0478 + s; si s = 1, adresse du slot = $0479
$0478 + s; si s = 2, adresse du slot = $047A
$0478 + s; si s = 7, adresse du slot = $047F
$04F8 + s, etc.

PORTS SERIE (I/O)

L'Apple *IIGS* est doté d'origine de deux ports série qui se substituent facilement aux connecteurs 1 et 2 prévus sur la carte mère. Le choix de l'un ou l'autre des ports s'effectue à travers le tableau de bord accessible à partir du clavier (Pomme-ouverte/Controle/Esc). L'utilisateur possède alors deux choix :

- exploiter les capacités des ports série intégrés, qui émulent parfaitement une interface aux normes RS-422;
- mettre en place sa propre carte interface série.

Les ports intégrés sont soutenus par un circuit intégré du type Zilog 8530, émulant parfaitement une carte super série standard Apple. Par définition, le slot 1 est réservé au port imprimante et le slot 2 au port de communication (Modem par exemple).

Adresses et protocoles sous Basic AppleSoft

Adresse	Description
$Cs00	Début de la routine d'initialisation intégrée dans la ROM interface. Le caractère stocké à cette adresse est chargé dans l'accumulateur.
$Cs05	Caractère d'identification du port, Read Character. La valeur est retournée dans l'accumulateur, les registres X et Y sont préservés.
$Cs07	Caractère d'identification de l'interface, Write Character. La valeur est transitée à travers l'accumulateur, les registres X et Y restent préservés.

Adresses et protocoles sous Pascal

Adresse	Description
$Cs0D	Valeur offset de l'adresse d'initialisation de la routine.
$Cs0E	Valeur offset de l'adresse de lecture (PRead).
$Cs0F	Valeur offset de l'adresse de sauvegarde (PWrite).
$Cs10	Valeur offset de l'adresse de la routine d'état (PStatus).
$Cs12	Valeur offset de l'adresse de la routine de contrôle pour une interface d'extension.

Améliorations apportées au niveau des ports serie

- tampon mémoire intégré, utilisé lors du transfert ou de l'édition des données (imprimante, modem, AppleTalk, etc.). Un tampon mémoire, d'une taille de 2 Ko par défaut, est réservé aussi bien dans le sens entrée (Input) que dans le sens sortie (Output) des données transitées. La gestion du tampon mémoire est effectuée par le Memory Manager (ToolBox), sa taille pouvant être étendue jusqu'à 64 Ko. La modification du tampon mémoire se réalise par le tableau de bord ou à l'aide de commandes appropriées.

- édition en tâche de fond, tout en pouvant travailler sur un programme. L'édition de données à partir d'un port sériel, en tâche de fond, est réalisée à travers les sous-programmes résidents dans la ROM (outils). Cette pratique permet de continuer le traitement d'un programme en cours d'exécution tout en imprimant des données par exemple. Le tampon mémoire, d'une capacité de 2 Ko par défaut, peut être étendu jusqu'à 64 Ko par le simple fait de modifier un paramètre dans la RAM non volatile.

- interface AppleTalk intégrée d'origine. L'utilisateur peut se connecter à un réseau AppleTalk à travers le port sériel de son choix (1 ou 2). Ce port est activé à travers le tableau de bord et peut occuper indifféremment les ports 1 ou 2. AppleTalk est un réseau local de communication, développé par Apple Computer. Il permet d'exploiter en commun, à partir de plusieurs postes de travail interconnectés et sous environnement GS ou Macintosh, les propriétés d'une imprimante ImageWriter *II* dotée d'une carte Apple talk ou LaserWriter par exemple.

PROTOCOLES DU HARDWARE & SOFTWARE

L'Apple *II*GS , en plus de ses nombreux perfectionnements, possède d'origine un port pour lecteur de disquettes. Ce support intégré pour périphérique standard, permet le raccordement de quatre lecteurs de disquettes (3,5 ou 5,25 pouces).

Le contrôleur de ce port dit intelligent (SmartPort), est émulé à travers un circuit intégré IWM (Integrated Woz Machine), connu de ceux qui pratiquent l'Apple *II*c. Sa capacité de gestion peut contrôler les slots 5 et 6 sur lesquels peuvent se connecter d'autres cartes interfaces: lecteurs de disques, RAM-Disk, ROM-Disk ou encore un disque dur.

La gestion des informations véhiculées à travers le SmartPort et transmises aux différents Blocks Devices, est assurée par le convertisseur de protocole (Protocol Converter). Le convertisseur de protocole est constitué par un ensemble cohérent de routines résidentes, destinées à faciliter les applications exécutées par l'intermédiaire du SmartPort.

La sortie normalisée du port d'entrée/sortie des devices nécessite une organisation dans le branchement des périphériques compatibles. Le raccordement sur le GS se fait dans l'ordre suivant:

- au maximum deux lecteurs du type dit non intelligent, dépourvus de la structure interne du SmartPort (Sony 3,5 pouces, Unified Disk 3,5 pouces);

- des lecteurs 3,5 pouces du type dit intelligent, intégrant l'électronique d'une structure similaire au SmartPort;

- des lecteurs de disquettes au format 5,25 pouces, de type 5,25 Apple, UniDisk ou DuoDisk 5,25 pouces.

PROTOCOLES RELATIFS AUX DEVICES

Par Device, il faut sous-entendre tout périphérique compatible GS, du type Block (drives) ou Character (clavier). Ces Devices traitent respectivement les données par bloc et caractère par caractère, aussi bien en mode lecture

qu'en mode écriture. Dans les commentaires qui suivent, un Device Block est assimilé par défaut à tout lecteur de disques, qu'il soit aux formats 3,5 ou 5,25 pouces, ou encore un disque dur. Ce genre de périphérique d'E/S traite particulièrement les données par blocs de 512 octets.

Un device est soutenu par un Device Driver, pilote de gestion faisant partie des structures de la ToolBox. Deux types de périphériques caractérisent la liste des Devices Drivers possibles :

- **Character Device Driver**

Ce sont les périphériques qui exploitent les nombreuses possibilités du caractère, tels la console du clavier, l'imprimante, l'écran vidéo, la souris ou tout autre matériel compatible. Ce type de Device est généralement appelé périphérique du type caractère.

- **Block Device Driver**

Ce sont les périphériques qui manipulent des données, par bloc de 512 octets. Les Blocks Devices désignent les lecteurs de disques en général, contrôlés soit par le SmartPort, soit par une carte interface mise en place dans un des slots. Ce type de périphérique peut se regrouper en plusieurs unités, et un seul Device Driver peut gérer plusieurs disques à la fois. Ce type de Device est généralement appelé périphérique du type bloc.

L'Apple *II*GS , de part sa construction, supporte toute une série de Devices. Le port intégré (IWM), du type intelligent (SmartPort), reconnaît un certain nombre de Devices par défaut.

Le sous-programme de démarrage, ou BootStrap, du SmartPort est implanté d'origine dans la ROM résidente du GS. Le port 5 émule un SmartPort qui autorise le raccordement de lecteurs de disques 3,5 et 5,25 pouces. Dans le cas d'une configuration mitigée, les lecteurs au format 5,25 pouces seront chaînés à la suite des lecteurs 3,5 pouces. Le port 5 est toujours réservé au format 3,5 pouces, le port 6 au format 5,25 pouces.

AFFECTATION DU SMARTPORT

Le slot 5, grâce à une électronique interne, possède les structures d'un port intelligent, réservé par construction aux lecteurs 3,5 pouces (800 Ko). La routine du BootStrap, séquence de démarrage du processus de lancement du système d'exploitation, se trouve dans la ROM résidente. Elle diffère de celle de la carte contrôleur du Disk *II* par le simple fait qu'elle appelle des routines de la ToolBox (boîte à outils).

Le slot 5 est défini comme SmartPort par défaut. C'est une adaptation de la technique intégrée dans les 32 Ko de la ROM système. Le SmartPort supporte deux drives 3,5 pouces, un RAM-Disk, et un Unidisk 3,5 pouces, offrant jusqu'à 127 combinaisons possibles.

AFFECTATION DU SLOT 6

Le slot 6 possède une structure classique, très proche du standard réservé au Disk *II* (140 Ko). Comme pour le slot 5, le BootStrap se trouve intégré dans la ROM résidente de la carte interface. La routine BootStrap, spécifique à la gamme des Apple *II*, permet de charger indifféremment les systèmes ProDOS, DOS 3.3 ou Pascal.

L'interface du slot 6, adresses I/0 $C600 à $C6FF, contient les routines indispensables pour piloter tout drive 5,25 pouces Apple ou compatible. Les adresses d'E/S (I/0 = $C600 à $C6FF) sont réservées à la copie de la ROM interface connectée dans le slot 6, ou à l'émulation du SmartPort. Le slot 6 supporte jusqu'à deux drives au format 5,25 pouces, le repérage physique étant réalisé par les chiffres 1 et 2 (D1 et D2). Uniquement un côté du support de sauvegarde est traité à la fois, ce type de drive n'étant équipé que d'une tête de lecture/écriture.

Tableau des caractéristiques communes au slot 6

Numéro du port	Input/output, port 6 drive 1 (D1) Input/output, port 6 drive 2 (D2)
Commande	A partir du Basic : IN£6 ou PR£6 A partir du Monitor : 6 Control-P
Hardware	Les adresses $C0E0 à $C0EF (Device I/0 Select), sont réservées à l'interface du lecteur connecté au slot 6 (réservés principalement au contrôleur du Disk *II*).
Adresses I/0	L'adresse $C600 est le point d'entrée de la routine de boot (port 6, adresses $C600-$C6FF). L'adresse $C65E est le point d'entrée de la routine de lecture du premier secteur de la piste $00. Elle débute l'exécution par le premier code machine trouvé.
Sortie vidéo	Les bancs $00 et $01 sont réservés aux tampons de visualisation des zones mémoire de Main et Aux Memory ($0400-$07FF). Le contenu des adresses réservées à des variables du slot n'est pas affiché à l'écran.

Routines et protocoles relatifs au SmartPort

Le SmartPort est une extension pour un périphérique du type Block Device, intégrant d'origine tous les Drivers réservés normalement aux slots 5 et 6. Les Drivers sont des routines pilotes constituées par une suite de codes machine. Une suite de codes cohérents, ou mnémoniques, forment des instructions assimilées à des programmes capables de gérer l'environnement des périphériques compatibles à ce type de port intégré. Le SmartPort interprète et décode les commandes qui lui sont transmises, et pilote en conséquence le device référencé par la commande.

TRANSMISSION D'UNE COMMANDE AU SMARTPORT

Le SmartPort, sous environnement ProDOS16, supporte et transmet des commandes à un périphérique du type Block Device, comme si elles étaient adressées à une application. Il simule une interface entre le système d'exploitation ProDOS16 et le périphérique référencé dans la commande. La commande au niveau le plus évolué se transmet par un JSR à l'entrée Dispatch du SmartPort.

Le Protocol Converter consiste à assurer la communication entre l'utilisateur, l'unité centrale représentée par le GS, le système d'exploitation ProDOS16 et le Block Device (lecteur de disques 3,5 pouces par exemple). C'est lui qui permet au drive 800 Ko d'analyser et d'exécuter la totalité des commandes qui lui sont transmises. La méthode consiste à appeler la routine localisée à une adresse déterminée par une signature. Le concepteur a doté l'interface d'une signature établie suivant un protocole défini par avance. Ce protocole est matérialisé par quatre octets caractéristiques permettant ainsi de détecter dans quel slot se situe la carte contrôleur du drive 3,5 pouces.

SIGNATURE DU SMARTPORT

Toute carte interface occupe un des slots d'extension assignés au GS et soudés à même la carte mère. Le SmartPort, comme tout autre port occupé par une carte interface, peut très facilement être déterminé en lisant la valeur des quatre octets de signature (ProDOS Block Device Signature). Ces octets font partie de la ROM interface copiée aux adresses d'E/S.

$Cs01	= $20
$Cs03	= $00
$Cs05	= $03
$Cs07	= $00 pour un drive 3,5 pouces = $3C pour un drive 5,25 pouces ou autre

Remarques

- s désigne le numéro du slot où se trouve la carte interface du contrôleur de disques;

- La signature pour les trois premiers octets est identique aux Blocks Devices 3,5 ou 5,25 pouces;

- L'octet $Cs07 est de valeur $00 pour un disque 3,5 pouces (800 Ko) et de valeur $3C pour un disque 5,25 pouces (140 Ko).

- En ce qui concerne un disque dur, la signature reste identique à celle d'un Block Device 5,25 pouces.

PROTOCOLE DE CONVERSION – PROTOCOL CONVERTER

Le terme Dispatch désigne l'adresse d'entrée du SmartPort. Ce vecteur correspond au premier code machine exécutable, utilisé pour démarrer le BootStrap du Block Device désigné. Pour calculer l'adresse du Dispatch, il faut ajouter à l'adresse $Cs00 le contenu de l'adresse $CsFF, augmentée de la valeur 3.

Dispatch = ($Cs00 + ($CsFF)) + 3

où,
– s désigne le numéro du slot;
– ($CsFF) représente le contenu de l'adresse $CsFF.

Le Protocol Converter autorise et favorise l'accès au Block Device, via le SmartPort, selon les mêmes principes mis en jeu avec les commandes ProDOS, seule la syntaxe change. Pour ce faire, un **JSR POINTEUR** est transmis au device, où **POINTEUR** représente l'adresse d'entrée du Dispatch. A la suite de la commande **JSR POINTEUR**, on trouve un numéro codé sur un octet qui personnalise la commande (CmdNum), suivi d'une adresse sur deux octets qui pointe la liste des paramètres du Block Device (CmdList).

TYPES DE COMMANDES TRANSMISSIBLES AU SMARTPORT

Deux types de commandes sont reconnues : la commande standard et la commande étendue.

- La **commande standard** s'adresse à toute interface supportant uniquement les périphériques compatibles Apple *II*. Ce type de commande est reconnu uniquement par un Block Device répondant au format 5,25 pouces (140 Ko). Dans ce cas, il appartiendra au programmeur, ou à l'application courante, de placer la table des paramètres (CmdList) dans le banc $00.
- La **commande étendue**, développée spécialement pour des applications tirant partie des propriétés du microprocesseur 65C816, permet d'exploiter d'autres bancs mémoire que le banc $00. Les blocs de données manipulés sont d'une taille plus importante et occupent maintenant des zones mémoire au-delà du banc $00. Ce type de commande est particulièrement utilisé avec un Block Device répondant au format 3,5 pouces (800 Ko). Dans ce cas, l'appel transmis par un **JSR ADRESSE** sera structuré en conséquence : la variable **ADRESSE** pointe toujours l'entrée Dispatch, mais le pointeur de la table des paramètres (CmdList) nécessite maintenant quatre octets. Cette table des paramètres (CmdList) pourra se trouver dans tout banc mémoire valide du GS.

La pile sera structurée selon le type de commande à transmettre, d'une taille variant de 30 à 35 bytes inclus. Un programme alloue d'office une taille de 35 bytes lors du premier appel de la commande.

Comme en mode natif, le SmartPort reste accessible en mode émulation. Le drapeau du mode émulation est positionné à la valeur 1 et les registres DBR (Data Bank Register) et DR (Direct Register) sont mis à zéro.

Transmission au SmartPort d'une commande standard

```
JSR Dispatch      ;Adresse d'entrée établie par le Protocol
                   Converter
DFB CmdNum        ;Numéro de la commande (1 octet)
DFB CmdList       ;Pointeur de la liste des paramètres
                   (2 bytes, dans l'ordre LByte, HByte)
BCS Error         ;Gestion de l'erreur
```

Transmission au SmartPort d'une commande étendue

```
JSR Dispatch      ;Adresse d'entrée établie par le Protocol
                   Converter
DFB CmdNum        ;Numéro de la commande (1 octet)
DFB CmdList       ;Pointeur de la liste des paramètres
                   (4 bytes, dans l'ordre LWord, HWord)
BCS Error         ;Gestion de l'erreur
```

Remarques

La liste des paramètres (CmdList) se compose d'une suite de paramètres faisant référence au Block Device désigné et destinés à traiter la commande

transmise au SmartPort. CmdList utilise un format particulier, très proche de la structure d'une table de paramètres associée à une commande ProDOS. La commande étendue, mise en place pour exploiter les capacités de programmation du GS, se différencie de la commande standard par deux points essentiels :

– d'une part, par son écriture : elle nécessite 4 octets au lieu de deux;

– d'autre part, par ses possibilités étendues, permettant de situer la liste des paramètres dans n'importe quel banc mémoire.

Structure de la CmdList

```
CmdList     DFB Parameter_Count     ; Un byte. Fixe le nombre de paramètres attribués à la commande, ce qui détermine le nombre total de bytes de la liste.

            DFB Unit_Number         ;Un byte. Reflète l'image binaire du byte affecté au périphérique device en ligne.
                                    Désigne le drive auquel s'adresse la commande transmise via le SmartPort.
                                    L'attribution s'effectue par progression ascendante, le numéro $01 étant le premier numéro assigné. Le RAM-Disk, le ROM-Disk et le drive 3,5 pouces sont des devices possibles, pouvant être assignés par la variable Unit_Number (valeurs correctes : $00, $01 à $7E)

            DFB Buffer_Address      ;Deux bytes. Ils pointent le tampon d'entrée/sortie réservé à l'exécution en cours. Le SmartPort pointe par défaut une zone mémoire référencée dans le banc $00.
                                    Une commande étendue permet de pointer tout autre banc mémoire.

            DFB Block_Number        ;Nombre de blocs logiques traités par le device en ligne. Pour l'utilisation d'une commande standard, ce paramètre est structuré sur 24 bits (3 bytes), pouvant être étendu à 32 bits (4 bytes) pour des applications particulières (commande étendue).

            DFB Byte_Count          ;Deux bytes. Ils désignent le nombre de bytes qui sont à transférer de la mémoire vers le device, ou inversement.

            DFB Address_Pointer     ;Désigne l'adresse du device en ligne.
```

EXEMPLES DE TRANSMISSION D'UNE COMMANDE

Commande du type standard

```
Standard    JSR Dispatch        ;Entrée du Dispatch, adresse déterminée par le Control Converter;
            DFB CmdNum          ;Numéro de la commande;
            DW CmdList          ;Pointe la table des paramètres;
            BCS Error           ;Gestion du code d'erreur (Carry = 1 si erreur rencontrée).
```

Commande du type étendue

```
Etendu      JSR Dispatch        ;Entrée du Dispatch déterminée par le Control Converter;
            DFB CmdNum + $40    ;La valeur $40 spécifie que c'est une commande étendue;
            DW CmdList          ;Mot le moins significatif codé sur deux octets, pointe la table des paramètres;
            DW CmdList          ;Mot le plus significatif codé sur deux octets, pointe la table des paramètres;
            BCS Error           ;Carry = 1 si erreur.
```

Situation des registres du microprocesseur au retour d'une commande adressée au SmartPort

• Valeurs possibles des flags du registre d'état et des registres du processeur au retour d'une commande aboutie, transmise au SmartPort :

	Status Byte	Registres du 65C816				
	n v m x d i z c	A	X	Y	PC	SP
Standard	N N 1 N 0 1 N 0	0	IN	IN	JSR +3	I
Etendue	I I 1 1 0 U X 0	0	IN	IN	JSR +5	I

• Valeurs possibles des flags du registre d'état et des registres du processeur au retour d'une commande non aboutie, transmise au SmartPort :

	Status Byte	Registres du 65C816				
	n v m x d i z c	A	X	Y	PC	SP
Standard	N N 1 N 0 1 N 1	ERR	N	N	JSR +3	I
Etendue	N N 1 X 0 1 N 1	ERR	N	N	JSR +5	I

N = Non défini.
I = Inchangé.
IN = Indéterminé : valeur au retour d'un transfert de données à partir du device courant, ou d'un nombre de bytes devant être transférés à partir du device.
ERR = Code d'erreur.

STATUS CODE DU SMARTPORT

Le SmartPort autorise de nombreuses manipulations grâce à des mécanismes très complexes mis en place par le concepteur. Il est particulièrement recommandé de connaître l'existence de la commande Status Call (CmdNum = $00 pour une commande standard et CmdList = $40 pour une commande étendue) qui permet de lire le statut du périphérique raccordé au port intelligent.

D'autres paramètres étant liés à cette commande, il est bon de faire un rapide tour d'horizon des divers bytes d'identification mis en présence avec le SmartPort :

• Le byte ID Type, identification de l'interface Device, se trouve à l'adresse $CsFB de l'interface considérée, où s représente le numéro du slot dans lequel se trouve la carte.

Identification du SubType (StatCode $03 – DIB)

Bits 7 6 5 4 3 2 1 0
1 1 0 x x x x x

Bit 7 = 1, commande étendue du SmartPort;
Bit 6 = 1, commutateur de dispatching de l'erreur;
Bit 5 = 0, Device amovible;
Bits 4 à 0, réservés pour des applications futures.

• Le Device Status, byte caractéristique du statut général du Device en ligne, est retourné par la commande Status Call (StatCode $00).

• La longueur de la chaîne ID (Identification Device) est donnée sur un byte, et se trouve en début de la chaîne d'identification. Cette longueur caractérise le nombre de caractères ASCII qui sont tolérés pour identifier le Device en ligne.

• La chaîne de caractères ID est constituée par un maximum de 16 bytes (codes ASCII) et qui identifient le Device en ligne. Chaque bit de poids fort des codes ASCII est mis à zéro (bit 7 = 0).

Identification ID Type ($CsFB)

Bits 7 6 5 4 3 2 1 0
E R R R R R S A

E = Commande du type étendue.
R = Réservé à une application future.
S = SCSI.
A = RamCard en place.

• Le byte SubType, statut byte caractéristique du Device Information Bloc (DIB), est le complément d'information relatif au Device en ligne. Il permet, en association avec le byte ID Type (adresse $CsFB) de reconnaître le Device courant. Le byte SubType est retourné suite à une commande Status Call transmise au SmartPort, où l'octet DIB personnalise le StatCode $03.

• Toute commande transmise au SmartPort requiert une syntaxe définie au préalable, à savoir un JSR qui exécute le premier code du sous-programme débutant à l'entrée du Dispatch, une table des paramètres valide et une gestion correcte de l'erreur.

Paramètres requis par la commande STATUS CALL

Parameter Count

Nombre de paramètres attribués à la commande : une valeur minimale de $03 est requise.

Unit Number

Un byte qui désigne le périphérique du type Device en ligne. Sa valeur pourra être $00 et $01 à $7E inclus.

CmdList

• Avec une commande standard, le pointeur est codé sur deux octets (un mot), et désigne la table des paramètres relatifs au Device référencé dans la commande. L'adresse de la table des paramètres du Device se trouve obligatoirement dans le banc $00.

Syntaxe du pointeur : LByte, HByte

• Avec une commande étendue, le pointeur est codé sur quatre octets (double mot), et désigne la table des paramètres relatifs au Device référencé par la commande. La table des paramètres pourra se trouver dans un banc mémoire quelconque.

Syntaxe du pointeur : Low Word, High Word

Code d'erreur (Status Code)

Un byte, valeur retournée après l'exécution d'une commande au SmartPort non aboutie. La valeur du Status Code sera comprise entre $00 et $FF inclus.
Le status (Status Code) de tout Device (Character ou Block) est personnalisé par l'écriture de quatre bytes :

– le Status Byte;
– le Device Control Block (DCB);
– le statut du type de Device référencé;
– le Device Information Block (DIB).

Status Code retourné

$00 = Device Status
$01 = Device Control block – DCB
$02 = Status Character
$03 = Device Information Block – DIB

• Le Status Byte (StatCode $00), ou byte caractéristique du périphérique Device, est significatif à travers chacun de ses bits :

Bit 7 = 1 si c'est un Block Device (drive), et 0 si c'est un Character Device (clavier, souris, etc.);
Bit 6 = 1, écriture autorisée;

Bit 5 = 1, lecture autorisée;
Bit 4 = 1, Device en ligne, ou disque dans le drive;
Bit 3 = 1, formatage autorisé;
Bit 2 = 1, Device protégé en écriture (drive uniquement);
Bit 1 = 1, interruption active (Apple *IIc* uniquement);
Bit 0 = 1, Device actif (uniquement pour les Character Devices, du type caractère (clavier, souris, etc.).

• Le Device Control Block (StatCode $01) ou DCB dépend du device en ligne. Il personnalise et effectue le contrôle de la commande transmise au SmartPort. Le premier byte référence la taille du bloc de contrôle en nombre d'octets. Si une valeur nulle est renvoyée, la taille du DCB est de 256 octets par défaut. Si la valeur retournée est 01, la longueur du DCB est de 1 octet. Les valeurs admises pour le DCB sont comprises entre 1 et 256 inclus.

• Newline Status (StatCode $02). Le SmartPort traite que des Blocks Devices, et ne reconnaît pas de périphérique du type Character Device, imprimante ou clavier par exemple. Par conséquent, le paramètre Newline renvoyé par la commande sera de valeur indéterminée. Newline est utilisé par le MLI du système ProDOS et nécessite un byte pour coder un caractère.

• Le Status Device Information Block (DIB) contient des informations qui permettent d'identifier par la suite le Device en ligne, son type et éventuellement ses attributs.

Format type du status DIB

Commande standard	Commande étendue
Device Status Byte	Device Status Byte.
Adresse du bloc de données (LByte)	Adresse du bloc de données (byte de poids faible contenu dans le mot de poids faible – Low byte, Low Word).
Adresse du bloc de données (MByte)	Adresse du bloc de données (byte de poids fort contenu dans le mot de poids faible – High byte, Low Word).
Adresse du bloc de données (HByte)	Adresse du bloc de données (byte de poids faible contenu dans le mot de poids fort – Low byte, High Word). Adresse du bloc de données (byte de poids fort contenu dans le mot de poids fort – High byte, High Word).
Longueur chaîne de caractères ID	Longueur de la chaîne de caractères ID.
Chaîne de caractères ID (16 bytes)	Chaîne de caractères ID (16 bytes).
Device SubType byte	Device Type byte.
Version du driver	Version du driver.

Devices Type et SubType assignés au SmartPort

Type	SubType	Device
$00	$00	Apple *II*, carte d'extension mémoire.
$01	$00	UniDisk 3,5 pouces.
$01	$C0	AppleDisk 3,5 pouces.
$03	$E0	Apple *II*,SCSI avec périphérique non amovible.

Devices Type et SubType non assignés au SmartPort

Type	SubType	Devices
$02	$20	Disque dur.
$02	$00	Disque dur amovible.
$02	$40	Disque dur amovible, avec gestion des erreurs.
$02	$A0	Disque dur compatible avec la commande étendue.
$02	$C0	Disque dur compatible avec la commande étendue et la gestion des erreurs.
$03	$C0	SCSI à travers un Device amovible.

SmartPort Driver Status

Byte 0	Nombre de Devices en ligne
Bytes 1 à 7	Réservés à des applications futures

COMMANDES RELATIVES AU SMARTPORT

Les commandes transmissibles au SmartPort sont référencées par un numéro marqué dans la table des paramètres (CmdList); Un numéro de commande transmise au SmartPort, diffère dans son écriture selon qu'elle soit du type standard ou étendue. Dans le cas d'une commande étendue, le numéro de la commande standard sera augmenté de la valeur $40.

Tableau des commandes adressables au SmartPort

Standard	Etendue	Signification de la commande
$00	$40	Status Call. Cette commande, très largement détaillée précédemment, permet de retourner au système le statut d'un Device particulier raccordé au SmartPort (4 bytes).
$01	$41	Read Block Call. Lecture d'un bloc de 512 bytes à partir du drive spécifié par le paramètre Unit_Number. Le bloc est lu, puis transféré dans le tampon mémoire spécifié par le pointeur Data_Buffer_Pointer.
$02	$42	Write Block Call. Ecriture d'un bloc de 512 bytes à partir du tampon mémoire spécifié, vers le drive référencé par le paramètre Unit_Number.

Standard	Etendue	Signification de la commande
		Le bloc, lu à partir de la zone mémoire spécifiée par le pointeur Data_Buffer_Pointer, est sauvegardé de la mémoire sur le support disque.
$03	$43	Format Call. Formatage du support contenu dans le Block Device. La routine de formatage n'est pas assemblée dans le système ProDOS, la routine Format Call (commande $03 et $43) prépare le périphérique aux séquences de lecture et d'écriture des blocs logiques.
$04	$44	Control Call. Cette commande effectue un contrôle du Device en ligne. L'information sera d'ordre général ou spécifique à un type de Device. En présence d'un Device de marque AppleDisk 3,5 pouces, cette commande éjecte le disque qui se trouve dans le drive.
$05	$45	Init Call. Effectue une remise à zéro du SmartPort en initialisant les valeurs standard (Reset SmartPort).
$07	$47	Close Call. Cette commande est étendue à un Character Device lors de séquences d'écriture ou de lecture. En présence d'une imprimante, il sera possible d'éditer les données implantées dans le tampon mémoire réservé au Device. Commande invalide envers un Block Device.
$08	$48	Read Call. Cette commande effectue la lecture des bytes, du Device vers le tampon mémoire dont le nombre est spécifié par le paramètre Byte_Count.
$09	$49	Write Call. Cette commande effectue l'écriture des bytes, du tampon mémoire vers le Device en ligne, dont le nombre est spécifié par le paramètre Byte_Count.

PROTOCOLES RELATIFS AU FIRMWARE

L'identification de certains bytes contenus dans les programmes de la ROM résidente (firmware) permet de reconnaître un type de machine parmi la gamme des Apple *II* existants. Le tableau suivant donne un aperçu des différents types de machine actuellement sur le marché, avec le byte ID permettant une identification correcte.

Type	Main ID ($FBB3)	SUB ID1 ($FBC0)	SUB ID2 ($FBBF)
II	$38	$60	$2F
II+	$EA	$EA	$EA
*II*e	$06	$EA	$C1
*II*e+	$06	$E0	$00
*II*GS	$06	$E0	$00
*II*c	$06	$00	$FF
*II*c+	$06	$00	$00

Remarque

On constate qu'il n'est pas possible de différencier le GS d'un Apple *II*e+ par le simple fait de vérifier le firmware, les différents bytes ID étant identiques. Par la mise en place de la nouvelle ROM du Moniteur étendu, Apple Computer a remédié au problème. A partir de l'adresse $FE1F débute une routine d'identification (IDROUTINE), permettant, grâce à la valeur retournée par certains registres, d'identifier le type de machine référencé par la commande.

Exemple d'identification de la machine

– Appel de la routine IDROUTINE

```
SEC             ;Carry = 1, débute de la routine
JSR $FE1F       ;Cette adresse contient un RTS en présence d'une version
                 antérieure à celle de l'Apple IIGS . Si c'est un GS, l'a-
                 dresse $FE1F contient le premier code exécutable de la
                 routine d'identification implantée dans sa ROM Moniteur.
BCS APPLE II    ;Si C = 1, c'est un ancien modèle Apple II, et les registres
                 A, X et Y restent inchangés.
BCC APPLE II    ;Si C = 0, c'est un Apple IIGS ou un modèle à venir. L'ac-
                 cumulateur C (registres A et B) et les registres d'index X
                 et Y seront, au retour de la routine IDROUTINE, chargés
                 avec des valeurs en conséquence.
```

Tableau des valeurs retournées au retour de IDROUTINE

Registre	Bits	Information retournée
A	7	Réservé, donc = 0.
	6	= 1 si une mémoire d'extension existe.
	5	= 1 si une interface IWM est présente.
	4	= 1 si une horloge en temps réel existe.
	3	= 1 si un Bus du type DeskTop existe.
	2	= 1 si un équipement SCC existe.
	1	= 1 si un slot d'extension est présent.
	0	= 1 si un port intégré est présent.
B	15-8	Réservés, donc = 0.
Y	15-8	Réservés, donc = 0.
	7-0	Identification machine : $00 = Apple IIGS . $01-$FF = Application future.
X	15-8	Réservés, donc = 0.
	7-0	Numéro de version de la ROM.

Remarque

Si l'identification de la machine est transmise en mode émulation, uniquement le contenu de poids faible des registres C, X et Y sera retourné, avec une Carry correcte. Si l'identification est transmise en mode natif, le format de la Carry sera le même, tandis que le contenu des registres sera retourné sur 16 bits. Le bit c, ou Carry, représente le bit de la retenue du registre d'état P.

VECTEURS PARTICULIERS SOUS PRODOS

Le démarrage dans le système d'exploitation n'est pas à confondre avec le processus de boot d'un disque en ligne et connecté à une interface d'un slot d'extension.

- La phase du boot consiste à démarrer une séquence de chargement, en exécutant le premier code machine d'un sous-programme implanté en mémoire et se trouvant initialement sur le disque. C'est le BootStrap intégré dans la ROM de la carte contrôleur qui se charge du processus initial. Il détermine par la suite le mode d'emploi destiné à effectuer le chargement du système d'exploitation.

- Le démarrage dans le système consiste à déclencher un mécanisme destiné à placer l'utilisateur soit dans le système d'exploitation courant (WarmStart), soit dans un système d'exploitation nouveau (ColdStart).

Le GS, comme tout Apple *II* doté d'une ROM autostart, permet deux modes de démarrage dans le système ProDOS : le démarrage à froid (ColdStart) et le démarrage à chaud (WarmStart).

- Un démarrage à froid purge les données présentes dans la mémoire vive (RAM), charge la routine de chargement des blocs $00 et $01 du disque et débute la mise en place du système ProDOS par exemple. La routine de chargement, communément appelée Loader, n'est pas à confondre avec le System Loader qui représente un ensemble de routines permettant de charger les programmes à leurs segments respectifs.

- Un démarrage à chaud dans le système arrête le cours de l'exécution du programme et place la machine dans le Basic AppleSoft. Dans ce cas, le programme implanté en mémoire reste intact et prêt à une autre utilisation.

Démarrage à chaud ou à froid du système?

Sous le système DOS 3.3, deux vecteurs d'entrées dans le système d'exploitation sont disponibles : le démarrage à chaud et le démarrage à froid. Chacun de ces vecteurs a son pointeur mémorisé dans la page 3, respectivement aux adresses $03D0 et $03D3. Le programmeur sollicite très souvent l'un ou l'autre de ces pointeurs, ne serait-ce que pour exploiter certains mécanismes de la programmation (appel à RWTS, table IOB, revectorisation du DOS, RESET, etc.)

Vecteurs de la page 3 sous DOS 3.3

$03D0- 4C BF 9D	JMP $9DBF	Redémarrage à chaud
$03D3- 4C 84 9D	JMP $9D84	Démarrage à froid

DEMARRAGE A FROID SOUS PRODOS

Sous DOS 3.3, le démarrage à froid s'effectue par la commande 3D3G formulée à partir du Moniteur. Elle annule toutes les variables qui se trouvent à cet instant en mémoire centrale et purge par la même occasion tout programme AppleSoft résident. Avec un démarrage à chaud, commande 3D0G, les variables et programmes restent en place et restent utilisables.

Sous le système d'exploitation ProDOS, Apple *II*GS sous tension, cette notion de démarrage à froid du système a totalement disparu pour laisser place à un compromis. Le seul démarrage à froid s'effectue au moment du boot du disque, Apple mis hors tension au préalable.

DEMARRAGE A CHAUD SOUS PRODOS

Sous ProDOS, un démarrage à chaud préserve tout programme AppleSoft résidant encore en mémoire vive, mais purge par contre ses variables. Ce processus est dicté par le saut inconditionnel, JMP $D665, qui se trouve en place à partir de l'adresse $ABF1 ($AC35 avec la version 1.1.1). Ce vecteur est sollicité lors d'un démarrage à chaud. L'adresse $D665 correspond à l'entrée dans la routine CLEAR de l'interpréteur AppleSoft. La notion de démarrage à froid et à chaud diffère totalement sous ProDOS.

- Le démarrage à froid initial n'est transmissible qu'à la mise sous tension de la configuration (ROM autostart) et uniquement à partir du drive 1.

- Le démarrage à chaud, processus intermédiaire par rapport au DOS 3.3, peut s'appliquer indifféremment à l'ordre établi des drives et des slots.

Vecteurs de la page 3 sous ProDOS

- Commande 3D0G ou 3D3G. Appel du pointeur se trouvant à cette adresse, en l'occurrence $BE00.

$03D0- 4C 00 BE	JMP $BE00	Démarrage à chaud
$03D3- 4C 00 BE	JMP $BE00	Démarrage à froid

- En $BE00 se trouve un saut à l'adresse $ABF1 du BASIC.SYSTEM (WarmStart).

$BE00- 4C F1 AB	JMP $ABF1	Connect Entry

- En $ABF1 se trouve un saut à l'adresse $D665 de la ROM AppleSoft (Routine CLEAR).

$ABF1- 20 65 D6	JSR $D665	AppleSoft CLEAR Routine

- A partir de l'adresse $D665, l'interpréteur AppleSoft effectue une purge des variables qui se trouvent dans la zone des variables par l'instruction CLEAR.

Touches CTRL-Y et RESET

La combinaison des touches Ctrl-Y, tapées à partir du moniteur, effectue un démarrage à chaud dans le système et efface ainsi toute variable implantée dans la mémoire vive. Par ailleurs, la combinaison des touches Ctrl-Reset, en plus de la propriété de purger les variables (démarrage à chaud), exécute un sous-programme qui gèle le prompt jusqu'à ce qu'une autre touche soit activée.

Commande de l'ampersand (&)

Lorsque l'ampersand est associé à un retour-chariot, le système effectue un saut à l'adresse $03F5 qui contient normalement un JMP $BE03. A cette adresse se trouve la première commande du BASIC.SYSTEM, qui effectue un JMP $A677 (JMP $A6B4 avec la version 1.1.1). Le résultat se traduit par une revectorisation de certains paramètres du Basic et rien de visible ne se passe.

Lorsque l'ampersand est associé à une autre commande non définie au préalable dans la table des vecteurs de la page 3, par exemple une commande ProDOS ou Basic, le système retourne un message d'erreur:

?SYNTAX ERROR

L'ampersand garde les mêmes propriétés que sous une exploitation du DOS 3.3, à savoir la possibilité de redéfinir le pointeur de la page 3 en modifiant au préalable le contenu des deux adresses $03F6 et $03F7.

Application simplifiée

Purge de la fenêtre de texte en tapant &
$03F5- 4C 58 FC JMP $FC58
En tapant, à partir du clavier, le & + Return, le système effectue un saut à l'adresse $03F5. Celle-ci contient la valeur 4C qui est le code machine de l'instruction du saut inconditionnel (JMP). L'interpréteur branche alors à l'adresse qui le suit, en l'occurrence $FC58. Le sous-programme qui débute à l'adresse $FC58 de la ROM Monitor efface l'écran dans les limites de la fenêtre de texte et place le curseur en haut et à gauche. C'est l'équivalent du traditionnel HOME en Basic AppleSoft.

Saut indirect à l'adresse $03FE

Lorsque ProDOS se trouve en mémoire et qu'une routine d'interruption est validée, un saut est effectué à l'adresse $03FE de la page 3. A cette adresse se trouve le pointeur du vecteur Interrupt Entry de la page globale de ProDOS: normalement EB BF (pointeur $BFEB). A l'adresse $BFEB débute un sous-programme de commutation du banc 1 de la carte langage. Par la suite un saut est réalisé à l'adresse $D13A. Une fois l'exécution aboutie, le message suivant s'affiche à l'écran:

INSERT SYSTEM DISK AND RESTART -ERR xx

où,

les xx représentent le code de l'erreur rencontrée. Le système boucle maintenant et seul un RESET pourra débloquer la situation:

```
D239- 4C 39 D2     JMP $D239        Boucle jusqu'au prochain RESET
```

Interrupt Entry Vector

```
BFEB- 2C 8B C0     BIT $C08B        Sélectionne la RAM en lecture
BFEE- 2C 8B C0     BIT $C08B        et en écriture.
BFF1- 4C 3A D1     JMP $D13A        Saut dans le MLI – IRQ Handler
```

Commandes et vecteurs d'entrées/sorties

Sous environnement ProDOS, les commandes PR£, IN£ et FF59G déconnectent les vecteurs d'entrées/sorties, tandis que le traditionnel CALL 1002 (3EAG) semble inerte et sans effet. Il existe néanmoins une parade à cette défaillance, en remplaçant l'ancienne commande par un CALL 39447 (9A17G).
Cette astuce rebranche à nouveau le système Basic grâce au sous-programme se trouvant à l'adresse $9A17 (Connect) qui devient désormais le vecteur de reconnexion.

Il existe une autre possibilité: réutiliser l'ancien vecteur se trouvant à l'adresse $03EA et inutilisé après le boot de la disquette. Il suffit simplement d'y placer un saut à l'adresse $9A17 (versions de ProDOS antérieures à la version 1.1.1).

REVECTORISATION DES ENTREES/SORTIES

```
$03EA- 4C 17 9A    JMP $9A17        Connect Entry
```

où,

4C représente le code machine du JMP, et les valeurs 17/9A sont respectivement les octets de poids faible et de poids fort de l'adresse $9A17. Après la modification, il suffit simplement de reconnecter le BASIC.SYSTEM par l'ancien CALL 1002 (3EAG).

Entry ProDOS Global Page (ProDOS8)

```
BE00- 4C F1 AB     JMP $ABF1        Démarrage à chaud – Warmstart
```

DEMARRAGE A CHAUD DANS LE SYSTEME

```
ABF1- 20 65 D6     JSR $D665        AppleSoft CLEAR.
ABF4- 20 17 9A     JSR $9A17        Reconnecte le système,
ABF7- A9 00        LDA £$00         Charge l'accumulateur,
ABF9- 85 24        STA $24          Annule HTAB.
ABFB- 4C 3F D4     JMP $D43F        ROM AppleSoft – Warmstart
```

Remarques

• Sous le système d'exploitation DOS 3.3, l'entrée du démarrage à chaud dans l'interpréteur AppleSoft se trouve à l'adresse $D43C et contient un JSR $DAFB. A cet emplacement de la ROM AppleSoft débute le sous-programme d'affichage du retour-chariot : cela se traduit à l'écran par un saut de ligne.

• ProDOS MLI n'utilise plus ce saut de ligne traditionnel avant l'ordre transmis, mais se connecte généralement à l'entrée directe des différentes routines de la ROM AppleSoft.

BOOTSTRAP ET CONVENTIONS

Le BootStrap, à ne pas confondre avec le démarrage dans le système ProDOS (ColdStart ou WarmStart), effectue la séquence initiale du boot transmis à un disque placé dans un drive courant. L'exécution du boot est indépendante du système d'exploitation à charger en mémoire. C'est le secteur physique $00 de la piste $00 du disque référencé qui est lu en premier par la routine de démarrage initial. Cette routine fait partie de la ROM résidente de l'interface qui pilote le drive qui lui est raccordé.

Remarques

• Pour un système ProDOS, l'occupation d'un slot quelconque par une interface Block Device est autorisée. Cette affirmation n'est valide que si cette structure n'affecte pas un autre port.

• Pour les systèmes d'exploitation CP/M et Pascal, le slot 6 est par convention le slot où devra se réaliser le processus du boot.

Vecteurs typiques de démarrage

AppleSoft sans système d'exploitation	$E003
DOS 3.3 (64 Ko)	$9DBF
ProDOS (lors du Boot)	$FF69
ProDOS (après le Boot)	$FF59
BASIC.SYSTEM	$BE00

Boot sous tension et hors tension

Le protocole du boot, mécanisme intimement lié à la carte contrôleur, diffère selon que le micro-ordinateur se trouve sous tension ('on') ou non ('off'). Il est sous-entendu que l'Apple *II*GS est équipé d'une ROM autostart.

BOOT A LA MISE SOUS TENSION

A la mise en route de la configuration par la commutation du bouton marche dans la position 'on', la ROM autostart exécute un sous-programme interne et recherche un drive en ligne suivant un protocole défini au préalable.

• **Apple hors tension**

Si pour une raison indéterminée, l'octet de test n'est pas celui attendu, le sous-programme résident du moniteur considère alors que le micro-ordinateur vient juste d'être mis sous tension.

La commande HOME ($FC58) est exécutée et le texte Apple *II*GS affiché en haut et au centre de l'écran. Un périphérique du type Block Device est recherché en ligne. L'occupation des slots est testée dans un ordre déterminé par le tableau de bord. Si un slot est occupé par une carte interface contrôleur, le contenu des adresses $Cs01, $Cs03, $Cs05 et $Cs07 de la carte contrôleur est testé dans l'ordre établi avec les valeurs $20, $00, $03 et $3C pour un disque 5,25 pouces, la dernière valeur étant $00 pour un disque 3,5 pouces. Si le résultat du test est positif, saut à l'adresse $Cs00, où s représente le numéro du slot concerné. Dans le cas contraire, le prochain slot sera recherché, et le même cycle recommence.
Si, durant le processus de recherche, aucun des slots n'est occupé par une carte contrôleur, ou pas de Block Device connecté, un démarrage à froid dans la ROM AppleSoft sera transmis et le sous-programme à l'adresse $E000 exécuté.

BOOT, MACHINE SOUS TENSION

• **Apple sous tension**

Si le GS reçoit un signal RESET transmis par la frappe des touches Pomme-ouverte, Control et Reset, puis relâchement de la touche Reset, le système effectue un saut à l'adresse $FA62 et exécute le sous-programme résident. Après diverses commutations et initialisations des vecteurs de la mé0004moire (mode TEXT, mode NORMAL, PR£, etc.), le pointeur mémorisé aux adresses $03F2-$03F3 de la page 3 est chargé et un saut est effectué à cette adresse. Un test de l'octet de validité, Power-Up byte de valeur $1B sous ProDOS ($03F4), est effectué au préalable. Cet octet est calculé en effectuant un OR (OU exclusif) du second octet du vecteur de réinitialisation, avec comme valeur constante $A5. Lors de chaque séquence de RESET, l'Apple utilise cet octet pour déterminer si le vecteur de réinitialisation est encore valide.

STRUCTURE & BOOT D'UNE DISQUETTE ProDOS – ProDOS 1.1.1., ProDOS8 ET ProDOS16

ProDOS 1.1.1., ProDOS8 et ProDOS16 désignent le système d'exploitation ProDOS sous ses différentes versions et formes possibles.

- ProDOS 1.1.1. représente le système standard des Apple *II*, *IIe*, *IIe+* et Apple *IIc*. Il ne traite que des mots de 8 bits et s'exécute tout aussi bien sur le GS, émulant alors un Apple *IIe*.

- ProDOS8 est la nouvelle version du système d'exploitation, ne traitant que des mots de 8 bits. Comme la version 1.1.1, ProDOS8 est bootable sur le GS, destiné essentiellement à maintenir une compatibilité relative avec la bibliothèque des programmes existants. ProDOS8 est compatible avec les instructions destinées aux microprocesseurs 6502 et 65C02. Il peut occuper un disque 5,25 pouces.

- ProDOS16 est le nouveau système d'exploitation destiné à traiter des mots de 16 bits de large, réservé exclusivement aux applications spécifiques du microprocesseur 65C816. Un système ProDOS complet ne pourra trouver place sur un disque 5,25 pouces (140 Ko), mais nécessite un lecteur AppleDisk 3,5 pouces d'une capacité de 800 Ko formatés.

Chaque système se laisse facilement configurer sur un disque selon le programme que l'on désire exécuter. Le support pourra tout aussi bien être un disque 5,25 pouces, 3,5 pouces ou encore un disque dur. Chaque configuration ProDOS sera adaptée en fonction de l'application à exécuter. Le système ProDOS, matérialisé sous la forme de plusieurs fichiers système, n'occupe pas de place précise sur le disque. Les commandes résidentes dépendent directement de la version courante.

DIFFERENTES STRUCTURES POSSIBLES

ProDOS, système d'exploitation performant, ne possède pas la même structure selon qu'il est destiné à traiter des programmes écrits pour des applications de 8 ou 16 bits (processeur 6502 ou 65C816). Dans le premier cas, structure 8 bits, deux configurations sont possibles: la version ProDOS 1.1.1 ou la version ProDOS8 regroupant tous les fichiers système nécessaires. Dans le cas de l'Apple *II*GS , et en mode natif, la structure ProDOS16 (16 bits) sera requise, le mode émulation faisant appel à ProDOS8.

Le raisonnement développé par la suite s'applique à tout périphérique du type Block Device, qu'il soit au format 5,25 ou 3,5 pouces, ou à tout disque dur partitionné au format ProDOS. Chaque structure est abordée en détail, dotant le lecteur de tous les moyens pour aborder par la suite n'importe quelle structure ProDOS. Tout cas particulier susceptible d'être rencontré sera mentionné en temps utile.

RAM-Disk

Lorsque ProDOS8 détecte une carte 80 colonnes étendue sur une configuration antérieure à celle du GS, il crée un disque virtuel dans l'espace mémoire auxiliaires des 64 Ko. Seulement 61 Ko seront utilisables par l'intermédiaire du préfixe /RAM. Ce volume de structure interne sera considéré comme étant situé au slot 3 drive 2 et pourra être appelé de deux manières.

APPELS DU PSEUDO-DISK (RAM-DISK)

Deux syntaxes sont disponibles à l'utilisateur:

- PREFIX/RAM

où,

PREFIX est la commande ProDOS
/RAM est la commande d'appel du disque virtuel.

- CAT,S3,D2

où,

CAT est la commande ProDOS
S3, désigne le paramètre slot 3
D2, désigne le paramètre drive 2.

C'est un pseudo-disque, dont le temps d'accès est beaucoup plus rapide, permettant la cohabitation de plusieurs programmes implantés en mémoire auxiliaire. Les donnés sont maintenues en mémoire tant que l'Apple reste sous tension; dans le cas contraire, tous les programmes et fichiers sont définitivement perdus. Un boot à froid vous fera perdre le bénéfice des programmes contenus dans le Directory de la RAM.

VOLUME DU RAM-DISK

Le Ram-disque ainsi créé en mémoire auxiliaire se compose de 127 blocs de 512 octets chacun, dotés de la structure suivante:

Blocs 0-1	Non disponibles
Bloc 2	Table des matières principale (Volume Directory) pour 12 fichiers.
Bloc 3	Volume Bit Map: table d'occupation des blocs.
Blocs 4-7	Non disponibles.
Blocs 8-126	Fichiers SubDirectory et fichiers programmes.

ROM-Disk

Le ROM-Disk est une structure mise en place et accessible à travers le système d'exploitation ProDOS16. Le ROM-Disk se trouve matérialisé sous forme de mémoire morte (ROM) implantée sur une carte interface.

INSTALLATION DU ROM-DISK

La zone d'adresses réservée au ROM-Disk d'une carte d'extension débute au vecteur $F0/0000. La zone mémoire comprise entre les adresses $F0/0000 et $F7/FFFF incluses peut être exploitée normalement par une carte d'extension de la ROM.

Pendant la phase du BootStrap transmis à travers le SmartPort, une routine résidente recherche en ligne une éventuelle carte d'extension de la ROM. Si une telle interface est détectée et reconnue, le SmartPort lui assigne un numéro à travers le paramètre Unit_Number. Si la chaîne de caractères ROMDISK est lue en début des adresses de la carte d'extension, l'initialisation du ROM-Disk est exécutée, via son entrée réelle (Dispatch). Si l'exécution aboutit normalement, le ROM-Disk Driver sera chaîné au SmartPort. Si une erreur est détectée pendant la recherche, le ROM-Disk sera déclaré inexistant.

Protocoles d'une ROM d'extension

- L'entrée de la carte interface ($F0/0000-0006), adresse de base de la routine pilote (Driver), devra contenir la variable ROMDISK. Les codes ASCII utilisés pour l'écriture de la chaîne de caractères seront au format majuscule (MSB 'on').

- Le point d'entrée de la routine pilote (Driver) se fera à partir de l'adresse $F0/0007.

PASSAGE DES COMMANDES AU ROM-DISK

Le passage de paramètres au ROM-Disk s'effectue à travers la page zéro. La commande sera du type étendue, même structure que pour la gestion du SmartPort (voir chapitre 3).

Les adresses $0040 à $0062 de la page zéro, adressage absolu page zéro, sont transmis à travers le SmartPort au ROM-Disk en ligne.

Adr.	Paramètre transmis	Type de commande
$42	Adresse du tampon mémoire (bits 0-7)	Toute commande
$43	Adresse du tampon mémoire (bits 8-15)	Toute commande
$44	Adresse du tampon mémoire (bits 16-23)	Toute commande
$45	Numéro de la commande	Toute commande
$46	Paramètre Parameter_Count	Toute commande
$47	Adresse du tampon mémoire (bits 24-31)	Toute commande
$48	Extension (Block) bits 0-7 Status Code ou Control Code Paramètre Byte_Count (bits 0-7)	Read Block & Write Block Status & Control Read & Write
$49	Extension (Block) bits 8-15 Paramètre Byte_Count (bits 8-15)	Read Block & Write Block Read & Write
$4A	Extension (Block) bits 16-23 Paramètre Address_Pointer (bits 0-7)	Read Block & Write Block Read & Write
$4B	Extension (Block) bits 24-31 Paramètre Address_Pointer (bits 8-15)	Read Block & Write Block Read & Write
$4C	Paramètre Address_Pointer (bits 16-23)	Read & Write
$4D	Paramètre Address_Pointer (bits 24-31)	Read & Write

Remarque

Au retour de la commande, le ROM-Disk retourne certains paramètres dans la page zéro (adressage direct page zéro). La syntaxe de ces paramètres est la suivante :

$00/0050-	Code de l'erreur éventuel.
$00/0051-	LByte, nombre de bytes réellement transférés.
$00/0052-	HByte, nombre de bytes réellement transférés.

Accès au disque en général

L'accès au disque détermine le mode d'emploi de lecture et d'écriture de celui-ci. C'est généralement une routine interne au système qui gère les contraintes d'accès au disque, aussi bien lors du formatage, que de la lecture ou de l'écriture. Ce procédé est largement utilisé par les concepteurs de logiciels qui tentent, souvent en vain, de protéger leurs produits contre le piratage informatique. Ce mode entraîne la mise en place d'une routine introduisant une contrainte au niveau du temps : chaque bit doit être écrit en quatre cycles-machine, ce qui fait 32 cycles pour un octet normal et 40 cycles pour un octet de synchronisation. Il devient alors impératif de compter le nombre de cycles de chaque instruction et tout dérapage dans ce domaine rendra la disquette inutilisable. Néanmoins, il existe un certain nombre d'utilitaires facilitant la mise au point à ce niveau de programmation. Les assembleurs, ProCODE par exemple, permettant de gagner du temps dans ce mode de programmation en regroupant les bytes synchro dans des macro-instructions de temporisation.

REPERTOIRE DES PISTES & SECTEURS (BLOCS) TRACK BLOCK LIST

Sous DOS 3.3, un secteur était systématiquement alloué lors de la sauvegarde d'un fichier pour former la Track Sector List ou répertoire des pistes et secteurs alloués à ce fichier. Sous système ProDOS, certains termes sont modifiés et l'organisation du disque s'en trouve modifiée. La Track Block List désigne la table d'allocation des blocs du fichier.

ALLOCATION DES BLOCS (BLOCS INDEX)

- Un fichier programme ou Volume SubDirectory (table des matières auxiliaire) de moins de 512 bytes, soit 1 bloc, est directement accessible.

- Un fichier d'une taille comprise entre 2 et 256 blocs nécessite un bloc pour le répertoire (Blocs Index).

- Un fichier d'une taille supérieure à 256 blocs utilise un bloc pour le répertoire principal (Master Index Block). Celui-ci pointe le premier bloc d'un répertoire auxiliaire (Blocs Index) d'une capacité limitée à 128 blocs. Chacun de ces blocs du répertoire auxiliaire peut à nouveau pointer 256 blocs de données.

NIBBLES BRUTS

Les nibbles bruts correspondent à l'information telle qu'elle est encodée sur une disquette, conséquence des restrictions internes du système d'exploitation. En plus de la transcription codée des données, les nibbles renferment une masse d'informations qui a pour seul rôle d'aider ProDOS dans son travail. Ce sont des marques et repères précieux pour le programmeur (décryptage d'une disquette) et indispensables à ProDOS pour traiter les données. En transposant des nibbles bruts, on peut lire uniquement l'image codée des données, telle qu'elle a été sauvegardée sur la piste sélectionnée en lecture.

ROUTINES READ_BLOCK ET WRITE_BLOCK

Les commandes READ_BLOCK et WRITE_BLOCK sollicitent à travers le MLI des routines chargées de lire et d'écrire les données au niveau du support physique (pistes & secteurs). On peut comparer ces routines à un traducteur effectuant le thème (écriture encodée) sur la disquette et la version (retour du programme décodé), implantée dans la mémoire centrale. READ_BLOCK est un filtre que agit uniquement dans le sens passage des données, de la disquette vers la mémoire centrale; il élimine les marques et repères devenus inutiles. ProDOS permet donc de retrouver les valeurs d'un programme originel, en filtrant les nibbles brut, pour laisser paraître les seuls bytes réels.

FORMAT LOGIQUE

C'est l'organisation interne des pistes et des secteurs, structure propre au système d'exploitation qui l'utilise. C'est une routine qui a comme fonction première de rendre transparente à l'utilisateur l'organisation des blocs éparpillés sur le disque. Un bloc (Block) est une unité de mesure spécifique au système d'exploitation ProDOS et regroupe deux secteurs physiques du disque.

SKEWING

Skewing est l'ordre dans lequel s'effectue le déplacement latéral de la tête de lecture d'un drive, au-dessus des pistes et des secteurs physiques. Les secteurs sur un disque sont formatés en pistes physiques, dans l'ordre $00-$0F (0 à 15 = 16 secteurs). La routine MLI, Disk-Driver de ProDOS, utilise la méthode **Ascending Skew 2. Ascending Skew** signifie que la tête de lecture se déplace dans le sens ascendant croissant de la numérotation des secteurs. Le 2 qui suit Skew fixe la valeur logique des secteurs traités par paires. Deux secteurs physiques sont égaux à un bloc logique et seize secteurs physiques à huit blocs logiques.

Sous ProDOS, les secteurs logiques d'une piste sont traités dans le sens ascendant : 0, 1, 2, 3, etc.; DOS 3.3 traite les secteurs dans le sens descendant : 15, 14, 13, etc. (Descending Skew 1).

DISPOSITION DES SECTEURS SUR UNE PISTE

– Sous DOS 3.3

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Secteur physique
00	0D	0B	09	07	05	03	01	0E	0C	0A	08	06	04	02	0F	Secteur logique

– Sous ProDOS

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Secteur physique
00	02	04	06	08	0A	0C	0E	01	03	05	07	09	0B	0D	0F	Secteur logique
G0	D0	G1	D1	G2	D2	G3	D3	G4	D4	G5	D5	G6	D6	G7	D7	Bloc logique

Légende :

G0 = bloc partie gauche secteur 0;
D0 = bloc partie droite secteur 0;

Remarque

Le tableau montre la disposition des secteurs et nous apprend que les seuls secteurs qui correspondent dans les modes physique et logique avec les systèmes DOS et ProDOS sont les secteurs $00 et $0F.

Organisation typique d'un disque ProDOS

Les systèmes d'exploitation ProDOS 1.1.1, ProDOS8 et ProDOS16 utilisent une table d'allocation des blocs, la volume bit map, pour gérer l'occupation des blocs au niveau du disque. A ne pas confondre avec la système bit map qui gère, quant à elle, l'espace mémoire de la RAM (page globale de ProDOS, adresses $BF58-$BF6F). Avec un Apple *IIGS* et ProDOS16 booté en mémoire, la Global Page n'est plus exploitée, la gestion de la mémoire étant prise en charge par l'outil Memory Manager qui se trouve en ROM résidente. Le Memory Manager, outil numéro 2, a pour mission de prendre en charge toute la gestion de la mémoire du GS. Il est à noter que ce dernier est conçu pour faire cohabiter plusieurs applications simultanément en mémoire et de les utiliser à tour de rôle en passant de l'une à l'autre (à ne pas confondre avec multitâches).

VOLUME BIT MAP

C'est la table d'allocation des blocs d'une disquette. Elle débute toujours au bloc 6 et se limite à un bloc pour les disques 3,5 et 5,25 pouces. En présence d'un disque dur compatible, l'extension de la bit map s'effectue en fonction de la partition allouée à ProDOS. Néanmoins, ProDOS fixe la limite de ses possibilités dans la taille d'un Volume à la barre des 32 Mo en allouant un maximum de 128 blocs à la bit map. Cette structure permet alors de gérer correctement 65 536 blocs de données.

- Un disque 5,25 pouces (35 pistes, 140 Ko) utilise les 35 premiers bytes de la volume bit map pour gérer l'occupation de ses blocs. Chaque byte de la table marque 8 blocs, ce qui fait un total de 280 blocs (35 * 8).

- Un disque 3,5 pouces (200 pistes, 800 Ko) utilise les 200 premiers bytes de la volume bit map pour gérer l'occupation de ses blocs. Chaque byte marque 8 blocs, ce qui fait un total de 1600 blocs disponibles (200 * 8).

- Un disque dur, partitionné en fonction de la configuration allouée à ProDOS, permet d'étendre considérablement les possibilités de sauvegarde de masse. La volume bit map est structurée en fonction du Volume alloué à ProDOS, mais limite à 65 536 blocs (32 Mo) le maximum de blocs pouvant être sauvegardés sur le disque. Un maximum de 128 blocs pourra être alloué à la volume bit map du disque dur.

OCCUPATION DE LA VOLUME BIT MAP

La table d'allocations de la volume bit map gère l'occupation des blocs du disque. Chaque bit d'un octet représentatif dans cette table représente un bloc : si le bit est positionné à 1, le bloc est libre; si par contre le bit est à 0, le bloc est occupé.

Table d'allocation des blocs

Byte $00 = piste $00 = blocs $000-$007
Byte $01 = piste $01 = blocs $008-$00F
Byte $02 = piste $02 = blocs $010-$017
.....
.....
.....
Taille en fonction du Device en ligne

Représentation d'un byte de la volume BitMap

Byte $00

blocs $000-$007 0 1 2 3 4 5 6 7
0 0 0 0 0 0 0 1 = 01

Le bloc $007 est libre et les blocs $000-$006 occupés.

Byte $01

blocs $008-$00F 8 9 A B C D E F
1 1 1 1 1 1 1 1 = FF

Les blocs $008-$00F sont tous libres.

DUMP DE LA VOLUME BIT MAP

Le dump de la volume Bit Map est celui d'une disquette 5,25 pouces venant juste d'être formatée. En ce qui concerne le format des autres Devices en présence, le raisonnement reste identique, uniquement les valeurs marquées libres changent.

Structure typique

Blocs 0-1	Loader. Routine de chargement du système
Bloc 2	Volume Directory – Key-Block -
Blocs 3-5	Volume Directory – Non Key-Block -
Bloc 6	Volume Bit Map
Blocs 7-xxx	Libres

où,

les xxx représentent le nombre limite des blocs alloués au Device courant. Cette valeur représente 279 pour un disque 5,25 pouces; 1599 pour un disque 3,5 pouces et selon la taille allouée à la partition pour un disque dur.

Pour un disque 5,25 pouces, la commande CAT affiche les paramètres suivants:
/VOLUME

NAME TYPE BLOCKS MODIFIED

BLOCKS FREE : 273 BLOCKS USED : 7

où,
/VOLUME désigne le nom du volume du disque
NAME, TYPE, BLOCKS et MODIFIED sont des rubriques vides pour le moment.

Remarque

7 blocs sont occupés pour l'instant et 273 blocs sont libres. La conversion en format physique nous fait apparaître 14 secteurs occupés et 546 secteurs libres. Le total de 560 secteurs correspond à la capacité d'un disque standard de 35 pistes.

Dump de la bit map d'un disque ProDOS

Track	$00/ Sector $03/ Block 00 06															
$00	01	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
$10	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
$20	FF	FF	FF	00	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	00	00	00	00
$40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

OCCUPATION TYPIQUE D'UN SUPPORT DISQUE

- Mise à zéro binaire de tous les blocs:

– $000-$117 ou 0-279 blocs pour un disque 5,25 pouces;
– $000-$640 ou 0-1600 blocs pour un disque 3,5 pouces;
– selon le partitionnage pour un disque dur.

- En présence de n'importe quel type de périphérique du type Block Device, les blocs $00 et $01 du disque contiennent le Loader, routine de chargement destinée à charger en mémoire le système d'exploitation ProDOS. Si la disquette est destinée à être utilisée comme disquette système, il faudra y copier un certain nombre de fichiers selon la configuration requise (ProDOS8 ou ProDOS16 ?)

- Les blocs 2 à 5 contiennent le Volume Directory, encore appelé table des matières principale. Les blocs sont chaînés entre eux par des pointeurs avant et arrière. Comme précédemment, la répartition de ces blocs est indépendante du type de Device présent (disque 5,25 ou 3,5 pouces, ou disque dur).

- Le bloc 6 contient la bit map ou table d'occupation des blocs; après le formatage de la disquette, les blocs 0 à 6 sont marqués occupés. Un bloc de la bit map peut gérer jusqu'à 4096 blocs d'un disque. Une bit map d'une taille d'un bloc est suffisante pour gérer les 280 blocs d'un disque standard de 5,25 pouces ou d'un disque de 3,5 pouces.

– pour un disque 5,25 pouces, les bytes $00 à $22 de la bit map sont utilisés;
– pour un disque 3,5 pouces, les bytes $00 à $C7 de la bit map sont utilisés;
– pour un disque dur, l'occupation est en fonction de la partition choisie. Comme exemple, un disque dur d'une taille de 20 Mo utilise pour sa bit map non moins de 6 blocs du disque (blocs $06 à $0B).

- Les blocs suivants sont destinés à des Volume SubDirectory et fichiers de données.

Allocation des blocs sur un disque

Après le formatage par un programme utilitaire, le disque présente un certain nombre de blocs occupés et de blocs libres.

Un disque est opérationnel après le formatage. A la fin de cette opération, le système effectue un certain nombre de manipulations : écriture du Loader, routine de chargement du système, sur les deux premiers blocs; constitution de la table des matières principale (Volume Directory) sur les blocs 2 à 5; écriture et mise à jour de la table d'occupation (bit map) sur le bloc 6 et retour au programme appelant.

Capacité d'un disque après formatage (5,25 pouces)

Blocs disponibles	273 blocs
Bytes disponibles	139 776 bytes

Remarque

A ce total il faudra retrancher le nombre de blocs nécessaires pour la sauvegarde des fichiers du système ProDOS.

ALLOCATIONS DES BLOCS SUR UN DISQUE 5,25 POUCES

Disque 5,25 pouces (140 Ko)
Blocs alloués par ProDOS

Blocs 0-1	<------- Loader. Routine de chargement ---------->
Blocs 2-5	<------- Volume Directory ---------->
Bloc 6	<------- Bit Map ---------->
Blocs 7-279	<------- Volume SubDirectory ---------->
	<------- Fichiers système ---------->
	<------- Utilisateur ---------->

Remarques

- Le bloc $02 est désigné sous l'appellation Key-Block et les blocs $03 à $05 Non Key-Block;
- La bit map gère un maximum de 35 pistes possibles ($00-$22).
- Les blocs $02 à $05, emplacement réservé au Directory, autorisent jusqu'à 52 entrées de noms, le premier étant réservé au nom du volume (disque) et les 51 autres à des noms de fichiers divers. Pour étendre la capacité de la table des matières principales (Directory), il est possible de créer des tables des matières auxiliaires (SubDirectory).

Capacité physique du disque 5,25 pouces

Nombre de face	1
Pistes	35 (0-34)
Secteurs par piste	16 (0-15)
Secteurs par disque	560
Bytes par secteur	256
Bytes par piste	4096
Bytes par disque	143.360 (140 Ko)

Capacité logique du disque 5,25 pouces

280 blocs par disque (0 à 279 ou $000-$117)
8 blocs par piste (0 à 7 ou $00-$07)
1 bloc = 2 secteurs
512 bytes par bloc (0 à 511 ou $000-1FF)
Un bloc se divise en deux parties : la partie gauche et la partie droite. Les 256 premiers bytes d'un bloc sont ceux de gauche, les 256 bytes suivants, ceux de droite.

ALLOCATIONS DES BLOCS SUR UN DISQUE 3,5 POUCES

Disque 3,5 pouces (800 Ko)
Blocs alloués par ProDOS

Blocs 0-1	<----- Loader. Routine de chargement ----->
Blocs 2-5	<----- Volume Directory ----->
Bloc 6	<----- Bit Map ----->
Blocs 7-1600	<----- Volume SubDirectory ----->
	<----- Fichiers système ----->
	<----- Utilisateur ----->

Remarques

- Le bloc $02 est désigné sous l'appellation Key-Block et les blocs $03 à $05 Non Key-Block.
- La bit map est étendue et gère un maximum de 200 pistes possibles ($00 à $C7).
- Les blocs $02 à $05, emplacement réservé au Directory, autorisent jusqu'à 52 entrées de noms, le premier étant réservé au nom du Volume (disque) et les 51 autres à des noms de fichiers divers. Pour étendre la capacité de la table des matières principale (Directory), il est possible de créer des tables de matières auxiliaires (SubDirectory).
- Les blocs disponibles à l'utilisateur s'étendent du bloc 007 au bloc 1600 (disque 800 Ko).

Capacité physique du disque 3,5 pouces

Nombre de faces	2
Pistes par face	100 (0-99)
Pistes totales	200 (0-199)
Secteurs par piste	16 (0-15)
Secteurs par disque	3 200
Bytes par secteurs	256
Bytes par piste	4 096
Bytes par disque	819 200 (800 Ko)

Capacité logique du disque 3,5 pouces

1 600 blocs par disque (0 à 1599 ou $000-$639)
8 blocs par piste (0 à 7 ou $00-07)
1 bloc = 2 secteurs
512 bytes par bloc (0 à 511 ou $000-$1FF)

Comme avec le disque 5,25 pouces, un bloc se divise en deux parties: la partie gauche et la partie droite. Les 256 premiers bytes d'un bloc sont ceux de gauche, les 256 bytes suivants, ceux de droite.

ALLOCATIONS DES BLOCS SUR UN DISQUE DUR

Disque dur (selon la partition)
Blocs alloués par ProDOS

Blocs 0-1 <----- Loader. Routine de chargement ----------->
Blocs 2-5 <----- Volume Directory ----------->
Bloc 6-11 <----- Bit Map ----------->
Blocs 7-xxxx <----- Volume SubDirectory ----------->
<----- Fichiers système ----------->
<----- Utilisateur ----------->

Remarques

- Les xxx représentent le nombre de blocs disponibles en fonction des partitions allouées au disque dur. En effet, la plupart des disques durs permettent de configurer la mémoire de masse disponible pour travailler avec plusieurs systèmes d'exploitation.

- Le bloc $02 est désigné sous l'appellation Key-Block et les blocs $03 à $05 Non Key-Block.

- La bit map est étendue et gère le nombre de blocs en fonction de la configuration des partitions allouées à chaque système d'exploitation.

- Les blocs $02 à $05, emplacement réservé au Directory, autorisent jusqu'à 52 entrées de noms, le premier étant réservé au nom du volume (disque) et les 51 autres à des noms de fichiers divers. Pour étendre la capacité de la table des matières principale (Directory), il est possible de créer des tables des matières auxiliaires (SubDirectory).

Capacité physique du disque dur

La capacité d'un disque dur est intimement liée aux caractéristiques de construction et à la structure de la partition allouée au système ProDOS. Dans ce contexte, il n'est pas possible d'avancer de chiffre de référence comme pour les disques 3,5 ou 5,25 pouces.

Capacité logique du disque 3,5 pouces

Comme pour la capacité physique du disque dur, l'allocation des blocs est en fonction de la partition et des caractéristiques mises en présence pour ce type de matériel.

- Comme pour les périphériques du type Block Device 5,25 et 3,5 pouces, le disque dur possède la même structure des blocs. Un bloc se divise en deux parties: la partie gauche et la partie droite. Les 256 premiers bytes d'un bloc sont ceux de gauche, les 256 bytes suivants, ceux de droite.

ProDOS 1.1.1., ProDOS8 ou ProDOS16?

La gamme des Apple *II* étant des plus variée, bon nombre d'utilisateurs supposent que ces machines sont totalement compatibles entre elles. Avec l'apparition du GS, cette confusion n'a fait que s'accentuer. Certains commerçants, appuyés par des développeurs sans scrupule, ont adopté ce créneau comme slogan de vente. La compatibilité n'est qu'une chose abstraite, et relative à chaque type de machine. Elle n'est mesurable qu'au niveau du microprocesseur équipant la machine considérée, avec la possibilité d'exécuter les instructions sous forme de codes machines. Ces codes, ou mnémoniques, différent très souvent d'un type de processeur à un autre, la gamme des Apple *II* n'échappant pas à la règle.

En mode émulation, le GS simule le fonctionnement d'un Apple *IIe* à travers un composant hybride nommé Mega *II*. L'utilisateur privilégié d'une telle machine possède, dans la version de base, tous les composants nécessaires au traitement des programmes écrits pour fonctionner avec des mots de 8 ou 16 bits. Par ailleurs, une compatibilité relative existe avec la bibliothèque des programmes Apple, compatibilité cependant ramenée à la seule compréhension du système mis en présence et de la machine. Il appartiendra à l'utilisateur d'avoir toujours la bonne structure ProDOS sur la disquette où se trouve le programme à exécuter: est-ce un programme écrit pour traiter des mots de 8 ou 16 bits? Ce paragraphe apporte certaines solutions pour guider le lecteur dans son choix.

STRUCTURE D'UN DISQUE SYSTEME STANDARD

Système ProDOS version 1.1.1 par exemple

Cette structure est celle diffusée au tout début de la sortie du système ProDOS, avec cependant une version récente (version 1.1.1). Une fois le disque formaté par l'utilitaire Filer par exemple, un certain nombre de blocs sont alors disponibles. Pour créer un disque système, l'utilisateur est tenu de configurer son support selon l'utilisation future. C'est ainsi qu'un disque ProDOS standard comporte au moins deux fichiers:

Contenu d'un disque ProDOS 1.1.1

- un fichier système Basic, généralement sauvegardé sous le nom de BASIC.SYSTEM. Il sert d'interface au programme d'application écrit en langage Basic pour transmettre des commandes à l'interpréteur AppleSoft qui se trouve en ROM résidente. Des programmes utilisant d'autres interfaces nécessitent alors des fichiers système adaptés pour la circonstance, mais dotés du suffixe .SYSTEM.

- un fichier système ProDOS, généralement sauvegardé sous le nom de ProDOS. Il contient les routines MLI et commandes ProDOS écrites en langage machine. Chaque version possède ses propres commandes, ces dernières n'étant pas toujours compatibles d'une version à une autre. Le fichier système est du type .SYS, uniquement bootable.

STRUCTURE D'UN DISQUE SYSTEME COMPLET

ProDOS8 et ProDOS16

L'environnement de l'Apple *II*GS dispose de capacités nouvelles et, de ce fait, utilise un système nouveau. Le disque système du GS contient les programmes système nécessaires à la bonne exécution des applications qui lui sont destinées.

L'organisation typique d'un tel disque système se compose essentiellement de programmes système, à savoir ProDOS8 et ProDOS16, le System Loader, les outils Tool Sets à charger en RAM, le patch pour transférer les outils résidant en ROM vers la RAM, les polices de caractères (Fonts), les accessoires de bureau (Desk Accesories), le Loader qui représente le programme d'initialisation du boot, et d'éventuels programmes d'applications diverses.

Deux versions de disques système sont possibles : un disque système complet et un disque système configuré selon l'exécution à transmettre (8 ou 16 bits programme?). Dans le premier cas, tous les fichiers système se trouvent dans la table des matières. En présence d'un disque d'application, uniquement les fichiers système requis sont sauvegardés dans le Directory.

Un disque système complet, doté de tous les fichiers système ProDOS8 et ProDOS16, nécessite une structure 800 Ko au niveau du Block Device (drive). Le format du support disque sera un disque micro-floppy, au format 3,5 pouces (135 tpi) et d'une capacité de sauvegarde de 800 Ko. Une disquette souple de 5,25 pouces (140 Ko) ne pourra pas contenir la totalité de la bibliothèque des programmes système.

Contenu d'un disque système complet

Directory/File	Signification
PRODOS	Ce fichier comporte les routines et sous-programmes pour charger par la suite le système d'exploitation approprié en fonction du programme à exécuter. Reboote le système sur un pilote au moment de quitter le programme en fin de traitement.
SYSTEM/	Fichier table des matières auxiliaire (SubDirectory), contenant les fichiers système et la sélection du programme d'application.
P8	Fichier système spécifique aux programmes développés pour traiter des mots de 8 bits.
P16	Fichier système spécifique aux programmes développés pour traiter des mots de 16 bits.
xxx.SYS16	SYSTEM LOADER, fichier de chargement du système de l'Apple *II*GS . Le nom du fichier possède le suffixe .SYS16.
START	Fichier standard de sélection du programme à exécuter.
LIBRAIRIE/	Fichier table des matières auxiliaire contenant les fichiers librairie nécessaires à ProDOS.
TOOLS/	Fichier SubDirectory, contenant tous les outils destinés à être chargés en mémoire vive.
FONTS/	Fichier SubDirectory, contenant toutes les polices de caractères exploitables sous ProDOS (Fonts).
DESK.ACCS/	Fichier SubDirectory, contenant tous les fichiers accessoires de bureau.
SYSTEM.SETUP/	Fichier SubDirectory, contenant les vecteurs d'initialisation des programmes à traiter. TOOL.SETUP, fichier contenant le patch et le sous-programme pour installer les outils de la ROM résidente. Ce fichier doit se trouver en tête dans le SubDirectory SYSTEM.SETUP/; il sera chargé avant tout autre fichier.
BASIC.SYSTEM	Fichier système, interface requise avec des applications faisant appel à l'interpréteur du Basic AppleSoft.

A propos de SYSTEM.SETUP/

C'est un fichier SubDirectory qui contient un certain nombre de types de fichiers. Lors de la phase du boot, ces divers fichiers ne sont pas chargés tous en même temps que le programme à exécuter, mais sont sollicités par certaines applications à un moment ou à un autre.

Contenu typique du SubDirectory SYSTEM.SETUP/

TOOL.SETUP

Ce fichier est indispensable, et sera recherché et chargé en premier dans la liste des fichiers présents dans cette table des matières auxiliaire. TOOL.SETUP installe et initialise le patch pour transférer les outils résidant en ROM (Tool Sets), dans la mémoire vive (RAM). Une fois l'exécution terminée, ProDOS16 charge et exécute la suite des fichiers contenus dans le SubDirectory SYSTEM.SETUP.

Permanent Init Files (FileType $B6)

Initialisation permanente de fichiers. Ce type de fichier est uniquement chargé et exécuté en association avec des applications standards (fichiers du type $B3). La fin de l'exécution en cours ne nécessite pas une gestion contrôlée pour le retour à un autre sous-programme.

Cette structure présente certaines caractéristiques :

- le fichier est chargé à une adresse mémoire quelconque;
- la pile accessible directement en adressage direct ne pourra pas être exploitée en permanence;
- l'abandon du programme en cours d'exécution devra se faire à travers un RTL (ReTurn from subroutine Long = retour long de sous-programme) au détriment de la commande MLI QUIT.

Temporary Init Files (FileType $B7)

Initialisation temporaire de fichiers. Comme pour les fichiers précédents, ce type de fichier est uniquement chargé et exécuté avec des applications standards (fichiers du type $B3). L'abandon du programme en cours d'exécution se fera à travers un RTL (ReTurn from subroutine Long = retour long de sous-programme) et non par la commande QUIT (MLI).

Accessoires de bureau additionnels (FileType $B8)

Ce type de fichier est chargé en mémoire et non exécuté. La mémoire allouée ne sera pas protégée contre l'écriture.

Accessoires de bureau standards (FileType $B9)

Ce type de fichier est chargé en mémoire et non exécuté. La mémoire allouée ne sera pas protégée contre l'écriture.

STRUCTURE D'UN DISQUE ProDOS16

Un disque système complet contient un certain nombre de programmes système pas toujours utiles pour n'importe quel programme à exécuter. Pour cette raison, il est souhaitable de pouvoir configurer un disque système selon les besoins actuels de l'utilisateur. Différentes applications nécessitent différentes structures de disque système.

Chaque application ou programme en général, nécessite pour son exécution un système d'exploitation, ne serait-ce que pour aller lire les données au niveau du disque. Un disque au format 5,25 pouces (140 Ko) ne pourra contenir que des applications du système ProDOS8. En effet, ce type de Block Device ne peut en aucun cas contenir la totalité des fichiers système de ProDOS16. En ce qui concerne l'exécution de toutes les applications écrites d'origine pour les processeurs 6502 et 65C02, ou pour fonctionner en mode émulation sur le GS, le système ProDOS8 sera de rigueur.

Contenu d'un disque ProDOS16

Directory/File	Signification
PRODOS	Fichier indispensable pour booter à partir d'un Apple *II*GS .
SYSTEM/	SubDirectory, structure requise pour le système à booter.
P16	Fichier système pour traiter des mots de 16 bits.
LOADER	Fichier de chargement et de mise en place ou système ProDOS.
SYSTEM.SETUP/	Fichier SubDirectory, structure indispensable pour charger et configurer les outils de la ROM (Tools Sets).
TOOL.SETUP	Fichier requis pour la mise en place des outils.

ROLE DU SYSTEME D'EXPLOITATION ProDOS

Définition des programmes système

Le but d'un ensemble informatique est de permettre le traitement des informations à partir d'une mémoire centrale (RAM). Associée à la mémoire vive, la mémoire morte (ROM) contribue pour une large part à l'efficacité du système ProDOS. En présence du GS, les outils résidant dans la ROM effectuent un travail considérable.

De par sa conception, le système d'exploitation a pour tâche de coordonner les différents maillons d'un ensemble et de traiter les fichiers au niveau de la mémoire ou du disque. ProDOS, Professionnal Disk Operating System, est le système d'exploitation par excellence, permettant de gérer des fichiers pour toute la famille des processeurs 65C... à partir de lecteurs de disques (5,25 ou 3,5 pouces) ou d'un disque dur adapté. Il a été mis au point par Apple Computer. Les unités de lecteurs et le disque dur sont utilisés comme mémoire de masse auxiliaire.

A un niveau plus élevé, ProDOS doit pouvoir classer les fichiers dans un certain ordre sur le disque et mettre à jour des tables permettant de connaître la position de chacun de ces fichiers. L'utilisateur doit pouvoir visualiser le contenu de la disquette, effectuer des copies ou sauvegarder et rappeler des fichiers.

ProDOS est un ensemble de petits programmes ou routines, qui font partie des programmes système, permettant une gestion, du disque à l'aide de commandes évoluées. Toutes les opérations sont prises en compte et gérées par le système : l'opérateur n'a pas à se soucier du déplacement de la tête du lecteur, ni à rechercher le bon emplacement d'un fichier, etc. Il est possible de gérer plusieurs lecteurs branchés sur un même ordinateur par l'intermédiaire des ports intégrés ou d'une carte contrôleur connectée dans un des slots de la carte mère (Apple *II*GS). Pour utiliser le système ProDOS, il faut d'abord le charger en mémoire. Cette opération est effectuée par le boot issu de la carte contrôleur et transmis au Loader du disque en ligne.

Démarrage d'un disque ProDOS

- Placer le disque système dans l'unité 1 sélectionnée comme dispositif de boot à l'aide du tableau de bord.

- Taper à partir du clavier : IN£s PR£s où s représente le numéro du slot affecté à l'unité considérée.

- Le sous-programme de l'interface du lecteur est exécuté (SmartPort par exemple, adresse $C500).

- Le chargement de ProDOS s'effectue à partir du disque. Cela se traduit par la mise en route du lecteur qui fait un bruit de mécanique dû au calage de la tête; puis celle-ci lit les données de la piste $00 à partir du secteur $00. A cet emplacement se trouve le Loader, routine de chargement du système, qui occupe les blocs logiques $00 et $01. Le rôle fondamental du Loader

consiste à reconnaître la configuration, puis à charger par l'intermédiaire du code de boot (Startup Boot code), les fichiers système appropriès. Cette routine de boot est identique pour toute la gamme des Apple *II* (*II*e, *II*c et *II*gs) et commune aux systèmes ProDOS8 et ProDOS16. La routine de chargement est d'abord placée à l'adresse $0801, puis exécutée.

BOOT D'UN DISQUE SYSTEME (DOS3.3, ProDOS8 et ProDOS16)

Pour permettre une comparaison des différents systèmes existants, les processus de boot existants sont détaillés par la suite.

Boot d'un disque DOS 3.3

Il est bon de schématiser au préalable le démarrage d'une disquette avec le DOS 3.3, afin d'avoir à l'esprit le mécanisme bien particulier du boot. Cela aidera à acquérir une meilleure compréhension de l'ensemble pour la suite des investigations avec le système ProDOS.

Boot sous DOS 3.3 – RAM de 64 Ko –

1- BOOT 0	ROM carte contrôleur – $C600 avec le GS	Charge BOOT 1 en RAM.
2- BOOT 1	Disque : Piste 0 secteur 0 RAM : $0800-$0900	Charge BOOT 2 et lui même.
3- BOOT 2	Disque : Piste 0, secteurs 1-9 Master: $3700-$4000 Charge le DOS Slave : - $B700-$C000	Contient RWTS, charge le DOS et le translateur.
4- SED	Disque : Piste 2, secteurs 4-0 Piste 2, secteurs F-0 Piste 0, secteurs F-C Master : $1D00-$3600 Slave : $9D00-$B600	Système d'exploitation du disque.
Relocator	Disquette Master : Piste 0, secteurs A-B RAM: $1B00-$1D00	Réinstalle le DOS à sa place définitive ($9D00-$BFFF).

Boot d'un disque ProDOS 1.1.1

Tout change à ce niveau de la technologie, l'emplacement du système d'exploitation sur le disque, la phase d'implantation en mémoire et l'occupation de la mémoire vive (RAM).

Procesus de boot du disque ProDOS_Version 1.1.1._

• BOOT 0. L'ensemble des opérations implique qu'un disque système (ProDOS 1.1.1 par exemple) soit introduit dans le lecteur de disques relié au slot 6 de l'Apple *II*GS par exemple. Deux cas peuvent alors se présenter : ou votre Apple est sous tension ou il est hors tension. Dans le premier cas, et si la sélection du tableau de bord a été réalisée dans cet état d'esprit, la mise sous tension entraînera le processus de boot. Dans le second cas, il faudra taper à partir du clavier l'une des commandes suivantes : IN£6, 6CTRL-P ou C600G. Par la suite, le processus de boot est exécuté à travers la routine du SmartPort ou de la carte contrôleur (si elle est en place). Si par définition, le disque système se trouve dans le drive connecté au slot 6, la routine de démarrage débute en $C600. Au départ, le sous-programme ignore tout du contenu de la disquette : il charge en premier une partie de la routine de chargement (Loader) qui se trouve matérialisée par les blocs logiques $00 et $01 du disque. Son but est de poursuivre le boot du disque en ayant au préalable déterminé la marche à suivre. Au niveau de la phase 1, la routine de chargement ne représente que la moitié du bloc 0 et sera logée à partir de l'adresse $0800.

• BOOT 1. Ensuite est effectué un saut à l'adresse $0801 par l'intermédiaire du sous-programme contenu dans la ROM du contrôleur qui exécute les instructions, puis charge l'autre moitié du bloc $00 et le bloc $01. C'est la phase 2 de la routine de chargement du système qui se termine.

• BOOT 2. Les blocs $00 et $01 forment un ensemble d'instructions qui déclenchent la phase 3 du processus. Si c'est un Apple III, alors le système cherche le fichier SOS dans le Directory du disque. Par contre, si c'est un Apple *II* comme défini au préalable, le système cherche le fichier ProDOS et le charge à l'adresse $2000. A ce système d'exploitation est associé un programme Move, dont le but est de déplacer une partie de ProDOS dans le banc 0 de la carte langage et cela à partir de l'adresse $D000. Ensuite est installé un programme de Reboot dans le banc 1 de la carte langage, à partir de l'adresse $D100. Si la configuration comporte une carte d'extension de 64 Ko, le système installe le RAM Disk-Driver à partir de l'adresse $0200 puis, pour finir, place la Global Page de ProDOS à partir de l'adresse $BF00 jusqu'à $BFFF.

• Ensuite ProDOS cherche sur le disque le premier fichier doté du suffixe .SYSTEM, qui est en général le BASIC.SYSTEM. Celui-ci est également chargé à l'adresse $2000, puis transféré à son adresse définitive, à partir de $9A00.

• Une fois le fichier BASIC.SYSTEM chargé en mémoire et exécuté, un programme Startup est recherché dans la table des matières principale (Directory) du disque. Ce fichier, du type AppleSoft ou Binaire, est exécuté dès qu'il est trouvé, sinon le système Basic Sollicite l'interpréteur à AppleSoft.

Boot d'un disque ProDOS8 ou ProDOS16

Sous un environnement Apple *II*GS , le disque système complet comporte les systèmes ProDOS8 et ProDOS16. Le principe de boot de ProDOS8 est

identique à celui de ProDOS16. Si nous déclarons que notre disque système a comme nom de volume VOLUME, le processus de boot sera le suivant :

• Lors de l'exécution de la routine de chargement (Loader), blocs logiques $00 et $01 du disque système, le code de démarrage est recherché en vue de sélectionner dans le Volume Directory le premier fichier répondant au nom de ProDOS (FileType $FF). Le préfixe sera /VOLUME/ProDOS, où ProDOS constitue la syntaxe exacte avec comme type de fichier $FF et /VOLUME/ désigne le nom du volume.

• Si le fichier ProDOS est trouvé, celui-ci est chargé en mémoire à partir de l'adresse $2000 (banc $00 du GS) et exécuté. Le fichier ProODOS est le maître d'oeuvre de la mise en place du chargement du système d'exploitation ProDOS complet, incluant entre autres les fichiers des outils de la RAM, des accessoires de bureau et éventuellement du programme à exécuter.

• Pour un Apple *II*GS , la notion du fichier ProDOS est différente et ne représente plus le fichier système d'exploitation, mais une routine élaborée pour charger et implanter ProDOS16.

• Si c'est un Apple *II*e ou *II*c, le fichier ProDOS sera remplacé par le système ProDOS8.

• La commande QUIT est relogée à son adresse définitive et référencée PQUIT pour le système. PQUIT contient le code de fin d'une application en cours de démarrage d'un autre programme, que celui-ci nécessite par la suite ProDOS8 ou ProDOS16.

• Le préfixe /VOLUME/ProDOS place le chemin d'accès pour charger le système ProDOS16 dont le chemin est par définition : /VOLUME/SYSTEM/P16

• Pour charger le système d'exploitation ProDOS16, /VOLUME/ProDOS est déclaré comme étant le chemin d'accès (Pathname) et prépare /VOLUME/SYSTEM/LOADER comme nouveau chemin d'accès. LOADER représente un fichier système avec comme suffixe .SYS16.

• /VOLUME/ProDOS prépare le chemin d'accès et initialise les vecteurs d'implantation avant l'exécution des fichiers stockés dans la table des matières auxiliaire dont la syntaxe est la suivante :

/VOLUME/SYSTEM/SYSTEM.SETUP

• Si un fichier du nom de TOOL.SETUP est détecté, celui-ci est exécuté en premier dans la liste des fichiers existants. Tool.Setup charge en RAM les outils de travail sous forme de fichiers, patche la RAM pour déplacer les outils de la ROM vers la RAM.

• Les fichiers stockés dans le SubDirectory SYSTEM.SETUP (Pathname complet /VOLUME/SYSTEM/SYSTEM.SETUP) doivent correspondre aux types de fichiers $B6, $B7, $B8 ou $B9. Ces différents types de fichiers sont commentés par la suite. Après l'exécution du fichier TOOL.SETUP, /VOLUME/ProDOS charge et exécute, un à un, d'autres fichiers stockés dans le SubDirectory.

• A partir de cet instant, le chemin d'accès formé par /VOLUME/ProDOS détermine et fixe le système d'exploitation nécessaire pour exécuter l'application future. Le chemin d'accès complet est élaboré pour charger par la suite le système d'exploitation sélectionné.

• En premier, un fichier du type $B3 est recherché dans le chemin /VOLUME/SYSTEM/START, où START représente le nom de ce fichier. Par défaut, START est le sélecteur pour le démarrage du programme à exécuter. Si le fichier START est trouvé, l'exécution se poursuit dans cette voie.

• Si aucun fichier du nom de START n'est trouvé dans le SubDirectory /SYSTEM, la structure /VOLUME/ProDOS recherche dans le volume ayant traduit le boot (Volume Directory), le premier fichier de l'un ou l'autre des systèmes et répondant aux critères suivants :

- un fichier compatible système ProDOS8, du type $FF, avec comme extension .SYSTEM;

ou

- un fichier compatible système ProDOS16, du type $B3, avec comme extension .SYS16.

Indépendamment du système courant, le premier fichier trouvé est chargé puis exécuté.

• Par la suite, /VOLUME/ProDOS passe le contrôle à la routine PQUIT (mise en place au début du boot). Si PQUIT ne pointe pas le chemin /VOLUME/ProDOS, le programme sélectionné auparavant est chargé en mémoire.

ROUTINE RESET

REMISE A ZERO DE LA ROM-AUTOSTART DU MONITEUR

– RESET DU MONITEUR AUTOSTART

ROM Moniteur commutée

Le point d'entrée du vecteur RESET se trouve à l'adresse $FA62. La copie du pointeur RESET se trouve aux adresses $FFFC-$FFFD (62 FA) de la ROM résidente du Moniteur du GS (hardware RESET Handler).

Processus du RESET, carte langage commutée

• Sous ProDOS, l'interruption à partir d'une commande ProDOS MLI provenant de la carte langage, emprunte le même chemin qu'un appel venant du moniteur : saut à l'adresse $FA62, en ayant au préalable reconnecté la ROM du Moniteur.

• Lorsque la carte langage est commutée 'on', le vecteur $FFFC-$FFFD contient le pointeur $FFCB; ce dernier, moniteur commuté 'on', contient un RTS (code 60).

Reset Hardware Handler (Apple *IIgs*

• **Entry Point**

FA62- LDA £01	Reset HardWare Handler Entrée de la routine de gestion de l'interruption RESET. On fixe le mode vidéo actuel en haute résolution 16 couleurs : Bit 7, inhibe le mode vidéo. Bit 6, linéarise les adresses $2000-$9FFF. Bit 5, mode noir et blanc. Bit 4 à 1 réservés. Bit 0, dernier banc mémoire.
FA64- TSB $C029	New Vidéo. Positionne le mode vidéo suivant la valeur du contenu de l'accumulateur.

FA67- LDX £FB
FA69- SEI
FA6A- CLD
FA6B- TXS
FA6C- JSR $FE36 — Saut dans la routine MOVE.

FA6F- LDA $C058 — Annonciateur 0 commuté High.
FA72- LDA $C05A — Annonciateur 1 commuté High.

FA75- LDY £09
FA77- JSR $F89C — Entrée dans la routine GET816LEN. Elle calcule l'adresse −1 de l'instruction du microprocesseur 65C816.

FA7A- LDA $CFFF — CLRROM. Désactivation des sous-programmes entre les adresses $C800-$CFFF.

FA7D- BIT $C010 — KBDSTRB. Mise à zéro de l'échantillonnage du clavier, permet de saisir un nouveau caractère au clavier.

FA80- JSR $FA23 — Saut dans la routine TEXT2COPY pour afficher la page courante et commuter le shadowing.

FA83- BNE $FA88
FA85- JSR $FD12 — Met le curseur en mode inverse clignotant et attend un caractère. C'est le vecteur de la routine d'entrée d'un caractère dont le pointeur est sauvegardé aux adresses $0038-$0039 de la page zéro (KSWL/KSWH).

FA88- JSR $F89A — Autre point d'entrée dans la routine GET816LEN. Elle calcule l'adresse −1 de l'instruction du microprocesseur 65C816.

FA8B- LDA $03F3 — RESET-Handler. Charge l'adresse de poids fort du pointeur de renvoi en cas de RESET (SOFTEV).

FA8E- EOR £A5 — Et effectue un test avec l'octet de réinitialisation qui a comme valeur constante £$A5.

L'adresse $03F4 contient la valeur £$38 avec le système d'exploitation DOS 3.3, et comme valeur £$1B avec le système ProDOS. En l'absence de tout système d'exploitation, l'adresse $03F4 contient la valeur £$45, tandis que $03F2-$03F3 pointe l'adresse $E000 (Basic ColdStart).

FA90- CMP $03F4 — PWRUP. Compare le résultat avec l'octet de test £$1B sous PRODOS.

FA93- BNE $FAA6 — Saut au vecteur de démarrage à froid du Basic AppleSoft.
$FAA6- JSR $F8F4 ColdStart

FA95- LDA $03F2 — RESET-Handler. Charge le pointeur LByte.

FA98- BNE $FADB — Si différent de zéro, démarrage à chaud dans le Basic.

FA9A- LDA £$EO — Charge la valeur £$E0, octet de poids fort du vecteur d'entrée à chaud dans le Basic AppleSoft ($E000).

FA9C- CMP $03F3 — Compare si un système d'exploitation est oui ou non implanté en mémoire. Est-ce que l'adresse en $03F2-$03F3 pointe $E000 (Coldstart du Basic)?

FA9F- BNE $FADB — Non, effectue alors un démarrage à chaud dans le Basic.

FAA1- LDY £$03 — RESET pointe le vecteur du démarrage à froid de Basic. Charge la valeur £$03, octet de poids faible de Basic Warmstart,

FAA3- JMP $FEED — Entrée obsolète pour un Apple *II*GS . Ancien vecteur de gestion d'une cassette.

ColdStart

- **PWRUP (System ColdStart Routine)**

FAA6- JSR $F8F4 — Effectue un HOME, puis affiche le logo Apple *II*GS .

FAA9- LDX £$05 — Initialise le registre d'index X,
FAAB- LDA $FAFC,X — Et positionne le pointeur. Charge les différentes données,
FAAE- STA $03EF,X — et les sauvegarde comme pointeurs dans la page 3, sauf pour RESET.
FAB1- DEX — Décrémente le compteur de la boucle,
FAB2- BNE $FAAB — Sauvegarde terminée?
FAB4- BRA £$C8 — Oui, met en place l'adresse du premier slot à solliciter sous la forme $Cs00, où s représente le numéro du slot. Premier slot = 7.
FAB6- STX $00 — Sauvegarde la valeur £$00.
FAB8- STA $01 — Sauvegarde la valeur £$C8:
$0000- 00 C8 Pointeur $C800. Premier slot $C700 (slot 7).

Sloop

- **Boucle de recherche des slots en ligne**

FABA- LDY £$05 — Charge le numéro du premier slot à solliciter (slot 5, SmartPort).
FABC- DEC $01 — Décrémente le pointeur de poids fort.
FABE- LDA £00 — Et met l'accumulateur à zéro.
FACO- LDA $01 — Charge le numéro du slot courant.
FAC2- CMP £C0 — Est-ce le slot numéro 1 (C100)?

FAC4- BEQ $FAF8	Non
FAC6- STA $07F8	Oui, sauvegarde le paramètre dans le vecteur réservé à la mémorisation des interfaces connectées dans le slot (s).
FAC9- LDA ($00),Y	Charge successivement le contenu des adresses $Cs07, $Cs05, $Cs03 et $Cs01 de la ROM contrôleur,
FACB- CMP $FB01, Y	et compare les valeurs trouvées respectivement avec £$20, £$00, £$03 et £$3C (DISKID du Disk *II*).
FACE- BNE $FABA	Pas de carte contrôleur dans le slot actuel, recherche du prochain slot.
FAD0- DEY	Décrémente deux fois le registre Y.
FAD1- DEY	
FAD2- BPL $FAC9	Test effectué jusqu'à ce que le byte à la position $Cs01 du slot considéré est trouvé (s = numéro du slot).
FAD4- JMP ($0000)	Ok, saut au vecteur $Cs00.

TABLE DES MATIERES SOUS ProDOS8 & ProDOS16

Ce chapitre développe particulièrement l'organisation et la structure des différentes entrées possibles dans une table des matières d'un volume quelconque. Cette disposition s'applique tout aussi bien au format 5,25 pouces qu'à celui de 3,5 pouces, voire même le disque dur. Le système ProDOS16 organise, au moment du formatage d'un support de sauvegarde, une table des matières principale destinée à y stocker par la suite un certain nombre d'informations utilisées pour la gestion des fichiers. D'origine, la table des matières sera toujours principale. Par la suite, elle pourra être étendue à une table des matières auxiliaire, pouvant à nouveau contenir des fichiers programmes et SubDirectory. Tout ce qui se rapporte à la structure d'une table des matières est traité sans différence par ProDOS8 et par ProDOS16.

FORMAT DU SUPPORT DE SAUVEGARDE

Au moment du formatage, ProDOS16 considère le Block Device en ligne comme un volume. Il divise la surface du support en pistes et secteurs physiques dont les nombres sont directement liés au type de lecteur courant. Pour accéder à un fichier quelconque, ProDOS16, par opposition au DOS 3.3, utilise la recherche logique et ignore totalement les pistes et secteurs physiques. Le format logique consiste à diviser la surface physique du disque en blocs logiques, plus imple à traiter par le système. C'est une routine interne à ProDOS qui effectue la conversion physique/logique, ou inversement, selon le travail courant (lecture ou écriture).

Un disque formaté sous ProDOS16 gère ses fichiers par l'intermédiaire de repères faisant référence à des blocs logiques. Un certain nombre d'informations sont directement écrites sur le disque et la bit map mise à jour par le système.

C'est ainsi que le programme de chargement du système d'exploitation (Loader) se trouve initialisé sur les blocs $00 et $01 de tout disque Système ProDOS.

• Une table des matières désigne soit un Directory, soit un SubDirectory.

– dans le premier cas, elle est dite principale et occupe 4 blocs par défaut. Un Directory a une taille fixe, limitée à 4 blocs.

– dans le second cas, elle est réservée à une table des matières auxiliaires et occupe un bloc par défaut. Un SubDirectory est extensible dans sa taille, chaque bloc supplémentaire rendu nécessaire, est chaîné à la suite du bloc existant.

Le nom d'un fichier stocké dans une entrée du Directory, peut se trouver dans une table des matières principale (Directory) ou auxiliaire (SubDirectory). Le bloc $02, appelé Key Block, désigne toujours le premier bloc de la table des matières d'un volume ProDOS. Les blocs $03 à $05, Non Key Blocks, sont chaînés à la suite du Key Block. Chaque table des matières permet la sauvegarde de paramètres et pointeurs faisant référence aux fichiers dont le contenu se trouve sur la surface du disque.

• Le bloc $06 est réservé à la table d'occupation du disque, c'est la bit map du support. Elle marque libre les différents blocs logiques encore disponibles sur la surface du disque. Si plusieurs blocs sont nécessaires à la bit map, ceux-ci sont chaînés les uns à la suite des autres. Le format d'une bit map est limité, en théorie, à la gestion de 4096 blocs logiques.

• Le reste des blocs du disque est réservé à la sauvegarde des données, comme par exemple des fichiers programmes, programmes sytèmes, etc.

TABLE DES MATIERES PRINCIPALE & AUXILIAIRE

Généralités concernant les Directory et SubDirectory

• Un Directory désigne une table des matières en général.

• Une table des matières contient à la fois des informations relatives aux fichiers programmes et tables des matières. Le Key Block est le premier bloc dans une table des matières.

• Le premier bloc de toute table des matières s'appelle le Key Block. La première entrée dans cette table des matières est réservée à l'en-tête:
– elle désigne le nom du Volume Directory quand elle se trouve dans une table des matières principale;
– elle désigne le nom du Volume SubDirectory quand elle se trouve dans une table des matières auxilliaire.

• Chaque entrée dans une table des matières se compose d'un nombre de bytes identiques. Le début réel dans une entrée réservée à l'en-tête d'une table des matières débute à la position relative $04 du Key Block, les positions $00-$03 sont réservées aux pointeurs avant et arrière du bloc courant.

• La différence fondamentale entre un Directory et un SubDirectory réside dans le premier champ (en-tête) du Key Block. Un Volume Directory Header contient le nom du volume (nom du disque), tandis que le Volume SubDirectory Header contient le nom du SubDirectory donné lors de sa création.

• Les quatre premiers bytes de chaque bloc d'une table des matières (Directory ou SubDirectory) sont réservés pour les pointeurs arrière et avant (Backward et Forward):
– le pointeur avant, ou Forward Pointer, désigne à l'intérieur du Directory le numéro du bloc qui précède son propre bloc;
– le pointeur arrière, ou Backward Pointer, désigne à l'intérieur du Directory le numéro du bloc qui suit son propre bloc;
– au niveau du Key Block, bloc 0002, le pointeur arrière de valeur 0000 désigne le premier bloc du Directory;
– le bloc 0005, Non Key Block, pointeur avant de valeur 0000 désigne le dernier bloc du Directory.

GENERALITES CONCERNANT UN VOLUME DIRECTORY

• Un volume désigne l'espace total de la surface d'un disque. Par extension de la syntaxe, un volume est par défaut un Block Device. Un Volume Directory représente la table des matières principale du volume cité en référence. Un nom est attribué à chaque volume au moment de la séquence de formatage, et se trouve copié dans la première entrée du Directory Key Block.

• Un Volume Directory représente toujours une table des matières principale; quatre blocs lui sont affectés pour stocker les noms des fichiers: blocs $02 à $05 du support disque.

• La table des matières principale occupe les blocs 0002 à 0005 de tout disque. Par comparaison au Volume SubDirectory, le Volume Directory ne permet pas d'extension au niveau de sa taille en nombre d'entrées (ou champs).

• Chaque entrée d'un Directory Key Block comporte les attributs suivants:
- un pointeur qui désigne le prochain bloc dans le Directory;
- une entrée principale (Header) qui renseigne sur le nom du Volume Directory;
- un certain nombre de champs réservés aux entrées de noms de fichiers accompagnés de leurs paramètres descriptifs et des pointeurs respectifs;
- un ou plusieurs bytes inutilisés.

• Chaque entrée d'un Directory Non Key Blocks comporte les attributs suivants:
- un pointeur qui désigne le bloc précédent auquel se trouve chaîné le bloc actuel;
- un certain nombre de champs réservés aux entrées de noms de fichiers accompagnés de leurs paramètres descriptifs et des pointeurs respectifs;
- un ou plusieurs bytes inutilisés.

GENERALITES CONCERNANT UN VOLUME SUBDIRECTORY

• Un Volume SubDirectory désigne l'espace du support disque affecté à l'extension d'un Directory. Cette extension est souvent rendue nécessaire

pour augmenter la capacité de sauvegarde en nombre de fichiers, limités initialement par la structure intrinsèque du Directory (blocs $02 à $05). Une organisation arborescente pourra très bien faire appel à un ou plusieurs SubDirectory.

• Un Volume SubDirectory représente une table des matières auxiliaire créée par la commande CREATE. Il occupe par défaut un bloc sur le disque, extensible au fur et à mesure de l'occupation de ses entrées. Le SubDirectory n'occupe pas de place précise sur le disque et peut, de ce fait, se trouver à tout emplacement non réservé intialement par les fichiers système.

Volume Directory (Table des matières principale)

Position relative des entrées
Configuration d'une entrée

Les entrées dans la table des matières principale, après l'écriture du nom du volume, sont réservées à des noms de fichiers quelconques. Le Key Block pointe le premier bloc appelé Non Key Block.

Structure type d'un bloc table des matières

Pointeurs avant et arrière	4 bytes
13 entrées noms de fichiers	507 bytes
Total	511 bytes

La position 512 de chaque Directory Block reste inutilisée, sa valeur est mise initialement à zéro. Chaque bloc d'une table des matières se divise en un certain nombre de champs ou d'entrées réservées à des noms de fichiers. Chaque entrée est matérialisée à l'intérieur de son Directory Bock par un pointeur relatif: adresse de début $00; adresse de fin $26, ce qui fait un total de £$27, soit 39 bytes réservés par entrée.

ENTREES POSSIBLES DANS UN VOLUME DIRECTORY

Numéro d'ordre	Positions Hex	Dec
Entrée 1	$004	004
Entrée 2	$02B	043
Entrée 3	$052	082
Entrée 4	$079	121
Entrée 5	$0A0	160
Entrée 6	$0CF	199
Entrée 7	$OEE	238
Entrée 8	$115	277
Entrée 9	$13C	316
Entrée 10	$163	355
Entrée 11	$18A	394
Entrée 12	$1B1	433
Entrée 13	$1D8	472

VOLUME DIRECTORY NON KEY BLOCKS

Structure generale d'un bloc: exemple du deuxième bloc du Volume Directory Block 00 03.
Les blocs 04 et 05 ont une structure identique.

Position	Contenu
$00-$01	Pointeur arrière – LByte, HByte. Pointe le bloc précédent : 02 00; 00 02 = bloc 00 02
$02-$03	Pointeur avant – LByte, HByte. Pointe le bloc suivant : 04 00; 00 04 = bloc 00 04.
$04-$2A	Première entrée d'un nom de fichier.
$2B-$51	Deuxième entrée d'un nom de fichier. ....
$1D8-$1FE	Treizième entrée d'un nom de fichier.
$1FF	Inutilisé.

VOLUME DIRECTORY HEADER (KEY-BLOCK)

Le nom du volume, qui par défaut est celui du disque, se trouve sauvegardé dans le premier champ (Header) d'une table des matières principale (bloc $02). L'écriture du nom du volume est réalisée après la séquence de formatage du disque. Le Volume Directory Header permet la sauvegarde d'un certain nombre de paramètres particuliers.

Entrée d'un Volume Directory Header

Position	Contenu
$00-$01	Backward_Reference (2 bytes). Pointeur arrière avec comme valeurs LByte, HByte. 0000 signifie que c'est le premier bloc.
$02-$03	Forward_Reference (2 bytes). Pointeur avant avec comme valeurs LByte, HByte. $0003 (03 00) pointe le bloc 3 dans le Directory.
$04	Storage_Type & Name_Length (1 byte). Les quatre bits de poids fort caractérisent le type de sauvegarde du fichier. Les quatre bits de poids faible désignent la longueur du volume. **Exemple**: $F6, où F représente les quatre bits de poids fort et désigne un fichier Volume; 6 représente les quatre bits de poids faible qui désignent

Position	Contenu
	la taille du nom : 6 caractères dans notre exemple. La longueur dépend du nom donné au moment du formatage du disque et peut être modifié par la commande MLI RENAME.
$05-$13	File_Name (15 bytes). Ce champ est réservé au nom du volume qui n'est autre que le nom du disque. Le nombre maximum de caractères autorisés se monte à 15, les codes ASCII ayant les bits 7 positionnés à zéro. Le nom du volume se modifie à l'aide de la commande MLI RENAME.
$14-$1B	Bytes réservés (8 bytes). Ce champ est réservé par Apple Computer pour une application future.
$1C-$1F	Created DATE & TIME (4 bytes). Ce champ se partage les bytes pour la date et l'heure de sauvegarde du nom du volume. Ces bytes sont lus par la commande MLI GET_FILE_INFO.
$20	Version (1 byte). Désigne la version de ProDOS ou ProDOS16 sous lequel a été initialisé le volume actuel. Avec la version 1.0, version est égal à 0.
$21	Mini_Version (1 byte). Réservé à une application future et destiné à tester la version mini autorisée avec des applications courantes.
$22	Access_Byte (1 byte). Détermine le mode d'accès au Directory. Selon la valeur de ce byte, le MLI autorise l'écriture, la lecture, la purge ou la modification du nom courant. **Exemple** : $C3 = Lecture et écriture permises.
$23	Entry_Length (1 byte). Désigne le nombre de bytes alloués à la première entrée dans le volume Directory actuel : – 39 bytes pour l'entrée du volume (Header).
$24	Entry_per_Block (1 byte). Désigne le nombre d'entrées possibles dans le Key Block : – 13 entrées dans le Key Block.
$25-$26	File_Count (2 bytes). Nombre d'entrées valides de noms de fichiers actuellement sauvegardés dans la table des matières principale. Une entrée valide se caractérise par un Storage_type différent de zéro.
$27-$28	Bit_Map_Pointer (2 bytes). C'est le pointeur de la bit map. Les deux bytes pointent le premier bloc de la table d'allocation des blocs du volume (06 00 = 0006 par défaut). La bit map occupe un

Position	Contenu
$29-$2A	nombre de blocs consécutifs déterminés au moment du formatage du volume. Elle peut gérer un maximum de 4096 blocs, ce qui fixe à 8 le maximum de blocs alloués à une bit map. Total_Blocks (2 bytes). Représente la capacité totale du volume courant : – 18 01 = 0118 ou 280 blocs pour un disque 5,25 pouces par exemple. – 40 06 = 0640 ou 1600 blocs pour un disque 3,5 pouces par exemple.

Paramètres détaillés d'un Volume Directory Header

Bytes $00-$01 Pointeur arrière – LByte, HByte

Pointe le bloc qui le précède. L'entrée réservée au nom du volume est par définition et par conception le premier champ dans le Key Block. Le Key Block se trouve au début de la table des matières principale et n'a pas d'autre bloc logique qui le précède. Le pointeur arrière a donc une valeur toujours nulle : 00 00.

Bytes $02-$03 Pointeur avant – LByte, HByte

Pointe le bloc suivant (bloc $03). Le bloc $02 du disque est le premier bloc dans la suite logique des blocs affectés à une table des matières principale. Il se nomme le Key Block. Les blocs suivants, Non Key Blocks, forment une suite chaînée possédant chacun un pointeur avant et un pointeur arrière. Le pointeur avant d'un Key Block possède toujours comme valeur 03 00 (0003).

Byte $04 Storage Type & Name Length

Ce paramètre est combiné par l'association de deux nibbles formant chacun 4 bits. Les 4 bits de poids fort caractérisent le type sauvegarde du fichier; les 4 bits de poids faible caractérisent la longueur du nom attribué au fichier considéré.

Paramètres Storage_Type possibles

$00	Nul, pas de nom ou fichier détruit.
$01	Seedling. Ce type de fichier fait référence au plus petit format (entre $0000 et $0200 bytes). Il occupe un seul bloc de données sur le disque et ne possède pas de bloc index. Les données sont directement pointées par l'entrée correspondante du fichier. Le pointeur d'un tel fichier se trouve stocké aux positions relatives $11-$12 de l'entrée fichier dans sa table des matières.

$02	Sapling. Ce type de fichier est d'une taille supérieure à un bloc (512 bytes), avec une capacité de $0201 à $20000 bytes. Un fichier Sapling peut comporter jusqu'à 256 blocs de données pointés par un seul et unique bloc index comme répertoire des blocs.
$03	Tree. Ce type de fichier personnalise la taille maximale que pourra avoir un fichier ProDOS. Il nécessite toujours plus d'un bloc index. La capacité d'un fichier Tree est comprise entre $020001 et $1000000 bytes inclus. Une telle structure peut nécessiter un Master Block qui peut pointer de 1 à 128 blocs index, dont chacun peut à nouveau pointer 256 blocs de données. Un rapide calcul montre qu'un tel fichier peut contenir 16.777.216 bytes, soit 16 mégabytes.
$04	USCD Pascal. Ce type de fichier est réservé à des applications en Pascal.
$0D	SubDirectory File. Ce Storage_type est attribué au nom du fichier stocké dans une table des matières et qui pointe un Volume Subdirectory.
$0E	Volume SubDirectory. Ce paramètre est attribué au nom du volume stocké dans le premier champ d'un Volume Subdirectory. C'est par défaut le nom du volume d'un SubDirectory attribué au moment de sa création.
$0F	Volume Directory. Ce paramètre est attribué au nom du volume stocké dans le premier champ d'un Volume Directory. C'est par défaut le nom du volume d'un disque attribué lors du formatage.

Paramètre Name_Length (Longueur du nom)

– Fichier programme ou SubDirectory

Les quatre bits de poids faible représentent la longueur du nom de tout fichier sauvegardé sur un disque ProDOS. La syntaxe est particulière, sa longueur ne devra pas dépasser 15 caractères choisis parmi les codes ASCII valides. Quatre bits permettent un codage maximum de 16 ($2^4 = 16$).

Bytes $05-$13 File Name (nom du fichier)

Un maximum de quinze caractères valides permettent de nommer tout type de fichier, qu'il représente un programme, un Volume Subdirectory ou tout simplement des données.

Exemple

Si USERS.DISK est le nom du Volume Directory, le byte Storage_Type & Name_Length sera représenté par l'association de deux nibbles et aura comme valeur FA, où F représente le paramètre affecté au type du Volume Directory et A la longueur du nom représentatif (L = 10 dans notre exemple). Les noms des fichiers sont sauvegardés dans la table des matières avec le bit 7 positionné à 0.

Bytes $14-$1B Bytes inutilisés

Ce champ du Key Block Header est réservé à des applications futures et contient actuellement 8 bytes nuls (citation du Technical Manual).

Bytes $1C-$1F Date et heure de création

Ce champ d'entrée est réservé à la date et à l'heure de création du Volume Directory. Ces paramètres sont mis en place lors du formatage du support disque.

Bytes $20-$21 Version de ProDOS

ProDOS Version et Mini_Version. Seul le byte $21 est testé par la commande MLI OPEN : si sa valeur est différente de $00 pour les versions ProDOS8 1.0.1 et 1.0.2, le système interrompt l'exécution et renvoie le message d'erreur ERR£4E INCOMPATIBLE FILE FORMAT.

Byte $22 Access_Byte

Ce paramètre détermine si un nom de fichier peut être renommé, effacé, lu ou si ses données peuvent être copiées. La valeur binaire des bits détermine le byte d'accès.

Représentation binaire d'Access_Byte

Bits	7 6 5 4 3 2 1 0 D R B x x x W R
Bit 7	Destroy bit. Si ce bit est à 1, la commande DELETE sera autorisée.
Bit 6	Rename bit. Si ce bit est à 1, la commande RENAME sera autorisée.
Bit 5	Backup bit. Ce bit est exploité par les commandes CREATE, RENAME, CLOSE, WRITE et SET_FIL_INFO pour permettre la copie des données. Test du bit 5 : Bits : 7 6 5 4 3 2 1 0 0 0 1 0 0 0 0 0 = £$20 L'instruction machine EOR effectue un test sur le bit 5 (Backup bit).

Bits 4-2	Inutilisés, donc nuls – 000 -.
Bit 1	Write. Si ce bit est à 1, autorise l'écriture.
Bit 0	Read. Si ce bit est à 1, autorise la lecture.

La lecture et l'écriture sont possibles avec le byte Access de valeur £$C3.

Exemples

Bits: 7 6 5 4 3 2 1 0
1 1 – – – – 1 – = UNLOCK (déverrouillé)
0 0 – – – – 0 – = LOCKED (verrouillé)
0 0 1 0 0 0 1 – = £$21 = SYS

Byte $23 Entry_Length

Ce paramètre désigne la longueur par défaut attribuée à chaque entrée dans une table des matières. Chaque entrée est ainsi utilisée par ProDOS pour y sauvegarder le nom du fichier et un certain nombre d'attributs qui sont des paramètres personnalisés (39 bytes pour l'entrée nom du volume).

Byte $24 Entries_per_Block

ProDOS8 et ProDOS16 autorisent un total de treize entrées de noms de fichiers par bloc, nom du Volume Directory compris.

Bytes $25-$26 File_Count

File_Count recense le nombre de noms de fichiers actuellement sauvegardés dans le Volume Directory. C'est le nombre d'entrées réellements occupées dans la table des matières principale.

Bytes $27-$28 Bit_Map Pointer

Ce champ contient le pointeur du premier bloc de la Volume Bit Map, table d'occupation des blocs du support disque. Ce pointeur possède une valeur par défaut: 06 00 = 0006. Théoriquement, ProDOS peut gérer une VBM de 4096 blocs de données; dans ce cas, les blocs de la bit map se suivent et sont chaînés entre eux.

Bytes $29-$2A Total_Blocks

C'est la capacité totale du volume courant en nombre de blocs. C'est en somme la mémoire de masse possible d'un disque ou support de sauvegarde: 280 blocs ($0118) pour une disquette standard 5,25 pouces et 35 pistes. Un disque de 3,5 pouces autorise le formatage d'un volume de 1600 blocs ($640 blocs). Théoriquement ProDOS peut gérer jusqu'à 65 536 blocs de données ($10000).

Remarque

L'entrée suivante est par défaut une entrée d'un nom de fichier quelconque. Chaque Directory Block peut contenir treize noms de fichiers, le Key Block contient d'origine le nom du volume sauvegardé lors du formatage.

VOLUME SUBDIRECTORY HEADER

Cette table de matières est particulière. Elle se retrouve très souvent sous l'appellation SubDirectory, sous-catalogue ou encore table des matières auxiliaire: c'est une extension du Directory. Le fichier qui pointe une table des matières auxiliaire est créé par la commande CREATE.

Le nom du fichier SubDirectory est sauvegardé dans le Volume Directory et pointe de ce fait le Volume SubDirectory. Ce dernier, appelé encore Key Block SubDirectory, occupe au départ un bloc.

La structure du SubDirectory Header permet de sauvegarder à nouveau des fichiers programmes ou SubDirectory. Un fichier SubDirectory est du type DIR et possède un Storage_Byte de valeur £$DC, tandis que dans un Volume SubDirectory, le nom du volume a comme valeur £$EC. Les quartets £$D et £$E, bits de poids fort, représentent Storage_Type, tandis que le quartet £$C, bits de poids faible, équivaut à Name_Length.

- Les quatre premiers bytes sont réservés aux pointeurs arrière et avant, suivant les mêmes principes que dans la représentation du Volume Directory. Deux bytes pour le pointeur arrière (Backward Pointer) et 2 bytes pour le pointeur avant (Forward Pointer).

- A la suite, se trouvent les 39 bytes réservés à la première entrée dans la table des matières auxiliaire (Volume SubDirectory Header).

En opposition au Volume Directory, le Volume SubDirectory peut avoir un nombre quelconque de blocs. Lorsque le SubDirectory est créé par la commande CREATE, le système lui attribue un seul bloc. S'il arrive par la suite que plus de treize noms de fichiers doivent y être sauvegardés, un deuxième bloc sera chaîné au premier (nom du Volume SubDirectory compris). Etant donné la structure particulière du chaînage d'un SubDirectory, les blocs ne peuvent en aucun cas se suivre dans l'ordre ascendant des valeurs.

Entrée d'un Volume SubDirectory Header

Position	Contenu
$00-$01	Backward Pointer (2 bytes). Pointeur arrière: 00 00 = $0000
$02-$03	Forward Pointer (2 bytes). Pointeur avant: 00 00 = $0000
$04	Storage_Type & Name_Length (1 byte). Les quatre bits de poids fort caractérisent le type de sauvegarde. Les quatre bits de poids faible désignent la longueur du nom du volume attribué au SubDirectory. La longueur du nom du volume peut varier en fonction du nombre de caractères lui étant attribués lors de sa création (commande RENAME).

Position	Contenu
$05-$13	File_Name (15 bytes). Le premier champ dans le Volume SubDirectory contient le nom du volume. Ce nom répond à une syntaxe particulière à ProDOS avec un maximum de 15 caractères ASCII possibles.
$14-$1B	Bytes réservés (8 bytes). Ce champ est réservé à des applications futures et comporte le nom de 'uHUSTON!' sauvegardé comme sigle. Il n'a aucune signification précise et n'est pas testé par le MLI.
$1C-$1F	Created_Date & Time. (4 bytes). Ce champ est réservé à la date et à l'heure de création du Volume SubDirectory.
$20	Version (1 byte). Désigne la version de ProDOS en général et de ProDOS16 en particulier avec laquelle le Volume SubDirectory a été créé.
$21	Mini_Version (1 byte). Désigne la version minimum à laquelle ProDOS pourra avoir accès par la suite.
$22	Access_Byte (1 byte). Détermine le mode d'accès au SubDirectory. Selon la valeur d'Access-_Byte, le MLI autorise l'écriture, la lecture, la purge ou la modification du nom courant. **Exemple**: $C3 = lecture et écriture permises.
$23	Entry_Length (1 byte). Désigne le nombre de bytes alloués à la première entrée dans le Volume SubDirectory actuel : – 39 bytes pour l'entrée du volume (Header).
$24	Entry_per_Block (1 byte). Désigne le nombre d'entrées possibles dans le Key Block : – 13 entrées dans le Key Block.
$25-$26	File_Count (2 bytes). Nombre d'entrées valides de noms de fichiers actuellement sauvegardés dans la table des matières auxiliaire. Une entrée valide se caractérise par un Storage_Type différent de zéro.
$27-$28	Parent_Pointer (2 bytes). L'adresse de ce champ pointe le bloc dans la table des matières (principale ou auxiliaire) où se trouve le nom du fichier qui pointe ce Volume SubDirectory. 02 00 = $0002 = bloc 00 02 si le nom du fichier DIR se trouve sauvegardé dans le bloc 00 02.
$29	Parent_Entry_Number (1 byte). Représente le

Position	Contenu
$2A	numéro de l'entrée dans le bloc parent du SubDirectory. Parent_Entry_Length (1 byte). C'est la longueur d'une entrée dans la table des matières du bloc parent, c'est-à-dire le Directory dans lequel se trouve le nom du fichier SubDirectory.

Paramètres détaillés d'un Volume SubDirectory Header

Bytes $00-$01 Pointeur arrière – LByte, HByte

Pointe le bloc qui le précède. L'entrée réservée au nom du Volume SubDirectory est par définition et par conception le premier champ dans le Key Block. Le Key Block se trouve au début de la table des matières auxiliaire et n'a pas d'autre bloc logique qui le précède. Le pointeur arrière a donc une valeur toujours nulle : 00 00. Lors de la création d'un Volume SubDirectory, le Key Block est le seul et unique bloc attribué. Par la suite, ProDOS alloue un bloc au Volume SubDirectory à chaque fois que son extension sera rendue nécessaire. Les nouveaux blocs sont chaînés au bloc précédent.

Bytes $02-$03 Pointeur avant – LByte, HByte

Pointe le bloc suivant. Le Key Block, bloc d'une table des matières auxiliaires est le seul bloc logique au départ. Le pointeur avant aura donc comme valeur 0000. Les blocs suivants, Non Key Blocks, seront attribués au fur et à mesure de l'extension du Volume SubDirectory et chaînés aux blocs précédents.

Byte $04 Storage_Type & Name_Length

Ce paramètre est combiné par l'association de deux nibbles formant chacun quatre bits. Les quatre bits de poids fort caractérisent le type de sauvegarde du fichier; les quatre bits de poids faible caractérisent la longueur du nom attribué au fichier considéré. Pour un Volume SubDirectory, la valeur de Storage_Type, quatre bits de poids fort, est $E.

Bytes $14-$1B Sigle 'uHUSTON!'

Dans un Volume Directory, ce champ est inutilisé et les bytes sont mis à zéro. En présence d'un Volume SubDirectory, le sigle **uHUSTON!** se trouve stocké à cet emplacement. Le MLI, en présence d'un Volume SubDirectory Header (Key Block), teste le byte $14; il doit avoir un minimum de 5 bits à 1, sinon un message d'erreur sera renvoyé : ERR£4E INCOMPATIBLE FILE FORMAT.

Bytes $27-$28 Parent_Pointer

Ce champ pointe le bloc de la table des matières dans laquelle se trouve le nom du fichier ayant donné naissance au Volume SubDirectory courant. Si

le pointeur est de valeur $02, le nom du fichier ayant donné naissance au Volume SubDirectory se trouve copié dans le bloc $02 du Volume Directory.

Byte $29 Parent_Entry_Number

Ce byte désigne le numéro de l'entrée dans la table des matières qui le contient (bloc parent) et dans laquelle se trouve copié le nom du fichier SubDirectory. Si par exemple ce byte est de valeur $03, le nom du fichier Volume SubDirectory se trouve alors à la troisième position dans la table des matières qui le contient.

Byte $2A Parent_Entry_Length

Ce paramètre signale le nombre de bytes alloués à l'entrée faisant partie de la table des matières qui contient le nom du fichier. Cette valeur attribuée par défaut est $27 (39 bytes).

FILE ENTRY (ENTREE D'UN NOM DE FICHIER)

- La position relative des paramètres stockés dans chaque champ réservé à un nom de fichier (39 bytes), que ce soit dans un Directory ou SubDirectory, est donnée par rapport à chaque entrée. Chaque pointeur débute avec la valeur $00 et prend fin avec la valeur $26 (£$27 ou 39 bytes alloués à chaque fichier).

- Chaque bloc d'une table des matières peut contenir treize noms de fichiers accompagnés de ses caractéristiques. Le byte à la position 512 est toujours inutilisé et de ce fait prend la valeur zéro.

- Une table des matières principale peut contenir un total de 52 noms de fichiers, le premier nom qui se trouve dans le Key Block est réservé au nom du volume. Dans le cas d'une table des matières principale, le nom du volume désigne un Directory qui est celui du disque par défaut. Dans le cas d'une table des matières auxiliaire, le nom du volume désigne un SubDirectory.

Entrée d'un nom de fichier

Position relative	Contenu
$00	Storage_Type & Name_Length (1 byte). Les quatre bits de poids forts représentent Storage_Type et désignent la caractéristique de sauvegarde du fichier : $1 = Seedling File (fichier de plus petit format) $2 = Sapling File (fichier supérieur à 512 bytes) $3 = Tree File (fichier qui utilise plus d'un bloc index) $4 = Pascal Area (réservé au Pascal) $D = SubDirectory File (fichier table des matières auxiliaire – SubDirectory) Les quatre bits de poids faible représentent la longueur du nom attribué au fichier.
$01-$0F	File_Name (15 bytes). C'est le nom du fichier programme et répond à une syntaxe particulière à ProDOS, avec un maximum de 15 caractères ASCII possibles.
$10	File_Type (1 byte). Ce byte est un descripteur et personnalise la structure interne du fichier. **Exemple** : $0B = BIN
$11-$12	Key_Pointer (2 bytes). Pointe le premier bloc du fichier. **Exemple** : 12 00 = 0012, pointe le bloc $0012. L'adresse pointe en général un bloc index d'un fichier avec une taille supérieure à 512 bytes. Si le pointeur s'adresse à un Volume SubDirectory, ce champ contient le numéro du Key Block.
$13-$14	Blocks_Used (2 bytes). Désigne le nombre total de blocs occupés par le fichier : 45 00 = 0045 blocs occupés. Pour un SubDirectory, le bloc fichier DIR est inclus dans le décompte du total.
$15-$17	EOF_Pointer (3 bytes). Ces trois bytes représentent le nombre de bytes pouvant être réellement lus dans un fichier. C'est par défaut la longueur d'un fichier texte ou la fin d'un fichier binaire : 55 00 00 = $000055 de longueur.
$18-$1B	Created_Date & Time (4 bytes). La date et l'heure de création du fichier sont copiés dans ce champ. Format : 00 00 00 00 (2 bytes pour la date et 2 bytes pour l'heure)
$1C	Version (1 byte). C'est le numéro de la version ProDOS qui a sauvegardé le fichier courant.
$1D	Mini_Version (1 byte). C'est le numéro de la version mini avec laquelle pourra être traité le fichier par la suite.
$1E	Access_Byte (1 byte). Ce paramètre détermine par la suite le mode d'accès au fichier courant. Permet suivant la position des bits, de déclarer les commandes MLI READ, WRITE, DESTROY ou RENAME.

Position relative	Contenu
$1F-$20	Aux_Type (2 bytes). Information complémentaire, désigne une adresse d'implantation du programme par exemple. Le Basic System utilise ce champ pour lire l'adresse d'implantation du programme Basic. C'est encore la longueur pour un fichier texte.
$21-$24	Last_Mod (4 bytes). Ce champ comporte la date et heure de la dernière modification effectuée sur un fichier par exemple. Il est mis à jour par la commande MLI SET_FILE_INFO lors d'un CLOSE fichier. Format : 00 00 00 00 (2 bytes pour la date et 2 bytes pour l'heure).
$25-$26	Header_Pointer (2 bytes). Ce champ contient l'adresse qui pointe le Key Block d'un Directory dans lequel se trouve sauvegardé le nom du fichier. Le pointeur est de la forme LByte, HByte.

Paramètres détaillés d'un NOM de FICHIER

Byte $00 Storage_Type & Name_Length

La commande DELETE met ce byte à zéro; les bytes $13-$14 (Blocks_Used) sont remis à zéro pour marquer l'attribution comme nulle, les autres données stockées dans l'entrée restent intactes. Cette structure de la table des matières ne permet plus de récupérer un fichier effacé. ProDOS laisse très peu de chance à de telles pratiques de récupération, à moins d'être un expert du clavier : il faudra remettre à jour la Volume Bit Map ainsi que la table des matières du fichier.

Bytes $01-$0F File_Name

Identique à l'entrée du nom d'un Volume Directory : les caractères sont du type ASCII valides.

Bytes $10 File_Type

Emplacement réservé pour stocker le type du fichier dont le nom est sauvegardé aux positions relatives $01-$0F de l'entrée correspondante. Le paramètre File_Type n'est pas à confondre avec Storage_Type.

Remarques concernant les fichiers programmes

• Les fichiers INTEGER ne sont plus traités, car ProDOS ignore totalement ce langage en raison de son occupation mémoire. Néanmoins les codes INT et IVR ($FA, $FB) figurent toujours parmi la table des types de fichiers (Technical Manual).

• Les fichiers de données reconnus par le BASIC.SYSTEM, lors de la commande CAT ou CATALOG, sont personnalisés par trois caractères : sous la rubrique TYPE, 3 bytes déclinent généralement les trois premiers caractères du type de fichier. (BAS = BASIC, BIN = BINAIRE, etc.).

• TXT désigne un fichier du type TEXTE. ProDOS sauvegarde les données différemment que le DOS 3.3 : les enregistrements des caractères sont réalisés avec le bit 7 à 0.

Exemple

Le mot TEXTE, sauvegardé sous les deux systèmes d'exploitation, pourra être lu au niveau de la disquette, par un utilitaire du genre DISKFIXER.

ProDOS	54	45	58	54	45	0D	= bits 7 à 0
DOS 3.3	D4	C5	D8	D4	C5	8D	= bits 7 à 1

Le programme CONVERT de la disquette utilitaire ProDOS fournie par Apple Computer, fait automatiquement la conversion entre les deux systèmes.

Les routines COUT ($FDED) et RDKEY ($FDOC) ne permettent plus un accès aux fichiers texte.

• Les fichiers binaires représentés par BIN. DOS 3.3 sauvegarde un programme binaire en lui associant deux paramètres : son adresse d'implantation en mémoire centrale et sa longueur (A$Adresse, L$Longueur). Au moment de la sauvegarde sur un disque, ces valeurs figurent en tête des données du programme : deux bytes pour l'adresse et deux bytes pour la longueur.

ProDOS8 et ProDOS16 traitent les données différemment : l'adresse d'implantation en mémoire centrale du programme se trouve stockée aux positions relatives $1F-$20 (Aux_Type) et la longueur aux positions relatives $15-$17 (E0F). Ces paramètres (adresse et longueur) ne sont plus sauvegardés avec les données dans le premier bloc du programme.

• Le fichier BAS est l'équivalent du fichier AppleSoft (A) sous DOS 3.3. Comme avec les fichiers BIN, la longueur n'est plus sauvegardée dans le premier bloc des données du programme, mais stockée aux positions relatives $15-$17 (E0F). Fait nouveau sous ProDOS : l'adresse d'implantation d'un programme AppleSoft, se trouve stockée aux positions relatives $1F-$20 de la table des matières (Aux_Types).

• Le fichier VAR est nouveau : il est créé par la commande STORE et contient les variables d'un programme en mémoire (variables + noms des variables). La commande RESTORE rappelle en mémoire les variables et leurs noms sauvegardés dans un fichier du type VAR. L'avantage essentiel de cette commande réside dans sa simplicité d'emploi : sauvegarde très rapide des variables d'un programme, utilisation ultérieure, etc.

• Le fichier SYS est nouveau. Ce type de fichier représente surtout un programme système, généralement écrit en langage machine (assembleur ou autre), dont l'accès se limite à une exécution directe en mémoire centrale. Pour charger en mémoire un fichier du type SYS, il faudra associer à la commande le paramètre TSYS.

Exemple

Le fichier ProDOS de la disquette utilitaire pourra se charger à l'adresse $2000 par la commande BLOAD ProDOS, TSYS, A$2000. Si le byte d'accès d'un fichier du type SYS est modifié en type BIN, celui-ci ne sera plus exécuté lors de son appel (non bootable).

• Le fichier DIR est nouveau. Ce type de fichier désigne une table des matières auxiliaire (Volume SubDirectory) et permet d'augmenter en nombre la capacité de sauvegarde des fichiers programmes ou SubDirectory.

Bytes $11-$12 Key_Pointer

Ce champ pointe un bloc de données d'un fichier lorsque celui-ci ne compte pas plus de 512 bytes. Dans le cas contraire, un bloc index pointe un ou plusieurs blocs de données. Si la taille du fichier est supérieure à $20000 bytes (131072 bytes ou 128 Ko), un Master Block sera rendu nécessaire. Un Master Block peut pointer 128 blocs index dont chacun peut à nouveau pointer 256 blocs de données. Dans le cas d'un fichier SubDirectory, le pointeur indique toujours le premier bloc du SubDirectory (Key Block du SubDirectory).

Bytes $15-$17 EOF – End Of File

Ce champ dispose de trois bytes pour représenter la longueur d'un fichier: LByte, MByte, HByte. Cette structure permet de pointer des valeurs comprises dans une fourchette allant de $000001 à $FFFFFF.

Exemple de valeur

LByte	MByte	HByte	
B2	12	01	= $0112B2

Formule de calcul

(LByte * 1) + (MByte * 256) + (HByte * 65536)

Le résultat du calcul précédent sera évalué en base 10, à condition d'avoir gardé les mêmes unités.
Le pointeur EOF désigne un byte logique, non matérialisé sur le disque. Il renseigne le système sur la longueur réelle d'un fichier.

Bytes $18-$1B Create_Date & Create_Time

Ce champ permet la sauvegarde de la date et de l'heure au moment de la création d'un fichier.

Bytes $1C-$1D Version & Mini_Version

Ce champ stocke les paramètres relatifs à la version de ProDOS ayant sauvegardé le fichier. Seul le byte $1D est testé par la commande OPEN.

Byte $1E Access_Byte

Ce paramètre suit les mêmes règles que celles faisant référence à une entrée d'un nom de volume.

Bytes $1F-$20 Aux_Type

Ce champ contient des informations complémentaires et représente une adresse d'implantation pour un fichier binaire par exemple. Certains types de fichiers n'utilisent pas ce champ de l'entrée correspondante.
Un fichier texte (TXT) du type aléatoire sauvegarde ici la longueur du dernier enregistrement effectué (LByte, HByte). Avec un fichier texte du type séquentiel, ces deux bytes sont inutilisés.

Pour les fichiers du type BIN et BAS, c'est l'adresse d'implantation en mémoire centrale qui est représentée (LByte, HByte).

Bytes $21-$24 Mod_Date & Mod_Time

Ce champ permet de stocker la date et l'heure lorsqu'un fichier est associé aux commandes MLI CLOSE et WRITE.

Bytes $25-$26 Header_Pointer

Pointe le numéro du Key Block Directory dans lequel se trouve sauvegardé le nom du fichier.

ProDOS16 ET LA GESTION DES FICHIERS

Ce chapitre, destiné particulièrement aux programmeurs de tout bord, permet de mieux cerner certains aspects faisant appel à la gestion des fichiers en général. Le paramètre Storage_Type, particularité des fichiers ProDOS, ainsi que les termes spécifiques faisant référence aux fichiers, sont passés en revue. Par la suite, de nombreux détails et astuces familiarisent le lecteur à cette pratique, permettant une meilleure compréhension des mécanismes exploités par ProDOS16 pour le traitement de ses fichiers.

Le traitement d'une application consiste à exécuter une suite de codes machine écrits dans un langage appelé langage machine. Un programme système sous ProDOS16 sera toujours du type $B3 à $BF. Il devra avoir comme extension de nom de fichier .SYS16 et être conforme au protocole établi par la commande QUIT.

TERMES PARTICULIERS AUX FICHIERS ProDOS

Volume Bit Map (VBM)

Chaque support de sauvegarde de masse (disque souple ou disque dur) nécessite pour sa gestion interne une table d'occupation, appelée généralement Volume Bit Map. Si plus de 256 blocs sont nécessaires pour le marquage de l'occupation des blocs logiques du support de sauvegarde, plusieurs bit map blocs sont chaînés à la suite. Un disque dur du type Profile peut facilement comporter après le formatage, six blocs pour sa table d'occupation (Volume Bit Map Blocks).

Dans le cas d'une VBM étendue, les blocs attribués à la table d'occupation sont disposés sur la surface du disque dans un ordre croissant des numéros des blocs. Par rapport à d'autres systèmes d'exploitation, ProDOS16 ne fait aucune référence au sujet de la VBM; seul le premier numéro du bloc se trouve copié dans le Header du Volume Directory, position relative $27-$28.

Le Volume Directory contient un pointeur (Parent_Pointer) qui désigne le premier bloc VBM, lequel est d'ailleurs l'unique bloc bit map pour un disque standard au format 5,25 pouces.

Dump de la Bit Map (bloc $06 du disque)

Cette bit map est celle d'un disque venant juste d'être formaté, encore exempt des fichiers système. Elle marque occupé uniquement les blocs réservés par le système.

$00	01	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
$10	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
$20	FF	FF	FF	00	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	00	00	00	00
$40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Représentation du byte $00

Blocs	0	1	2	3	4	5	6	7	
Bits	0	1	2	3	4	5	6	7	
	0	0	0	0	0	0	0	1	= $01
									= 1 bloc libre (bloc 7)

Le 7e bit représente le bloc 7 du disque.

Représentation du byte $01

Blocs	8	9	A	B	C	D	E	F	
Bits	8	9	A	B	C	D	E	F	
	1	1	1	1	1	1	1	1	= $FF
									= 8 blocs libres (8-15)

Remarques

- Un bloc de données consiste en une suite de 512 bytes. A l'intérieur de la VBM, chaque byte représente 8 blocs. Une VBM standard peut gérer $1000 ($200 * 8) blocs de données, ce qui équivaut à 2 mégabytes. Pour les disques 3,5 et 5,25 pouces cela suffit largement, mais pour un disque dur d'une capacité de 5 mégabytes, il faudra une VBM étendue : trois blocs VBM sont alors nécessaires, ce qui implique l'occupation des blocs $06, $07 et $08.

- Ce sont les commandes MLI CREATE & WRITE qui marquent les blocs occupés, tandis que les commandes DESTROY & SET_EOF libèrent les blocs.

Master Block

C'est un bloc répertoire principal alloué par ProDOS : il pointe par extension d'autres blocs index. Un Master Block devient indispensable à chaque fois que le nombre de blocs de données dépasse la valeur 256.

Structure type d'un Master Block

- un Master Block désigne au maximum 128 Index Blocks;
- chaque Index Block peut à nouveau pointer jusqu'à 256 Data Blocks;
- un Data Block a, par conception, une capacité de 512 bytes.

Un fichier avec un Storage_Type $03 nécessite un Master Block qui peut pointer au maximum 128 Index Blocks. La limite de ce type de bloc réside dans le codage du pointeur EOF à qui l'on a attribué trois bytes pour déterminer la longueur d'un fichier (3 bytes = 24 bits). La capacité de codage sur 24 bits est de 16 Mo (256 * 256 * 256). La taille maximale d'un fichier sous ProDOS16 est limitée à 16 Mo (128 * 256 * 512). Un tel fichier nécessite 128 blocs index dont chacun pointe 256 blocs de données, chaque bloc de données ayant une capacité totale de 512 bytes.

Index Block

ProDOS16 alloue un bloc index à un fichier lorsque celui-ci dépasse la taille de 512 bytes de données. Un bloc index peut pointer un maximum de 256 blocs de données. Si un deuxième bloc index est nécessaire, celui-ci sera chaîné au premier par un pointeur.

Un fichier avec un Storage_Type $02 nécessite un seul bloc index. Un fichier avec un Storage_Type $03 nécessite au maximum 128 blocs index. Les adresses des blocs de données, contenues à l'intérieur du bloc index, sont répertoriées sur deux pages : une page pour les pointeurs de poids faible et une page pour les pointeurs de poids fort. Les 256 premières valeurs contiennent les bytes de poids faible, les 256 bytes suivants, les bytes de poids fort.

Data Block

Un Data Block est réservé à la sauvegarde des données du fichier. Par définition et par conception, un bloc de données a une capacité maximale de 512 bytes. C'est la taille minimale attribuée par ProDOS à un fichier de données. Le contenu peut être quelconque.

Si un bloc de données est effacé par la commande MLI DESTROY, toutes les données restent néanmoins présentes au niveau du support. Le système libère ensuite les blocs dans la table des matières (VBM) et une mise à jour transmise. Un éventuel Master Block ou des blocs index sont purgés de leur contenu par la commande SET_EOF, via DESTROY. Un bloc index ainsi libéré ne contiendra en fin d'opération que des bytes nuls (00).

FORMAT DES FICHIERS (STORAGE_TYPE)

Généralités (Master, Index et Data Blocks)

ProDOS8 et ProDOS16 gèrent plusieurs Storage_Type qui sont des indices caractéristiques propres à la taille de chaque fichier. Ce paramètre est attribué au fichier au moment de la sauvegarde de ses données sur le disque et de ses paramètres dans sa table des matières correspondante.

• Un fichier d'une taille ne dépassant pas 512 bytes nécessite un seul bloc pour sa sauvegarde.

• Dès que sa taille dépasse 512 bytes, un bloc répertoire sera nécessaire. Le premier bloc de tout fichier supérieur à 512 bytes est le bloc index. Ce bloc index répertoire, appelé bloc index ou encore Index Block, contient les pointeurs des différents blocs de données du fichier. Le système ProDOS16 consulte au préalable le bloc répertoire avant d'aller lire et de charger en mémoire les données du fichier.

• Un fichier qui dépasse 128 Ko nécessite alors un Master Block (bloc maître). Le Master Block peut à nouveau pointer jusqu'à 128 blocs index. Chaque bloc index peut à nouveau pointer jusqu'à 256 blocs de donnéées. Le système ProDOS16 consulte en premier le Master Block et détermine par la suite les adresses des différents blocs index. Chaque bloc index renseigne le système sur les adresses de chaque bloc de données.

Différents Storage_Type possibles

$00 — Nul, pas de nom ou fichier effacé.

$01 — Seedling. C'est un fichier du plus petit format, comportant un nombre de bytes inférieur à 512 ($000 <= EOF <= $0200). Ce fichier n'occupe qu'un bloc de données sur le disque et n'a pas de bloc index. Les données sont directement pointées par le nom du fichier : pointeur sauvegardé aux bytes $11-$12 dans l'entrée de sa table des matières.

$02 — Sapling. C'est un fichier d'une taille supérieure à 512 bytes (un bloc), pouvant atteindre 128 Ko ($0200 < EOF <=$20000). Ce type de fichier comporte un seul bloc index qui peut pointer jusqu'à 256 blocs de données. Le premier bloc d'un fichier Sapling est le bloc index.

$03 — Tree. C'est un fichier dont la taille dépasse 128 Ko, pouvant atteindre les 16 Mo ($020000 < EOF < $1000000). Il nécessite plus d'un bloc index. Une telle structure nécessite alors un Master Block qui peut pointer de 1 à 128 blocs index, chaque bloc index peut à nouveau pointer 256 blocs de données. Un rapide calcul montre qu'un tel fichier peut contenir 16.777.216 bytes, soit 16 méga-octets (Mo). Le Master Index contient les pointeurs des blocs index qui contiennent eux-mêmes les adresses des blocs de données. Le premier bloc d'un fichier Tree est le Master Block.

$04 — USCD Pascal. Désigne un fichier d'une structure Pascal.

$0D — SubDirectory File. Ce paramètre est attribué à un nom de fichier qui se trouve dans un Volume Directory et qui pointe un Volume SubDirectory.

$0E — Volume SubDirectory. Ce paramètre est attribué au nom du volume qui se trouve dans la première entrée dans un Volume SubDirectory. C'est le nom du volume déclaré avec la commande CREATE par exemple.

$0F — Volume Directory. Ce paramètre est attribué au nom du volume qui se trouve dans la première entrée dans un Volume Directory. C'est le nom du disque déclaré lors du formatage.

CONFIGURATIONS POSSIBLES D'UN VOLUME DIRECTORY

Ce paragraphe effectue une synthèse complète des différentes possibilités de sauvegarde d'un fichier de données dont le Storage_Type est attribué en fonction de sa taille.

Disque venant juste d'être formaté

L'exemple traité par la suite est effectué sur un disque au format 5,25 pouces avec une capacité totale de 280 blocs. Un support de sauvegarde d'une capacité plus grande aura des blocs en nombre plus important, le raisonnement restant le même.

Structure du Directory après le formatage

Blocs 0-1	Loader (routine de chargement du système)
Blocs 2-5	Volume Directory
Bloc 6	Volume Bit Map
Blocs 7-279	Libres

Sauvegarde d'un fichier du type Seedling

Lorsqu'un fichier de données est créé par la commande CREATE, le système lui alloue d'office un bloc. Le nom du fichier sera stocké en deuxième position dans la table des matières principale (Volume Directory), à partir de la position $2B (43 en décimal) et pointe le bloc 0007. C'est le premier bloc de données alloué au fichier, celui-ci étant vide pour l'instant.

Création d'un nom de fichier par CREATE.

CREATE/VOLUME/PROGRAMME, TBAS

où,

CREATE est le nom de la commande ProDOS qui permet de créer un fichier vide de données.

/VOLUME/ est le nom du Volume Directory.

PROGRAMME est le nom du fichier.

TBAS est le paramètre qui détermine le type de fichier (BAS = Basic).

Structure du Directory après la création du fichier

Blocs 0-1	Loader (Routine de chargement du système)
Blocs 2-5	Volume Directory
Bloc 6	Volume Bit Map
Bloc 7	Data Block, bloc de données
Blocs 8-279	Libres

Allocation des blocs

1 bloc de données – vide pour l'instant.
Total : 1 bloc.

Sauvegarde d'un fichier du type Sapling

Dès qu'un fichier contient plus de 512 bytes de données, le système lui alloue un bloc index. Ce bloc particulier pointe à son tour des blocs de données avec une capacité maximale de 256 blocs. Chaque bloc de données nécessite 2 bytes pour le pointeur (LByte, HByte). Ce type de fichier se compose au minimum d'un bloc index et de deux blocs de données. La taille maximale d'un type de fichier Sapling est de 128 Ko.

Structure du Directory après la création du fichier

Blocs 0-1	Loader (Routine de chargement du système)
Blocs 2-5	Volume Directory
Bloc 6	Volume Bit Map
Bloc 7	Data Block 0, premier bloc de données
Bloc 8	Index Bloc 0, bloc index
Bloc 9	Data Block 1, deuxième bloc de données
Blocs 10-279	Libres

Allocations des blocs

- 1 bloc index
- 2 blocs de données

T: 3 blocs.

Sauvegarde d'un fichier du type Tree

Si la capacité d'un fichier dépasse les 128 Ko, il nécessite alors plus d'un bloc index. Le système crée alors un bloc répertoire principal (Master Block) qui peut pointer 128 blocs auxiliaires (Index Blocks); chaque bloc index peut à son tour pointer 256 blocs de données. Un tel type de fichier peut à lui seul avoir une capacité de 16 méga-octets (16 777 716 bytes).

Exemple de création d'un fichier Tree

Pour illustrer la création d'un fichier Tree, le programme Basic qui suit permet de créer sur un disque, un fichier texte avec la structure suivante :

- 1 Master Block;
- 2 blocs index;
- 257 blocs de données.

L'écriture sur le disque prend un certain temps : elle marque 260 blocs, ce qui représente 90 % de la capacité de sauvegarde. Le détail du Directory est donné par la suite.

Programme de mise en place d'un fichier Tree

```
10 REM Fichier Tree.
20 PRINT CHR$ (4); 'OPEN FICHIER'
30 PRINT CHR$ (4); 'WRITE FICHIER'
40 REM  Mise en place de 257 blocs de données :
        1 Master Block et 2 Index Blocks
50 FOR X = 1000000 T0 1016384 : PRINT X
60 NEXT X
70 PRINT CHR$ (4); 'CLOSE FICHIER'
```

Structure du Directory après la création du fichier

Blocs 0-1	Loader (Routine de chargement du système)
Blocs 2-5	Volume Directory
Bloc 6	Volume Bit Map
Bloc 7	Data Block 0, premier bloc de données
Bloc 8	Index Block 0, premier bloc index
Bloc 9	Data Block 1, deuxième bloc de données
Bloc 10	Data Block 2, troisième bloc de données
.....	.....
Bloc 263	Data Block 255, 256e bloc de données
Bloc 264	Master Index Block, bloc répertoire principal
Bloc 265	Index Block 1, deuxième bloc index
Bloc 266	Data Block 256, 257e bloc de données
Blocs 267-279	Libres

Allocations des blocs

- 1 bloc index principal – Master Block;
- 2 blocs index – Index Block;
- 257 blocs de données.

Total : 260 blocs.

GENERALITES SUR LES FICHIERS

Fichiers sous ProDOS

Directory, SubDirectory et programmes

Un disque ProDOS peut contenir plusieurs types de fichiers: les fichiers tables des matières principales, auxiliaires (DIR); et les autres qui regroupent les fichiers programmes (TXT, BAS, VAR, BIN, REL, SYS, ceux du type $Fx, etc.). A partir d'un environnement Basic.System, un certain nombre d'instructions permettent une manipulation aisée avec les divers fichiers.

COMMANDES DE GESTION GLOBALE D'UN FICHIER

CAT ou CATALOG	Pour visualiser le contenu d'un Directory.
PREFIX	Pour définir un nouveau préfixe ou pour connaître le préfixe actuel.
CREATE	Pour placer un nom de fichier dans le Directory d'une disquette (Volume Directory ou Volume SubDirectory).
RENAME	Pour changer le nom d'un fichier.
DELETE	Pour supprimer le nom d'un fichier.
LOCK	Pour verrouiller un fichier et empêcher toute action des instructions RENAME et DELETE.
UNLOCK	Pour déverrouiller un fichier. Cette instruction est complémentaire de LOCK.

COMMANDES D'UTILISATION D'UN FICHIER

-	Ou Dash, sert à lancer l'exécution d'un programme.
RUN	Pour charger en mémoire et lancer l'exécution d'un programme écrit en langage Basic.
LOAD	Pour charger en mémoire un programme Basic, sans l'exécuter.
SAVE	Pour sauvegarder sur la disquette un programme Basic.
BRUN	Pour charger en mémoire et exécuter un programme écrit en langage machine.
BLOAD	Pour charger en mémoire un programme machine, sans l'exécuter.
BSAVE	Pour sauvegarder sur le disque un programme machine.

COMMANDES DE PROGRAMMATION D'UN FICHIER

CHAIN	Pour enchaîner l'exécution de deux programmes tout en préservant les variables.
STORE	Pour sauvegarder les noms et les valeurs des variables.
RESTORE	Pour charger en mémoire les variables et leurs noms sauvegardés par STORE.
PR£/IN£	Pour sélectionner un périphérique d'entrées/sorties en ligne, autre que l'écran et le clavier.

TYPES DE FICHIERS STANDARDS

DIR	Désigne une table des matières, ou sous-catalogue: Directory ou SubDirectory.
TXT	Fichier de données, exploitable par l'utilisateur et appelé généralement fichier texte (Text File). Cette catégorie regroupe aussi les fichiers EXEC.
BAS	Ce sont les programmes écrits en langage Basic AppleSoft.
VAR	Ce type de fichier contient la sauvegarde des variables d'un programme Basic AppleSoft.
BIN	Fichier binaire qui contient des données codées en langage machine ou autre.
REL	Programme binaire en format relogeable (Relocatable); il peut, à l'aide d'un utilitaire adéquat, se charger en mémoire et y être exécuté à n'importe quelle adresse.
$F£	Désigne un type de fichier pouvant être redéfini par l'utilisateur. L'indice £ peut prendre comme valeur un nombre compris entre 1 et 8 inclus. Donc ce sont encore 8 types de fichiers qui peuvent être nomenclaturés au choix du programmeur.

DATE DE CREATION ET DERNIERE MODIFICATION

La date et l'heure de création ou de la dernière modification d'un fichier sont des attributs copiés dans l'entrée correspondante du nom du fichier. ProDOS16 alloue à chaque entrée d'un fichier dans la table des matières, un total de quatre bytes: deux bytes pour la date et deux autres bytes pour l'heure. Ce sont les variables Create_Date, positions relatives $18-$19, et Create_Time, positions relatives $1A-1B, qui représentent les paramètres.

Représentation du paramètre Create_Date

Bits	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
	A	A	A	A	A	A	A	M	M	M	M	J	J	J	J	J

A = Année M = Mois J = Jour

Représentation du paramètre Create_Time

Bits	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
	0	0	0	H	H	H	H	H	0	0	M	M	M	M	M	M

H = Heure
M = Minute

Catégories de fichiers

Les fichiers se regroupent en deux grandes catégories :

- **Fichiers tables des matières Directory & SubDirectory**

L'entrée d'un fichier Directory pointe toujours une extension d'un Volume Directory. Un Directory désigne deux types possibles : une table des matières principale et une table des matières auxiliaire.

La table des matières principale (Volume Directory) peut contenir des noms de fichiers programmes et des noms de fichiers tables des matières auxiliaire (Volume SubDirectory). Chaque table des matières auxiliaire, ou Volume SubDirectory, peut à son tour contenir des fichiers programmes ou tables des matières auxiliaire.

Le Volume Directory est créé une fois pour toute lors du formatage du volume : c'est la première table des matières d'une disquette ProDOS d'où le nom de table des matières principale.

- **Fichiers de données – Data File**

Ce sont les types de fichiers autres que les fichiers Directory. Ces types de fichiers ont tous la même particularité : ils pointent les données d'un programme ou d'un fichier texte et nécessitent un bloc index (Index Block) lorsque leur taille dépasse 512 bytes.

Notion de fichier

Le système ProDOS16 ignore le bit de poids fort des caractères dans un texte et change d'office le jeu de caractères : c'est ainsi que tous les codes ASCII rentrés au clavier ont leurs bits 7 à 0 alors que le DOS 3.3, lui, les met à 1.

ProDOS accepte les minuscules, et lorsque celles-ci sont attribuées à des commandes ProDOS, il les interprète comme des majuscules. Un nom de fichier ne doit former qu'un ensemble de noms indissociables.

Exemples corrects

LETTRE
PROGR.UTIL
JEU.08

Exemples incorrects

1LETTRE
PROGR UTIL

Comme le nom d'un volume, un nom de fichier suit les mêmes règles et contraintes d'écritures. Sous ProDOS8, un chemin complet, partiel ou un préfixe peut comporter un maximum de 64 caractères avec le '/' inclus. Il doit impérativement débuter par une lettre et peut se composer de lettres, de chiffres ou de points. Il peut être complet ou partiel.

Notion de chemin

Un chemin est une liste de tables des matières, principale ou auxiliaire, qui désigne le chemin d'accès à travers la structure hiérarchisée du volume pour arriver au fichier que l'on veut exécuter par exemple. Un chemin complet, partiel ou un préfixe possède chacun une syntaxe propre. Préfixe et chemin partiel s'ajoutent au moment de la recherche, ce qui permet d'accroître le nombre de caractères. La limitation en nombre de caractères des différents chemins n'est pas la même sous ProDOS8 ou ProDOS16.

Chemin partiel (Partial Pathname)

Un chemin partiel est une structure qui sera complète par le préfixe. Ce type de chemin d'accès est une portion du chemin complet qui situe le nom du fichier dans le volume considéré. Un chemin partiel ne débute jamais par le '/' suivi d'un nom de volume, mais se contente uniquement du nom d'un fichier SubDirectory et du nom de l'application recherchée.

Remarques

- en ce qui concerne ProDOS8, la longueur d'un tel chemin est limitée à 64 caractères.

- la version 2.0 de ProDOS16 autorise un chemin partiel pouvant atteindre 128 caractères.

Syntaxe : SUBDIRECTORY/UTILITAIRE

où SUBDIRECTORY/ désigne la table des matières courante

et UTILITAIRE le fichier recherché.

Préfixe – PREFIX

Le préfixe est un chemin par défaut qu'il sera nécessaire d'ajouter à celui que l'on veut spécifier. ProDOS additionne de lui-même le préfixe au début d'un chemin partiel. Le préfixe est un chemin valide déclaré au préalable par la commande PREFIX. ProDOS effectue une recherche suivant le nom du volume déclaré et teste par la suite tous les périphériques en ligne.

Remarques

- sous ProDOS8, la longueur maximale d'un préfixe est de 64 caractères, tandis que sa longueur minimale peut être réduite à zéro, ce qui annule sa déclaration préalable. Un Pathname, association préfixe et chemin partiel,

autorise une longueur de 128 caractères. Le slash au début de la déclaration du préfixe est obligatoire, tandis que celui de la fin est facultatif.
- la version 2.0 de ProDOS16 modifie quelque peu la notion de préfixe. Un préfixe peut maintenant atteindre une taille de 128 caractères valides.

Syntaxe : /USERS.DISK/SUBDIRECTORY/

où /USERS.DISK est le nom du volume

et /SUBDIRECTORY/ le nom du Volume SubDirectory par exemple.

Chemin complet – (Pathname)

Un chemin complet débute toujours par le nom du volume, et se trouve complété d'un ou de plusieurs noms de fichiers. Cette succession de noms désigne le chemin d'accès à un fichier sauvegardé dans une table des matières, ce dernier pouvant être du type SubDirectory. Ce type de chemin est toujours précédé par le '/' qui est un slash. Chemin et préfixe s'ajoutent, ce qui augmente le nombre de caractères.

Remarques

- sous ProDOS8 un chemin complet autorise un maximum de 128 caractères;
- la version 2.0 de ProDOS16 autorise un chemin complet de 128 caractères.

Exemples de Pathname

/USERS.DISK/UTILITAIRES/NOMFICH

où

/USERS.DISK est le nom du Volume Directory;

/UTILITAIRES/ est le nom du Volume SubDirectory;

et NOMFICH le nom d'un programme.

Création d'un fichier

• CREATE est la seule commande MLI permettant de créer un nouveau fichier. Il pourra être du type table des matières (Volume SubDirectory) ou fichier de données. ProDOS lui alloue alors un bloc par défaut. Ce bloc, vide initialement, sera réservé pour la sauvegarde des données futures.
• Le nom de fichier ainsi créé, pourra être modifié par la commande RENAME et effacé par DELETE, à condition que le fichier ne soit pas verrouillé. Si c'est un fichier table des matières (Directory), le Volume SubDirectory pointé ne devra pas contenir de programmes pour permettre son effacement par la commande DELETE, mais RENAME pourra le renommer.

Identité d'un fichier

Chaque fichier se caractérise par un certain nombre d'attributs que le système lui accorde à un moment ou à un autre pendant le traitement. Les principales caractéristiques sont les suivantes :

• **chemin**

Déclaré lors de la création d'un fichier, il devra obligatoirement être celui utilisé pour la recherche ultérieure de ce même fichier. Le nom du fichier sera placé derrière le nom du Directory existant.

• **byte d'accès**

Il est matérialisé dans la table des matières par l'astérisque '*', dont la valeur interne détermine si le fichier pourra être effacé ou renommé, c'est-à-dire s'il est verrouillé ou déverrouillé; le premier état permettant la lecture seule et le second la lecture et l'écriture du fichier.

• **type de fichier**

Le byte qui le caractérise détermine son format physique au moment de la sauvegarde sur le disque et n'affecte pas son contenu.

• **byte caractéristique**

Cet attribut détermine si le fichier est une table des matières principale ou auxiliaire, ou encore simplement un fichier programme : Volume Directory, Volume SubDirectory ou fichier de données.

• **date et heure**

Possibilité de gestion de la date et de l'heure.
- format de la date : Année/Mois/Jour
- format de l'heure : Heure/Minute

FICHIER TEXTE

Généralités

La structure d'un fichier texte (TXT) est utilisée pour traiter des données d'une façon simple et rationnelle. Par la suite, ces données écrites sont susceptibles d'être modifiées ou l'écriture à un emplacement précis souhaité. Tout fichier ouvert devra être fermé avant de quitter le programme, sinon les données risquent fort d'être perdues.

Un fichier texte peut être aléatoire, séquentiel ou exec.

Les fichiers texte sont répertoriés dans la table des matiéres à l'aide de leurs noms. L'attribut Access_Byte, position relative $1E de l'entrée nom du fichier dans la table des matières, donne l'état du fichier: verouillé (LOCK) ou déverouillé (UNLOCK). C'est le bit 7 (Destroy bit) qui détermine si l'accès au fichier permet l'écriture, ou s'il n'autorise que la lecture.

Représentation du paramètre Access_Byte

	7	6	5	4	3	2	1	0
Bits	D	N	B	0	0	0	W	R

Le bit à 1 autorise la fonction.
Le bit à 0 n'autorise pas la fonction.

Signification des différents bits
ProDOS version 1.1.1

Bit 7 — Destroy Enable Bit. Il détermine si le nom du fichier peut être effacé ou non par la commande MLI DESTROY (DELETE en Basic). Avec un Access_Byte de valeur $C3, le fichier pourra être effacé.

Bit 6 — Rename Enable Bit. Si ce bit est à 1, il sera permis de modifier le nom du fichier par l'instruction RENAME.

Bit 5 — Backup Needed Bit. Ce bit informe le système qu'une donnée peut être copiée au non. Cette situation est mise à profit par les commandes MLI CREATE, RENAME, CLOSE, WRITE et

SET_FILE_INFO. A l'appel d'une de ces routines, le bit 5 sera positionné à 1 pour permettre la copie des données. Au retour des sous-programmes, le bit 5 sera remis à 0 par une routine implantée en $D078 (EXIT).

```
- EXIT
D078- A9 00       LDA £$00
D07A- 8D 95 BF    STA $BF95 BACKUP
D07D- ::::::::::::::::::
```

A l'adresse $E9D9 (SET_FILE_INFO command), le système effectue un OU exclusif pour incorporer la valeur du bit 5 dans le byte d'accès:

```
E9D9- AD 95 BF    LDA $BF95
E9DC- 49 20       EOR £$20
E9DE- 2D 3D F0    AND $F03D
E9E1- 29 20       AND £$20
E9E3- 8D 74 F0    STA $F074
E9E6- ::::::::::::::::::
⇒ où £$20 = 00100000
```

Bits 4-2 — Inutilisés, donc toujours positionnés à 0.

Bit 1 — Write Enable Bit. Autorise l'écriture des données, si le bit est à 1.

Bit 0 — Read Enable Bit. Autorise la lecture des données, si le bit est à 1.

Structure d'un fichier texte

ENREGISTREMENT 1 (RETURN)

ENREGISTREMENT 2 (RETURN)

ENREGISTREMENT FIN (RETURN) (NUL)

Remarque

Un enregistrement est un ensemble de caractères et de signes écrits sur le support disque avec des codes ASCII dont le bit 7 est à zéro. L'écriture se trouve sous la forme d'une suite de codes ASCII et d'une longueur déterminée par avance. RETURN est le code ASCII £$0D; NUL est le code ASCII £$00. NUL n'est pas apparent après un enregistrement normal, mais seulement dans le dernier enregistrement.

FICHIERS TEXTE SEQUENTIELS

Les fichiers séquentiels sont des enregistrements réalisés dans un ordre structuré en séquences. Pour lire ou écrire dans un champ donné de ce type de fichier, il faudra d'abord lire ou écrire toutes les rubriques précédentes. Les caractéristiques énoncées par la suite sont valables sous un environnement Basic et ProDOS8 actif.

Caractéristiques d'un fichier séquentiel

- Il se compose d'une suite de séquences, dont chacune peut se composer:
  - d'une suite de 0 ou de caractères;
  - d'un caractère RETURN (retour-chariot code 13);
- La première séquence a le numéro 0;
- La dernière séquence possible est 65 535;
- Pour placer une valeur dans une séquence, il faut utiliser l'instruction PRINT, suivie des caractères à placer dans le fichier;
- Pour relire une séquence il faut utiliser l'instruction INPUT;
- Une séquence peut se composer de rubriques, et chaque rubrique est séparée de la précédente par une virgule;
- La lecture des rubriques d'une séquence donnée se fait par INPUT (les variables successives qui lui sont affectées iront directement lire les rubriques respectives des séquences).
- Comme la virgule est utilisée pour séparer les rubriques, il ne sera pas possible d'utiliser INPUT pour lire un champ contenant une virgule. A cette occasion l'instruction GET sera tout indiquée en testant la présence du caractère retour chariot.

FICHIERS TEXTE ALEATOIRES

Les fichiers aléatoires sont structurés dans une suite d'enregistrements, lesquels peuvent à nouveau se subdiviser en un ou plusieurs champs. Chacun de ces enregistrements, ou suite de champs, contient un nombre identique d'informations définies en bytes (caractères). Ils sont plus simples à utiliser que les fichiers séquentiels. Ils permettent de lire et d'écrire dans n'importe quel champ, quelle que soit sa position.

Caractéristiques d'un fichier aléatoire

- La longueur totale d'un fichier aléatoire est déterminée par la somme des enregistrements;
- Un enregistrement dans un fichier aléatoire est déterminé par la somme des champs de données;
- Un champ de données est suivi par un retour-chariot, puis par un autre champ de données suivi à nouveau par un retour-chariot, etc.;
- La somme des champs de données forme un enregistrement qui se termine par un retour-chariot;
- Un champ de données est la plus petite partie contenue dans un fichier aléatoire;
- Chaque enregistrement se termine par un retour-chariot (RETURN);
- Les enregistrements seront toujours de la même longueur;
- Si la sauvegarde d'un enregistrement ne comporte pas le nombre de caractères ASCII déclarés par le paramètre L£, le système les complète à l'aide de bytes nuls (Ctrl-à);
- Si un enregistrement comporte plusieurs champs de données, la longueur des champs les plus courts sera complétée par des bytes nuls (Ctrl-à):
- Le nombre d'enregistrements n'est pas absolu;
- Pour placer des données dans le fichier, il faudra préciser le numéro de l'enregistrement à utiliser par l'instruction WRITE;
- Les enregistrements peuvent être adressés dans un ordre quelconque;
- Le retour chariot qui sépare deux enregistrements ou deux champs de données, est comptabilisé pour la longueur;
- A la création d'un fichier, il sera nécessaire d'indiquer la longueur de l'enregistrement. Cette longueur sera stockée dans la table des matières. La longueur par défaut sera de 1 sous ProDOS;
- Lorsque le nom du fichier texte existe, l'option longueur sera facultative et le système utilisera la valeur spécifiée lors de sa création. Si pour une raison particulière, vous précisez une taille différente lors de la réouverture d'un fichier, cette valeur sera prise en considération pendant cette ouverture. La valeur spécifiée au cours de la création du fichier ne sera pas modifiée par la suite;
- La longueur déclarée devra être supérieure à celle de l'enregistrement à effectuer.

FICHIERS EXEC

Un fichier EXEC est un fichier de commande du type séquentiel. Il contient des lignes de commandes ProDOS ou AppleSoft en mode direct, telles qu'elles seraient tapées directement à partir du clavier. Ces commandes sont exécutées par l'instruction EXEC ou par le '–'(dash) et permettent l'exécution d'une suite automatique d'ordres et de commandes précises.

TRAITEMENT D'UN FICHIER TEXTE

Ouverture d'un fichier texte

Avant toute opération de lecture ou d'écriture dans un fichier texte, il faudra ouvrir celui-ci par la commande OPEN qui appelle son sous-programme MLI. Pour ouvrir un fichier texte, il faut spécifier au préalable: le chemin et l'adresse du tampon d'entrée/sortie mémoire.

Chemin d'accès

Si le fichier est du genre séquentiel, seul le chemin comportant le nom du fichier, le connecteur et le lecteur de disques où se trouve le fichier sera permis. Si par contre le fichier est du genre aléatoire, l'option longueur (valeur entre 1 et 65 535) sera tolérée. Si par ailleurs, le type de fichier est autre, le paramètre T pourra y être adjoint (BAS, BIN, etc.).

Adresse mémoire – I/O_Buffer

L'adresse du tampon mémoire est gérée par le système; celui-ci alloue 1 024 octets pour chaque fichier ouvert et provoque une décrémentation de HIMEM du même ordre de grandeur.

Sous ProDOS8, un maximum de 8 fichiers peuvent être ouverts simultanément, tandis que ProDOS16 ne possède aucune limitation. Si un fichier est déjà ouvert et qu'il est à nouveau sollicité par OPEN, le système renverra le texte suivant: FILE ALREADY OPENED.

Fermeture d'un fichier texte

Instruction CLOSE

Sous environnement Basic, l'instruction CLOSE ferme tout fichier ouvert au moment où elle est transmise à travers le système. Lorqu'un fichier est ouvert, ProDOS se réserve en mémoire un tampon de 1024 octets pour y placer des données avant d'effectuer des transferts périodiques sur le disque.

La commande CLOSE permet donc de forcer le vidage de ce tampon de travail sur le disque et de libérer en même temps la place utilisée en mémoire centrale.

Si plusieurs fichiers sont ouverts, alors CLOSE les ferme tous; si par contre un fichier particulier doit être fermé, CLOSE sera suivi du paramètre ou chemin particulier à ce fichier.

CLOSE peut être déclaré en mode immédiat ou par programme et n'est pas rendu nécessaire lors de la lecture seule d'un fichier.

Si une erreur est rencontrée lors de l'utilisation d'un fichier ouvert, CLOSE en mode immédiat fermera ce fichier. Tout autre fichier ouvert sera fermé par la même occasion.

Instruction FLUSH

Cette instruction Basic sauvegarde sur le disque les données encore contenues dans le tampon d'un fichier ouvert, sans pour autant le fermer. Si la commande FLUSH est transmise en présence de plusieurs fichiers ouverts au même instant, alors tous les tampons des fichiers actuels seront sauvegardés sur le disque.

Lorsque plusieurs fichiers sont ouverts et que l'on désire effectuer une sauvegarde sélective, l'instruction FLUSH devra être suivie par le nom du fichier concerné.

Lecture/écriture dans un fichier

Les sous-programmes MLI READ et WRITE, permettent le transfert des données entre un fichier et la mémoire centrale. Avant l'appel de ces sous-programmes, il est nécessaire de spécifier certains paramètres:

- Le numéro de l'enregistrement à l'intérieur du fichier ouvert: il est spécifié par le paramètre R£.

- L'ouverture d'un tampon mémoire (buffer) qui gère les données entre le fichier qui se trouve sur le disque et la mémoire de l'Apple: le système utilise un tampon par défaut.

- La position de l'enregistrement dans le fichier: elle est indiquée par un nombre de bytes à ignorer à partir de la position courante.

Remarque

Il est impératif de fermer un fichier texte par la commande CLOSE, avant de quitter le traitement en cours ou avant la frappe des touches CTRL-RESET, sous peine de perdre le bénéfice des données se trouvant encore dans le tampon utilisé. En effet, le système utilise un tampon de 1024 octets pour effectuer le transfert des données entre le disque et la mémoire. La commande FLUSH permet d'effectuer un vidage instantané du tampon actif en sauvegardant les données sur le disque.

Structure et sauvegarde d'un fichier

Un fichier Directory (DIR) occupe généralement moins de place sur le disque qu'un fichier de données. Par ailleurs, son accès est en principe du type séquentiel: ProDOS utilise une forme simplifiée de sauvegarde sur le disque. Par contre, tous les types de fichiers suivent les mêmes règles concernant la disposition et l'allocation du nombre de bytes par bloc. Les données sont stockées sur le support de sauvegarde, par blocs de 512 bytes.

Si un fichier texte se compose de 512 bytes ou moins, alors son nom stocké dans le Directory pointe directement le bloc qui contient ses données. Si, par contre, celui-ci comporte plus de 512 bytes, alors le nom (fichiers) pointe au préalable un bloc (Index Block) contenant la liste des blocs de données. Cette liste s'appelle la Blocks List ou Index Blocks. Si le fichier nécessite plusieurs blocs index, ceux-ci sont chaînés entre eux par des pointeurs avant et arrière, permettant au système d'effectuer une recherche sélective.

COMMANDES RELATIVES A UN FICHIER TEXTE

Commandes et syntaxes

La gestion et le traitement des données d'un fichier du type texte (TXT) imposent l'utilisation d'une syntaxe définie au préalable. Sous environnement Basic, un certain nombre d'instructions leur sont particulièrement destinées, permettant à l'utilisateur averti d'effectuer des manipulations à travers des applications diverses.

OPEN	Pour ouvrir et créer un fichier.
CLOSE	Pour fermer un ou plusieurs fichiers ouverts.
WRITE	Pour sélectionner un fichier en mode écriture.
READ	Pour sélectionner un fichier en mode lecture uniquement.
APPEND	Pour ajouter des données à la fin d'un fichier.
FLUSH	Pour vider le tampon d'un ou de plusieurs fichiers encore ouverts. Permet de transférer les données du tampon mémoire sur le disque par exemple.
POSITION	Pour désigner un emplacement particulier à l'intérieur d'un fichier et pouvoir par la suite y lire ou y écrire des données.

Description des Commandes

Instruction APPEND

Cette instruction est utilisable uniquement à partir d'un programme et permet l'ajout de données à la fin d'un fichier.

Syntaxe : APPEND NOM, L£, S£ D£

où,

NOM représente le nom du fichier ouvert
L£ représente la longueur du fichier
S£ et D£ représentent respectivement le slot et le drive.

APPEND effectue automatiquement l'ouverture d'un fichier, la recherche de la fin de celui-ci et l'exécution de l'instruction WRITE avec le nom du fichier courant. Cette disposition particulière permet de placer des informations dans le fichier, à la suite d'un PRINT.

Avec un fichier aléatoire, il est permis de préciser la longueur de l'enregistrement; si celle-ci correspond à celle de sa création, APPEND place les données dans le premier enregistrement qui suit la fin du fichier. Si la longueur est différente de celle utilisée au moment de la création du fichier, ProDOS calcule la position du premier caractère suivant la dernière sauvegarde. Le WRITE implicite dans la commande APPEND prend fin à la rencontre de la prochaine commande ProDOS. Après avoir ajouté des données dans un fichier texte, il faudra fermer celui-ci par CLOSE.

Instruction CLOSE

Cette instruction ferme un fichier texte et par la même occasion indique la fin de son utilisation. La commande est réalisable en mode immédiat ou à l'intérieur d'un programme.

Syntaxe : CLOSE NOM

où,

NOM représente le nom du fichier.

La commande CLOSE peut être déclarée seule ou suivie d'un nom de fichier. Elle a pour fonction de fermer le ou les fichiers encore ouverts. La conséquence directe en sera la sauvegarde sur un disque des données encore stockées dans le tampon mémoire. Si elle n'est pas suivie d'un nom de fichier, cette commande ferme tous les fichiers ouverts, force le vidage des tampons sur le disque et libère de la place en mémoire.

Instruction FLUSH

Cette instruction s'exécute soit en mode immédiat, soit à partir d'un programme. Elle force le vidage du tampon mémoire et sauvegarde les données sur un disque par exemple.

Syntaxe : FLUSH NOM

où,

NOM représente le nom du fichier.

Lorsque ProDOS travaille avec un fichier texte, il utilise une technique de sauvegarde particulière. En effet, lors des accès d'écriture ou de lecture, les instructions PRINT transfèrent les données d'un fichier, par blocs, dans un tampon mémoire. Cette pratique est rendue nécessaire afin d'éviter tout

abus en ce qui concerne la sollicitation du drive (accès pour chaque octet par exemple). La déclaration de l'instruction OPEN force ProDOS à allouer un tampon d'une taille de 1 024 octets à chaque fichier texte. Au fur et à mesure du remplissage du tampon mémoire, le système effectue des sauvegardes ponctuelles sur le disque. Si, pour une raison particulière, l'on désire effectuer une sauvegarde des données du tampon, sans pour autant fermer le fichier, la commande FLUSH sera de rigueur.

Instruction OPEN

Cette instruction se transmet uniquement à partir d'un programme et permet d'ouvrir un fichier texte pour y placer ou retirer des informations.

Syntaxe : OPEN NOM, S£, D£, L£, Ttype

où,

NOM représente le nom du fichier.

S£ et D£ sont des paramètres qui désignent respectivement le slot et le drive.

L£ est le paramètre qui désigne la longueur d'un fichier aléatoire.

Ttype désigne le type de fichier.

Instruction POSITION

Cette instruction, uniquement autorisée à partir d'un programme, permet de se positionner à l'intérieur d'un fichier pour y lire ou y écrire des données. Elle est commune aux fichiers séquentiels et aléatoires.

Syntaxe : POSITION NOM, F£, B£

où,

F£ désigne un chiffre qui permet d'ignorer un nombre de champs d'enregistrement représentés par £.

B£ désigne le nombre d'octets à ignorer et représentés par £.
Uniquement exploitable avec un fichier séquentiel.

Instruction READ

Cette instruction, uniquement utilisable à partir d'un programme, permet de lire des données à l'intérieur d'un fichier texte.

Syntaxe : READ NOM, R£, F£, B£

où;

NOM désigne le nom d'un fichier.

R£ désigne le numéro de l'enregistrement dans un fichier aléatoire.

F£ désigne le nombre de champs d'enregistrement à sauter.

B£ désigne le nombre d'octets à sauter.

Instruction WRITE

Cette instruction, transmissible uniquement à partir d'un programme, permet d'écrire dans un fichier texte des données ou des informations diverses.

Syntaxe : WRITE NOM, R£, F£, B£

où,

NOM désigne le nom du fichier.

R£ désigne le numéro de l'enregistrement dans un fichier aléatoire.

F£ désigne le nombre de champs d'enregistrement à ignorer.

B£ désigne le nombre d'octets à ignorer.

POINTEURS D'UN FICHIER TEXTE

Pointeurs EOF et MARK

Lorsqu'un fichier texte est ouvert par la commande OPEN pour y permettre l'écriture de données, un premier pointeur met à jour l'adresse de fin, tandis qu'un autre se déplace à l'intérieur du fichier, au fur et à mesure des enregistrements. Ces pointeurs, appelés EOF (End Of File) et MARK (position actuelle du caractère considéré), désignent respectivement un pointeur logique et un pointeur physique à l'intérieur du fichier considéré.

POINTEUR MARK

Le pointeur MARK trouve son utilisation à travers les commandes MLI SET_MARK et GET_MARK (ProDOS8 et ProDOS16). Le pointeur est sauvegardé en mémoire dans le File Control Block aux positions relatives £$12-£$14, de l'entrée de la table des matières qui correspond au fichier traité. Le FCB se trouve en mémoire aux adresses $F300-$F3FF et peut contenir jusqu'à 8 entrées de paramètres dans une table. Chaque fichier se voit alloué 32 bytes qui forment alors une entrée.

MARK est un pointeur qui correspond à la position actuelle à l'intérieur d'un fichier texte. Il est utilisé lors des accès disque et déterminé par un paramètre associé à une commande ProDOS. La valeur du pointeur est calculée par une routine MLI, à partir d'une position absolue dans un fichier texte ($000000) et désigne le caractère actuel à lire ou à écrire. Ce pointeur se trouve désigné sous le nom de **Position**. Sous environnement Basic, et en association avec les instructions READ et WRITE, le pointeur MARK effectue la somme équivalente des paramètres R£, F£ et B£ déclarés à partir d'une position absolue dans le fichier.

POINTEUR EOF

Ce pointeur trouve son utilisation associée avec les commandes MLI SET_EOF et GET_EOF (ProDOS8 et ProDOS16). Le pointeur est sauvegardé en mémoire dans la File Control Block, aux positions relatives £$15-£$17 de l'entrée de la table qui correspond au fichier traité. Lorsque le fichier est sauvegardé sur un disque, EOF est mis à jour dans la table des matières du fichier correspondant, positions relatives $15 à $17 de son entrée. Il occupe 3 bytes et désigne la longueur d'un fichier texte.

EOF pointe la fin des données et matérialise ainsi la longueur d'un fichier qui peut prendre toute valeur comprise entre .$000001 et $FFFFFF (LByte, MByte, HByte). EOF sous-entend la position d'un byte pointé dans un fichier, juste à la suite du dernier caractère de fin, qui est un retour-chariot (code ASCII $0D). Cette position de fin est matérialisée par un byte logique, dont le pointeur a comme valeur MARK + 1; ProDOS considère ce byte comme imaginaire et n'est donc pas matérialisé sur le disque au moment de l'enregistrement.

Le pointeur EOF désigne une longueur relative, prenant son origine au début du fichier texte et avec la valeur 1 par défaut.

CONCLUSION

Ces pointeurs se trouvent gérés automatiquement par le système ProDOS, ce qui permet d'écrire à une place bien précise en indiquant au préalable la position ou le nombre de bytes à ignorer. Des paramètres d'extension, associés aux instructions WRITE et READ, permettent une meilleure gestion des fichiers.

Applications autorisées

– dans un fichier aléatoire, il est possible de préciser le numéro d'enregistrement pour l'écriture ou la lecture, par l'intermédiaire du paramètre R£.

– le paramètre B£ sert à ignorer un nombre de bytes déterminés par l'indice £, à partir de la position courante.

– le paramètre F£ permet d'ignorer un certain nombre de champs d'enregistrement ProDOS effectue alors le comptage des retours-chariot (RETURN = code $0D) dont le nombre limite est fixé par l'indice £. Un retour-chariot délimite en somme une ligne de texte d'un enregistrement effectué sur le disque, dont la longueur équivaut à un champ de données.

Particularité relative aux pointeurs

L'instruction Basic OPEN, à l'intérieur d'un fichier ouvert, positionne le pointeur sur le premier byte déclaré par le paramètre. A chaque écriture ou lecture dans ce fichier, le pointeur est automatiquement incrémenté de la valeur d'un byte, la somme des bytes étant égale à la taille de l'enregistrement. Si par exemple le pointeur se déplace 25 fois, 25 bytes sont écrits dans le fichier.

Lorsqu'un byte est écrit ou lu dans un enregistrement, la marque courante se déplace d'une unité pour pointer la position byte +1. Cet emplacement correspond au prochain byte à écrire ou à lire. Cependant, la position extrême de lecture/écriture dans un fichier ne pourra excéder sa taille fixée par le pointeur de fin de fichier (EOF).

Toute opération d'enregistrement déplace vers l'avant la marque courante et la position de fin du fichier, du nombre de bytes écrits à un moment précis. Le nombre de données ajoutées fixe la nouvelle taille du fichier (EOF) et détermine automatiquement la position de la marque courante du pointeur. ProDOS attribue au fichier une nouvelle marque de fin (EOF) et met à jour le pointeur dans la table des matières du disque (Directory).

La taille du fichier est sauvegardée dans la table des matières au moment du premier enregistrement. C'est la commande OPEN, suivie de la longueur choisie (paramètre L£ pour les fichiers aléatoires), qui effectue ce travail. Si le paramètre longueur n'est plus précisé par la suite, la longueur totale du fichier ne pourra pas excéder la taille spécifiée au moment de sa création. Une nouvelle valeur de longueur, déclarée à la réouverture du même fichier, sera prise comme valeur courante pour les opérations à venir. L'ancienne valeur, sauvegardée dans la table des matières au moment de la création du fichier, ne sera pas modifiée pour autant. Si cette taille venait à être inférieure à un enregistrement initial, le pointeur de fin de fichier serait alors placé en avant de la fin représentative et le système complètera avec des bytes nuls. Ces types de fichiers sont appelés fichiers 'creux' ou Sparse Files, car ils possèdent des zones de bytes non significatifs (bytes nuls – Ctrl-à). Le Technical Manual appelle les fichiers creux des Sparse Files.

Compatibilité entre fichiers

Dans la majorité des cas, les données sont sauvegardées par des enregistrements séquentiels dans un fichier texte. Les pointeurs de la position où l'écriture est réalisée (MARK), ainsi que le pointeur de fin (EOF) se déplacent au fur et à mesure du traitement. Ces enregistrements pourront être réalisés en mode séquentiel ou aléatoire.

Une quelconque compatibilité entre les fichiers séquentiels et aléatoires est hors de question en raison d'une structure différente et une commande malencontreuse pourra créer des situations particulières.

Les fichiers se caractérisent par leur organisation de sauvegarde sur la surface du support disque. Celle-ci pourra être hiérarchisée ou du type éparse et comportera dans ce dernier cas des enregistrements indésirables.

STRUCTURE HIERARCHISEE

En présence d'une structure hiérarchisée, ProDOS utilise pour la sauvegarde des fichiers un nombre de blocs en fonction de leur capacité en caractères enregistrés:

– si un fichier se compose de moins de 512 bytes, il n'utilise qu'un seul bloc;

– s'il nécessite plus de 512 bytes, il lui faudra 2 blocs pour ses données et un bloc pour sa table répertoire ou Bloc Index. Une table répertoire peut pointer au maximum 128 blocs de données et devra utiliser un deuxième bloc répertoire au-delà de ces valeurs. Un Master Block est utilisé si la structure de sauvegarde nécessite plus d'un bloc index. Dans ce cas, le Master Block est considéré comme un bloc répertoire principal qui peut à son tour, pointer à nouveau des blocs auxiliaires (Blocks Index).

STRUCTURE D'UN SPARSE FILE

• Un fichier aléatoire est une suite d'enregistrements qui se compose d'un certain nombre de sauvegardes successives de données.

• Chaque enregistrement contient le même nombre de données, défini en bytes (caractères), valeur attribuée au moment de son ouverture.

• Un enregistrement peut contenir un ou plusieurs champs de données. Le total des informations placées dans un enregistrement définit la longueur de cet enregistrement.

• Chaque champ, ou enregistrement, est identifié par un nombre indiquant sa position absolue à l'intérieur du fichier: le premier enregistrement porte le numéro 0, le suivant le 1, etc. Pour définir ce type de fichier, il faudra inclure le paramètre longueur: L£, où £ désigne la longueur en bytes de chaque enregistrement.

Paramètre Longueur

```
10 PRINT CHR$ (4); 'OPEN TEST,L100'
20 ::::::::::::::
```

⇒ La ligne 10 indique au système d'ouvrir un fichier aléatoire du nom de TEST, avec une longueur de chaque enregistrement de 100 bytes.

• Pour éliminer des enregistrements indésirables inclus dans un fichier aléatoire, ce qui permet de réduire sa taille, il sera nécessaire de recopier les enregistrements utiles dans un nouveau fichier à accès aléatoire.

• Les enregistrements utiles se trouvent emprisonnés dans le contenu global d'un fichier aléatoire. Les données indésirables font par ailleurs, partie intégrante de la structure. Ce type de fichier occupe petit à petit une place importante sur le disque, tout en limitant sa capacité réelle. On dit que ce sont des fichiers 'creux' ou Sparse Files.

EXEMPLE DE SPARSE FILE

Le fichier traité en exemple vient juste d'être créé. Il contient par définition 1000 enregistrements, avec un paramètre L£ = 100 (longueur). Si l'on écrit dans ce fichier aléatoire, ouvert au préalable, des données uniquement dans l'enregistrement 1000 et que les enregistrements qui se trouvent en retrait deviennent inutiles, ProDOS marque comme occupé la longueur initiale du fichier. Cet exemple de situation extrême fait nettement ressortir le gaspillage que représente un tel fichier. Pour une occupation effective d'un seul bloc de données, le fichier occupe pratiquement toute la surface d'un disque 5,25 pouces.

L'inconvénient des fichiers aléatoires réside dans le fait qu'ils écrivent des codes nuls (CTRL-à) pour compléter la longueur déclarée, si celle-ci n'est pas totalement utilisée par les données. Si un fichier comporte une multitude d'enregistrements du type 'creux', on se rend compte qu'un disque se trouve rapidement envahi par des blocs contenant un nombre impressionnant de données inutiles (bytes nuls). Un remède à ce mal consiste à reformater ce type de fichier, pour ne réutiliser que les données utiles et purger les zones

occupées par les codes nuls. Cette opération de reformatage se réalise le plus simplement, en écrivant dans le fichier aléatoire un caractère ASCII : le code 32 par exemple (espace). Ce fichier aléatoire sera alors utilisé comme un fichier séquentiel et comportera le caractère 'espace' dans chaque enregistrement.

Programme de reformatage

```
10 PRINT CHR$ (4); 'OPEN ESSAI'
20 PRINT CHR$ (4); 'WRITE ESSAI'
30 FOR X = 0 TO 1000
40 FOR Y = 1 TO 99
50 PRINT CHR$ (32); :REM écrit des espaces
60 NEXT Y : PRINT
70 NEXT X
80 PRINT CHR$ (4). 'CLOSE'
```

Les valeurs sont ajustées selon la longueur et le nombre des enregistrements du fichier à reformater.

ORIGINE D'UN SPARSE FILE

Les fichiers aléatoires du type Sparse File sont engendrés de trois manières différentes :

• Lorsqu'un fichier ouvert ne comporte qu'un champ dans chaque enregistrement, par exemple :

RO : MARCELr
R1 : LUCr
R2 : RODOLPHEr

où,

r représente physiquement le retour chariot, code £$0D.

Les enregistrements plus courts sont complétés par des bytes nuls (Ctrl-à) après le retour chariot.

Si le paramètre L£ = 9, L'enregistrement sur le disque apparaît ainsi :

```
R0              R1              R2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
M A R C E L r 0 0 L U C  r  0  0  0  0  0  R  O  D  O  L  P  H  E  r
```

• Lorsqu'un fichier ouvert comporte plus d'un champ par enregistrement :

RO : MARCELrLUCrRODOLPHEr

Si l'enregistrement est plus court que la longueur déclarée, le reste des bytes est complété avec des Ctrl-à, après le dernier retour-chariot.

Si le paramètre L£ = 25, l'enregistrement sur le disque apparaît ainsi :


```
R0
1 2 3 4 5 6 7 8 9 10 1112 13 14 15 16 1718 19 20 21 22 23 24
M A R C E L r L U C r R O D O L P H E r 0 0 0
```

• Si en présence d'un ancien fichier on effectue une nouvelle sauvegarde de plusieurs enregistrements sans tenir compte de la valeur des emplacements chronologiques du paramètre R£. Par exemple, si l'on déclare un numéro d'enregistrement R£, tout en sachant qu'il existe encore des enregistrements en retrait, donc < R£, et dont les champs n'ont pas encore été écrits.

Remarque

Chaque table des matières (Directory) contient un certain nombre de renseignements, dont un byte particulier caractérise le fichier. Ce byte se trouve sauvegardé à la position relative $00 d'une entrée table des matières d'un fichier et se nomme Storage_Type. En fait, Storage_Type ne nécessite que les quatre bits de poids fort, tandis que les quatre bits de poids faible sont attribués à Name_Length qui désigne la longueur du nom d'un fichier. Storage_Type peut prendre les valeurs 1, 2, 3, etc. pour désigner successivement un fichier sans bloc index, avec 1 bloc index, 1 bloc index principal et 1 ou plusieurs blocs auxiliaires, etc.

STRUCTURE DE SAUVEGARDE D'UN FICHIER

Identité d'un fichier

Tout fichier ProDOS comporte en règle générale, une structure interne permettant d'écrire ou de lire un certain nombre de données par l'intermédiaire de deux pointeurs : MARK et EOF. Les fichiers, comme nous l'avons vu, regroupent deux grandes familles : les fichiers table des matières (Directory et SubDirectory) et les fichiers de données (généralement assimilés aux programmes).

FICHIER – DIRECTORY

Un Volume SubDirectory est toujours pointé par l'entrée du nom de fichier qui lui a donné naissance. Ce type de volume peut à nouveau contenir des fichiers Directory ou tout simplement des fichiers de données.

En début de chaque table des matières auxiliaire se trouve sauvegardé le nom du Volume Directory. Celui-ci comporte en préfixe un byte caractéristique :

• Les quatre bits de poids fort l'identifient par rapport au système. Si c'est un Volume Directory, le paramètre a comme valeur $F; un Volume SubDirectory aura comme valeur $E, un fichier qui pointe son Volume SubDirectory aura comme valeur $D et un fichier de données aura une valeur selon sa taille de sauvegarde ($1, $2 ou $3).

• Les quatre bits de poids faible désignent la longueur du nom du fichier. Le byte est sauvegardé à la position relative $00 de chaque entrée d'une table des matières. Le Technical Manual nomme les quatre bits de poids fort Storage_Type et les quatre bits de poids faible Name_Length.

Exemple

Si nous déclarons USERS.DISK comme nom du Volume SubDirectory, sa première entrée dans le Key Block comportera ce nom précédé du byte caractéristique de valeur FB :

- F pour signifier que c'est un Volume SubDirectory;
- B pour représenter la longueur du nom (10 en décimal pour notre exemple).

Si un Volume Directory contient un ou plusieurs fichiers qui pointent un ou plusieurs Volumes SubDirectory, les différents blocs répertoire seront chaînés dans le sens avant et arrière par des pointeurs. Cette technique est rendue nécessaire afin de pouvoir pointer le bon chemin lors de la recherche d'un fichier de données dans un Volume SubDirectory. Un nom de fichier qui pointe un Volume SubDirectory est le parent de ce dernier : c'est l'endroit où il a pris naissance.

Exemple

Volume Directory ----->
<----- Volume SubDirectory

Un Volume SubDirectory possède une structure de volume, avec son nom sauvegardé au début de la table des matières auxiliaire. Le système lui alloue au départ un bloc par défaut, volume extensible au fur et à mesure de son occupation par des noms de fichiers.

FICHIERS DE DONNEES – (PROGRAMME)

Un fichier de données représente en général le programme utilisateur et désigne tout autre type de fichier de données (Binaire, AppleSoft, Texte, Exec, etc.). Ces différents types de fichiers peuvent être lus ou écrits à partir d'une table des matières du volume d'un disque.

Les fichiers utilisateurs permettent l'exécution de différents programmes à partir d'un langage spécifié. Les fichiers que ProDOS gère se regroupent ainsi :

BAS	Ce sont les fichiers programmes écrits en langage Basic.
TXT	Ce sont les fichiers de données, sous forme de sauvegarde de codes ASCII et appelés fichiers texte.
VAR	Ce sont les fichiers de sauvegarde des variables programmes, créés par l'instruction STORE.
SYS	Ce sont les fichiers système, uniquement exécutables.
BIN	Ce sont les fichiers programmes du type binaire, écrits pour la plupart avec l'aide d'assembleurs.
REL	Ce sont les fichiers de programmes relocatables.
F£	Ce sont des fichiers qui peuvent être définis par l'utilisateur.

Notion de blocs index en présence d'un fichier

Un bloc index principal (Master Block) pointe des blocs index auxiliaires (Index Blocks) jusqu'à une limite de 128 blocs index. Chaque index auxiliaire peut pointer à nouveau un maximum de 256 blocs de données. Cette technique permet d'élaborer des fichiers structurés, d'une forme hiérarchisée, dont le principe est le propre du système d'exploitation ProDOS.

L'allocation des différents types de blocs est effectuée sous le contrôle du système ProDOS et dans une suite consécutive de secteurs à condition que la configuration du disque le permette. Dans le cas contraire, les blocs alloués seront chaînés entre eux et les pointeurs sauvegardés dans la table répertoire du bloc index.

Allocation de blocs consécutifs

- la disquette sera exempte de toute information;
- le nombre de blocs consécutifs doit être suffisant;
- le fichier à sauvegarder comporte un nombre de blocs inférieur à celui déjà alloué au fichier du même nom figurant sur le disque.

Allocation maximale des blocs à un fichier programme

Master Block =›Index Block 0
=›Index Block 1
=›Index Block 2
=›Index Block 3
:::: ::::::::
:::: :::::::
=›Index Block 127

=›Data Block 0
=›Data Block 1
=›Data Block 2
::::::::::::
::::::::::::
=›Data Block 255

Méthode de recherche d'un fichier

ProDOS16 effectue toujours la sauvegarde d'un fichier dans un volume, qu'il soit du type Volume Directory ou Volume SubDirectory. Ce choix de volume peut être déterminé d'une part par l'utilisateur et d'autre part par le programme en cours d'exécution. Le volume répond à une structure hiérarchisée : Volume Directory ou Volume SubDirectory.
La commande PREFIX/VOLUME/SUBDIR, où /VOLUME/ désigne le nom du volume (ou disque) venant d'être mis en place dans le drive 1 du slot 6 (drive encore ignoré par le MLI), déclenche une suite de recherches dont le processus pourrait être le suivant :

• Le chemin, ou Pathname complet, sera placé et vérifié à l'intérieur du MLI à partir de l'adresse $F100 par le sous-programme utilisateur.

• Le Volume Control Block (VCB) situé après le nom VOLUME, sera recherché; dans notre exemple ce nom ne figure pas encore dans le FCB,

disquette nouvellement introduite dans le lecteur. Le système consulte alors, les uns après les autres, tous les paramètres dans la Device Table : il recherche en premier une entrée libre dans le VCB, puis lit le paramètre Unit_Number qui représente l'image binaire du byte d'un périphérique en ligne.

• Le nom du Volume Directory trouvé (nom du volume) sera comparé avec toutes les entrées du VCB : si celui-ci ne figure dans aucune des entrées, il y sera sauvegardé. Sur cette base et dans un ordre donné, à partir du slot ayant le numéro le plus élevé, tous les paramètres des périphériques en ligne seront sauvegardés. La recherche comparative se poursuivra, jusqu'à ce que le nom VOLUME soit trouvé.

• Si pour une raison indéterminée la commande n'aboutit pas, une des erreurs suivantes sera renvoyée : VOLUME CONTROL BLOCK TABLE FULL, ou VOLUME NOT FOUND. Cette dernière erreur survient en général dans le cas où le nom VOLUME reste introuvable dans le Volume Directory de notre exemple.

• Si le nom VOLUME est trouvé, le travail du MLI ne sera pas pour autant terminé : il reste encore à trouver SUBDIR, qui pointe vers un Volume SubDirectory, lui même sauvegardé dans le Volume Directory (ou table des matières principale).

• Le numéro du premier bloc de données de SUBDIR se trouve stocké dans l'entrée du Volume Directory : il a été sauvegardé en même temps que les autres caractéristiques.

• Après avoir vérifié qu'il s'agit bien d'un volume du type SubDirectory, le nouveau nom/VOLUME/SUBDIR sera copié dans l'entrée PREFIX ($F100-$F1FF) de telle façon que la fin coïncide avec l'adresse $F1FF, le caractère R sera placé en $F1FF. Le byte Storage_Type sera copié à l'adresse $F100 et dans PREFIX ($BF91). Exemple de Storage_Type valide : $F6 = VOLUME, où F représente un fichier Directory et 6 le nombre de caractères du nom VOLUME.

• Ensuite ProDOS sauvegarde le numéro du SubDirectory Key Block SUBDIR, aux adresses $F060-$F061 et non dans le VCB comme on aurait pu le croire.

• Le processus de la routine de recherche SET_PREFIX prend fin et le retour se fera, pour ProDOS8 via la page globale de PRODOS, au sous-programme appelant. En ce qui concerne ProDOS16, le processus de recherche est effectué à travers les outils de la ToolBox.

Attributs relatifs aux fichiers ProDOS16

Le paramètre File_Type, stocké dans la table des matières à la position relative $10 de son entrée, est un descripteur de fichier. Il détermine la caractéristique interne du fichier dont ProDOS16 gère les attributs, permettant par la suite de différencier un type de fichier d'un autre. Ce paramètre n'est pas à confondre avec Storage_Type qui personnalise le fichier selon sa taille ou son type de volume (Directory ou SubDirectory).

File_Type	Utilisation préconisée
$00	Fichiers non définis (SOS et ProDOS8)
$01	Bad Block File Fichier contenant la liste des blocs défectueux
$02	Pascal Code File Fichier réservé au système Pascal
$03	Pascal Text File Fichier texte – Système Pascal
$04	ASCII Text File (SOS & ProDOS8) Fichier Texte au format ASCII (SOS et ProDOS8)
$05	Pascal Data File Fichier de données – Système Pascal
$06	General Binary File (SOS & ProDOS8) Fichier binaire (SOS et ProDOS8)
$07	Font File Fichier réservé aux fontes et matrices de caractères
$08	Graphics Screen File Fichier réservé aux écrans graphiques
$09	Business Basic Programm File Programme de gestion – Langage Basic
$0A	Business Basic Data File Fichier de données du programme gestion – Basic
$0B	Word Processeur File Fichier langage machine
$0C	SOS System File Fichier système SOS
$0D-$0E	SOS Reserved Réservés au système SOS
$0F	Directory File – SOS & ProDOS8 Fichier table des matières – SOS et ProDOS8
$10	RPS Data File Fichier de données – RPS
$11	RPS Index File Fichier réservé à des pointeurs – RPS
$12	AppleFile Discard File Fichier annexe
$13	AppleFile Model File Fichier témoin ou source
$14	AppleFile Report Format File Fichier de compte rendu
$15	Screen Library File Fichier Librairie ou de documentation
$16-$18	SOS Reserved Réservés au système SOS
$19	AppleWorks Data Base File Fichier de données relatif à AppleWorks
$1A	AppleWorks Word Processeur File Fichier codes machine relatif à AppleWorks
$1B	AppleWorks Spreadsheet File Fichier étendu relatif à AppleWorks
$1C-$EE	Réserved Reservés
$B0	Source File

File_Type	Utilisation préconisée
	Fichier source. Le paramètre Aux_Type définit le langage utilisé.
$B1	Object File Fichier objet relatif à un langage donné
$B2	Library File Fichier librairie relatif à un langage donné
$B3	ProDOS16 Application Fichier d'application – ProDOS16
$B4	ProDOS16 Run-Time Library file Fichier de lancement (librairie) – ProDOS16
$B5	ProDOS16 Shell Load File Source pour le chargement d'un fichier – ProDOS16
$B6	ProDOS16 Startup Load File Source pour le démarrage d'un fichier – ProDOS16
$B7-$BE	Reserved for ProDOS16 Load File Réservés à des fichiers de chargement – ProDOS16
$BF	ProDOS16 Document File Fichier document – Système ProDOS16
$EF	Pascal Area on a Partionned Disk Fichier Pascal supportant un disque dur
$F0	ProDOS8 CI Added Command File ProDOS8, extension d'un fichier de commande
$F1-F8	ProDOS8 User Defined Files 1-8 Fichiers redéfinissables (indices 1-8) – ProDOS8
$F9	ProDOS8 Reserved Réservé au système ProDOS8
$FA	Integer Basic Program File Fichier programme Basic – Integer
$FB	Integer Basic Variable File Fichier contenant des variables Basic – Integer
$FC	AppleSoft Program File Fichier programme AppleSoft
$FD	AppleSoft Variable File Fichier des variables AppleSoft
$FE	Relocatable Code File (EDASM) Fichier relogeable – codes EDASM
$FF	ProDOS8 System File Fichier système – ProDOS8

CONVENTIONS ET PROTOCOLES SOUS ProDOS16

Ce chapitre traite plus particulièrement la version 2.0 du système d'exploitation ProDOS16, nouveau système adapté à l'Apple *IIGS* en particulier. La version initiale 1.0 était distribuée sous forme de version bêta, et n'était pas à jour avec certaines structures du MLI, en particuliers les commandes RENAME ($03), GET_ENTRY ($07), WRITE_PROTECT ($0C), GET_DIB ($21), START_PRODOS ($25), END_PRODOS ($26), SAVE_STATE ($2B), RESTORE_STATE ($2C) . Les commandes ALLOC_INTERRUPT et DEALLOC_INTERRUPT ont changé de numéro et sont repérées respectivement $30 et $31.

Nous développerons les possibilités adaptées au traitement des fichiers (File Management) et de la gestion des entrées/sorties (Input/Output) en présence de l'environnement de la machine et des logiciels adaptés. Certains aspects particuliers sont abordés pour permettre au lecteur hobbyiste ou non de maîtriser le MLI (Machine Language Interface).

L'auteur a mis l'accent sur les particularités relatives à l'environnement et aux conventions fixées par Apple Computer en ce qui concerne ProDOS16.

SYSTEMES NOUVELLE GENERATION – ProDOS8 et ProDOS16

Origine du système ProDOS16

Au départ, le système ProDOS fut mis au point pour le professionnel travaillant sur un disque dur, afin de lui permettre la gestion de gros fichiers. Par la suite, l'évolution aidant, les programmes système furent adaptés pour pouvoir exécuter des applications sur toute la gamme des Apple *II*, *IIe*, *IIc*, *IIe*+ et Apple *IIGS* dotés d'une capacité mémoire de 64 Ko minimum. La structure de ProDOS est telle, qu'elle permet de gérer des périphériques de sauvegarde d'une capacité pouvant atteindre jusqu'à 32 Mo (méga-octets), la taille d'un fichier étant elle-même étendue à 16 Mo (1 octet = 1 caractère; 1

Ko = 1024 octets; 1 Mo = 1024 * 1024 = 1.0.48.576 octets; 16 Mo = 16.777.216 octets).

On constate qu'un tel système ouvre de nouvelles perspectives, laissant au concepteur la possibilité de créer du matériel de plus en plus performant: capacité de stockage plus grande, accès aux données plus rapide, etc. Par la suite, le GS ayant fait son apparition sur le marché, l'ancien système ProDOS a du être revu et corrigé pour laisser sa place à deux systèmes bien distincts: ProDOS8 et ProDOS16.

ProDOS8 est le système d'exploitation réservé à des applications traitant des mots de 8 bits de large. Il est légèrement différent dans son écriture par rapport à la version 1.1.1 de ProDOS. C'est en fait une adaptation du système ProDOS pour lui permettre une mise à niveau envers du matériel nouveau, en particulier le micro-ordinateur *II*GS.

ProDOS16 est devenu le noyau du système d'exploitation, destiné dans l'immédiat à des applications sur l'Apple *II*GS en mode natif, c'est à dire capable de traiter des registres de 16 bits de large. La routine de chargement (Loader) charge ProDOS16 à un vecteur de la mémoire déterminé par un code du système Loader. Bon nombre d'extensions ont été rendues possibles grâce à ce nouveau système:

- les périphériques ont été totalement revus et réadaptés à la technologie nouvelle;
- la structure et le traitement des fichiers ont subi de profondes mutations;
- l'environnement a été adapté au nouveau type de machine;
- un certain nombre de commandes MLI nouvelles sont apparues;
- la mémoire morte intègre désormais les outils de travail du Macintosh (ROM ToolBox), la mémoire adressable est étendue à 16 Mo, dont 8 Mo sont réservés à de la ROM et les 8 autres à de la RAM;
- la RAM se subdivise en bancs mémoire de 64 Ko chacun;
- l'adressage mémoire se gère par le Memory Manager, outil intégré dans la ROM ToolBox.

L'utilisateur averti possède maintenant les moyens pour utiliser le nouveau système et pour créer ses propres applications. La ROM ToolBox, complément indispensable au programmeur, contient bon nombre d'outils de travail qu'il sera nécessaire de connaître au préalable. Toute une éducation technique est à refaire, les nouveaux moyens disponibles sur ce type de machine vont dans ce sens.

ProDOS8 et ProDOS16

ProDOS16 est un système d'exploitation issu de la technique et du savoir-faire voués initialement à ProDOS8. Ce système, à l'avant-garde des développements logiciels, était initialement destiné aux applications de la gamme des Apple *II* en général, munis d'une mémoire de 64 Ko minimum. Le concepteur a adapté l'ancien système aux nouvelles fonctions apparues sur le GS, tirant profit au maximum des structures des programmes système existants. Le résultat se concrétisa par l'apparition de ProDOS16 sous la forme d'une version bêta baptisée 1.0. La version 2.0, revue et corrigée, est considérée par Apple Computer comme celle devant être opérationnelle sur le GS.

POINTS COMMUNS ET DIFFERENCES FONDAMENTALES

Par la suite, une synthèse des systèmes ProDOS8 et ProDOS16 permet d'apprécier pleinement les différences et les points communs qu'offrent ces deux systèmes.

- Le microprocesseur 65C816 possède une caractéristique essentielle: il émule les fonctions essentielles d'un processeur 8 bits (6502) et traite en mode natif les instructions des applications 16 bits. ProDOS16 autorise des appels au MLI à travers les modes 8 ou 16 bits. ProDOS8 n'accepte que des commandes faisant référence à des applications du mode 8 bits.

- L'Apple *II*GS possède en version de base une mémoire vive de 256 Ko, facilement extensible à 4 Mo, voire même plus. ProDOS16 possède la propriété d'exploiter pleinement les capacités d'adressage du microprocesseur 65C816, dont les limites sont fixées par construction, à une taille de 16 Mo (ROM + RAM). Les adresses mémoire possibles s'étendent de $000000 à $FFFFFF et sont disponibles à de la mémoire vive ou de la mémoire morte. ProDOS8 se cantonne à exploiter la zone mémoire dans les limites fixées jusqu'alors (Apple *II*e et *II*c), c'est-à-dire des adresses $0000 à $FFFF, ce qui limite la zone mémoire à 64 Ko.

- ProDOS16 exploite à travers les outils du Memory Manager de la ROM ToolBox une réelle gestion de la mémoire, délaissant totalement la Global Page ($BF00-$BFF). ProDOS16 gère l'allocation et la désallocation des pages mémoire par l'intermédiaire de routines intégrées dans la ROM résidente. ProDOS8 gère les pages mémoire à travers la bit map de la Global Page, adresses $BF58 à $BF6F.

- Les applications sous ProDOS16 nécessitent au préalable l'allocation mémoire d'implantation, l'accès aux variables générales du système, les valeurs courantes de la date et de l'heure, le niveau d'ouverture des fichiers (System Level) et les adresses réservées au tampon d'entrées/sorties (I/O_Buffer). Cette disposition est rendue nécessaire par le fait que la Global Page est ignorée sous ProDOS16. ProDOS8 gère toutes ses variables MLI à travers la Global Page initialisée lors du boot système, adresses $BF00-$BFFF du banc $00 réservé exclusivement aux applications en mode émulation.

- ProDOS16 a nécessité la mise au point d'un protocole défini par Apple Computer et applicable par le développeur. Ce protocole particulier à ProDOS16 devient caduque sous ProDOS8.

- ProDOS8 et ProDOS16 supportent la gestion des données par blocs de 512 bytes transmis à travers des périphériques appelés Blocks Devices. Un Block Device est assimilé par défaut à un lecteur de disque par exemple.

- ProDOS8 ne reconnaît pas les types de fichiers $B0 à $BF, destinés exclusivement à des applications sous ProDOS16.

• ProDOS16 ne reconnaît pas les fichiers système du type $FF et les fichiers binaires du type $06, destinés exclusivement à des applications sous ProDOS8.

• Le système d'exploitation par défaut, exécutable sur le GS (Coldstart ou Warmstart) est soit ProDOS8, soit ProDOS16, le choix étant essentiellement une organisation des fichiers du disque système mis en présence et l'application à exécuter.

• Sous ProDOS8, la Global Page garde toute sa signification initiale : sauvegarde de certaines variables du système et point d'entrée du vecteur pour les appels au MLI.

ProDOS8 (Global Page)	ProDOS16
MLI : entrée dans ProDOS8	Ignoré par ProDOS16
Driver Table ($BF10-$BF2F)	Ignoré par ProDOS16
Devices Table ($BF32-$BF3F)	Commande VOLUME ($08)
System Bit Map ($BF58-$BF6F)	Outil du Memory Manager
Pointer I/O_Buffer ($BF70-$BF7F)	Commande OPEN ($10)
Interrupt Table ($BF80-$BF87)	Commande ALLOC_INTERRUPT ($30)
System Time ($BF92-$BF93)	ReadTime de Miscellaneous Tools
System Level ($BF94)	Commande GET_LEVEL ($1B)
MACHID ($BF98)	Ignoré par ProDOS16
ProDOS Version ($BFFC-$BFFF)	Commande GET_VERSION ($2A)

• La majorité des commandes MLI sous ProDOS8 sont présentes sous ProDOS16 et se retrouvent sous le même nom. Sous environnement ProDOS16, chaque bloc d'une commande transmise au MLI aura sa syntaxe modifiée par rapport à ProDOS8 :

- le JSR sera remplacé par un JSL, saut long;
- le numéro de la commande sera codé sur deux bytes;
- le pointeur de la table des paramètres sera codé sur quatre bytes;
- les autres adaptations rendues nécessaires sont établies à travers la table des paramètres de la commande et du format requis sous ProDOS16.

COMPATIBILITE ASCENDANTE

ProDOS16, associé à des fonctions hardware et software, n'est pas compatible dans le sens ascendant à ProDOS8, ce qui limite l'exécution des programmes écrits sous ProDOS8 à ce seul système d'exploitation. ProDOS16 ne s'adresse pas aux Apple *II*, *II+*, *IIc* et *IIe*, ces types de machines n'étant pas équipées d'une configuration mémoire adaptée. Par ailleurs, elles sont dotées d'un microprocesseur différent du 65C816 qui autorise quant à lui le mode natif du GS. Cependant, ces deux systèmes présentent quelques similitudes et points communs, autorisant une compatibilité ascendante restreinte :

• Les deux systèmes possèdent une structure basée sur une conception commune :

- les commandes MLI présentent dans leur origine des aspects communs, mais sont plus performantes dans leurs exécutions en présence de ProDOS16;
- les commandes transmises à travers le MLI sont effectuées d'une manière semblable pour les deux systèmes ProDOS;
- ProDOS16 et ProDOS8 appliquent les mêmes règles face à un fichier. Celui-ci peut être lu, modifié et réécrit indifféremment à partir d'un volume, qu'il ait été créé à partir de l'un ou de l'autre des systèmes ProDOS. Les deux systèmes traitent un fichier à partir d'un support disque comme un sous-ensemble faisant partie d'une structure d'un volume courant (disque et volume logiques).

• Les deux systèmes ProDOS sont exécutables à partir de l'Apple *IIGS*. ProDOS8 est un système spécialement destiné aux applications de la famille des Apple *II* en général. ProDOS16 est destiné particulièrement aux applications mises au point pour être exécutées sur un Apple *IIGS*.

Remarque

Le système adéquat, ProDOS8 ou ProDOS16, est chargé en mémoire puis sollicité par le programme d'application mis en présence. Si c'est un programme écrit pour traiter des mots de 8 bits, il charge en mémoire le système ProDOS8. Si un programme est élaboré pour traiter des mots de 16 bits et qu'il est exécuté, il charge en mémoire le système ProDOS16, si ce dernier ne l'est pas encore.

COMPATIBILITE DESCENDANTE

Les applications développées pour fonctionner sous le système ProDOS16 ne sont pas exécutables sur les Apple *II*, *IIc* et *IIe*. Cette situation est due essentiellement à la différence de conception de la mémoire disponible sur le GS (256 Ko de RAM et 128 Ko de ROM) et aux instructions supplémentaires reconnues par le microprocesseur 65C816.

COMMANDES NOUVELLES ET CELLES DISPARUES

Certaines fonctions sous ProDOS8 sont équivalentes à d'autres fonctions sous ProDOS16, la syntaxe étant alors le plus souvent différente. Certaines commandes, rendues superflues sous ProDOS16, ont été purement éliminées, d'autres ont été rajoutées, quelques unes sont équivalentes, mais se retrouvent sous un autre nom.

Commandes disparues sous ProDOS16

RENAME ($30) La version 1.0 de ProDOS16 est dépourvue de la commande RENAME, mais exploite une commande similaire qu'on retrouve sous le nom de CHANGE_PATH ($40). A partir de la version 2.0 de ProDOS16, la commande RENAME est réapparue et traite les mêmes fonctions que ProDOS8.

GET_TIME ($82) Sous ProDOS16, les paramètres relatifs à la date et à l'heure sont recherchés à travers l'appel d'une routine résidente en ROM (Miscellaneous Toolset de la ToolBox).

SET_BUF ($D2) Sous ProDOS16, l'allocation et la gestion du tampon d'entrées/sorties réservé aux données du fichier sont gérées à travers le Memory Manager et la ROM résidente (ToolBox).

GET_BUF ($D3) Cette commande n'est plus nécessaire sous ProDOS16 car la commande OPEN, ouverture d'un fichier courant, rebranche le système à la routine de gestion du tampon d'entrées/sorties (I/O Buffer). La routine se trouve résidente dans la ROM.

ON_LINE ($C5) Cette commande est éliminée sous ProDOS16 et trouve son équivalence dans la commande VOLUME ($08).

Caractéristiques du système ProDOS16

La liste des caractéristiques, différenciant ProDOS16 d'autres programmes système, est longue. Par la suite, une liste non exhaustive, dresse les possibilités essentielles du système ProDOS16, mettant l'accent sur le côté technique des détails.

GENERALITES RELATIVES A ProDOS16

- ProDOS16 est un système d'exploitation simple tâche, supportant une structure hiérarchisée de ses fichiers (volumes ou programmes).
- il permet la gestion et la sauvegarde d'un fichier d'une taille étendue à 16 Mo (Storage_Type = Tree file).
- la gestion des périphériques du type Blocks Devices autorise l'attribution d'un nom et permet la sauvegarde de données jusqu'à 32 Mo.

PARTICULARITES RELATIVES AUX COMMANDES MLI

- les commandes, à travers le MLI de ProDOS16, sont transmises par l'instruction machine JSL pour appeler l'entrée du système située dans un autre banc mémoire.
- si une erreur est rencontrée lors d'un appel transmis au MLI, le système retourne le code d'erreur dans le registre accumulateur (A) tandis que le registre d'état (P) sera positionné en conséquence.
- le contenu de tous les autres registres du microprocesseur 65C816 reste préservé.
- deux modes sont accessibles sur le GS : le mode natif permettant une exécution de toutes les instructions du 65C816; le mode émulation, limitant les instructions à celles compatibles au processeur 6502.
- toute la mémoire du GS est exploitable par le système ProDOS16, la limite n'étant plus fixée au seul banc $00 (ProDOS8).
- chaque bloc mémoire est disponible, soit pour y placer des données, soit pour transférer des données d'un bloc mémoire vers un autre.
- l'adressage est réalisé par un pointeur codé sur 24 bits, permettant de gérer jusqu'à 16 Mo d'adresses disponibles. Cet espace mémoire reste commun à la mémoire vive (RAM) et à la mémoire morte (ROM).

GESTION DES FICHIERS ProDOS16 (FILE MANAGEMENT)

- la gestion des fichiers est dite structurée, mettant à la disposition de l'utilisateur des outils performants. Les fichiers sont catalogués suivant une structure hiérarchisée, leurs noms répertoriés dans une table des matières et un attribut (Storage_Type) personnalisent le fichier selon sa taille (Seedling, Saapling ou Tree) et son identité (Directory, SubDirectory, etc.).
- avec la version 1.0, huit préfixes pouvaient être déclarés. A partir de la version 2.0, cette valeur est ramenée à quatre préfixes.
- attribue un pointeur d'accès aux fichiers (Directory ou Data File).
- alloue des blocs logiques au moment de la sauvegarde des données d'un fichier, soit dans une suite continue, soit dans une suite discontinue à la surface du support Block Device.
- gère aussi les fichiers du type Sparse File (voir le chapitre consacré aux types de fichiers gérés sous ProDOS16).
- alloue automatiquement des tampons mémoire au fichier en cours de traitement.
- gère un byte d'accès au fichier, attribut stocké dans l'entrée correspondante au nom du fichier. Le paramètre Access_Byte peut déterminer si un fichier est accessible en lecture ou en écriture, s'il peut être renommé (Rename), détruit (Destroy bit) ou dupliqué (Backup bit).
- attribue un indice (Level) à chaque fichier ouvert, ce qui détermine le nombre de fichiers ouverts à un instant donné. CLOSE exploite le paramètre Level pour fermer tous les fichiers ouverts.
- en présence d'un fichier, la date et l'heure sont automatiquement gérés, ce qui permet de personnaliser des programmes suivant les critères de sauvegarde (date et heure de mise à jour).
- la version 1.0, autorise jusqu'à huit fichiers ouverts simultanément. A partir de la version 2.0, cette limitation est étendue à la seule capacité de la mémoire, ce qui permet un nombre indéterminé de fichiers ouverts.
- la version 1.0 limite à quatorze le nombre de volumes en ligne. A partir de la version 2.0, cette limitation n'existe plus.
- le traitement des données s'effectue par blocs de 512 octets.
- un volume est limité à la taille de 32 Mo tandis qu'un fichier permet une sauvegarde jusqu'à 16 Mo de données.
- un total de 128 caractères valides est autorisé dans la déclaration d'un chemin d'accès complet (Pathname).
- un préfixe autorise jusqu'à 128 caractères valides.
- un total de quinze caractères valides sont autorisés pour décliner soit un nom de volume (Directory File ou SubDirectory File), soit un nom de fichier (Data File).

GESTION DES PERIPHERIQUES DU TYPE DEVICE

- ProDOS16 supporte jusqu'à trois protocoles différents de Block Device: protocole relatif aux Blocks Devices sous ProDOS8, protocole du convertisseur de protocole (Protocol Converter) et protocole du convertisseur de protocole étendu.

- chaque Block Device est nommé.

- le nombre de Blocks Devices n'est plus limité avec la version 2.0.

- autorise jusqu'à 15 caractères pour nommer chaque Device.

GESTION DES INTERRUPTIONS DU SYSTEME

- ProDOS16 détecte et transmet les interruptions à un ensemble matériel (hardware) si elles ne sont pas reconnues par les routines résidentes (Firmware).

- transmet les interruptions à des routines de service mises en place par le programme utilisateur pour gérer les différentes interruptions rencontrées.

- permet de gérer jusqu'à 16 routines de service à travers le handler des interruptions. La routine de service sera mise en place au préalable par la commande ALLOC_INTERRUPT ($30).

GESTION MEMOIRE

- ProDOS16 gère à travers la ToolBox, la mémoire dynamique disponible sur un Apple *II*GS . Alloue automatiquement une zone mémoire (Buffer) à tout fichier ouvert à un instant donné.

- accède directement jusqu'à 16 Mo de mémoire autorisés par la conception matérielle du microprocesseur 65C816. La version 2.0 permet de traiter par la suite une mémoire pouvant atteindre jusqu'à 2^{32} bytes.

- autorise des applications écrites pour traiter des mots de 16 bits avec un minimum de 256 Ko de RAM disponibles.

COMMANDE COMPLEMENTAIRE A ProDOS16

• La commande QUIT ($29) est un 'must' du système ProDOS16, permettant à l'application en cours de se terminer soit normalement dans le système, soit de relancer une autre application mémorisée au préalable dans une pile réservée à cet effet (Stack). ProDOS16 gère automatiquement le pointeur de retour de la pile (Quit Return Stack).

• Si QUIT pointe sur un retour normal, tous les fichiers ouverts à ce moment sont fermés (attribut Level mis à 0) et annule une éventuelle routine de service destinée à la gestion de l'interruption.

• Dans tous les cas, ProDOS16 autorise en fin de traitement d'une application en cours d'exécution:

- de lancer une nouvelle application spécifiée à partir du programme courant et avant de quitter ce dernier;

- de réaliser automatiquement une séquence d'exécution d'un programme choisi au préalable, dont le pointeur se trouve sauvegardé au sommet de la pile réservée.

Equivalence blocs/secteurs

Le système ProDOS en général se divise en deux parties essentielles: le Command Dispatcher and Block File et le Disk Driver.

• Le Command Dispatcher and Block File Manager est l'équivalent du File Manager du DOS 3.3. Cette partie gère l'allocation des fichiers sur un disque et traite des blocs de données, donc 512 octets à chaque fois.

• Le Disk-Driver traite le Block Device au niveau physique (pistes/secteurs). La routine RWTB (Read Write Track Block) n'est plus capable de formater un disque; elle est rebaptisée Disk-Driver pour la circonstance. C'est l'équivalence des routines employées pour les systèmes Pascal et CP/M, ce qui leur confère des similitudes frappantes avec ProDOS (gestion identique des blocs). Associée à un disque dur, la routine Disk-Driver comporte un module différent.

Sous le système DOS 3.3, la routine RWTS (Read Write Track Sector) peut manipuler des secteurs d'une taille de 256 bytes. Un secteur est la plus petite zone de sauvegarde possible en une seule fois, ce qui permet d'effectuer des opérations d'écriture et de lecture dans un fichier.

Sous le système ProDOS en général, la notion de secteur existe uniquement au niveau de la piste physique, la gestion logique étant réalisée à travers un bloc logique. Un bloc logique se caractérise par une capacité de sauvegarde continue de 512 bytes. ProDOS traite les données d'un Block Device sous forme de blocs, dont chacun peut être comparé à une unité logique se composant de deux secteurs physiques. Un disque standard au format 5,25 pouces, formaté 35 pistes physiques, comporte 280 blocs logiques (35 * 16 / 2 = 280 blocs).

Volumes et syntaxes valides

SYNTAXE D'UN CHEMIN D'ACCES (PATHNAME) ET PREFIXE

Sous ProDOS16, un chemin consiste en une suite de noms de fichiers, précédés par le slash (/). Le premier nom dans le chemin énoncé représente le nom du Volume Directory. La suite des noms successifs consiste en un chemin que le système aura à parcourir avant d'atteindre le nom du fichier de données. Sous ProDOS8 et ProDOS16 version 1.0, la longueur maximale d'un chemin est de 64 caractères valides, slash compris. A partir de la version 2.0 de ProDOS16, la notion de chemin diffère, et autorise désormais 128 caractères valides soit pour un Pathname, soit pour un préfixe.

Pour nommer un volume il faut

- débuter le chemin par une barre oblique, Slash, suivi d'une lettre;
- composer la suite du chemin par des lettres, chiffres et points;
• lettres entre A et Z

- chiffres entre 0 et 9
- le point '.'

– que le premier caractère soit une lettre,
– que le chemin ne contienne pas d'espace ou de caractère de ponctuation autre que le point servant au format et à la présentation du nom;
– utiliser au maximum quinze caractères par nom de fichier, y compris le point, mais barre oblique exclue;

Voici quelques exemples valides

/USERS.DIKS
/UTILITIES
/COURRIER.18.09
/FACTURATION

et quelques exemples non autorisés

/EXEMPLE.TROP.LONG	Plus de quinze caractères
/BIG MAG	Espace après BIG interdit
/5.UTILITAIRES	Débuter par un chiffre est interdit
/.COURRIER	Débuter par un point est interdit

Définitions

Pathname

Un Pathname représente soit un chemin complet, soit un chemin partiel, permettant l'accès au fichier nommé. Cette syntaxe consiste en une suite de noms qui désignent des tables des matières, Volume Directory et Volume SubDirectory, juxtaposées l'une à la suite de l'autre et représentant le chemin à parcourir pour arriver au fichier d'application. Ce chemin peut être partiel ou complet. Un chemin complet commence par le nom du /VOLUME, tandis qu'un chemin partiel est complété par le préfixe.

Prefix

Le préfixe est un chemin partiel par défaut, qu'il sera nécessaire d'ajouter à celui que l'on veut nommer. Ses différentes possibilités permettent une simplification dans l'élaboration des chemins: la frappe de la totalité du chemin n'est pas rendue nécessaire.

Exemple de chemin complet

/USERS.DISK/UTILITAIRES/FICHIER

Exemples de chemin partiel

/UTILITAIRES/FICHIER

ou encore

UTILITAIRES

Exemples de préfixes

/UTILITAIRES/NOM.FICH

ou

/UTILITAIRES/JEUX

Remarques

- Les caractères minuscules sont automatiquement convertis en majuscules.
- Toute commande transmise au système et qui nécessite un nom de fichier, autorise la syntaxe d'un chemin partiel ou complet.
- Un chemin partiel est un chemin dont le nom du volume aura été tronqué.
- Un chemin complet, partiel ou préfixe, peut être constitué d'un nombre de caractères valides limités par le système ProDOS actif.
- ProDOS8 et ProDOS16 version 1.0 autorisent un Pathname et un préfixe de 64 caractères chacun.
- La version 2.0 de ProDOS16 autorise un Pathname et un préfixe de 128 caractères chacun.

NOTION DE VOLUME (SUB)DIRECTORY

Un Volume Directory peut contenir 52 noms de fichiers, nom du volume compris. ProDOS16 autorise la gestion de Block Devices pouvant atteindre jusqu'à 32 Mo en taille. Un fichier peut atteindre la taille limite de 16 Mo. Il est alors souhaitable de pouvoir disposer dans ces cas d'une capacité de sauvegarde de masse plus importante, un disque dur par exemple. ProDOS16 permet d'étendre la capacité de son Volume Directory initial, limité à 52 entrées possibles, en créant simplement une table des matières auxiliaire (Volume SubDirectory). C'est la commande MLI CREATE ($01), seul moyen disponible à l'utilisateur, qui permet d'ouvrir un nouveau fichier Volume SubDirectory.

Un Volume SubDirectory peut à nouveau contenir des fichiers programmes et des fichiers tables des matières (SubDirectory). Un fichier qui pointe une table des matières auxiliaire est doté par le système d'un bloc unique. La table des matières qui contient le fichier SubDirectory est le volume 'parent' du SubDirectory: il sont chaînés entre eux par des pointeurs.

ORGANISATION HIERARCHISEE

Il est possible de regrouper plusieurs types de fichiers suivant une suite logique et fonctionnelle ou d'après leur appartenance à une famille de programmes. Comme exemple, un Volume SubDirectory répondant au nom de **UTILITAIRES** pourra contenir des programmes utilitaires, tandis qu'un Volume SubDirectory **JEUX** contiendra les programmes de jeux. Cette structure permettra par la suite la sauvegarde des programmes utilitaires suivant le chemin:

VOLUME/UTILITAIRES/FICHIER.UTIL

et les programmes jeux suivant le chemin :

/VOLUME/JEUX/FICHIER.JEUX

Détails des noms employés

/VOLUME	est le nom de Volume du disque
/UTILITAIRES/	est le nom du Volume SubDirectory contenant par la suite tous les fichiers utilitaires.
/JEUX/	est le nom du Volume SubDirectory contenant par la suite tous les fichiers de jeux.
FICHIER.UTIL	est un nom quelconque de fichier utilitaire.
FICHIER.JEUX	est un nom quelconque de fichier de jeux.

CREATION D'UN FICHIER

La commande CREATE permet de créer un fichier dans un volume existant. Pour créer un fichier quelconque, un certain nombre de règles sont à observer. Le programmeur sera tenu d'assigner au préalable et sur la même ligne de commande ProDOS, un ou plusieurs paramètres valides.

Paramètre Pathname

Le Pathname, chemin d'accès unique au fichier, permet par la suite de retrouver le fichier ainsi créé. Il sera composé de caractères valides et devra être accessible normalement dans un volume existant.

Paramètre Access_Byte

Le byte d'accès au fichier, paramètre Access_Byte, sera déterminé en fonction de l'utilisation future du fichier : possibilités de lecture uniquement, écriture autorisée, effacement ou affectation d'un nouveau nom.

Paramètre File_Type

Un byte caractéristique sera affecté comme attribut au fichier. Ce byte est un descripteur de fichier, utilisé par certaines applications pour reconnaître sa structure intrinsèque (fichier SYS, SOS, ProDOS8, ProDOS16, etc.).

Paramètre Storage_Type

Un byte descriptif détermine le format physique du fichier placé sur le disque. Storage_Type est un attribut affecté à deux formats de fichiers possibles : les fichiers tables des matières (Directory et SubDirectory) et les fichiers de données (Data File).

Remarque

La commande MLI GET_FILE_INFO ($06) permet de lire les informations relatives aux attributs affectés à un fichier. Sous ProDOS16 version 1.0, pour renommer un fichier, c'est la commande CHANGE_PATH ($04) qui sera nécessaire. A partir de la version 2.0, la commande RENAME ($03) est réapparue. Si pour une raison ou une autre, un ou plusieurs attributs d'un fichier sont à modifier, c'est la commande SET_FILE_INFO ($05) qu'il faudra utiliser.

OUVERTURE D'UN FICHIER

Avant de pouvoir lire ou écrire des données dans un fichier, il faudra ouvrir celui-ci en déclarant la commande MLI OPEN ($10). Si OPEN est actif, le fichier devra exister dans un volume valide et le chemin d'accès déclaré au préalable (Pathname). La commande OPEN retourne via le MLI, un code de référence (File_Reference_Number), affectant ainsi à chaque fichier ouvert un numéro de référence et un pointeur du tampon mémoire (I/O_Buffer). La variable I/0_Buffer pointe un tampon mémoire qui servira pour y transférer, à partir du disque ou vers le disque, des données du fichier courant. Un tel fichier ouvert par la commande OPEN ($10) devra être fermé en fin de traitement par la commande CLOSE ($14).

Chaque fichier ouvert occupe un tampon mémoire qui lui est propre, celui-ci restera attribué durant tout le temps de traitement. ProDOS16 version 1.0 autorise jusqu'à huit fichiers ouverts en même temps et à un moment donné. A partir de la version 2.0, cette limitation n'existe plus. Lorsqu'un fichier est ouvert, un certain nombre de paramètres sont stockés dans le File Control Block (FCB).

Un FCB est un espace mémoire réservé pour chaque entrée d'un nom de fichier de données ou de table des matières. Divers renseignements sont alors disponibles au système : numéro du bloc d'entrée dans le Volume Directory, numéro du premier bloc de données, pointeur interne au fichier (Mark), pointeur de fin d'un fichier (EOF), byte de référence du Device en ligne (Unit_Number), etc. Un FCB possède une structure pour la sauvegarde de ses entrées caractéristiques aux fichiers. Chaque entrée dans un FCB occupe 32 octets, ce qui alloue un total de 256 octets pour un FCB contenant huit fichiers.

POINTEURS INTERNES AU FICHIER EN GENERAL

Pour pouvoir écrire ou lire des données à l'intérieur d'un fichier, ProDOS16 dispose d'une série de pointeurs qu'il manipule à travers des routines MLI.

• EOF (End Of File) est un pointeur qui indique la fin dans le fichier. Ce pointeur est sauvegardé dans le FCB, aux positions relatives $15-$17 de son entrée correspondante. Lorsqu'un fichier est sauvegardé dans un volume, EOF est copié dans son entrée correspondante de la table des matières, aux positions relatives $15-$17. Il occupe trois bytes et désigne par défaut la longueur du fichier. EOF est en somme le nombre de bytes pouvant être lus dans un fichier. Si le premier byte dans un fichier occupe la position 0, EOF utilisé comme pointeur désigne la première position directement à la suite du caractère valide.

• MARK est un pointeur qui désigne une position courante dans un fichier. Le pointeur est sauvegardé dans le FCB, aux positions relatives $12-$14 réservées à l'entrée du fichier. Le pointeur MARK n'est pas copié dans une entrée d'une quelconque table des matières. MARK pointe une position courante dans un fichier pendant son traitement pour permettre les accès du disque. La valeur du pointeur est continuellement mise à jour par ProDOS, à partir d'une position absolue dans le fichier, désignant ainsi le caractère actuel à lire ou à écrire. Les commandes MLI Read et Write utilisent le pointeur MARK pour calculer l'emplacement des enregistrements, effectuant à chaque fois la somme, par association, des paramètres auxiliaires R£, F£ et B£.

Le pointeur MARK ne pourra en aucun cas être de valeur supérieure à EOF.

Remarque

Sous environnement Basic, R£ désigne le numéro de l'enregistrement; F£ indique le nombre de rubriques à ignorer avant de lire ou d'écrire des données; B£ désigne le nombre d'octets à ignorer à partir de la position courante.

LECTURE ET ECRITURE DANS UN FICHIER

Les commandes MLI READ ($12) et WRITE ($13) permettent respectivement de lire ou d'écrire des données dans un fichier. ProDOS16 transfère de (ou vers) la mémoire tampon réservée au fichier, des données vers (ou du) disque en ligne. READ et WRITE exploitent en commun trois paramètres relatifs aux commandes MLI :

- le paramètre Reference_Number assigné au fichier lorsque celui-ci a été ouvert;
- l'adresse de la zone mémoire (Data_Buffer) qui contient les données du fichier en cours de traitement, transférée périodiquement du/ou vers le disque. La variable Data_Buffer pointe le plus souvent l'adresse représentée par la variable I/O_Buffer.
- le nombre de bytes réellement transférés au moment du traitement du fichier (Transfer_Count).

ProDOS16 ET LA MEMOIRE SYSTEME

Configuration de la mémoire sous ProDOS16

Par définition, la mémoire de gestion est totalement séparée de celle réservée au système d'exploitation. ProDOS16 n'exploite pas les propriétés de la Global Page ($BF00-$BFFF) mais appelle un outil résident dans la mémoire morte : le Memory Manager.

L'Apple *II*GS , par construction, est capable d'adresser via son microprocesseur 65C816, 16 Mo de mémoire. Dans sa version de base, la configuration est dotée de 256 Ko de mémoire vive (RAM) et de 128 Ko de mémoire morte (ROM). Ces deux types de mémoire se partagent la totalité des 16 Mo disponibles.

L'espace mémoire total se partage en 256 bancs mémoire de 64 Ko chacun. Les bancs $00 et $01 sont réservés à des applications logiciels destinés à la gamme des Apple *II*e et *II*c (mode émulation). Les bancs $E0 et $E1 sont réservés essentiellement à des applications en mode émulation faisant appel à l'affichage vidéo sollicité par le graphisme haute résolution et à quelques variables système et outils divers. La zone mémoire particulière de chaque banc contient un certain nombre de commutateurs logiques, des adresses d'entrées/sorties réservées (I/O) et des tampons réservés à l'affichage du mode des Apple *II*e et *II*c. Les bancs $FE-$FF sont réservés à la ROM résidente contenant entre autres l'interpréteur Basic AppleSoft et les outils de travail de la ToolBox.

Composantes actives de l'Apple *II*GS

Composante	Valeur respective
Banc mémoire	65 536 bytes, ou 256 pages mémoire
Block	512 bytes, ou équivalent à 2 secteurs
Page mémoire	256 bytes
Mot long	DWord. Double mot, ou 4 bytes, ou 32 bits
Mot	Word. Mot, ou 2 bytes, ou 16 bits
Byte	8 bits
Nibble	4 bits, ou 1/2 byte

Une carte d'extension mémoire enfichée dans le slot réservé à cet effet sur la carte mère, permet d'étendre la capacité mémoire de la configuration. Pour une extension de 1 Mo, 16 bancs mémoire supplémentaires sont désormais disponibles à l'utilisateur et aux programmes capables de reconnaître une telle interface. Les bancs $02 à $11 sont maintenant alloués à la mémoire d'extension de 1 Mo. La mémoire d'extension ne se substitue en aucun cas aux bancs mémoire $00-$01 et $E0-$E1, ces derniers étant spécialement réservés à des applications particulières (mode émulation entre autres).

System Loader sous ProDOS16

Le System Loader, à ne pas confondre avec le Loader, consiste en une particularité de ProDOS16, permettant l'accès à partir d'une application aux différents outils de travail. Ces outils peuvent se situer dans la ROM (ToolBox) ou dans la RAM (outils mis en place par le programme utilisateur).

Le système d'exploitation ProDOS16 et le System Loader occupent pratiquement toute la zone mémoire des adresses $D000 à $FFFF dans les bancs $00 et $01. Cette zone mémoire correspond à celle initialement occupée par ProDOS8 sur les Apple *II*e et *II*c, adresse de la carte langage à partir de $D000. Il va sans dire que cette disposition n'est rendue possible que par l'intermédiaire de la commutation des bancs mémoire de la carte langage.

ProDOS16, à travers l'outil du Memory Manager, se réserve une extension mémoire de 10,7 Ko dans le banc $00, juste en dessous de l'adresse $C000 et se déployant vers le bas de la mémoire. Cette zone mémoire correspond à l'ancien vecteur d'occupation du BASIC.SYSTEM sous ProDOS 1.1.1. Cette mémoire d'extension sert en principe à gérer les tampons d'entrées/sorties, les tables de translation réservées à ProDOS8 et d'autres variables.

Une partie des variables du système, communs à ProDOS8 et ProDOS16, est stockée dans les bancs $E0 et $E1 et accessible à ProDOS par commutation logique. Le reste des adresses disponibles dans les bancs $E0 et $E1 sont réservées à des applications diverses.

Banc $00, adresses réservées

$00/9000-BFFF : Réservé au système d'exploitation ProDOS16

$00/D000-DFFF : 2 Ko réservés à ProDOS16 et 2 Ko réservés à d'autres systèmes.

$00/E000-FFFF Réservé à ProDOS16

Banc $01, adresses réservées

$01/D000-E7FF : Réservé au System Loader de ProDOS16

$01/E800-FFFF : Réservé au système d'exploitation ProDOS16

Bancs $02-$3F (extension mémoire)

$02-$3F/0000-FFFF : Réservé aux applications ou à d'autres systèmes

Bancs $E0-$E1, adresses réservées

$E0/D000-DFFF : 2 Ko réservés au système ProDOS16

$E1/0000 : Point d'entrée du System Loader

$E1/00A8 : Point d'entrée de ProDOS16

$E1/D000-DFFF : 2 Ko réservés au système ProDOS16

Le point d'entrée du système d'exploitation ProDOS16 se trouve dans le banc $E1 à partir de l'adresse $00A8. Ce vecteur représente le point d'entrée de toutes les commandes transmises au MLI. Pour mémoire, ProDOS8 possède son point d'entrée dans la Global Page, adresse $BF00 (MLIEntry).

Le point d'entrée du System Loader se trouve dans le banc $E1 à partir de l'adresse $0000. Ce vecteur représente le point d'entrée de tous les appels d'outils implémentés soit dans la ROM ToolBox, soit dans la RAM et installés par le programme d'application en cours d'exécution.

Vecteurs d'entrée du System Loader

$E1/0000-0003 DISPATCH1. Première entrée dans la routine du dispatcher des outils en ROM (JSL, saut adresse absolue longue). JSL du code utilisateur directement vers l'adresse pointée en ROM.

$E1/004-0007 DISPATCH2. Deuxième entrée dans la routine du dispatcher des outils en ROM (JSL, saut adresse absolue longue). JSL du code utilisateur directement vers l'adresse pointée en ROM.

$E1/0008-000B UDISPATCH1. Première entrée dans la routine du dispatcher des outils en RAM. (JSL, saut adresse absolue longue). JSL du code utilisateur directement vers l'adresse pointée en RAM.

$E1/000C-000F UDISPATCH2. Deuxième entrée dans la routine du dispatcher des outils en RAM. (JSL, saut adresse absolue longue). JSL du code utilisateur directement vers l'adresse pointée en RAM.

Vecteur d'entrée des commandes MLI de ProDOS16

$E1/00A8-00BF PRO16MLI. Ensemble des adresses réservées à ProDOS16 pour des appels provenant des commandes MLI.

$E1/00A8-00AB Point d'entrée des commandes MLI du système ProDOS16.

$E1/00AC-00B9 Réservés.

$EA/00BA-00BB 2 bytes mis à zéro.

$E1/00BC Byte d'attribut de la variable OS_KIND. La lecture de cette adresse permet de déterminer le système actuel valide :
$00 = ProDOS8
$01 = ProDOS16

$E1/00BD Byte d'attribut de la variable OS_BOOT. La lecture de cette adresse permet de déterminer le système ayant été booté d'origine :
$00 = ProDOS8
$01 = ProDOS16

$E1/00BE-00BF Drapeau sur 16 bits. Le bit 15 détermine si le système ProDOS16 est actif ou non :
bit 15 = 0, ProDOS16 est inactif
bit 15 = 1, ProDOS16 est actif
Les bits 14 à 0 étant inutilisés.

Memory Manager (outil de travail)

ProDOS16 exploite les nombreuses propriétés du Memory Manager. Chaque allocation de zone mémoire, ou toute modification intervenant dans l'attribution d'un pointeur système, est réalisée en association avec le Memory Manager. C'est un outil de travail faisant partie de l'ensemble des outils qui composent la ToolBox de l'Apple *II*GS . Le Memory Manager réside en mémoire morte (ROM) et contrôle l'allocation des adresses dans les différents bancs mémoire.

La bit map de la zone mémoire est gérée en association avec ProDOS16 à travers le System Loader et le Memory Manager.

Le résultat a pour conséquence l'attribution des pages mémoire avant le chargement d'un programme, ou pour des applications nécessitant des tampons mémoire pour le traitement des données (fichier ouvert).

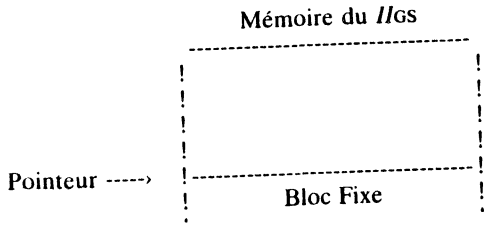
Le Memory Manager a pour mission essentielle de prendre en charge toute la gestion des bancs mémoire disponibles, et d'en tirer profit au maximum.

Utilisation des blocs du Memory Manager

Type de bloc	Signification
Blocs fixes	Ce bloc devra être logé à une adresse spécifiée au préalable. Ces blocs sont repérés par des pointeurs.
Blocs relogeables	Ce bloc sera déplacé par le Memory Manager dès qu'il le jugera nécessaire. Ces blocs relogeables sont repérés par des handles (gestionnaires), qui sont en fait des pointeurs auxiliaires sur des pointeurs maîtres.

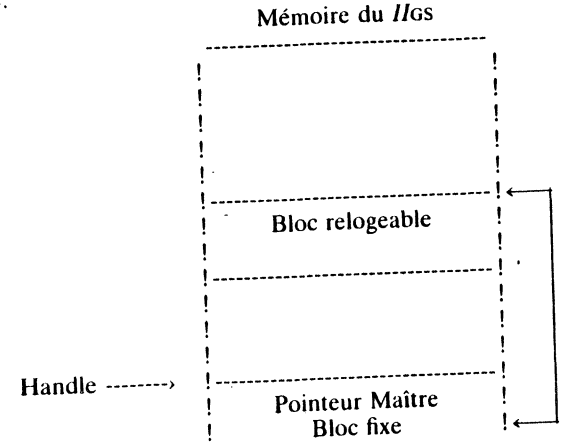
Qu'est-ce qu'un pointeur?

Un pointeur est une adresse qui pointe l'adresse de début d'un bloc normalement fixe. Sous ProDOS16, un tel pointeur est codé sur 32 bits, ce qui équivaut à un mot long codé sur 4 bytes (DWord).



Qu'est-ce qu'un pointeur maître?

Par analogie à un pointeur normal, un pointeur maître est un bloc fixe contenant l'adresse d'un bloc relogeable. Le handle, ou gestionnaire, connaît par définition l'adresse d'un bloc fixe qui pointe l'adresse d'un bloc relogeable. Quand le bloc change de banc mémoire, c'est à la charge du Memory Manager de stocker sa nouvelle adresse dans le pointeur maître qui le désigne. En aucun cas, la handle n'est modifié. Sous ProDOS16, un tel pointeur est codé sur 32 bits, ce qui équivaut à un mot long codé sur 4 bytes (DWord).



L'accès à un point d'entrée situé dans un bloc relogeable ne pourra se réaliser par un simple pointeur. C'est au Memory Manager de reloger le bloc et de modifier le pointeur en conséquence. Tout comme un bloc fixe, un bloc relogeable est repéré dans la mémoire du GS par un handle. C'est à l'application de gérer le pointeur en réajustant à chaque fois le handle.

Zone mémoire pouvant être allouée par le Memory Manager

$00/0800-BFFF
$01/0800-BFFF
$02-$3F/0000-FFFF

$E0/2000-BFFF
$E1/2000-BFFF

Remarque

Le reste des adresses des différents bancs mémoire ne pourra être alloué à travers le Memory Manager.

ProDOS16 ET SES PERIPHERIQUES EXTERNES

Généralités sur les Devices

Un périphérique externe est un équipement extérieur à l'unité centrale, constitué en général par du matériel électronique (hardware) pouvant transférer des informations sous forme de données codées selon la technique du tout ou rien (0 ou 1). Ce procédé d'encodage des données est exploité depuis le début avec la gamme des Apple*II* en général et avec le GS en particulier.

Un lecteur de disques (Disk*II*, 800 Ko, ou disque dur), une imprimante, la souris et le joystick sont classés parmi les périphériques dits Devices externes. Le clavier et l'écran vidéo sont par définition aussi assimilés à des Devices externes.

Sous ProDOS16, les devices regroupent deux grandes catégories, ceux qui véhiculent les informations en entrée (Input Device) et ceux qui véhiculent les informations en sortie (Output Device). Un Input Device transfère les données via le microprocesseur intégré dans l'unité centrale, vers un Output Device. Un Input/Output Device effectue le transfert des données dans les deux sens. Comme exemple, on pourrait citer le lecteur de disques comme Input/Output Device.

Block Devices

DEVICE DRIVERS – APPLE *II*GS

• Un Device driver, dans le sens le plus large du terme, est un ensemble de sous-programmes ou routines de pilotage, permettant au système ProDOS16 de communiquer sans problèmes avec les différents périphériques en ligne.

• Associé à un Block Device, un driver représente un ensemble de programmes élémentaires, destinés à organiser les transferts et liaisons entre plusieurs périphériques via le système d'exploitation, permettant ainsi une extension des possibilités de ProDOS16.

• Les programmes Device driver sont le fruit d'une association de multiples routines, souvent très courtes, et liées entre elles par des appels imbriqués.

• Un Device driver, une fois mis en place, fait partie de l'environnement du système d'exploitation. Par ailleurs, une partie souvent résidente de device driver existe d'origine, implémentée soit dans la ROM système, soit dans la ROM se trouvant sur une carte interface par exemple.

• Deux types de périphériques sont disponibles sur le GS : le Block Device et le Character Device.

PERIPHERIQUES DU TYPE BLOCK DEVICE

Par définition, un Block Device est un périphérique destiné à lire ou à écrire des informations sous forme de blocs de données et dans un temps alloué au préalable. Chaque bloc se compose de données, fixé à 512 bytes sous ProDOS16. Par ailleurs, ce type de Devices est classé dans les périphériques à accès aléatoire : l'accès à un bloc bien déterminé peut se faire sélectivement ou à travers le bloc qui le précède ou celui qui le suit. Un Block Device permet d'étendre les possibilités de sauvegarde de masse, tout en autorisant l'accès aux informations à partir de l'unité centrale. On dit que les informations véhiculent tout aussi bien en entrée qu'en sortie (du MPU) : c'est un Block Device Input/Output. Un lecteur de disques répond à ces critères de Block Device Input/Output.

Le système d'exploitation ProDOS16 est totalement compatible envers les Blocks Devices grâce aux commandes MLI appropriées. Associé aux commandes READ ($12), WRITE ($13) et à celles utilisant les propriétés d'accès aux fichiers, ProDOS16 autorise en plus quatre commandes MLI dites de bas niveau. Ces appels particuliers au MLI retournent après l'exécution de la routine un certain nombre d'informations à travers la table des paramètres de la commande. Ces paramètres sont exploités par la suite par les Blocks Devices concernés.

GET_DEV_NUM ($20)	Retourne l'attribut Dev_Num associé au nom du Block Device et alloué par le système ProDOS16. La valeur de Dev_Num sera comprise entre $00 et $FF inclus.
GET_DIB ($21)	Retourne dans le champ DIB le pointeur de la table des paramètres relatifs aux Blocks Devices (Device Information Block).
READ_BLOCK ($22)	Lecture d'un bloc de données d'une taille de 512 bytes à partir du Block Device spécifié.
WRITE_BLOCK ($23)	Ecriture sur le Block Device sélectionné et à partir du tampon mémoire spécifié, d'un bloc de données d'une taille de 512 bytes.

Remarques

• Le paramètre Dev_Num est requis pour le traitement de certaines commandes MLI transmises à ProDOS16, en particulier les commandes READ_BLOCK et WRITE_BLOCK. Ces accès sollicitent le device sur lequel se trouve stocké le fichier en cours de traitement.

• Un Block Device nécessite en général un device driver. Le driver consiste en général en une routine résidente au système ProDOS16 et destinée à effectuer la conversion des blocs logiques en pistes et secteurs au moment de sauvegarder les données sur le disque. Consulter à cet effet l'ouvrage **Système ProDOS de l'Apple *II*GS**, du même auteur, qui traite un certain nombre de routines ProDOS, en particulier les Drivers du DISK *II*.

• Le Device driver peut faire partie intégrante de l'interface qui pilote le Block Device, dans ce cas elle n'utilise pas la routine implémentée d'origine à ProDOS16 (Drivers Disk *II*, début adresse $F800). Le device driver peut se trouver en ROM résidente, ce qui est le cas de l'Unidisk 3,5 pouces ou encore faire partie du système ProDOS16 (Disk *II*).

• Le RAM Disk, Block Device virtuel, est un artifice créé par logiciel système sous ProDOS8 et destiné à simuler un device classique. ProDOS16 ne sollicite pas particulièrement le RAMDisk, mais permet de l'assimiler à un Block Device externe du même rang que les périphériques du même type.

DEVICE INFORMATION BLOCK – DIB

ProDOS16 gère une table d'information allouée à chaque Block Device en ligne. Cette table, nommée Device Information Block (DIB), est initialisée au moment du boot du système, et mise à jour à chaque recherche d'un device en ligne.

Table Device Information Block

Pos.	Utilisation
$00-$03	Pointeur de la prochaine table DIB. Contient des zéros s'il n'y a pas d'autre table DIB.
$04-$05	Device Reference Number (Dev_Num). Ce paramètre représente le numéro du device dont les caractéristiques sont définies dans cette table DIB.
$06-$15	Device Name (Dev_Name). Ce champ représente le nom du device, codé sur 16 caractères, avec un caractère espace à la suite. Le nom du device est attribué à chaque phase de recherche des périphériques en ligne (Device Scan). – Disk *II* – UniDisk – DuoDisk – ProFile
$16-$25	Nom complémentaire, codé sur 16 caractères, avec un caractère espace à la suite. Ce champ n'est pas exploité par ProDOS16, sa raison d'être n'étant que purement informative. – DISK *II* – UNIDISK – DUODISK – PROFILE

Pos.	Utilisation
$26-$27	Caractéristiques du Device. Deux bytes renseignent le système sur le Device en ligne : Block Device ou Character Device.
$28-$29	Identification du Device. Ce champ permet d'identifier le matériel Apple : toute valeur de 0 à 32767 inclus est réservée à Apple; toute valeur de 32768 à 65535 inclus est réservée à du matériel externe à Apple et attribué par le concepteur.
$2A	Numéro du slot. Il identifie le numéro du slot ou du port auquel fait référence le Device.
$2B	Unit_Number. C'est un paramètre particulier, dépendant directement du protocole défini au préalable et permettant de situer un Unit raccordé à un slot ou à un port interne donné.
$2C-$2F	Driver Entry Point. Adresse d'entrée de la routine pilote du Device (driver device).
$30-$33	Compteur de blocs logiques. Est utilisé en association avec un Block Device pour comptabiliser les blocs logiques.
$34-$35	Version Driver. Désigne le numéro de la version du pilote (driver). La valeur zéro spécifie que le pilote n'a pas de numéro de version.
$36-$3F	Réservés à des applications futures.

Encodage d'un Block Device (offset $26-$27)

Bits	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
Valeurs	1	W	R	F	RM	0	0	I	E	A	P	P	P	L	L	L

Encodage d'un Character Device (offset $26-$27)

Bits	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
Valeurs	0	W	R	0	0	0	0	I	E	A	P	P	P	L	L	L

Remarques

- si le bit F (15) est nul (bit 15 = 0), le périphérique est un Character Device.
- si le bit F (15) est à 1 (bit 15 = 1), le périphérique est un Block Device.
- les autres valeurs sont communes aux deux types de Devices.

Signification des différentes valeurs

W = 0 écriture interdite
W = 1 écriture autorisée

F = 0 Device ne supporte pas de commande
F = 1 Device autorise la transmission de commandes

RM = 0 le périphérique n'est pas amovible
RM = 1 le périphérique est amovible

I = 0 le Device ne peut générer une interruption
I = 1 le Device peut générer une interruption

E = 0 le pilote du Device fonctionne en mode natif
E = 1 le pilote du Device fonctionne en mode émulation

A = 0 le pilote est neutralisé
A = 1 le pilote est actif

PPP = identifie le protocole relatif au Device

PPP = 000 Protocole initial
PPP = 001 Protocole du Protocol Converter
PPP = 010 Protocole du Protocol Converter étendu
PPP = 011 Protocole relatif au Pascal 1.1
PPP = 100 Protocole relatif à un cas particulier
PPP = 101 Réservé
PPP = 110 Réservé
PPP = 111 Réservé

LLL = identifie le type de pilote (driver en ROM?)
LLL = 000 ROM résidente sur la carte interface
LLL = 001 ROM d'expansion utilisée
LLL = 011 RAM
LLL = 100 Réservé
LLL = 101 Réservé
LLL = 110 Réservé
LLL = 111 Réservé

ProDOS Block Device – I/O Protocoles

Les protocoles relatifs aux Devices supportés par le système ProDOS8 sont théoriquement valides sous ProDOS16. En présence de manipulations particulières avec des Blocks Devices, ProDOS16 nécessite une extension des protocoles définis avec du matériel antérieur à la sortie du GS. C'est ainsi que les Devices de la nouvelle génération utilisent une adresse d'entrée définie par le Protocol Converter. Ce vecteur d'entrée correspond au début de la routine pilote du Device.

Le SmartPort supporte tous les Blocks Devices de la nouvelle génération. Le Disk *II* faisant partie de la génération précédente, nécessite une carte contrôleur comme interface. Le port intégré permet une extension des possibilités d'un driver classique et autorise, en plus des commandes MLI, des commandes étendues, particulières au SmartPort : STATUS, READ BLOCK, WRITE BLOCK, FORMAT, CONTROL, INIT, OPEN, CLOSE, READ et WRITE.

L'extension d'un protocole défini pour le SmartPort s'étend uniquement à l'écriture du pointeur, qui sera codé sur 4 bytes au lieu de 2 à 3 bytes comme initialement.

Character Devices

Un Character Device désigne les périphériques qui exploitent les nombreuses possibilités du caractère, tels le clavier, les imprimantes, le moniteur vidéo, les ports de communication, etc.

Par définition, un Character Device est un périphérique destiné à lire ou à écrire des flots d'informations sous forme de données, dans un ordre et un temps alloué au préalable. Par ailleurs, ce type de Devices est classé parmi les périphériques à accès séquentiel: l'accès à une donnée se fera selon un protocole bien établi, par flots consécutifs de données à un emplacement prévu et débutant à une position déterminée au préalable. Ce type de Devices est utilisé en général pour transmettre des données d'une unité centrale vers une autre, et dans un sens bidirectionnel. Comme les Blocks Devices, les Character Devices se divisent en trois groupes: Input Devices, Output Devices et Input/Output Devices.

ProDOS16 ne supporte pas directement ce type de Devices, mais nécessite au préalable l'appel d'une commande appropriée à travers le MLI.

Named Device

ProDOS16 autorise en présence de Blocks Devices, de les différencier en leur attribuant un nom. Cet artifice est une opportunité réservée à un programme utilisateur (ou au programmeur), permettant de retrouver un device dans certaines structures complexes de chemins d'accès (Pathname) ou de volumes en lignes.

De quelque manière que ce soit, ProDOS16 supporte une syntaxe limitée dans l'attribution des noms de Devices et retourne du MLI certains résultats concrets en employant les commandes GET_DIB ($21), VOLUME ($08) et GET_DEV_NUM ($20).Certaines commandes nécessitent la mise en place préalable du nom du volume, ou retournent un attribut particulier: avec la commande GET_DEV_NUM par exemple, le paramètre Dev_Num est retourné dans la table des paramètres.

La syntaxe autorisée dans l'attribution d'un nom de Device répond à une convention établie au préalable par Apple Computer. Par ailleurs, un nom assigné à un device pourra être modifié par la suite.

Syntaxe d'un Block Device: .Dn

Où n représente un nombre codé sur un ou deux digits.

Remarques

• ProDOS16 version 1.0 supporte au total jusqu'à 14 Devices actifs, tandis que la version 2.0 n'est plus limitée dans le nombre de Devices en ligne. L'Apple *II*GS autorise jusqu'à 4 Devices connectés au SmartPort et deux devices supplémentaires par slot d'extension (slots 1 à 7).

• Lors du boot du système ProDOS16, celui-ci recherche à travers tous les slots en ligne un ou plusieurs Devices actifs (Pseudo slot de la ROM et ROM physique d'une carte interface). Chaque Device trouvé, et compatible hard et soft, sera placé dans la liste des Devices gérée par ProDOS16 en mémoire (table Device Information Block – DIB). Chaque Device sera affecté de deux attributs: un numéro de device (Dev_Num) et un nom de device (Dev_Name).

• Le tableau de bord permet à l'utilisateur de vérifier l'attribution des Devices aux différents slots existants (slots 1 à 7). ProDOS16 ne peut en aucun cas gérer simultanément des Devices connectés aux ports internes et externes assignés au même slot (ROM résidente du port intégré, ou ROM d'une carte interface mise en place dans le slot).

Volume Control Block

Un Volume Control Block (VCB) identifie par la suite les entrées des différents Devices en ligne, mémorisés au moment du boot de ProDOS16. La raison d'être d'un VCB est essentiellement la notion de volume introduite par un Block Device. C'est ainsi que le nom du Volume Directory d'un disque sera copié dans le VCB aux positions relatives $01 à $0F de son entrée, tandis que le numéro du bloc du Key Block sera copié à la position relative $16.

Lorsque ProDOS16 est exécuté pour la première fois, les slots sont parcourus à la recherche des Devices valides et une table est créée en mémoire: c'est la VCB. Elle contient les caractéristiques essentielles relatives aussi bien au volume qu'au Device courant, et pourra contenir jusqu'à huit entrées par table. Sous environnement ProDOS16 et GS, le nombre de Devices en ligne n'étant nullement limité, uniquement huit pourront être actifs à un instant donné. Si la table VCB est pleine et qu'un autre Device est sollicité, le système retourne le code d'erreur $55, VCB Table Full.

Gestion des interruptions

Le GS possède une structure telle qu'elle autorise l'interception de ses différents niveaux d'interruptions soit par routines résidentes en ROM (firmware), soit par logiciel (software). Les pointeurs par défaut des différents vecteurs d'interruptions se trouvent implémentés dans la ROM résidente du Moniteur étendu. Ce sera au programme utilisateur de veiller à la mise en place du bon vecteur destiné à intercepter par la suite l'interruption à gérer.

Si une interruption est détectée en présence d'un Apple *II*GS et sous environnement ProDOS16, le vecteur de prise en charge se trouve à partir de l'adresse $00/03FE de la page 3 (banc $00).

• Si l'interruption intervient en mode émulation, la fréquence d'horloge sera standard et cadencée à 1 Mhz, les registres au format 8 bits et l'appel transmis au vecteur **Interrupt Entry** à travers la Global Page de ProDOS, adresse $BFEB.

Processus d'interruption en mode émulation

- le mode émulation sera le mode courant par défaut;
- l'horloge du processeur sera calée sur 1 Mhz;
- la page 1 texte sera celle affichée par défaut;
- la mémoire principale sera celle accessible par défaut;
- l'espace mémoire $D000-$FFFF sera copié dans le banc $00 et aux mêmes adresses;
- la pile sera placée à la page 1 du banc $00;
- la page zéro sera initialisée dans le banc $00;

– sauvegarde du pointeur de la pile auxiliaire (mémoire auxiliaire) et restaure le pointeur de la pile actuelle (mémoire principale).

• Si l'interruption est prise en compte en mode natif, l'horloge sera cadencée à une fréquence de 2,5 Mhz, les registres exploités dans leur totalité (16 bits) et l'appel transmis via l'adresse de la routine de service. ProDOS16 supporte jusqu'à 16 routines de service destinées à la gestion des interruptions possibles. Lors d'une interruption IRQ, l'occupation des adresses de la table Interrupts sera testée individuellement et le vecteur programmé exécuté.

Pour la mise en place de chaque routine de service, il faudra faire appel à la commande ALLOC_INTERRUPT ($30). Elle a pour but de mettre en place une routine de service, ce qui effectue une mise à jour de la liste des pointeurs. La commande DEALLOC_INTERRUPT ($31) purge le pointeur d'une routine de service mise en place initialement dans le vecteur d'interruptions.

Pointeur des interruptions IRQ (IRQLOC)

$00/03FE-03FF Adresse de prise en charge des interruptions du type IRQ (ProDOS16).

– normalement, l'adresse (LByte, HByte) pointe le vecteur MON du Moniteur, adresse $FF65. C'est l'entrée standard dans le Moniteur avec un beep transmis dans le haut-parleur.

– si ProDOS16 est booté en mémoire, le vecteur IRQLOC pointe l'adresse d'entrée de gestion de l'interruption IRQ.

ProDOS16 ET SON ENVIRONNEMENT

Système disque complet

L'Apple *IIGS* , de part sa conception hardware et software, autorise des applications en modes émulation et natif. En mode émulation, une grande partie des programmes de la bibliothèque existante pourra être exécutée. La compatibilité ascendante envers les programmes reste cependant une notion relative, ne permettant pas de fixer de limites réelles. Dans le mode natif, tous les programmes développés pour exploiter pleinement les capacités du microprocesseur 65C816, pourront être exécutés. Pour ce faire, le disque d'application devra être configuré en conséquence afin de supporter les programmes courants.

Un disque sytème complet, doté de tous les fichiers système ProDOS8 et ProDOS16, nécessite un Block Device de 800 Ko au minimum (Unidisk 3,5 pouces ou disque dur).

Contenu d'un disque système complet

Directory/File	Signification
PRODOS	Ce fichier comporte les routines et sous-programmes pour charger par la suite le système d'exploitation approprié, en fonction du programme à exécuter. Reboote le système sur un pilote au moment de quitter le programme en fin de traitement.
SYSTEM/	Fichier table des matières auxiliaire (SubDirectory), contenant les fichiers système et la sélection du programme d'application.
P8	Fichier système spécifique aux programmes développés pour traiter des mots de 8 bits.
P16	Fichier système spécifique aux programmes développés pour traiter des mots de 16 bits.
xxxx.SYS16	LOADER, fichier de chargement du système de l'Apple *IIGS* . Le nom du fichier possède le suffixe .SYS16.
START	Fichier standard de sélection du programme à exécuter.
LIBRARY/	Fichier table des matières auxiliaire contenant les fichiers librairie nécessaires à ProDOS.
TOOLS/	Fichier SubDirectory, contenant tous les outils destinés à être chargés en mémoire vive.
FONTS/	Fichier SubDirectory, contenant toutes les polices de caractères exploitables sous ProDOS16.
DESK.ACCS/	Fichier SubDirectory, contenant tous les fichiers accessoires de bureau.
SYSTEM.SETUP/	Fichier SubDirectory, contenant les vecteurs d'initialisation des programmes à traiter. – TOOL.SETUP, fichier contenant le patch et le sous-programme pour installer les outils de la ROM résidente. Ce fichier doit se trouver en tête dans le SubDirectory SYSTEM.SETUP/, il sera chargé avant tout autre fichier s'y trouvant.
BASIC.SYSTEM	Fichier système, interface requise avec des applications faisant appel au Basic AppleSoft.

Démarrage et retour d'une application

A la fin de l'exécution d'un programme, le système ProDOS est doté d'un mécanisme permettant de passer le contrôle à une routine particulière : PQUIT. Cette routine, mise en place à la fin du boot, est commune aux deux systèmes (ProDOS8 et ProDOS16). PQUIT est sollicitée au moment de quitter l'application courante.

• PQUIT est une routine dispatcher, permettant de faire un choix en ce qui concerne le système d'exploitation à exécuter par la suite. Après la séquence de démarrage et d'installation du système ProDOS, PQUIT et le System

Loader sont implantés, et résident par la suite en permanence en mémoire. PQUIT charge ProDOS16 à travers une commande issue du System Loader.

• La routine MLI PQUIT possède deux points d'entrée : P8PQUIT et P16PQUIT.

– à n'importe quel moment d'une application en cours d'exécution sous ProDOS8, le contrôle pourra être passé à la routine PQUIT par le simple fait de solliciter l'entrée P8PQUIT

– de même, sous environnement ProDOS16, l'exécution d'une application pourra être abandonnée et le contrôle passé au système via l'entrée P16PQUIT de la routine PQUIT

– après avoir quitté une application en cours, que ce soit sous ProDOS8 ou ProDOS16, le premier programme stocké dans la suite du chemin mémorisé dans le VCB sera exécuté : /VOLUME/PRODOS, où /VOLUME désigne le nom du volume sélectionné et PRODOS le fichier système de démarrage du système ProDOS

– par la suite, le bon système sera exécuté (ProDOS8 ou ProDOS16) à travers la commande QUIT ($29), ce qui aura comme effet de donner le contrôle à l'entrée correspondante de la routine PQUIT (P8PQUIT ou P16PQUIT)

– la routine PQUIT supporte en réalité trois types de commandes à travers la commande MLI QUIT : la commande QUIT standard sous ProDOS8, la commande QUIT étendue sous ProDOS8 et la commande QUIT sous ProDOS16.

COMMANDE QUIT STANDARD SOUS ProDOS8

La commande QUIT standard sous ProDOS8 nécessite, avant l'appel du MLI, la mise en place dans la table des paramètres (Parameter Block) du champ Parameter Count.

Table des paramètres de la commande QUIT standard – ProDOS8

$00	Parameter_Count = $ $04
$01	Champ nul – byte
$02-$03	Champ nul – Word, ou 2 bytes
$04	Champ nul – byte
$05-$06	Champ nul – Word, ou 2 bytes

La variable Parameter_Count spécifie que quatre champs sont disponibles dans la table des paramètres. Parameter_Count devra être déclaré, et sa valeur portée à $04. Le reste des champs sera nul afin de garder une symétrie avec la commande QUIT étendue sous ProDOS8.

Avec la version 1.1.1, tous les champs devront être déclarés suivant le modèle présenté, et les champs nuls dotés de zéros. Cette pratique annule tous les chemins déclarés au préalable et nécessite la mise en place du chemin (Pathname) de la prochaine application à exécuter.

COMMANDE QUIT ETENDUE SOUS ProDOS8

La commande QUIT étendue de la version 1.3 de ProDOS8 diffère tant soit peu de celle de la version standard. La variable Parameter_Count est toujours égale à $04. Les deux premiers champs, Quit_Type et Pathname, admettent maintenant des valeurs, tandis que les champs suivants sont mis à zéro.

Table des paramètres de la commande QUIT étendue – ProDOS8

$00	Parameter_Count = $04
$01	Quit_Type – byte
$02-$03	Pathname – Word, ou 2 bytes
$04	Champ nul – byte
$05-$06	Champ nul – Word, ou 2 bytes

• La commande étendue diffère de la commande standard de par la déclaration de ses deux premiers paramètres.

• Le premier paramètre (Byte) définit le chemin du retour.

– si Quit_Type est nul, le mode de retour de l'exécution en cours sera identique à celui de la commande standard;

– si Quit_Type a comme valeur $EE, le système ProDOS8 interprète le champ Pathname (Word, 2 bytes) comme un pointeur codé sur deux bytes. Ce pointeur désigne la chaîne de caractères valides qui représente en fait le chemin (Pathname) de la prochaine application à exécuter.

• Sous ProDOS8, la commande QUIT étendue est significative en présence d'un environnement GS et uniquement si la routine PQUIT est présente au moment de son interprétation comme telle. En présence d'une configuration différente, la commande QUIT étendue se comporte de façon équivalente à la commande standard.

• Par ailleurs, si la commande QUIT étendue est transmise sous un environnement différent de celui décrit précédemment, certaines perturbations dans la suite du déroulement des opérations sont à craindre. En effet, le chemin déclaré (Pathname) sous ProDOS8 utilise certains vecteurs de la mémoire, en particulier les adresses $00/0200-02FF qui correspondent au tampon clavier.

COMMANDE QUIT SOUS ProDOS16

La commande QUIT ($29) est spécifique à la version 2.0 de ProDOS16. Elle diffère de la version 1.0 de par son écriture des champs de la table des paramètres.

Table des paramètres de la commande QUIT ProDOS16 version 2.0

$00-$01	Caller_Id (Identifie la commande ProDOS16)
$02-$09	Null_Field (8 bytes nuls, réservés pour des applications futures).

A propos de la version 1.0 de ProDOS16

La commande QUIT ($29) disponible avec la version 1.0 de ProDOS16, comporte une table de paramètres différente de la version 2.0. La composition de ses deux champs est la suivante :

$00-$03	Chain_Path – Mot long, codé sur 4 bytes
$04-$05	Return_Flag – Mot, codé sur 2 bytes

Chain_Path

C'est un pointeur codé sur quatre bytes (DWord = mot long). Il désigne le chemin contenant le nom de la prochaine application à exécuter. Format du pointeur : High Word, Low Word, où le byte le moins significatif de High Word est toujours de valeur zéro.
Valeurs possibles de Chain_Path : $0000 0000 à $00FF FFFF

Return_Flag

C'est un drapeau structuré selon une combinaison binaire des deux bytes représentatifs. Ce Flag signifie à la routine PQUIT si le retour doit se faire à un programme référencé par l'adresse sur le sommet de la pile, ou si au contraire une application résidente devra être exécutée. La pile est une structure mise en place après la phase du boot, et maintenue en mémoire sous ProDOS16.

– si le résultat du drapeau est 'vrai', PQUIT dépile l'adresse qui pointe le Device courant, contenant le programme à exécuter. Comme exemple d'un tel mécanisme, on pourrait citer le programme du sélecteur (Start File Selector) qui passe le contrôle au programme à exécuter et sauvegarde comme adresse de retour l'identificateur du Device (ID) sur le sommet de la pile. Au retour de l'application en cours d'exécution, le sélecteur sera le programme par défaut à exécuter. Le mécanisme, automatisé par la routine PQUIT, est rendu actif à travers la commande MLI QUIT. Cette forme de retour de programme n'est pas disponible sous ProDOS8, la pile interne gérée par ProDOS16 ne lui étant pas accessible.

– si le résultat du drapeau est 'faux', PQUIT ne fera pas appel à la pile interne.

Outils et routines sollicitées par ProDOS16

Memory Manager

Le Memory Manager est l'outil numéro deux dans la liste des outils résidant dans la ROM. C'est le gestionnaire de la mémoire du GS en mode natif, permettant de faire cohabiter les blocs fixes et les blocs relogeables. Ce sera à l'application de gérer les pointeurs, en déférençant le handle. Un programme en cours d'exécution va allouer, à un moment ou à un autre, des blocs de mémoire et va les désallouer dès qu'il aura terminé avec son utilisation. Dans le flot d'attribution ou d'annulation des blocs mémoire, l'espace mémoire total sera à un moment ou à un autre totalement fragmenté. C'est pourquoi le Memory Manager tente régulièrement de compacter la mémoire. Sa fonction essentielle consiste alors à déplacer des blocs relogeables de telle sorte qu'ils comblent le vide intialement laissé par les allocations de blocs. Les blocs fixes ne sont pas affectés par cette pratique, le Memory Manager essaie d'allouer ce type de blocs vers le bas de la mémoire.

System Loader

Le System Loader est un outil de travail non résidant en ROM, mais chargé en mémoire lors de la mise en place du système ProDOS16. Il occupe la position 17 dans la liste des outils de travail. Le System Loader comporte un certain nombre de routines permettant à une application d'être fragmentée en plusieurs segments. Une des ses fonctions essentielles sera de placer en mémoire vive (RAM) les segments non présents, indispensables à la bonne exécution de l'application.

Le System Loader fait partie intégrante du système ProDOS16 et supporte les modes de chargement statique et dynamique des segments programme et routines librairies.

Par ailleurs, le System Loader charge les fichiers conformes au format établi (format du module objet); recherche dans un volume désigné par le chemin un nom de fichier conforme à la structure de programmation appliquée à l'Apple *II*GS (Programmer's Workshop Linker) et autres composantes compatibles.

Remarque

Un segment statique est chargé en mémoire au début du programme et doit y rester jusqu'à la fin de son exécution. Un segment dynamique est chargé en mémoire au fur et à mesure des besoins, puis purgé par le Memory Manager quand il n'est plus référencé. C'est le Segment Loader qui a la tâche de détecter les segments à charger ou à purger, et le Memory Manager les exécute.

Scheduler

Scheduler est un outil de travail résidant en ROM. Il se trouve dans la liste des outils sous le numéro 7. Scheduler s'occupe essentiellement de la gestion de certains délais quand des accessoires de bureau ou d'autres tâches essaient de solliciter des ressources occupées. Scheduler effectue encore un contrôle (checksum) périodique d'un drapeau interne pointé par un mot actif tout en maintenant en ligne un accessoire de bureau qui sera actif lorsque le drapeau changera d'état. La routine d'interruption valide sera prise en compte par l'outil Scheduler à travers une commande ProDOS16.

UserID Manager

L'UserID Manager fait partie des outils appelés Miscellaneous Tools référencés sous le numéro 3 de la liste des outils résidant en ROM. Sous l'appellation Miscellaneous Tools se regroupe toute une panoplie de routines sans lien apparent, toutes placées en ROM.

La routine UserID affecte un numéro à un nom de programme faisant partie d'un chemin d'accès (Pathname). Chaque bloc mémoire alloué par le Memory Manager est marqué par l'intermédiaire de UserID, ce qui permet de savoir à quel programme système, quelle application, ou quel accessoire de bureau il appartient.

Chaque bloc de mémoire possède un nombre déterminé d'attributs stockés sur un mot de 16 bits et attribués par UserID : c'est le paramètre ID Type. Il décrit le type de segment à charger et son occupation mémoire. Tous les blocs attribués sous ProDOS8 et ProDOS16 sont du type 3; les blocs attribués par le System Loader sont du type 7; les blocs contrôlés par des applications sont du type 2; les blocs contenant des segments d'applications sont du type 1.

System Death Manager

Comme le Scheduler, le System Death Manager fait partie des outils Miscellaneous Tolls résidant dans la ROM. La routine gère les erreurs fatales imputées le plus souvent au plantage du système d'exploitation.

Toutes les erreurs fatales, celles de ProDOS16 comprises, sont transmises au System Death Manager. Au retour de l'exécution, un message d'erreur est affiché à l'écran ou, si la gestion de l'erreur est interceptée par une routine de service (interruption) mise en place au préalable, un pointeur revectorise l'ensemble et affiche un message particulier sous la forme d'une chaîne de caractères (codes ASCII). Le programme d'application termine son exécution après que le System Death Manager aura été appelé.

Routines d'extension sous ProDOS16

Une certaine extension des routines permet une meilleure gestion des Devices sous ProDOS16. Des routines de bas niveau, implémentées d'origine dans la ROM ToolBox, peuvent être considérées comme faisant partie intégrante du système. Elles sont le plus souvent totalement transparentes aux applications. Deux types de routines sont disponibles : les gestionnaires des interruptions et les pilotes (drivers) des Blocks Devices.

GESTIONNAIRES D'INTERRUPTIONS (INTERRUPT HANDLERS)

L'Apple *IIGS* , doté d'origine d'une extension des routines d'interruption, supporte sous ProDOS16 jusqu'à 16 routines de service programmables à travers la commande ALLOC_INTERRUPT ($30). La commande DEALLOC_INTERRUPT ($31), à l'inverse de la commande ALLOC_INTERRUPT ($30), désalloue une ou plusieurs routines de service. L'écriture d'un gestionnaire d'interruptions (Interrupt handlers) nécessite de la part de l'utilisateur l'application de conventions établies par le concepteur.

Conventions du gestionnaire des interruptions

Les routines de service écrites spécialement pour le GS répondent d'une part à certains critères de la machine, et d'autre part, à un protocole établi au préalable. Le dispatcher des interruptions teste l'origine et l'état de l'interruption avant de passer le contrôle au handler :

e = 0	Bit du signe, résultat positif;
m = 0	Sélection mémoire/A, commuté en mode 16 bits;
x = 0	Sélection du registre X, commuté en 16 bits;
i = 1	Inhibition des interruptions, interruption inhibée;
c = 1	Retenue, résultat attendu.

Vitesse = rapide (cycle horloge = 2,5 Mhz)

Sous ProDOS16, lorsqu'une application se termine, et avant le retour au sous-programme appelant, le handler devra restaurer les registres suivants :

e = 0	Résultat positif;
m = 0	Mode natif;
x = 0	Mode natif;
i = 1	Inhibe les interruptions.

Vitesse = rapide (cycle horloge = 2,5 Mhz)

Remarques

• Si une interruption est rencontrée, le bit de la Carry sera purgé et mis à zéro (c = 0); dans ce cas le résultat n'est pas celui attendu. Si, par contre le résultat est celui attendu, le bit de la carry sera positionné à 1 (c = 1). Le retour de l'interruption se fera par un RTL (retour long de sous-programme).

• Si une interruption survient dans le déroulement d'une application, ProDOS16 dépile l'adresse de la routine de service, pointeur mis en place par ALLOC_INTERRUPT ($30), et met à jour l'indicateur de la Carry (C = ?).

• Si le mode émulation est actif, l'instruction RTI (retour d'interruption) donne le contrôle à la routine de service résidente et gère l'interruption système.

Mise en place d'une routine de service

Une routine de service ressemble le plus souvent à un petit programme destiné à gérer une interruption associée directement à une application particulière. Elle est destinée à traiter une interruption, ce qui empêche le plus souvent un plantage du système, puis de continuer l'application directement à la suite de l'interception de l'interruption. La routine de service sera chargée en mémoire en même temps que le programme utilisateur.

La routine de service est installée à travers la commande ALLOC_INTERRUPT ($30) du MLI qui lui affecte un numéro et positionne le pointeur sur son adresse courante.

Le dispatcher du système ProDOS16 gère une table des pointeurs (Interrupt Vector Table) permettant la mise en place jusqu'à 16 vecteurs d'interruptions. A chaque mise en place d'une routine de service, le dispatcher met à jour la table des pointeurs en ajoutant le nouveau pointeur à la suite de la table. Le principe de gestion de la table des pointeurs subit les mêmes règles et contraintes que la pile système : le premier pointeur mis en place dans la table possède la priorité absolue (priorité 1), le second pointeur la priorité 2, et ainsi de suite.

La variable Int_Code pointe dans la table des pointeurs du handler l'adresse d'implantation de la routine de service. La variable Int_Num identifie l'interruption mise en place en lui affectant un numéro dans le dispatcher des interruptions.

La commande DEALLOC_INTERRUPT ($31) permet de purger un pointeur dans la table des pointeurs et de ce fait annuler une routine de service

mise en place précédemment. Le fait de retirer une routine de service ne perturbe en rien l'ordre établi dans la table des pointeurs. En effet, chaque routine de service est gérée par un numéro de référence (Int_Num) pris en considération par ProDOS16 au moment de l'interception de l'interruption. La purge d'un vecteur d'interruption dans la table des pointeurs, déplace la suite des pointeurs encore valides vers le haut de la table, d'un nombre de bytes équivalent à la taille du pointeur éliminé. La fin de la table sera mise à jour par des zéros et disponible pour un nouveau pointeur.

DEVICE DRIVERS

En règle générale, le matériel connecté au GS devra être compatible à ce type de machine. La compatibilité ne s'arrête pas uniquement au matériel (hardware), mais les programmes résidents (software) qui l'équipent devront répondre à un protocole établi par avance. C'est ainsi que les Blocks Devices, lecteurs de disques, entrent dans cette catégorie de matériel à protocole. La ROM résidente, implémentée d'origine sur la carte contrôleur, devra être conforme au cahier des charges établi par Apple Computer. D'autre part, la routine de pilotage (driver routine) de l'interface du contrôleur devra utiliser les paramètres stockés aux adresses réservées de la page zéro (direct-page).

Conventions concernant les Blocks Devices

Comme ProDOS8, le système ProDOS16 supporte plusieurs types de Blocks Devices en entrée et en sortie pour véhiculer des blocs de données (I/O devices). Une autre catégorie de devices, le SmartPort et le SmartPort étendu, sont inclus dans la liste des protocoles fixés par Apple Computer. Tous ces périphériques permettent au mieux d'exploiter les capacités de sauvegarde de masse sous ProDOS16.

Remarque

Par la suite et uniquement dans ce paragraphe, la syntaxe ProDOS fait référence aux systèmes ProDOS8 et ProDOS16.

ProDOS Blocks Devices protocoles
Conventions relatives à la ROM

Au moment du démarrage du système, ProDOS recherche à travers les slots d'extension et ports intégrés, les Blocks Devices en ligne. Pour chaque slot d'extension ou port intégré occupé (Csxx), trois bytes significatifs sont testés envers un quelconque Block Device actif:

$Cs01 = $20
$Cs03 = $00
$Cs05 = $03

Par la suite, ProDOS effectue un contrôle au niveau de l'octet contenu à l'adresse $CsFF, où s représente le numéro du slot d'extension ou du port intégré.

- Si le contenu de $CsFF = $00, le sytème considère que le périphérique est un Disk *II* équipé d'une ROM au format 16 secteurs par piste. Le device est répertorié dans la table interne des devices (DIB) pour ProDOS16 et dans la Device Driver Table de la Global Page pour ProDOS8 (adresses $BF10-$BF2F). La routine de pilotage (driver routine) du Disk *II* fait partie inégrante du système ProDOS et supporte des lecteurs de disques souples (16 secteurs, 35 pistes) d'une capacité de volume de 280 blocs.

- Si le contenu de $CsFF = $FF, le système considère que le périphérique est un Disk *II* équipé d'une ROM au format 13 secteurs par piste. ProDOS ne supporte plus ce format, il ignore le device en ligne et continue sa recherche à travers un autre slot.

- Si le contenu de $CsFF est différent des valeurs $00 et $FF, le système considère que le contrôleur en ligne est du type intelligent (Smart). Si la valeur du Status Byte à l'adresse $CsFE fait ressortir que le Device peut être exploité en lecture, et son statut lu, ProDOS16 met à jour sa table des Devices interne, tandis que ProDOS8 copie dans la Global Page l'adresse du Device driver, où l'octet de poids fort est égal à $C et l'octet de poids faible au contenu trouvé dans l'adresse $CsFF. L'adresse du Device driver a comme valeur offset le contenu de $CsFF dont l'adresse de base se trouve être le point d'entrée de la routine de pilotage du slot 's'.

Adresses réservées dans la ROM interface

$CsFC-$CsFD — LByte, HByte. Donne le nombre total de blocs que peut gérer le Block Device. Cette information est utilisée au moment de la mise à jour de la bit map du disque par le programme de formatage. L'entrée du Volume Directory, positions relatives $29-$2A, est également mise à jour par le contenu de $CsFC-$CsFD.
Si la valeur des deux bytes est nulle ($0000), le nombre de blocs disponibles sur le device sera obtenu à travers la commande STATUS du Device considéré (SmartPort).

$CsFE — Status Byte. Les bits 0 et 1 seront significatifs, sinon ProDOS n'installe pas le driver dans son vecteur respectif (Global Page pour ProDOS8).
Bit 7 le Device est amovible.
Bit 6 gestion d'interruption du Device.
Bits 5-4 numéro du Volume du Device (0-3).
Bit 3 formatage supporté par le Device.
Bit 2 écriture autorisée par le Device.
Bit 1 lecture autorisée par le Device.
Bit 0 lecture du status device autorisée.

$CsFF — LByte. Offset de l'adresse du vecteur de pilotage du device (driver routine). ProDOS ajoute l'adresse offset contenue dans $CsFF à l'adresse $Cs00.

APPEL DE LA COMMANDE STATUS ($00)

Les seules commandes transmises par ProDOS à une routine Disk Driver font références à un Status Byte où les bits 0 à 3 sont positionnés à 1 (Status, Read, Write et Format autorisés).

Si le contenu des adresses $CsFC-$CsFD est nul, il sera nécessaire de faire appel à la commande STATUS ($00) qui permet de retourner dans les re-

gistres d'index X et Y le nombre de blocs supportés par le Device. Dès que la commande STATUS ($00) est transmise au driver, ce dernier effectue un contrôle et vérifie si le Device est apte à la lecture et à l'écriture.

- Si le Device n'est pas compatible, le driver positionne la Carry (C = 1) et retourne par l'intermédiaire de l'accumulateur, le code d'erreur approprié.

- Si le Device est compatible, le driver purge le contenu de la Carry (C = 0), charge un zéro dans l'accumulateur (A = 0) et retourne le nombre de blocs pouvant être gérés par le Block Device considéré. La valeur du nombre de blocs sera chargée dans les registres d'index X et Y, où X contiendra l'octet de poids faible et Y l'octet de poids fort.

Remarque

Si plus de deux Devices doivent être installés à un même slot d'extension, il faudra solliciter le SmartPort du slot 5. Ce slot émule à travers un port interne, un port intelligent qui permet de chaîner plusieurs types de Devices.

Unit_Number	ProDOS slot et drive
SmartPort 1	Slot 5, drive 1
SmartPort 2	Slot 5, drive 2
SmartPort 3	Slot 2, drive 1
SmartPort 4	Slot 2, drive 2

Table des paramètres (page zéro)

Adresse	Paramètre	Signification
$42	Command-_Number	0 = lecture du Status autorisé 1 = lecture autorisée 2 = écriture autorisée 3 = formatage autorisé
$43	Unit_Number	Bit 7 = numéro du Drive (0-3) Bits 6-4 = numéro du slot (1-7) Bits 3-0 = inutilisés
$44-$45	Buffer_Pointer	Pointe l'adresse de début du tampon (512 bytes) alloué à un fichier. Les données sont transférées du disque vers le tampon ou inversement.
$46-$47	Block_Number	Numéro de bloc du disque dont les données sont transférées du ou vers le tampon.

Codes d'erreurs possibles

Si la commande STATUS ($00), relative au Device, n'aboutit pas, la routine Device driver positionne la retenue (Carry = 1) et retourne un code d'erreur à travers l'accumulateur.

Code d'erreur	Message d'erreur
$27	I/O error Erreur d'entrée/sortie
$28	No device Connected Pas de device en ligne
$2B	Write Protected Protégé contre l'écriture

MACHINE LANGUAGE INTERFACE – MLI COMMANDES ProDOS16

Ce chapitre passe en revue les commandes relatives au MLI (Machine Language Interface) du système d'exploitation ProDOS16 (version 2.0). Les appels au MLI se regroupent en cinq grandes familles de commandes : les commandes relatives au Directory, aux fichiers, aux Blocks Devices, à l'environnement du GS et au contrôle des interruptions.

APPELS MLI SOUS ProDOS8 & ProDOS16

Mise au point et généralités concernant ProDOS

Le MLI forme un ensemble de routines écrites en assembleur (langage machine). Elles regroupent les différents modules aidant au fonctionnement du système d'exploitation ProDOS16. La version Bêta (version 1.0) remplacée par la version 2.0 est celle destinée à la commercialisation. Elle se compose de routines, dont chacune comporte un point d'entrée standard pouvant être appelé par un programme externe.

ProDOS16 se définit comme un **Disk Operating System** ayant atteint sa pleine maturité. Cette affirmation se justifie par l'expérience accumulée par ce système pendant plusieurs années avec une version ProDOS8.

CLASSIFICATION DU SYSTEME ProDOS16

- Simple tâche, multi-gestionnaire, permettant de partager l'environnement de l'unité centrale (Apple *II*GS).

- Structure hiérarchisée de ses fichiers, et attributs permettant une sélection selon la taille du fichier (Seepling, Sapling et Tree).

- Blocks Devices pouvant traiter des données en entrée comme en sortie (I/O Devices).

COMMANDES DU SYSTEME (SYSTEM CALLS)

• Les appels sont transmis à travers un JSL, instruction machine de saut long, faisant référence au contenu d'une table des paramètres (Block Parameter).

• Gestion de l'erreur dans le cas d'une exécution non aboutie : un code d'erreur est retourné par l'intermédiaire de l'accumulateur (A), tandis que la position des registres d'état (P) détermine le type de l'erreur.

• Le contenu des autres registres du microprocesseur reste préservé.

• Les appels doivent avoir une syntaxe propre au mode natif du 65C816.

• Toute commande transmise au MLI pourra être adressée de n'importe quel banc mémoire. De même l'adresse de destination pourra pointer n'importe quel banc mémoire.

• La table des paramètres relative à la commande transmise, pourra se situer à n'importe quel emplacement mémoire ou banc mémoire.

• Chaque pointeur faisant partie d'une table des paramètres, pourra être adressé de telle façon qu'il pointe n'importe quelle zone mémoire.

• Une donnée pourra être transférée d'un banc mémoire vers un autre, et dans n'importe quelle zone mémoire.

GESTION DES FICHIERS SYSTEME (FILE MANAGEMENT SYSTEM)

• Structure hiérarchisée des fichiers.

• Déclaration et facilité dans la gestion des préfixes à l'intérieur d'un chemin (Pathname). La version 1.0 permet de traiter jusqu'à huit préfixes, tandis que la version 2.0 est limitée à quatre préfixes.

• Pour le traitement des fichiers, deux types d'accès sont rendus possibles : fichier table des matières (Directory File) et fichier de données (Data File).

• Gestion des données d'un Block Device, que la sauvegarde des données soit continue ou non à la surface du support.

• Allocation automatique d'un tampon de données au moment d'ouvrir un fichier par OPEN.

• Allocation dynamique des tampons alloués dans une zone mémoire donnée.

• Gestion des données par blocs de 512 bytes.

• Attribut d'accès au fichier permettant de sélectionner uniquement sa lecture (Read), son écriture (Write), son changement de nom (Rename) et sa destruction (Destroy) ou plusieurs possibilités à la fois.

• Attribut Level, mis en place par la commande OPEN, qui désigne le niveau du fichier courant ouvert.

• Mise en place automatique dans l'entrée correspondante d'une table des matières, de la date et de l'heure au moment de traiter un fichier en écriture.

• Gestion des fichiers du type Sparse File.

• Gestion des volumes en ligne par affectation d'un numéro.

• Gestion des fichiers par affectation d'un numéro.

• La taille d'un support de sauvegarde de masse pourra atteindre jusqu'à 32 Mo.

• La taille d'un fichier de données pourra atteindre jusqu'à 16 Mo.

• Depuis la version 2.0, le chemin d'accès à un fichier (Pathname) pourra avoir une taille de 128 caractères valides (64 caractères valides avec la version 1.0).

• Depuis la version 2.0, un préfixe pourra avoir une taille de 128 caractères valides (64 caractères valides avec la version 1.0).

• Un maximum de 15 caractères valides peuvent être attribués à un nom de volume.

• Un maximum de 15 caractères valides peuvent être attribués à un nom de fichier.

GESTION DES PERIPHERIQUES EN LIGNE

• Support des Blocks Devices définis suivant un protocole préalable : ProDOS8 Blocks Devices, protocole relatif à la ROM des interfaces et protocole des interfaces d'extension.

• A chaque Block Device se trouve affecté un nom.

• Tout Device est muni d'un numéro d'affectation dans la liste des devices recencés en ligne.

GESTION MEMOIRE

• Un maximum de 256 Ko est requis pour des applications sous ProDOS16.

• La taille mémoire limite est fixée à 16 Mo.

• Les tampons mémoire alloués sont gérés d'une façon dynamique et relogeables.

GESTION DES INTERRUPTIONS

• Jusqu'à 16 routines de service (d'interruptions) peuvent être mises en place, ce qui permet une grande souplesse dans la gestion des interruptions.

• Dispatche les différents vecteurs d'interruption vers la routine de service correspondante grâce à un handler.

• Transmet à un sous-ensemble matériel (hardware) une interruption non reconnue par le programme résident (firmware).

• La commande QUIT transmet l'exécution à une nouvelle application au moment de quitter un programme en cours.

Point d'entrée du MLI sous ProDOS8

Le point d'entrée du MLI pour la version 1.0.2 se situe à l'adresse $D000. Sous ProDOS8, Apple Computer s'est donné les moyens d'un changement éventuel du point d'entrée MLI, en utilisant la Global Page ($BF00) comme vecteur intermédiaire.

La version 1.1.1 comporte certaines modifications par rapport aux versions précédentes : elle appelle le MLI par un JMP à partir de la page globale, et a maintenant son point d'entrée à l'adresse $DEOO. Toutes les routines ont leur point de sortie via $DE78 avec MLIEXIT, adresse $BFA0. Font exception à cette règle : REBOOT, l'appel des routines d'interruption IRQ et SYSTEM DEATH pendant l'exécution des interruptions.

Méthode standard d'appel du MLI (ProDOS8 & ProDOS16)

TRANSMISSION D'UNE COMMANDE SOUS ProDOS8

```
ProDOS8   JSR MLI          Adresse $BF00 de la page globale.
          HEX 00           Call_Number, Un byte, numéro de
                           la commande.
          DA 0000          Parm_Block. LByte, HByte. Poin-
                           teur de la table des paramètres de
                           la commande.
Error     BCS ERROR        Récupération du code d'erreur. l'ac-
                           cumulateur contiendra le code de
                           l'erreur rencontrée et la Carry sera
                           positionnée (C = 1).
```

• Sous ProDOS8, la transmission d'une commande MLI se fait par un JSR MLI-ENTRY à travers la page globale, adresse $BF00. Ce type d'appel est préféré à un JMP, saut inconditionnel, car le système sauvegarde l'adresse de retour sur le sommet de la pile et mémorise le numéro de la commande avec ses paramètres. Le numéro de la commande et ses paramètres sont gérés par des tables dont ProDOS8 tient le contrôle : table des commandes en $EF25 et table des paramètres en $EF45.

• A la suite de l'appel d'une commande MLI se trouvera le numéro de la commande MLI représenté par la variable Command_Number.

• Suit ensuite le pointeur de l'adresse où se trouve la table des paramètres; format : Low Byte, High Byte (LByte, HByte).

• Au retour du MLI, le contenu des registres X et Y reste inchangé, leurs valeurs intermédiaires sont sauvegardées dans la page globale de ProDOS.

– si la commande aboutit normalement, l'accumulateur contiendra la valeur $00 et la retenue sera annulée (Carry = 0). A ce niveau du déroulement, l'appel MLI pourra être abandonné par l'instruction BCS.

– si une erreur survient pendant le déroulement d'une commande MLI, l'accumulateur sera changé avec le code de l'erreur et la retenue positionnée (Carry = 1). L'instruction BCS permet une gestion de l'erreur en la dispatchant vers le dispatcher du handler, sous-programme de traitement de l'erreur.

• La liste des paramètres pourra se trouver à une adresse fixée par le programmeur ou l'application en cours : elle devra se loger au bas des 64 Ko de la mémoire, ou à la suite de la commande. Chaque appel MLI devra comporter une liste de paramètres, d'une structure semblable à celle de notre exemple.

TRANSMISSION D'UNE COMMANDE SOUS ProDOS16

```
PRO16MLI  EQU $E100A8      Fixe le point d'entrée de Pro-
                           DOS16.
          ::: :::::::
          ::: :::::::
          JSL PRO16MLI     Appel du MLI, ProDOS16 Entry.
          DC  $0000        Call_Number. Numéro de la com-
                           mande codé sur deux bytes (LByte,
                           HByte).
          DC  $00000000    Parm_Block. Pointeur de l'adresse
                           où se trouve la table des paramètres
                           de la commande. Mot long, codé
                           sur quatre bytes suivant le format :
                           mot de poids faible, mot de poids
                           fort (Low Word, High Word).
          BCS ERROR        Error Handler. Si la Carry est posi-
                           tionnée, saut à la routine de gestion
                           de l'erreur, le code d'erreur sera
                           chargé dans l'accumulateur. Dans le
                           cas contraire, l'exécution se
                           poursuit dans la routine actuelle.
```

• ProDOS16 nécessite l'appel du MLI à travers un JSL, appel long d'un sous-programme dans un autre banc, et des directives de l'assembleur (DC).

• Le JSL est rendu nécessaire du fait de la capacité mémoire du GS étendue au délà des 64 Ko de la gamme initiale des Apple *II*.

• L'instruction JSL nécessite trois bytes pour l'écriture de l'adresse qui lui suit, tandis que l'instruction JSR ne nécessite que deux bytes pour pointer une adresse.

• L'instruction BCS Error est un exemple possible de récupération et de gestion d'une erreur éventuelle.

• La variable ERROR pointe dans ce cas le gestionnaire de l'erreur (Error Handler).

COMPARAISON ENTRE DIFFERENTS PARAMETRES

ProDOS8 et ProDOS16, malgré de nombreuses similitudes, présentent des différences fondamentales dans l'écriture et la structure de leurs paramètres.

- **Transmission d'une commande vers le MLI**

– sous ProDOS8, toute commande transmise au MLI s'effectue par un JSR via l'entrée MLI Entry du système située dans la global page ($BF00).

– sous ProDOS16, tout appel au système s'effectue par l'intermédiaire d'un JSL (saut long) à l'adresse du point d'entrée au MLI, située dans le banc $E1 et non dans le banc $00.

- **Emplacement de la table des paramètres en mémoire**

– sous ProDOS8, la table des paramètres se situe par défaut dans le banc $00. L'écriture du pointeur qui la désigne est réalisée sur deux bytes.

– sous ProDOS16, la table des paramètres peut se situer dans n'importe quel banc mémoire. Parm_Block, pointeur de la table des paramètres, se compose alors de quatre bytes (mot long);

- **Taille des différents pointeurs**

– sous ProDOS8, tous les pointeurs de la table des paramètres ont une taille de deux bytes; exception à la règle : les pointeurs EOF (End Of file) et Position;

– sous ProDOS16, tous les pointeurs qui désignent une zone mémoire quelconque sont codés sur quatre bytes. Cette structure permet de pointer n'importe quel banc mémoire.

- **Taille des différents paramètres**

– sous ProDOS8, de nombreux paramètres sont codés sur un byte, en particulier Access_Byte, File_Type, Unit_Number, Reference_Number, Enable_Mask, etc.

– sous ProDOS16, tous les paramètres anciennement codés sur un byte sont étendus à une écriture sur deux bytes. Cette structure est requise par le fait que le microprocesseur 65C816 travaille sur des mots de seize bits de large.

- **Taille des pointeurs internes à un fichier**

– sous ProDOS8, la taille de ces champs est de 3 bytes au maximum (EOF, Mark – Position).

– sous ProDOS16, tous les champs relatifs à une position (EOF) et à une spécification d'un bloc (numéro du bloc ou compteur de blocs), sont codés sur quatre bytes. Cette structure a été établie en fonction d'une évolution possible en ce qui concerne les programmes système futurs. Apple Computer se laisse une porte entrouverte en ce qui concerne la gestion des fichiers et des volumes, les tailles respectives étant fixées à 16 Mo pour les fichiers et à 32 Mo pour les volumes.

Remarque

Sous ProDOS16 booté sur le GS, il n'existe plus de champ codé sur trois bytes dans la table des paramètres. Toute adresse qui pointe une zone mémoire, ainsi que les compteurs de blocs, voient leur taille étendue à quatre bytes. Dans ce cas, le byte de poids fort (HByte) du mot de poids fort (High Word) est mis à zéro en prévision à des applications futures.

Table des paramètres sous ProDOS16

GENERALITES

La table des paramètres (Parameter Block) est une liste formatée, contenant des paramètres dont l'écriture est formulée dans une suite consécutive de bytes. Chaque table des paramètres, implantée en mémoire, est directement liée à la commande à laquelle elle fait référence. Une table de paramètres se compose de valeurs codées, chacune d'elle pouvant être représentée soit par un mot (Word = 2 bytes), soit par un mot long (DWord = 4 bytes). Ces différentes informations personnalisent la commande désignée, et se regroupent dans des champs. Deux types de champs composent une table de paramètres : ceux qui contiennent les paramètres à transmettre à la commande et ceux qui sont chargés par des valeurs au retour de l'exécution.

Chaque champ dans une table des paramètres contient un seul et unique paramètre. Il existe trois formes ou types de paramètres, chaque type ayant une signification particulière : une valeur, un résultat ou un pointeur. Chacun de ces formats est régi par des règles très strictes, trouvant une application soit pour transmettre une valeur en entrée (Input Parameter), soit pour recevoir un résultat au retour du MLI (Output Parameter), soit pour pointer une zone mémoire. Dans le dernier cas, le pointeur représente un point d'entrée d'une zone mémoire pour toutes les commandes ProDOS16. Cette zone mémoire pourra à nouveau contenir des données (Data), une adresse utilisée comme pointeur éventuel, ou une autre zone tampon où ProDOS16 pourra traiter des données diverses.

INPUT & OUTPUT PARAMETER

- **Input Parameter (paramètre valeur)**

Si le paramètre est transmis au MLI par la commande courante et à travers la table des paramètres (Parameter Block), sa valeur représentera une quantité numérique, écrite sur un ou plusieurs mots (2 bytes ou multiple de deux).

- **Output Parameter (paramètre résultat)**

Si, au retour de l'exécution de la commande, le paramètre est issu du MLI, le résultat représentera une quantité numérique écrite sur un ou plusieurs mots (2 bytes ou multiple de deux).

- **Input or Output Parameter (paramètre pointeur)**

Si le paramètre représente un pointeur désignant une zone mémoire, celle-ci pourra alors contenir des données, des codes particuliers, une adresse, ou

une zone mémoire utilisée par ProDOS16 comme tampon (Buffer). Dans ce cas, l'écriture de ce paramètre nécessitera un mot long (4 bytes). Le paramètre pointeur représente toujours une adresse d'entrée exploitée par la suite par chaque commande ProDOS16. Une donnée pointée par le paramètre pointeur pourra être utilisée comme paramètre d'entrée ou de sortie (Input or Output Parameter).

Chaque commande ProDOS16 transmise au MLI nécessite une table des paramètres valide, formatée selon le descriptif précédent et pointée par une adresse codée sur quatre bytes (mot long). C'est au programme appelant de transmettre la commande MLI avec la table des paramètres formatée.

FORMULATION DES PARAMETRES SOUS ProDOS16

Les paramètres sont de trois types : valeur (Input Parameter), résultat (Output Parameter) et pointeur. L'ensemble des paramètres est stocké dans une table formatée qui n'est autre qu'une liste de valeurs propre à chaque commande ProDOS16. Chaque liste est formulée suivant une syntaxe et un nombre de champs déterminés au préalable.

STRUCTURE D'UNE TABLE DE PARAMETRES (ProDOS16)

Une table de paramètres, ou Parameter Block, se compose d'un ou de plusieurs champs. Chaque champ peut représenter un numéro de bloc, un compteur de blocs, une valeur partielle d'un fichier, ou tout autre paramètre représentatif d'un fichier ou d'un volume. Dans ce dernier cas, l'écriture du champ requiert quatre bytes (DWord = mot long). Cette structure est rendue nécessaire pour adapter par la suite les devices à venir, pouvant traiter des volumes de tailles plus grandes. La taille minimale d'un champ est adaptée aux possibilités actuelles et futures du microprocesseur 65C816 qui traite des mots de seize bits de large. Certains pointeurs utilisent un champ de quatre bytes même si trois bytes suffisent le plus souvent pour pointer la zone mémoire. Dans ce cas, le byte de poids fort est mis à zéro.

L'écriture des pointeurs s'effectue suivant le format : Low Word, High Word. Chaque pointeur offset, Low Word et High Word, s'écrit suivant la syntaxe LByte, HByte.

Par comparaison au système ProDOS8, l'écriture de la table des paramètres est sensiblement différente sous ProDOS16. En effet, le système facilite l'écriture de la table des paramètres en réduisant dans certains cas sa syntaxe à un champ unique.

Dans certains cas, le champ d'une table de paramètres autorise une valeur numérique plus élevée que celle utilisée généralement pour identifier le paramètre. Cette situation est imposée d'une part, pour maintenir une certaine harmonie entre la taille des champs pour différentes commandes de la même famille, et, d'autre part, pour actualiser les commandes pour le futur. Dans ce dernier cas, les champs sont surdimensionnés en prévisions du matériel et du logiciel à venir. Comme exemple on pourrait citer la commande CREATE ($01) avec le champ Aux_Type codé sur quatre bytes : uniquement deux bytes sont utilisés, les deux autres étant mis à zéro (le mot de poids fort est nul = 0000).

IMPLANTATION DE LA TABLE DES PARAMETRES EN MEMOIRE

Chaque commande, à travers le MLI de ProDOS16, nécessite la mise en place d'un pointeur pour désigner la table des paramètres. Le pointeur sera représenté par un mot long, c'est-à-dire quatre bytes. Cette syntaxe permet par la suite de placer la table des paramètres dans n'importe quel banc mémoire. Sous environnement GS et ProDOS16 booté en mémoire, il existe deux méthodes pour implanter une table des paramètres :

- En intégrant la table des paramètres au programme appelant la commande MLI : l'adresse d'entrée de la table sera référencée par un label. Après assemblage de l'ensemble du programme, l'appel de la commande sera implémenté à l'application. Par la suite, la table des paramètres sera chargée en mémoire en même temps que le programme utilisateur. Dans l'exemple précédent, traitant la transmission d'une commande sous ProDOS16, la table des paramètres est pointée par la variable Parm_Block.

- En utilisant les propriétés des outils Memory Manager et System Loader.

– nécessite au préalable la déclaration du banc mémoire à travers l'outil Memory Manager. Le bloc alloué sera du type purgeable ou verrouillé.

– recherche du pointeur du bloc à travers le gestionnaire mémoire (Memory handle) de l'outil Memory Manager.

– établissement de la table des paramètres, puis la placer à l'adresse trouvée précédemment.

VALEUR DES DIFFERENTS REGISTRES DU PROCESSEUR

La transmission d'une commande au MLI ne nécessite pas de la part du programme utilisateur, ou du développeur, la sauvegarde ou la manipulation d'un quelconque registre du microprocesseur. ProDOS16 sauvegarde et restaure tous ses registres, à l'exception de l'accumulateur (A) et du registre d'état (P). Ces deux registres subissent des modifications au fur et à mesure de l'exécution de la commande. Au retour d'une commande transmise au MLI, les registres du microprocesseur 65C816 auront les valeurs suivantes :

Accumulateur Low ()	= 0 si la commande aboutit normalement ou contient le code d'erreur dans le cas contraire
Registre d'index X	= inchangé
Registre d'index Y	= inchangé
Pointeur de pile S	= inchangé
Registre Direct D	= inchangé
Registre d'état P	= valeur sauvegardée dans le MLI
Data Bank Register DB	= inchangé
Program Bank Reg. PB	= inchangé
Compteur ordinnal PC	= PC pointe maintenant l'adresse directement à la suite de la table des paramètres de la commande.

Valeurs possibles des bits du registre d'état P

Bit du signe (n)	= indéterminé
Bit de débordement (v)	= indéterminé
Bit d'accès mémoire/accumulateur (m)	= inchangé

Bit de sélection du registre d'index (x)	= inchangé
Bit du mode décimal (d)	= 0
Bit d'inhibition des interruptions (i)	= inchangé
Bit du résultat nul (z)	= indéterminé
Bit de la retenue (c)	= 0 avec une commande aboutie et 1 dans le cas contraire
Bit du mode (émutation ou native-e)	= inchangé

Remarques

• Le terme 'inchangé' signifie que ProDOS16 a initialement sauvegardé ces valeurs et les a à nouveau restaurées au retour du MLI. Les valeurs sauvegardées sont celles que possèdent les différents registres cités, juste avant d'effectuer le JSL à l'entrée MLI de ProDOS16.

• ProDOS16 affecte différemment certains bits du registre d'état. Au retour d'une commande MLI, les bits du signe (n) et du résultat nul (z) sont indéterminés sous ProDOS16, tandis que la retenue (c) est positionnée suivant l'issue de la commande (c = 1 si erreur et c = 0 si pas d'erreur). Sous ProDOS8, les bits n et z sont déterminés suivant le contenu de l'accumulateur (A), tandis que les bits c et z déterminent tous deux une erreur détectée.

COMMANDES MLI SOUS ProDOS16

Les commandes MLI se divisent en cinq groupes ou familles : les commandes relatives au Volume, les commandes réservées à la gestion des fichiers, les commandes relatives aux Devices, les commandes qui traitent l'environnement du système et les commandes de contrôle des interruptions.

Commandes relatives au Volume

Elles constituent les routines de création, d'opération de mise en forme, d'effacement ou de moyens pour renommer un fichier dans un Volume. Innovation par rapport à ProDOS8, la commande VOLUME ($08) retourne le nom du Volume affecté au Device, le nombre total de blocs qu'il peut gérer, le nombre de blocs libres et le numéro d'identification du système.

Ces commandes permettent de modifier la structure et le statut de tout ce qui se trouve dans un Volume. Par exemple, le traitement d'un fichier nécessite un tampon (Buffer) utilisé comme mémoire auxiliaire avant la sauvegarde des données sur une disquette.

Syntaxe des commandes

CREATE ($01)	Crée un nouveau fichier dans le Volume courant.
DESTROY ($02)	Efface un fichier du Volume courant.
RENAME ($03)	Permet de renommer un fichier courant.

SET_FILE_INFO ($05)	Affecte des attributs à un fichier.
GET_FILE_INFO ($06)	Retourne la valeur des attributs.
GET_ENTRY	Permet la recherche des tables des matières et crée une table des matières destinée à des fichiers utilitaires.
VOLUME ($08)	Retourne le nom du Volume qu'occupe un device, ainsi que certains attributs relatifs au device.
SET_PREFIX ($09)	Assigne un nouveau chemin au préfixe existant.
GET_PREFIX ($0A)	Retourne le chemin du dernier préfixe déclaré.
CLEAR_BACKUP_BIT ($0B)	Purge le bit de l'attribut Backup affecté au paramètre Access_Byte.
WRITE_PROTECT ($0C)	Détermine la protection en écriture envers un volume.

Commandes réservées à la gestion de fichiers

Elles sont utilisées pour le transfert et le traitement des données d'un fichier. La commande OPEN devra être opérante avant l'écriture dans un fichier : elle lui alloue un tampon mémoire de 1024 bytes. Ces commandes regroupent des routines écrites principalement pour faciliter le traitement des fichiers en général.

Syntaxe des commandes

OPEN ($10)	Prépare le fichier à l'accès aux données.
NEWLINE ($11)	Met en fonction ou désactive la fonction Newline pour pouvoir lire dans tout fichier ouvert.
READ($12)	Transfère des données à partir d'un support de sauvegarde dans le tampon mémoire réservé.
WRITE ($13)	Transfère des données à partir du tampon mémoire réservé dans le fichier qui se trouve sur un support de sauvegarde.
CLOSE ($14)	Termine l'accès à un fichier.
FLUSH ($15)	Sauvegarde les données sur un support disque par exemple et purge le contenu du tampon mémoire réservé à un fichier.
SET_MARK ($16)	Fixe la position courante dans le fichier (MARK).
GET_MARK ($17)	Retourne la position courante dans un fichier.
SET_EOF ($18)	Fixe la grandeur logique du fichier.
GET_EOF ($19)	Retourne la valeur de la grandeur logique d'un fichier.

SET_LEVEL ($1A) Affecte une valeur au niveau d'ouverture du fichier courant.

GET_LEVEL ($1B) Retourne la valeur du paramètre Level attribué en dernier.

Commandes relatives aux devices

Ces commandes, nouvelles sous ProDOS16, sont destinées à faciliter le traitement et les manipulations des Devices. Les commandes 'Devices Calls' permettent dans certains cas un accès direct aux mécanismes de sauvegarde des données. Sous ProDOS8, cet accès aux données n'est possible qu'au travers les commandes relatives aux Volumes et aux fichiers.

Syntaxe des commandes

GET_DEV_NUM ($20) Retourne le paramètre Dev_Num attribué par ProDOS 16 lors de la phase du boot. Ce paramètre n'est utilisé qu'avec les commandes relatives aux Devices.

GET_DIB ($21) Retourne l'adresse de la table Device Information Block (DIB). Chaque Device, qu'il soit du type Character Device ou Block Device, possède sa propre table DIB.

READ_BLOCK ($22) Lecture de 512 bytes à partir du Device courant et transfert dans le tampon pointé par Data_Buffer.

WRITE_BLOCK ($23) Ecriture sur un device (Dev_Num) d'un bloc de 512 bytes (Block_Num) pointé par Data_Buffer.

Commandes relatives à l'environnement

Ce type de commandes fait appel à l'environnement logiciel et matériel. Elles sont liées directement à la configuration en présence. C'est une disposition particulière au système ProDOS16, permettant de réaliser des branchements presonnalisés au moment de quitter une application courante.

Syntaxe des commandes

START_PRODOS ($25) Retourne l'identificateur à placer en début de chaque table des paramètres d'une commande MLI (Caller_Id).

END_PRODOS ($26) Signifie à ProDOS que la commande MLI courante prend fin et que Caller_Id, identificateur attribué à la commande, se trouve à nouveau disponible.

GET_PATHNAME ($27) Retourne le chemin complet de l'application courante.

GET_BOOT_VOL ($28) Retourne le nom du Volume ayant booté le système ProDOD16.

QUIT ($29) Prend en compte la procédure pour quitter l'application courante.

GET_VERSION ($2A) Retourne le numéro de la version du système ProDOS16 courant.

SAVE_STATE ($2B) Sauvegarde momentanément le paramètre Save_Num qui affecte ainsi un numéro à l'état courant du matériel et du système d'exploitation au moment où la commande est transmise.

RESTORE_STATE ($2C) Retourne l'état du matériel et du système d'exploitation référencé par le paramètre Save_Num.

Commandes de contrôle des interruptions

Ces commandes regroupent le mode opératoire pour la mise en place et l'annulation des routines de service. Ces routines de service permettent de gérer jusqu'à 16 vecteurs d'interruption.

Syntaxe des commandes

ALLOC_INTERRUPT ($30) Installe une routine de service en spécifiant le pointeur de son implantation (Int_Code).

DEALLOC_INTERRUPT ($31) Purge une routine de service spécifié par le paramètre Int_Num.

SET_INT_MODE ($32) Drapeau d'identification permettant la gestion correcte de l'interruption en cours.

PARAMETRES COMMUNS AUX COMMANDES MLI

Généralités

Chaque commande est accessible au MLI au travers d'une liste de paramètres stockée dans la table des paramètres (Parameter Block). Chaque table des paramètres se compose d'un nombre variable de champs, directement liés à la commande. Les noms affectés aux différents champs, ainsi que certains termes techniques, sont donnés en version d'origine afin de garder une compatibilité dans le verbe. L'interprétation de chaque paramètre d'une commande MLI est largement commentée dans ce paragraphe.

Liste en détail des paramètres valides

- **CALLER_ID – 2 BYTES (LOW BYTE, HIGH BYTE)**

Ce paramètre, attribué par la commande START_PRODOS, permet de situer par la suite la commande MLI.

La commande START_PRODOS devra précéder toute commande MLI, et retournera par la suite un identificateur à mettre en place en début de chaque table des paramètres.

La commande END_PRODOS permet de libérer le paramètre Caller_Id qui pourra alors être utilisé pour un autre appel au MLI.

- **PATHNAME – 4 BYTES (LOW WORD, HIGH WORD)**

Pointe dans la zone tampon réservée la chaîne de caractères qui désigne le chemin d'accès complet au fichier. Un chemin en général, peut être un chemin partiel, un préfixe, ou un chemin complet (Pathname). Un Pathname peut comporter jusqu'à 128 caractères (ProDOS16, version 2.0).

Le champ Pathname trouve son utilisation à travers les commandes MLI: CREATE, DESTROY, RENAME, SET_FILE_INFO, GET_FILE_INFO et OPEN. Si le nom déclaré débute par un slash (/), le chemin complet sera traité, sinon ce sera un chemin partiel, dont le nom correspond à celui d'un fichier par exemple.

Chemin – Pathname

Un chemin complet commence par le nom du Volume, tandis qu'un chemin partiel sera complété par le préfixe. Cette succession de noms a pour seul but de fixer le chemin d'accès à un fichier dont le nom se trouve stocké dans la table des matières. Le fichier considéré pourra être un fichier de données (data File) ou de table des matières (SubDirectory File). Un chemin complet est toujours prédédé par le slash (/) et désigne le Pathname. Sa taille limite est fixée à 128 caractères valides avec la version 2.0.

Exemple

/USERS.DISK/UTILITAIRES/NOMFICH

Chemin partiel

Un chemin partiel représente une autre méthode de recherche. Il représente en général une partie du chemin complet et nécessite le préfixe pour le compléter. Le chemin partiel ne débute pas par le nom du Volume précédé du slash, mais se contente d'une procédure de recherche limitée: le chemin d'accès au fichier se situe à l'intérieur du SubDirectory dont il fait partie. La longueur d'un tel chemin est limitée à 128 caractères valides (ProDOS16 version 2.0).

ProDOS16 alloue quatre bytes au pointeur Pathname: Low Word, High Word, où chaque mot (Word) répond à la syntaxe LByte, HByte.

Représentation des 4 bytes

Bits	‹---------- Low Word ----------›		‹---------- High Word ----------›	
	FEDCBA98	76543210	FEDCBA98	76543210
	LLLLLLLL	HHHHHHHH	LLLLLLLL	HHHHHHHH

- **NEW_PATHNAME – 4 BYTES (LOW WORD, HIGH WORD)**

Pointe dans la zone tampon la chaîne de caractères qui désigne le nouveau chemin d'accès au fichier courant. Ce paramètre est particulièrement utilisé avec la commande CHANGE_PATH pour situer un fichier déplacé à l'intérieur d'un Volume donné.

Le pointeur est adressé par un double mot: Low Word, High Word, où chaque mot répond à la syntaxe LByte, HByte.

Représentation des 4 bytes

Bits	‹---------- Low Word ----------›		‹---------- High Word ----------›	
	FEDCBA98	76543210	FEDCBA98	76543210
	LLLLLLLL	HHHHHHHH	LLLLLLLL	HHHHHHHH

- **CHAIN_PATH – 4 BYTES (LOW WORD, HIGH WORD)**

A partir de ProDOS16 version 2.0, le champ Chain_Path a totalement disparu de la table des paramètres. Avec la version 1.0, ce paramètre est utilisé uniquement par la commande QUIT, et occupe le premier champ dans la table des paramètres. Il pointe dans la zone tampon la chaîne de caractères qui désigne le chemin d'accès de la prochaine application à exécuter.

Le pointeur est adressé par un double mot: Low Word, High Word, où chaque mot répond à la syntaxe LByte, HByte.

Représentation des 4 bytes

Bits	‹---------- Low Word ----------›		‹---------- High Word ----------›	
	FEDCBA98	76543210	FEDCBA98	76543210
	LLLLLLLL	HHHHHHHH	LLLLLLLL	HHHHHHHH

- **PREFIX – 4 BYTES (LOW WORD, HIGH WORD)**

Utilisé avec les commandes SET_PREFIX et GET_PREFIX, le paramètre Prefix occupe le dernier champ de la table des paramètres. Il pointe dans la zone tampon la chaîne de caractères qui désigne le nom Directory valide.

Préfixe – Prefix

Le préfixe est un chemin par défaut qu'il faut ajouter à celui que l'on veut spécifier. La commande MLI SET_PREFIX permet de déclarer l'un des quatre types de préfixes valides. La longueur maximale d'un préfixe est limitée à 128 caractères valides, sa longueur minimale pouvant être réduite à zéro, ce qui annule sa déclaration préalable (ProDOS16 version 2.0). Le slash au début d'un préfixe est obligatoire, tandis que celui de la fin est facultatif.

ProDOS16 gère ses différents chemins, Pathname et Prefix, par l'intermédiaire d'un artifice très particulier. Il lui suffit en effet d'aller lire le premier byte dans le chemin courant (Pathname ou Prefix) pour être renseigné sur sa validité. C'est encore ce même byte qui est mis à zéro lorsqu'un préfixe est declaré nul.

Représentation du tampon réservé à un préfixe

Byte $00	Compteur de la longueur du préfixe
Bytes $01-$80	Codes ASCII du nom du préfixe

Exemple

Préfixe courant : /V2/NOM/

où /V2 désigne le nom du Volume Directory et /NOM/ le nom du Volume SubDirectory. La représentation des codes ASCII sera la suivante :

```
   /  V  2  /  N  0  M  /    = caractères
09 2F 56 32 2F 4E 4F 4D 2F   = codes ASCII, bit 7 = 0

09                           = 9 bytes de longueur
2F 56 32 2F 4E 4F 4D 2F      = /V2/NOM/
```

Prefix désigne le pointeur d'une zone tampon et se trouve adressé par un double mot : Low Word, High Word, où chaque mot répond à la syntaxe LByte, HByte.

Représentation des 4 bytes

	‹---------- Low Word ----------›		‹---------- High Word ----------›	
Bits	F E D C B A 9 8	7 6 5 4 3 2 1 0	F E D C B A 9 8	7 6 5 4 3 2 1 0
	L L L L L L L L	H H H H H H H H	L L L L L L L L	H H H H H H H H

- **PREFIX_NUM – 2 BYTES (LBYTE, HBYTE)**

Le champ Prefix_Num permet d'attribuer une valeur numérique de 0 à 3 inclus au préfixe courant pointé par le champ Prefix. Le préfixe sera soit un Pathname complet, soit un Pathname partiel, dans ce cas il prendra naissance à une origine du chemin complet fixée par le champ Préfix. Un Pathname partiel, sans numéro de préfixe (Prefix_Num), possède par défaut le numéro zéro (O/I). L'attribut 0 (ou nul) est initialisé au moment d'exécuter une nouvelle application. Le préfixe par défaut comporte le nom du Volume booté sur le Block Device courant, ce qui correspond en réalité au chemin d'accès de la dernière application lancée par ProDOS16.

Le champ Prefix_Num désigne l'un des quatre numéros possibles ($0000-$0003) pouvant être attribué à un préfixe et sous système ProDOS16, version 2.0. La version 1.0, quant à elle, autorise jusqu'à huit numéros attribuables à un préfixe.

Comme le champ Prefix, Prefix_Num est utilisé uniquement avec les commandes SET_PREFIX et GET_PREFIX. ProDOS16 gère jusqu'à quatre préfixes auxquels il permet d'attribuer un numéro valide de 0 à 3 inclus, terminé par un slash. Les trois premières valeurs autorisent des attributs par défaut : 0/, 1/ et 2/.

Signification des attributs

0/ Le préfixe courant contient le nom du Volume ayant booté le système ProDOS16 (Boot Volume).

1/ Le préfixe courant contient le Pathname complet du chemin d'accès à la table des matières (Directory ou SubDirectory) dans lequel se trouve l'application.

2/ Le préfixe courant contient le Pathname complet du chemin d'accès au fichier Library de l'application à exécuter (/BOOTVOL/SYSTEM/LIB).

3/ Préfixe disponible à une application particulière.

- **DIB – 4 BYTES (LOW WORD, HIGH WORD)**

Pointe dans la zone mémoire la table relative aux informations des devices recensés par ProDOS16. La table Device Information Blocks (DIB) contient un certain nombre de renseignements des Devices en ligne et reconnus par le système.

- **DATA_BUFFER – 4 BYTES (LOW WORD, HIGH WORD)**

Pointe dans la zone mémoire l'adresse de début où seront implantées les données lues à partir d'un disque, ou encore l'emplacement mémoire à partir duquel seront sauvegardées des données sur un disque. A ne pas confondre avec I/O_Buffer qui pointe le tampon d'entrées/sorties, zone transitoire des données traitées par un device courant. La zone pointée par I/O_Buffer a une taille limitée à 1024 bytes, et se trouve allouée par la commande OPEN. Cette zone de transition permet de traiter momentanément des données lues ou écrites du/ ou vers le disque.

Tampon des données réservé aux programmes (Data_Buffer)

Cet espace mémoire occupe en général la zone mémoire attribuée par le Memory Manager au moment de traiter l'application courante. Un tampon mémoire ne trouve pas forcément son origine à la limite d'une page mémoire (Page Boundary) mais peut tout aussi bien chevaucher deux pages, voire même deux bancs mémoire. A noter qu'un fichier, sous ProDOS16, peut atteindre une taille de 16 Mo, ce qui équivaut à un total de 16777216 bytes ($000000 à $FFFFFF).

- **I/O_BUFFER – 4 BYTES (LOW WORD, HIGH WORD)**

Memory Handle. Pointe dans la zone mémoire l'adresse du tampon d'entrées/sorties des données alloué par ProDOS16 lors de l'ouverture d'un fichier par la commande OPEN. Quatre bytes sont nécessaires pour l'écriture du pointeur, qui désigne le tampon d'entrée/sortie : Low Word, High Word.

Structure d'un tampon I/O_Buffer

Un tampon mémoire, alloué par ProDOS16 au moment de l'ouverture d'un fichier, possède une taille de 1024 octets (2 blocs logiques). Les 512 premiers bytes sont utilisés comme bloc de données. Les 512 bytes suivants comme index des blocs gérés. Par ailleurs, la commande CATALOG utilise la première partie du tampon comme zone mémoire intermédiaire avant l'affichage sur l'écran des données de la table des matières actuelle.

Représentation des 4 bytes

	‹---------- Low Word ----------›		‹---------- High Word ----------›	
Bits	F E D C B A 9 8	7 6 5 4 3 2 1 0	F E D C B A 9 8	7 6 5 4 3 2 1 0
	L L L L L L L L	H H H H H H H H	L L L L L L L L	H H H H H H H H

Tampon d'entrées/sorties – Input/Output

I/O_Buffer pointe le tampon d'entrées/sorties alloué par ProDOS16 au moment de l'ouverture d'un fichier. En effet, le système attribue une zone mémoire de 1024 bytes à chaque fichier ouvert par la commande OPEN, ce qui permet d'utiliser cet espace mémoire pour les transferts fichier/disque. Le tampon I/O nécessite de débuter à la limite d'une page mémoire et se réserve à chaque fois 1024 bytes ($0400 bytes). Lorsque le système Basic est actif (ProDOS8 + Basic.System), le premier tampon d'entrées/sorties par défaut se trouve dans la zone des adresses $9600-$99FF. Un tampon supplémentaire sera mis en place avec chaque commande OPEN : c'est ainsi que trois nouveaux tampons sont alloués avec trois fichiers ouverts.
La routine MLI READ BLOCK n'utilise pas ces tampons d'entrées/sorties.

Un byte est nécessaire par caractère ou adresse mémoire occupée par une donnée, tandis qu'un tampon I/O occupe 1024 octets réservés par la commande OPEN, même s'il n'est exploité que partiellement.

CREATE_DATE – 2 BYTES (LBYTE, HBYTE)

Attribue au fichier courant la date de sa création. Les positions relatives $18-$19 (24-25) dans chaque entrée de la table des matières d'un nom de fichier correspondent à sa date de création. Create_Date autorise la sauvegarde sur un support disque, la date du jour au moment de la création du fichier par la commande CREATE, par exemple. CATALOG affiche sous la rubrique Created, la date de création du fichier tandis que la commande CAT l'ignore.

Si la valeur du champ de Create_Date dans la table des paramètres est nulle, ProDOS16 recherche alors la date courante à travers le hard de l'horloge en temps réel intégré à l'Apple *IIGS* .

Deux bytes sont réservés au paramètre Create_Date dans chaque entrée d'une table des matières d'un nom de fichier.

Représentation des 2 bytes

	‹----- byte 1 -----›	‹----- byte 0 -----›	
Bits	F E D C B A 9 8	7 6 5 4 3 2 1 0	A = Année
	A A A A A A A M	M M M J J J J J	M = Mois
			J = Jour

Bits F-9 = 7 bits pour l'année
Bits 9-5 = 4 bits pour le mois
Bits 4-0 = 5 bits pour le jour

• CREATE_TIME – 2 BYTES (LBYTE, HBYTE)

Attribue au fichier courant l'heure de sa création. Les positions relatives $1A-$1B (26-27) dans chaque entrée de la table des matières d'un nom de fichier correspondent à l'heure de sa création. Create_Time, associé avec la commande CREATE par exemple, sauvegarde sur un disque l'heure de la création d'un fichier. CATALOG affiche sous la rubrique Created, l'heure de création du fichier tandis que la commande CAT l'ignore.

Si la valeur du champ de Create_Time dans la table des paramètres est nulle, ProDOS16 recherche alors l'heure courante à travers le hard de l'horloge en temps réel intégré à l'Apple *IIGS* .

Représentation des 2 bytes

	‹----- byte 1 -----›	‹----- byte 0 -----›	
Bits	F E D C B A 9 8	7 6 5 4 3 2 1 0	H = Heure
	0 0 0 H H H H H	0 0 M M M M M M	M = Minute
			0 = bits nuls

Les bits F-D sont inutilisés, donc nuls
Les bits C-8 sont réservés pour les heures
Les bits 7-6 sont inutilisés, donc nuls
Les bits 5-0 sont réservés pour les minutes

• MOD_DATE – 2 BYTES (LBYTE, HBYTE)

Permet de modifier la date pour différencier deux versions différentes d'un fichier par exemple. Les positions relatives $21-$22 (33-34) dans chaque entrée de la table des matières d'un nom de fichier correspondent à la date de la dernière modification apportée.

Ce paramètre, par l'intermédiaire d'une commande appropriée, sauvegarde sur un disque la date courante en même temps que l'enregistrement des données d'un fichier par exemple. Cette pratique permet par la suite de différencier deux versions différentes d'un programme. CAT et CATALOG affichent, sous la rubrique Modified la date de la dernière modification intervenue envers un fichier.

Si la valeur du champ de Mod_Date dans la table des paramètres est nulle, ProDOS16 recherche alors la date courante à travers le hard de l'horloge en temps réel intégré à l'Apple *IIGS* .

Deux bytes sont nécessaires pour coder le paramètre Mod_Date.

Représentation des 2 bytes

	‹----- byte 1 -----›	‹----- byte 0 -----›	
Bits	F E D C B A 9 8	7 6 5 4 3 2 1 0	A = Année
	A A A A A A A M	M M M J J J J J	M = Mois
			J = Jour

Bits F-9 = 7 bits pour l'année
Bits 9-5 = 4 bits pour le mois
Bits 4-0 = 5 bits pour le jour

• **MOD_TIME – 2 BYTES (LBYTE, HBYTE)**

Permet de modifier l'heure pour différencier deux versions d'un fichier par exemple. Les positions relatives $23-$24 (35,36) dans chaque entrée de la table des matières d'un nom de fichier correspondent à l'heure de la dernière modification apportée. Ce paramètre, transmis à travers une commande appropriée, sauvegarde sur un disque l'heure courante en même temps que l'enregistrement des données d'un fichier. Cette structure permet de différencier deux versions différentes d'un programme par exemple. CAT et CATALOG affichent, sous la rubrique Modified, l'heure de la dernière sauvegarde d'un fichier.

Si la valeur du champ de Mod_Time dans la table des paramètres est nulle, ProDOS16 recherche alors l'heure courante à travers le hard de l'horloge en temps réel intégré à l'Apple *IIGS* .

Deux bytes sont nécessaires pour coder le paramètre Mod_Time.

Représentation des 2 bytes

	‹----- byte 1 -----›	‹----- byte 0 -----›	
	F E D C B A 9 8	7 6 5 4 3 2 1 0	
Bits	0 0 0 H H H H H	0 0 M M M M M M	H = Heure M = Minute 0 = bits nuls

Les bits F-D sont inutilisés, donc nuls
Les bits C-8 sont réservés pour les heures
Les bits 7-6 sont inutilisés, donc nuls
Les bits 5-0 sont réservés pour les minutes

• **ACCESS – 2 BYTES (LBYTE, HBYTE)**

Détermine l'accès au fichier courant. La position relative $1E (30) dans chaque entrée de la table des matières d'un nom de fichier détermine le moyen d'accès du système à un tel fichier : lecture ou écriture autorisée, ou encore les deux à la fois, le fichier a été modifié ou non depuis sa dernière copie de sauvegarde, etc.

Ce paramètre, codé sur deux bytes, autorise par la représentation binaire du mot, le moyen d'accès par ProDOS16 au fichier.

Représentation du mot (Word)

	‹----- byte 1 -----›	‹----- byte 0 -----›
	F E D C B A 9 8	7 6 5 4 3 2 1 0
Bits	0 0 0 0 0 0 0 0	D N B - - - W R

Les bits F-8 sont inutilisés, donc nuls.
Les bits 4-2 sont réservés à une application future, et devront avoir une valeur nulle (bits 4 à 2 = 0).
Si le bit est à 1, la fonction correspondante sera reconnue comme valide. Si par contre, le bit est nul (0), la fonction correspondante sera ignorée.

Signification des fonctions d'attribution

D — Destroy enable bit. Ce bit détermine si le fichier pourra être effacé ou non.
Si le bit D = 1, la fonction est autorisée.

N — Re-Name enable bit. Ce bit détermine si le fichier pourra être renommé ou non.
Si le bit N = 1, la fonction est autorisée.

B — Backup needed bit. Ce bit prend deux significations :
• Sous ProDOS8, le bit 5 stocké dans la page globale (adresse $BF95) est exploité par des routines MLI pour le positionner en conséquence : si une écriture intervient dans le fichier, le bit 5 est mis à un, et à zéro dans le cas contraire. Cet état renseigne le système si une modification est intervenue ou non depuis sa dernière copie (Backup).
• Sous ProDOS16, la page globale étant totalement ignorée, c'est la commande CLEAR_BACKUP_BIT ($0B) qui permet d'annuler le bit 5.
– si le bit B = 0, aucune modification n'est intervenue au niveau du fichier depuis sa dernière copie de sauvegarde.
– si le bit B = 1, le fichier à été modifié par rapport à une version prédédente.

W — Write enable bit. Ce bit détermine si le fichier est accessible à l'écriture par le système.
Si le bit W = 1, la fonction est autorisée.

R — Read enable bit. Ce bit détermine si le fichier courant est accessible à la lecture par le système.
Si le bit R = 1, la fonction est autorisée.

Exemple

	‹----- byte 1 -----›	‹----- byte 0 -----›	
	F E D C B A 9 8	7 6 5 4 3 2 1 0	
Bits	0 0 0 0 0 0 0 0	D E B 0 0 0 W R	
	0 0 0 0 0 0 0 0	1 1 0 0 0 0 1 1	= $00C3
	0 0 0 0 0 0 0 0	1 1 1 0 0 0 1 1	= $00E3

Un champ Access de valeur $00C3 autorise l'accès au fichier en lecture et écriture. Les commandes RENAME et DELETE sont autorisées, aucune modification n'est intervenue au niveau du fichier, et des données pourront y être écrites. Un champ de valeur $00E3 autorise la recopie des données vers un autre support de sauvegarde, mais signale au système que le contenu du fichier a changé par rapport à une version précédente (bits 7, 6, 5, 1 et 0 positionnés à 1).

• **FILE_TYPE – 2 BYTES (LBYTE, HBYTE)**

Détermine le type de fichier (texte, binaire, système, etc). La position relative $10 (16) dans chaque entrée de la table des matières d'un nom de fichier correspond au type de fichier ProDOS16. Ce champ est utilisé par les commandes CREATE, SET_FILE_INFO et GET_FILE_INFO. File-

_Type nécessite l'écriture sur deux bytes (LByte, HByte). Exemple : la valeur $FC correspond à un fichier du type Basic AppleSoft (BAS).

Représentation des différents fichiers

$00		Fichier de type indéterminé
$04	TXT	Fichier texte (ASCII).
$06	BIN	Fichier binaire.
$0F	DIR	Fichier sous-catalogue.
$C0-$EF		Réservés pour ProDOS.
$F0	CMD	Fichier de commandes.
$F1-$F8		Fichiers redéfinissables.
$F9		Réservés pour ProDOS.
$FA	INT	Fichier Basic Integer.
$FB	IVR	Fichier Basic (variables).
$FC	BAS	Fichier AppleSoft (programmes).
$FD	VAR	Fichier AppleSoft (variables).
$FE	REL	Fichier codes relocatables.
$FF	SYS	Fichier Système.

• STORAGE_TYPE – 2 BYTES (LBYTE, HBYTE)

Détermine la structure intrinsèque du fichier. Uniquement un nibble est exploité à ce jour. La position relative $00 (00) dans chaque entrée de la table des matières d'un nom de fichier correspond à sa caractéristique intrinsèque. Deux paramètres combinés en un byte (Storage_Type et Name_Length) permettent de personnaliser le fichier tant dans son genre que dans sa longueur relative à son nom.

Dans le champ de la table des paramètres, Storage_Type est codé seul, sur deux bytes. Il détermine d'une part si le fichier contient des données, alors sa taille est représentée par un paramètre, et d'autre part, si ce fichier désigne une table des matières, alors le paramètre représente soit un Volume Directory, soit un Volume SubDirectory.

Les commandes CREATE et GET_FILE_INFO utilisent à travers le champ Storage_Type, la caractéristique dont l'image binaire traduit le genre du fichier (fichier de données ou de Volume).

Représentation du byte complet

	‹----- byte 1 -----›								‹----- byte 0 -----›								
Bits	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	0	0	0	0	0	S	S	S	S	= $000D
	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	= Directory File

$0000 Non défini. Représente un fichier effacé ou un fichier catalogue encore vide.

$0001 Seedling File. C'est la plus petite structure d'un fichier sauvegardé sur un disque. Il occupe au total 1 bloc. C'est généralement un fichier de données, pointé directement par son entrée correspondante dans le Volume qui contient son nom. Le pointeur, stocké aux positions relatives $11-$12 (17-18) dans l'entrée de la table des matières du fichier (Key_Pointer), désigne directement le numéro du bloc de données. La capacité de sauvegarde d'un tel fichier est de $0000 à $0200 bytes (0-512 bytes) et comporte un bloc de données et pas de bloc index.

$0002 Sapling File. C'est un fichier d'une taille supérieure à un fichier Seedling. Il peut contenir des données d'une taille de $0201 à $20000 bytes (513 à 131 072 bytes). Structure typique d'un tel fichier : l'entrée correspondante du nom du fichier pointe un Index Block, ou bloc index, qui peut à nouveau pointer jusqu'à 256 blocs de données (256 * 512 = 131 072 bytes ou 128 Ko).

$0003 Tree File. Ce paramètre désigne la limite maximale que peut atteindre la taille d'un fichier. Il peut contenir des données d'une taille allant de $020001 à $1000000 bytes (131 073 à 16 777 216 bytes ou 16 Mo). Structure typique d'un tel fichier : un Master Block, ou bloc index principal qui peut pointer jusqu'à 128 blocs index. Chaque bloc index peut à nouveau pointer jusqu'à un maximum de 256 blocs de données. Capacité maximale d'un fichier Tree : 32 768 Data Blocks pointés par 128 Index Blocks et 1 Master Block.

$0004 UCSD Pascal Language. Les fichiers UCSD sont réservés à un disque partionné pour contenir ce type de langage.

$000D Directory File. Représente le nom du fichier sauvegardé dans un Volume Directory et qui pointe un Volume SubDirectory (Nom d'un fichier SubDirectory).

Remarque

Les paramètres de valeurs $000E et $000F ne sont pas valides, ce sont des identificateurs de noms de volumes stockés dans le Key Block respectif (SubDirectory et Directory).

• AUX_TYPE – 4 BYTES (LOW WORD, HIGH WORD)

Ce paramètre est un attribut complémentaire, pour permettre au système de personnaliser certains types de fichiers (longueur d'un fichier texte, adresse d'implantation d'un fichier binaire, etc.).

Les positions relatives $1F-$20 (31-32) dans chaque entrée de la table des matières d'un nom de fichier correspondent à un attribut complémentaire. Il permet à ProDOS16 de stocker dans l'entrée du nom du fichier (disque) des valeurs complémentaires relatives soit à une longueur, soit à une adresse.

Le champ Aux_Type de la table des paramètres, codé sur quatre bytes, est utilisé à travers les commandes CREATE, SET_FILE_INFO et GET FILE_INFO et désigne soit un pointeur, soit une longueur. ProDOS16 n'utilise pas toujours le paramètre dans la table des matières du fichier, mais s'il est présent, la commande CATALOG l'affiche sous la rubrique Sub-Type. Les renseignements ainsi retournés peuvent être d'origines diverses : avec un fichier texte aléatoire, la longueur déclarée par le paramètre LE sera sauvegardée à l'emplacement Aux_Type (format : LByte, HByte).

Représentation des 4 bytes

	‹ Low Word ›		‹ High Word ›	
Bits	FEDCBA98	76543210	FEDCBA98	76543210
	LLLLLLLL	HHHHHHHH	LLLLLLLL	HHHHHHHH

Remarques

• Les fichiers séquentiels suivent une autre règle : ils ne nécessitent pas la déclaration de la longueur de l'enregistrement, et dans ce cas Aux_Type sera nul.

• Les fichiers binaires et Basic comportent dans l'entrée Aux_Type du fichier, un pointeur qui désigne l'adresse de début d'implantation du programme en mémoire sous la forme LByte, HByte. En Basic AppleSoft (File Type BAS), le pointeur sera représenté par 01 08, ou l'adresse $0801 représente l'adresse d'implantation par défaut en mémoire.

• BLOCKS_USED – 4 BYTES (LOW WORD, HIGH WORD)

Désigne le nombre total de blocs occupés par le fichier courant. Les positions relatives $13-$14 (19-20) dans chaque entrée de la table des matières d'un nom de fichier correspondent au nombre de blocs occupés par celui-ci. La commande GET_FILE_INFO retourne sur quatre bytes, à travers le champ du paramètre Block_Used, le nombre de blocs occupés par le fichier courant : blocs index et blocs de données compris.

– un fichier Seeding occupe 1 bloc.

– un fichier Sapling occupe au minimum 3 blocs : 1 bloc index et 2 bloc de données. Jusqu'à 257 blocs peuvent être gérés par un fichier Sapling : 1 bloc index et 256 blocs de données.

– un fichier Tree occupe au minimum 260 blocs : 1 Master Block, 2 blocs index et 257 blocs de données. Jusqu'à 32 897 blocs peuvent être gérés par un fichier Tree : 1 Master Block, 128 blocs index et 32 768 blocs de données.

L'occupation théorique des blocs peut prendre toute valeur de $0001 à $FFFF, soit 1 à 65 535 blocs. La commande CAT ou CATALOG affiche l'occupation des blocs pour chaque fichier sous la rubrique BLOCKS.

Représentation des 4 bytes

	‹ Low Word ›		‹ High Word ›	
Bits	FEDCBA98	76543210	FEDCBA98	76543210
	LLLLLLLL	HHHHHHHH	LLLLLLLL	HHHHHHHH

• TOTAL_BLOCKS – 4 BYTES (LOW WORD, HIGH WORD)

Si la commande est transmise à un Volume, le nombre de blocs total pouvant être géré par ce Volume est retourné via le paramètre Total_Blocks.

Les positions relatives $29-$2A dans chaque entrée d'un Volume Directory correspondent au nombre total de blocs occupés dans le Volume (disque). Uniquement deux bytes sont disponibles dans l'entrée du Volume.

Si la commande GET_FILE_INFO est exécutée à l'encontre d'un Volume, le champ Total_Blocks, assigné aux positions relatives $08-$0B dans la table des paramètres, retourne le nombre total de blocs occupés dans le Volume. Quatre bytes sont alors disponibles au paramètre Total_Blocks.

Si, par contre, la commande GET_FILE_INFO est transmise à l'encontre d'un fichier de données, le champ Block_Used retourne le nombre de blocs occupés par le fichier. Dans ce cas, le dernier champ (4 bytes) de la table des paramètres est utilisé.

Représentation des 4 bytes

	‹ Low Word ›		‹ High Word ›	
Bits	FEDCBA98	76543210	FEDCBA98	76543210
	LLLLLLLL	HHHHHHHH	LLLLLLLL	HHHHHHHH

• EOF – 4 BYTES (LOW WORD, HIGH WORD)

Pointe l'emplacement logique dans un fichier. EOF représente aussi le nombre total de bytes pouvant être lus à partir du fichier courant.

Les positions relatives $15-$17 (21-23) dans chaque entrée de la table des matières d'un nom de fichier correspondent à sa longueur (trois bytes).

Le champ EOF, d'une taille de quatre bytes, est utilisé par les commandes SET_EOF et GET_EOF.

Le pointeur EOF désigne dans un fichier un emplacement logique (fictif et non matérialisé), ce qui correspond à la longueur des données : Low Byte, Middle Byte, High Byte. La longueur théorique d'un fichier se situe entre les valeurs $000000 à $FFFFFF bytes.

Exemple

LByte	MByte	HByte	
B0	10	02	= $0210B0

Conversion hexadécimale/décimale

$B0 = 176 ; $10 = 16 ; $02 = 02
(LByte * 1) + (MByte * 256) + (HByte * 65 536)
(176 * 1) + (16 * 256) + (02 * 65 536) = 176 + 4096 + 131 072 = 135 344

- EOF détermine par son pointeur un byte logique, directement à la suite du caractère de fin des données physiques (MARK), et d'autre part, la longueur totale des données de ce même fichier.

Exemple typique

Sauvegardons le texte APPLE sous forme d'un fichier texte sur un disque.

La représentation des bytes physiques et logiques sera la suivante :

```
A  P  P  L  E CR          CR = retour – chariot ($0D)
41 50 50 4C 45 0D 00 00   Codes ASCII
00 01 02 03 04 05 06 07   Pointeur actuel $00 = MARK
01 02 03 04 05 06 07 08   Longueur 1, 2, etc. = EOF
```

– la position de MARK est matérialisée par le byte à la position relative $05, représenté par la valeur $0D, code ASCII du retour-chariot. C'est la position relative dans le texte du fichier, souvent désignée comme position courante pour lire ou écrire la donnée.

– la position de EOF est matérialisée par le byte logique à la position absolue $06 : les données ont une longueur de six bytes. Ce sixième byte correspond à la position absolue dans le fichier, matérialisée par le byte physique de valeur $0D, qui est le retour-chariot.

– le pointeur débute à la position relative $00 : il compte les caractères physiques et ajoute 1 au résultat. La valeur totale du compteur est en somme la longueur totale de l'enregistrement courant.

Commentaire relatif au texte APPLE

– la première rangée, utilisée comme texte témoin, représente les caractères réels sauvegardés sur le support disque, suivis par le retour-chariot ($0D).

– la deuxième rangée contient les codes ASCII des caractères de la première rangée, tels qu'ils peuvent être lus sur le disque avec un utilitaire approprié. Leur bits 7 est mis à zéro (0).

– la troisième rangée affiche les différentes positions relatives du pointeur à l'intérieur du fichier créé. La première valeur est $00. Ce type de pointeur est désigné le plus souvent sous le nom de MARK et débute à la position $00.

– la quatrième rangée affiche les positions absolues du pointeur à l'intérieur du fichier, ce qui détermine en même temps la longueur totale des données. Le pointeur débute à la position $01.

– le retour-chariot matérialise la fin de chaque enregistrement à l'intérieur d'un fichier. Le nombre d'enregistrements est limité par la taille du fichier possible. Le code CR ($0D) ne matérialise pas la fin du fichier mais uniquement celle de l'enregistrement courant. EOF serait alors le prochain byte à la suite du retour-chariot placé par le dernier enregistrement.

Remarques

• Si la fin d'un enregistrement sur un disque correspond avec la fin d'un bloc logique, plus aucun byte physique ne suivra et le dernier code ASCII sera $0D.

• Si par contre, la fin d'un fichier se trouve en avant de la fin d'un bloc sur le disque, le complément des bytes formant ce bloc sera représenté par des zéros (bytes nuls = 00). Le premier 0 après le dernier code ASCII du retour-chariot ($0D) matérialise la fin du fichier : il sera vu par ProDOS16 comme un byte logique (EOF).

• FILE_ENTRY – 2 BYTES (LBYTE, HBYTE)

Ce mot identifie la position de l'entrée d'un nom de fichier à l'intérieur d'une table des matières. Les fichiers effacés, sont ignorés et non comptabilisés pour l'affichage par la suite.

Un pointeur partiel, représenté par le champ File_Name_Offset, pointe successivement le nom du fichier désigné à un moment donné par File_Entry.

Un champ File_Entry qui débute avec la valeur $0000 représente le Directory Header, c'est-à-dire les noms et les attributs des différents fichiers comptabilisés à travers la table des matières. Par la suite, les informations qui suivent sont celles obtenues par la représentation du contenu de la table des matières 'scannée', formatées et renvoyées à l'écran. Le format affiché sera similiaire à celui obtenu par la commande CATALOG, transmise sous Basic.System.

Un champ File_Entry qui termine avec la valeur $FFFF représente la fin de la chaîne de caractères pointée par Entry_String. Ce type de chaîne de caractères comporte en plus les informations relatives à l'occupation du disque courant (BLOCKS FREE : et BLOCKS USED :).

• POSITION – 4 BYTES (LOW WORD, HIGH WORD)

Pointe la position courante dans un fichier. C'est la position relative, évaluée en nombre de bytes à partir du début du fichier. Position permet de lire ou d'écrire à la suite des données dans un fichier.
Le champ Position est utilisé par les commandes SET_MARK et GET MARK. Il est codé sur quatre bytes (Low Word, High Word), et représente la position courante du caractère dans un fichier. La valeur de Position ne peut en aucun cas excéder celle du pointeur EOF. Le pointeur MARK désigne la position d'un byte physique à l'intérieur d'un fichier : c'est le caractère courant lors du traitement des données. La fin d'un enregistrement de données sur un disque est matérialisée par le retour-chariot, ce qui détermine la valeur du champ Position. Le pointeur débute avec la position $00, compte les bytes physiques de l'enregistrement, et se positionne sur le caractère à lire ou à écrire.

• NEWLINE_CHAR – 2 BYTES (LBYTE, HBYTE)

Quel que soit le caractère défini par le byte de poids faible (LByte), il représente toujours le caractère Newline dans le fichier.

Le champ Newline_Char est utilisé par la commande NEWLINE et nécessite deux bytes pour coder le paramètre. Lorsque le champ Enable_Mask est nul, Newline_Character est passif.

Mise au point

Pointeur EOF

EOF désigne la longueur totale des enregistrements successifs réalisés dans un fichier, ou encore le pointeur de la position absolue du dernier byte des données. EOF n'est pas matérialisé sur le disque, mais pointe un byte logique et imaginaire qui se trouverait théoriquement à la suite du dernier enre-

gistrement. Le premier caractère de l'enregistrement suivant se trouverait à l'emplacement désigné actuellement par EOF.

Pointeur MARK

MARK pointe la position de la fin des données de l'enregistrement courant dans un fichier. Sa position est matérialisée sur le disque par un retour-chariot, code ASCII $0D. C'est une marque physique et réelle, visible dans le fichier, et qui représente la marque de la fin d'un enregistrement. Un fichier peut comporter une ou plusieurs marques de fin, selon qu'il contienne un ou plusieurs enregistrements de données.

Pointeur courant

La position courante du pointeur est calculée en fonction du pointeur MARK, ce qui détermine le caractère à lire ou à écrire dans le fichier. La valeur du pointeur est calculée en fonction de la position absolue du début des données (début = $000000).

- **ENABLE_MASK – 2 BYTES (LBYTE, HBYTE)**

Désigne le masque pour déterminer la valeur du caractère courant. Le système effectue un AND (ET logique) sur le caractère représenté par le byte de poids faible.

Le champ Enable_Mask est principalement utilisé par la commande NEWLINE. Il nécessite deux bytes pour coder un masque, ce qui permet au moment de la saisie des caractères, de forcer des bits à 1. Son utilisation en programmation machine pourrait avoir la syntaxe suivante :

```
LDA CARACTERE      Saisie d'un caractère du texte.
AND Enable_Maks    ET logique avec le masque.
CMP Newline_Char   Compare avec le dernier caractère du texte.
BEQ TERMINE        Caractère trouvé.
BNE CONTINUE       Continue avec le prochain caractère.
```

Lorsque l'accumulateur est chargé avec le code ASCII de Newline_Char, la saisie des caractères du fichier est interrompue.

Valeurs du masque

– si la valeur du paramètre est $7F (01111111), le bit 7 des caractères sera ignoré, ce qui a comme conséquence de forcer l'interpréteur à évaluer de façon identique les codes ASCII $0D et $8D : $0D possède le bit 7 = 0, et $8D a son bit 7 = 1.

– si la valeur du paramètre est $FF (11111111), chaque bit sera significatif tandis que la valeur $00 annulera le mode NEWLINE.

- **REF_NUM – 2 BYTES (LBYTE, HBYTE)**

Ce paramètre, alloué par la commande OPEN, référence le fichier ouvert pointé par Pathname. Le champ Ref_Num, codé sur deux bytes, est utilisé par les commandes OPEN, CLOSE, FLUSH, NEWLINE, READ, WRITE, SET_MARK, GET_MARK, SET_EOF et GET_EOF. Pour le traitement d'un et même fichier, il sera nécessaire d'utiliser la même valeur de Ref_Num.

ProDOS16 attribue un numéro d'identification (Ref_Num) à tout fichier au moment de déclarer la commande OPEN. Ce numéro de référence est retourné dans le champ Ref_Num. Par la suite, Ref_Num est utilisé par le système comme référence interne du MLI : il désigne un numéro d'ouverture du fichier et se trouve copié dans le File Control Block (FCB) à la position relative $00 de chaque entrée d'un fichier.

Toutes les commandes relatives au traitement et à la gestion des fichiers utilisent le paramètre Ref_Num comme numéro d'identification. Cette disposition permet, en association avec les commandes CLOSE et FLUSH, de traiter sélectivement les fichiers.

Les commandes CLOSE et FLUSH, transmises avec un Ref_Num de valeur $0000, sont opérationnelles envers des fichiers ouverts précédemment avec un paramètre Level égal ou plus grand que le dernier Level déclaré.

- **REQUEST_COUNT – 4 BYTES (LOW WORD, HIGH WORD)**

Désigne le nombre total de bytes devant être transférés. Le champ Request Count, codé sur quatre bytes (Low Word, High Word), est utilisé par les commandes READ et WRITE.

– avec la commande READ, c'est le nombre de caractères devant être transférés dans un tampon mémoire pointé par le champ Data_Buffer. Le nombre de bytes est limité par le nombre de pages mémoire libres. Lorsque la valeur de Newline_Char est fixée au préalable, tous les caractères à partir d'une position déterminée seront lus jusqu'au Newline byte, sinon ce sera Request_Count qui en fixera le nombre.

– avec la commande WRITE, c'est le nombre de caractères ou données devant être transférés du tampon mémoire pointé par Data_Buffer, vers les blocs logiques du disque.

- **TRANSFER_COUNT – 4 BYTES (LOW WORD, HIGH WORD)**

Désigne le nombre de bytes total réellement transférés. Le champ Transfer_Count, codé sur quatre bytes (Low Word, High Word), est utilisé par les commandes READ et WRITE. Ce sont les bytes réellement lus ou écrits, par comparaison à Request_Count qui désigne le nombre total des données à lire ou à écrire dans un fichier.

- **LEVEL – 2 BYTES (LBYTE, HBYTE)**

Le niveau d'initialisation par défaut de Level est $0000. Le champ Level se laisse modifier par la commande SET_LEVEL et détermine par la suite le niveau des commandes CLOSE et FLUSH. L'entrée du File Control Block est mise à jour, position relative $1B de l'entrée réservée au fichier (Level).

La commande OPEN alloue à chaque fichier ouvert un byte caractéristique représenté par le paramètre Ref_Num et utilisé par la suite par les commandes relatives au traitement des fichiers en général.

La commande SET_LEVEL autorise au champ Level toute valeur de $0000 à $00FF, assignant ainsi à toutes les commandes OPEN à venir le même niveau pour tous les fichiers ouverts. Par la suite, un champ Ref_Num de va-

leur $0000, transmis à travers les commandes CLOSE et FLUSH, traitera tous les fichiers ouverts avec un Level de niveau égal ou supérieur au dernier Level déclaré.

La commande GET_LEVEL retourne la valeur courante du champ Level. SET_LEVEL et GET_LEVEL ont été rendus nécessaires par le fait que ProDOS16 n'utilise pas la Global Page et ignore totalement l'adresse $BF94 qui contient normalement le paramètre Level (ProDOS8).

• DEV_NAME – 4 BYTES (LOW WORD, HIGH WORD)

Pointe dans la zone mémoire l'emplacement du tampon contenant la chaîne de caractères représentant le nom du Device courant.

Le champ Dev_Name, codé sur quatre bytes (Low Word, High Word), est utilisé par la commande GET_DEV_NUM pour pointer dans le tampon mémoire, la chaîne de caractères qui représente le nom du Device courant. La commande retourne alors le numéro de référence attribué à ce Device (Dev_Num).

Représentation des 4 bytes

	‹ Low Word ›		‹ High Word ›	
Bits	F E D C B A 9 8	7 6 5 4 3 2 1 0	F E D C B A 9 8	7 6 5 4 3 2 1 0
	L L L L L L L L	H H H H H H H H	L L L L L L L L	H H H H H H H H

• VOL_NAME – 4 BYTES (LOW WORD, HIGH WORD)

Pointe dans la zone mémoire l'emplacement du tampon qui contient la chaîne de caractères représentant le nom du Volume, premier slash inclus.

Le champ Vol_Name, codé sur quatre bytes (Low Word, High Word), est utilisé par la commande VOLUME pour renseigner le système sur le nom du Volume courant, le nombre de blocs total, le nombre de blocs libres et les identificateurs du Volume.

Représentation des 4 bytes

	‹ Low Word ›		‹ High Word ›	
Bits	F E D C B A 9 8	7 6 5 4 3 2 1 0	F E D C B A 9 8	7 6 5 4 3 2 1 0
	L L L L L L L L	H H H H H H H H	L L L L L L L L	H H H H H H H H

• DEV_NUM – 2 BYTES (LBYTE, HBYTE)

Représente le numéro du Device courant, paramètre utilisé exclusivement avec les commandes relatives aux Devices.

Le champ Dev_Num, codé sur 2 bytes (Low Byte, High Byte), est utilisé par les commandes GET_DEV_NUM, GET_DIB, READ_BLOCK et WRITE_BLOCK. Ce paramètre combine sur un byte utile l'image binaire des numéros du slot et du drive déclarés.

Représentation du champ Dev_Num

	‹ byte 1 ›	‹ byte 0 ›	
Bits	F E D C B A 9 8	7 6 5 4 3 2 1 0	
	0 0 0 0 0 0 0 0	D S S S x x x x	D = drive S = slot x = Inutilisés

Le byte 1 n'est pas utilisé pour le moment, mais réservé àv des applications futures. Le byte 0 trouve son utilisation pour représenter, par l'intermédiaire d'une combinaison binaire des bits 7 à 4, les Devices et slot courants.

DRIVE

Bit 7 = 0, donc Block Device 1
Bit 7 = 1, donc Block Device 2

SLOT

Bits : 6 5 4
0 0 1 = slot 1
1 0 0 = slot 4
1 1 0 = slot 6
1 1 1 = slot 7

Valeurs possibles de Dev_Num

Slots	7	6	5	4	3	2	1
Drive 1	70	60	50	40	30	20	10
Drive 2	F0	E0	D0	C0	B0	A0	90

Si le paramètre Dev_Num est nul, tous les slots sont mémorisés et le résultat copié dans le tampon Data_Buffer (le byte 1 est ignoré).

Exemples

Bits	7 6 5 4 3 2 1 0	
	1 1 1 0 0 0 0 0	= $E0
		= device 2, slot 6
Bits	7 6 5 4 3 2 1 0	
	0 0 1 1 0 0 0 0	= $30
		= device 1, slot 3
Bits	7 6 5 4 3 2 1 0	
	0 0 0 0 0 0 0 0	= $00
		= tous les slots mémorisés

• BLOCK_NUM – 4 BYTES (LOW WORD, HIGH WORD)

Représente le numéro du bloc qui sera lu ou écrit sur le Device courant. Le champ Block_Num, codé sur quatre bytes (Low Word, High Word), est utilisé par les commandes READ_BLOCK et WRITE_BLOCK pour désigner le numéro du bloc courant à lire ou à écrire sur le Block Device (drive) ré-

férencé par Dev_Num. La zone mémoire où se trouvent stockées les données est référencée par le champ Data_Buffer.

• RETURN_FLAG – 2 BYTES (LBYTE, HBYTE)

Ce champ est uniquement valide avec la version 1.0 de ProDOS16. A partir de la version 2.0, la commande QUIT ignore totalement ce paramètre.

Représente une combinaison binaire d'un mot codé sur 16 bits, destiné à dispatcher la prochaine application au moment de quitter un programme courant. Le champ Return_Flag, codé sur deux bytes, est utilisé uniquement par la commande QUIT (version 2.0) pour renseigner le système sur la prochaine application à lancer au moment de quitter le programme en cours d'exécution. Le paramètre n'est autre qu'un drapeau (flag) de situation, dont la combinaison binaire permet à ProDOS16 d'effectuer un choix dans l'exécution à venir : Return Flag (drapeau de retour) et Restart Flag (drapeau d'exécution). Dans le premier cas, l'adresse de la prochaine application est recherchée dans la pile ID (UserID), tandis que dans le second cas un programme est exécuté à partir de la mémoire.

La routine PQUIT, sollicitée par la commande QUIT, exécute une procédure de retour. Si la commande QUIT, au moment de quitter une application en cours d'exécution, ne spécifie pas de chemin, PQUIT prend en charge le déroulement des opérations.

– si le bit 15 du paramètre Return_Flag (Word) est égal à un (bit 15 = 1), l'adresse du programme (UserID) ayant transmis la commande QUIT est empilée comme adresse de retour. Si le drapeau est nul (bit 15 = 0), la procédure actuelle est abandonnée.

– si le bit 14 du paramètre Return_Flag (Word) est égal à un (bit 14 = 1), un programme déjà implanté en mémoire est exécuté. Si le drapeau Restart Flag est nul (bit 14 = 0), le programme à exécuter devra être chargé à partir d'un support disque.

Remarques

• La variable ID représente la pile interne, gérée par ProDOS16, et qui permet de placer en attente les adresses des chemins d'accès pour des applications futures.

• La variable UserID, stockée dans la pile ID, représente l'adresse du programme d'application courant à exécuter par la commande QUIT (UserID = programme utilisateur sur la pile interne). Le System Loader est par exellence le programme type qu'on pourrait citer en exemple, et dont l'adresse est gérée comme UserID (programme utilisateur par défaut).

• A chaque fois qu'un pointeur stocké comme UserID est utilisé comme adresse de retour de l'application en cours d'exécution, le drapeau Return_Flag est mis à jour.

• VERSION – 2 BYTES (LBYTE, HBYTE)

Représente le numéro de la version courante du système actif. Le champ Version, codé sur deux bytes, est utilisé exclusivement par la commande GET_VERSION pour retourner le numéro de la version actuelle de ProDOS16. Par version, il faut sous-entendre les numéros version et sous-version.

Représentation des 2 bytes

	‹ ---- High Byte -----›	‹ ----- Low Byte -----›
	‹ ------- byte 1 --------›	‹ ------- byte 0 --------›
Bits	F E D C B A 9 8	7 6 5 4 3 2 1 0
	B V V V V V V V	S S S S S S S S

– le bit F (High Byte) compose le bit significatif du mot contenu dans le champ Version, où la valeur 1 (B = 1) est attribuée pour une version finale, et la valeur 0 (B = 0) pour une version prototype (bêta).

– le byte de poids fort (High Byte) représente le numéro de la version, où les bits E à 8 sont de valeur 1 pour un système ProDOS16 version 1.0.

– le byte de poids faible (Low Byte) représente le numéro de la sous-version, où les bits 7 à 0 sont de valeur 0 pour un système ProDOS16 version 1.0.

• INT_NUM – 2 BYTES (LBYTE, HBYTE)

Identifie l'interruption par un numéro assigné au dispatcher des interruptions. Le champ Int_Num, codé sur 2 bytes, est utilisé par les commandes ALLOC_INTERRUPT et DEALLOC_INTERRUPT qui sont des gestionnaires des interruptions ProDOS16. Le paramètre représente le numéro alloué à la routine de service mise en place pour gérer une interruption précise.

• INT_CODE – 4 BYTES (LOW WORD, HIGH WORD)

Représente l'adresse de la zone mémoire où se trouve le début de la routine de service destinée à gérer l'interruption courante.

Le champ Int_Code, codé sur quatre bytes, est utilisé par la commande ALLOC_INTERRUPT, permettant de mettre en place une routine de service destinée à gérer une interruption courante.

• INT_MODE – 2 BYTES (LBYTE, HBYTE)

Le champ Int_Mode, codé sur deux bytes, est uniquement utilisé par la commande SET_INT_MODE, permettant d'attribuer un drapeau (Flag) de priorité à une interruption.

Le système d'exploitation ProDOS16, au moment d'intercepter une interruption, dépile successivement chaque adresse de la table du handler. Si aucune routine de service n'est valide, une recherche sera transmise à travers le vecteur IRQ. Si le vecteur IRQ est à l'état haut, le déroulement des opérations sera laissé au programme utilisateur, et la main rendue à ProDOS16.

Si le drapeau contenu dans le champ Int_Mode est nul, ProDOS16 exécute alors un retour de routine d'interruption à travers l'instruction machine RTI (ReTurn form Interrupt). L'application en cours d'exécution pourra alors continuer normalement.

Si le drapeau contenu dans le champ Int_Mode est positionné, ProDOS16 recherche à travers le SYSTEM DEATH le message correspondant à l'erreur fatale rencontrée, l'affiche à l'écran et suspend l'exécution en cours.

- **SAVE_NUM – 2 BYTES (LBYTE, HBYTE)**

Le champ Save_Num, codé sur deux bytes, représente un numéro attribué par la commande SAVE_STATE de l'état courant de la machine et du système mis en présence. Cette commande, transmise pendant le déroulement d'une application, permet de sauvegarder les informations relatives au matériel (Hard) et au système (System) mis en présence à cet instant précis. C'est ainsi que les informations relatives aux FCB et VCB restent préservées, ainsi que les vecteurs d'interruption programmés et les statuts des commutateurs logiques.

La commande RESTORE_STATE permet de rétablir à travers le champ Save_Num, l'état d'un environnement établi au moment de l'appel de la commande SAVE_STATE. La valeur autorisée du champ Save_Num est de $0000 à $00FF inclus, uniquement le byte de poids faible étant significatif.

- **FILE_SYS_ID – 2 BYTES (LBYTE, HBYTE)**

Le byte de poids faible du mot (Word) identifie le système d'exploitation auquel appartient ou pourra être traité le fichier ou le Volume courant.

Les systèmes d'exploitation ProDOS8, ProDOS16 et SOS utilisent le même principe dans la structure hiérarchisée des fichiers. Cependant, certains des fichiers stockés sur le support de sauvegarde ne pourront être lus que par l'un ou l'autre des systèmes et rarement par les trois en même temps. Les systèmes Pascal et DOS utilisent un format de fichier différent. Les différents fichiers initialisés sous d'autres systèmes sont comparés par une procédure interne à ProDOS, dont le champ File_Sys_Id est un des éléments de comparaison possible.

ProDOS8

ProDOS16 et ProDOS8 utilisent les mêmes identificateurs, sauf en ce qui concerne les fichiers du type $B3, $B4, $B5 et $B6, ces derniers étant réservés exclusivement à ProDOS16.

SOS

Les fichiers SOS sont reconnus en général par ProDOS16 et peuvent être du type fichiers table des matières (Directory File), de données (Text file) et binaires (Binary File). Ces trois types de fichiers sont compatibles SOS et ProDOS16.

DOS

Le système d'exploitation DOS (Disk Operating System) ne possède pas de structure hiérarchisée. ProDOS16 ne pourra donc pas lire directement ces types de fichiers mais nécessitera un utilitaire de transfert.

Pascal

Le système d'exploitation Pascal (Apple *II*) ne possède pas de structure hiérarchisée. ProDOS16 ne pourra donc pas lire directement ces types de fichiers, mais nécessitera un utilitaire de transfert.

Paramètres File_Sys_Id affectés aux différents systèmes

00 = réservé
01 = ProDOS/SOS File
02 = DOS 3.3
03 = DOS 3.2, DOS 3.1
04 = Apple *II* Pascal
05 = Macintosh
06 = Macintosh (HFS)
07 = LISA
08 = Apple *II* CP/M
09-255 réservés à des applications futures

Lecture sous DOS 3.3 et Pascal

Les disques 5,25 pouces (140 Ko) formatés sous DOS 3.3 et ProDOS8 utilisent la technique de division de la piste en 16 secteurs. Cette disposition permet à ProDOS16, à travers les commandes READ_BLOCK et WRITE_BLOCK, d'avoir accès aux données des fichiers créés sous ces deux systèmes. Les commandes READ_BLOCK et WRITE_BLOCK ne nécessitent pas une quelconque structure de la part des fichiers d'un disque, mais traitent directement les blocs logiques.

Piste/secteur en bloc logique

Cette table permet de déterminer le numéro d'un bloc logique par rapport à sa situation physique sur un disque (situation physique = piste et secteur). En premier, il faudra multiplier le numéro de la piste par 8, puis ajouter au résultat la valeur Offset du secteur correspondant. La rubrique 1/2 bloc logique signale la position du secteur dans la partie du bloc correspondant : 1 représente les 256 premiers bytes du bloc (256 bytes), et 2, les 256 bytes suivants du même bloc.

Secteur	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F,
Secteur Offset	0	7	6	6	5	5	4	4	3	3	2	2	1	1	0	7
1/2 bloc logique	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	2

Exemple

Si nous voulons connaître le numéro du bloc équivalent se trouvant à l'emplacement physique de la piste 2 secteur 6, le procédé de calcul sera le suivant :

Numéro du bloc = 8 * numéro de la piste physique + secteur Offset

Numéro du bloc = 8 * 2 + 4
Numéro du bloc = 20

Nous constatons que la piste physique 2 secteur 6 correspond au bloc logique $14 (20 en décimal), et seconde partie du 1/2 bloc.

- **ENTRY_STRING – 4 BYTES (LOW WORD, HIGH WORD)**

Ce champ contient l'adresse qui pointe une chaîne de caractères valides d'une longueur pouvant atteindre jusqu'à 255 codes ASCII, code(s) du retour-chariot inclus. Chaque caractère aura son bit de poids fort positionné 'off', c'est-à-dire bit 7 = 0. Les fichiers delétés sont ignorés et non comptabilisés pour l'affichage par la suite.

Chaque chaîne de caractères pointée par Entry_String se compose d'un certain nombre d'informations relatives aux noms des fichiers et de leurs attributs. En fin de lecture à travers les tables des matières d'un Volume, la chaîne de caractères sera formatée avant d'être affichée sur l'écran.

Sous ProDOS16, la chaîne de caractères retournée sera formatée d'une façon similaire au résultat qui serait obtenu par la commande CATALOG transmise sous BASIC.SYSTEM.

- **FILE_NAME_OFFSET – 2 BYTES (LBYTE, HBYTE)**

Ce champ représente un pointeur partiel, qui désigne la position du nom du fichier à l'intérieur de la chaîne de caractères retournée par Entry_String.

- **FILE_NAME_LEN – 2 BYTES (LBYTE, HBYTE)**

Ce champ donne la longueur du nom du fichier à l'intérieur de la chaîne de caractères retournée par Entry_String.

- **NULL_FIELD – 2, 4 OU 8 BYTES NULS**

Utilisés avec les commandes SET_FILE_INFO, GET_ENTRY et QUIT, ces bytes nuls sont placés dans la table des paramètres uniquement pour maintenir une symétrie avec d'autres commandes. Le système ProDOS16 ignore totalement le champ Null_Field.

COMMANDES RELATIVES AU VOLUME

Le Technical Manual les appelle 'File Housekeeping Calls': elles regroupent les commandes pour créer, renommer, effacer et se renseigner au niveau des fichiers dans un Volume courant.

Conventions relatives aux champs d'une table des matières

Pour lire correctement les données relatives à chaque commande MLI, certaines conventions ont été adoptées par le concepteur. C'est ainsi que chaque paramètre dans une table des paramètres, occupe un champ. Chaque champ possède une taille conditionnée suivant deux valeurs possibles: deux bytes pour désigner une valeur ou un résultat; quatre bytes pour désigner un pointeur, une valeur ou un résultat. Un champ dans une table de paramètres est utilisé soit pour transmettre des informations, soit pour interroger le MLI.

Champs Valeur, Résultat et Pointeur

Le manuel de référence les appelle respectivement Value, Result et Pointer.

– **Valeur, ou Value** désigne une quantité numérique, codée sur un ou deux mots (16 ou 32 bits), toujours utilisée pour transmettre un paramètre au MLI (Input Parameter).

– **Résultat, ou Result** désigne une quantité numérique, codée sur un ou deux mots (16 ou 32 bits), toujours utilisée pour retrouvrer un paramètre à partir du MLI (Output Parameter).

– **Pointeur, ou Pointer** désigne une adresse, codée sur deux mots (32 bits), toujours utilisée pour transmettre un paramètre au MLI (Input Parameter). La donnée pointée par le champ représente un pointeur utilisé par la suite soit pour transmettre, soit pour solliciter un paramètre du MLI (Input ou Output).

Syntaxe des pointeurs et valeurs

Les pointeurs, ainsi que toute valeur transmise ou renvoyée par le système, répondent à une syntaxe conditionnée par le microprocesseur mis en présence. Le 65C816, microprocesseur 16 bits, n'échappe pas à cette règle.

Par la suite, chaque champ interne à une table des paramètres est précédé d'un numéro désignant sa position relative dans la table, suit ensuite le nom du paramètre, le terme conventionnel (Valeur, Pointeur ou Résultat), le nombre de bytes affecté au champ courant et les valeurs numériques autorisées.

• Un pointeur en mode natif est généralement écrit sur quatre bytes, ce qui nécessite deux mots (Word) ou un double mot (DWord). Format d'écriture d'un double mot: Low Word, High Word, où Low Word représente le mot de poids faible et High Word le mot de poids fort.

• L'écriture d'une valeur courante nécessite le format: Low Byte, High Byte, où Low Byte (LByte) représente le byte de poids faible et High Byte (HByte) le byte de poids fort.

Liste des commandes du Directory

$01	CREATE	Crée un fichier programme ou Directory
$02	DESTROY	Efface un nom de fichier
$03	RENAME	Renomme un fichier courant
$04	CHANGE_PATH	Modifie le chemin d'accès à un fichier
$05	SET_FILE_INFO	Assigne des attributs à un fichier
$06	GET_FILE_INFO	Lit les attributs d'un fichier
$07	GET_ENTRY	Recherche et construit un Directory
$08	VOLUME	Lit les attributs d'un Volume
$09	SET_PREFIX	Attribue un numéro au préfixe
$0A	GET_PREFIX	Lit le numéro assigné au préfixe
$0B	CLEAR_BACKUP_BIT	Purge l'attribut Backup dans Access
$0C	WRITE_PROTECT	Protège le Volume contre l'écriture

Commandes Directory détaillées

CREATE ($01)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Pathname	Pointeur. 4 bytes ($0000 0000-$00FF FFFF)
$06-$07	Access	Valeur. 2 bytes ($0000-$00E3)
$08-$09	File_Type	Valeur. 2 bytes ($0000-$00FF)
$0A-$0D	Aux_Type	Valeur. 4 bytes ($0000 0000-$0000 FFFF)
$0E-$0F	Storage_Type	Valeur. 2 bytes ($0000-$000D)
$10-$11	Create_Date	Valeur. 2 bytes (00 00 pas de date)
$12-$13	Create_Time	Valeur. 2 bytes (00 00 pas d'heure)

• CREATE est la seule commande permettant de créer un fichier, qu'il soit table des matières ou tout simplement fichier de données. Un Volume (Volume Directory) ne sera pas créé par la commande CREATE, mais nécessitera un utilitaire de formatage. Par ailleurs, le système Pascal UCSD utilise CREATE pour initialiser certaines zones de la partition d'un disque dur par exemple.

• A la création d'un fichier de données ou de Volume SubDirectory, le système sauvegarde à la première position dans son entrée correspondante, un byte d'identité. Celui-ci se compose de la combinaison binaire de Storage_type et Name_Length: quatre bits sont nécessaires pour coder chacun d'eux. ProDOS16 permet, grâce à cet artifice, de différencier des fichiers au niveau de leur organisation logique:

$00 = fichier delété ou entrée non affectée
$01 = Seedling File
$02 = Sapling File
$03 = Tree File
$04 = UCSD Pascal
$0D = SubDirectory File

Création d'un Volume SubDirectory

CREATE Volume SubDirectory effectue la sauvegarde du nom d'un fichier SubDirectory dans le Volume courant, détermine l'emplacement du premier bloc du Volume SubDirectory Key Block et met à jour sa table des matières.

Création d'un fichier vide

CREATE File effectue la sauvegarde du nom d'un fichier dans son Volume correspondant, détermine le premier bloc des données qui sera vide pour le moment, et met à jour sa table des matières.

CREATE fixe automatiquement le byte Access et préserve ainsi le fichier de toute fausse manœuvre: le bit 1 (Write) sera mis à 1 (bit 1 = 1).

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$10	Device not found Device non trouvé en ligne
$27	I/O error Erreur d'entrées/sorties
$2B	Disk write protected Disque protégé en écriture
$40	Invalid pathname syntax Syntaxe ou chemin incorrect
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$45	Volume not found Volume non trouvé
$46	File not found Fichier non trouvé
$47	Duplicate pathname Nom du chemin existant déjà
$48	Volume full Disquette pleine ou dépassement du pointeur EOF
$49	Volume directory full Volume saturé (51 noms de fichiers stockés)
$4B	Unsupported (or incorrect) Storage_Type L'organisation logique du fichier est incorrecte
$52	Unsupported volume type Le Volume ProDOS courant est incompatible
$53	Parameter out of range Champ incorrect dans la table des paramètres
$58	Not a Block Device Le device attendu n'est pas un Block Device
$5A	Block number out of range Le format de la bit map est incorrect

DESTROY ($02)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Pathname	Pointeur. 4 bytes ($0000 0000-$00FF FFFF)

• La commande DESTROY efface un fichier spécifié dans le chemin courant (pathname), et met à jour la table des matières correspondante.

• Le nom du Volume Directory, ou un fichier muni d'un Storage_Type incompatible, ou encore un fichier ouvert ne peuvent être effacés de leur table des matières correspondante.

• Un fichier SubDirectory devra pointer un Volume SubDirectory vide avant de pouvoir être supprimé.

• La commande DESTROY met à zéro le byte Storage_Type et laisse intact le nom du fichier. Le contenu du bloc index est également mis à zéro.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$10	Device not found Device non trouvé en ligne
$27	I/O error Erreur d'entrées/sorties
$2B	Disk write protected Disque protégé en écriture
$40	Invalid pathname syntax Syntaxe ou chemin incorrect
$42	FCB table full Table FCB saturée
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$45	Volume not found Volume non trouvé
$46	File not found Fichier non trouvé
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible
$4B	Unsupported (or incorrect) Storage_Type L'organisation logique du fichier est incorrecte
$4E	Access not allowed Erreur d'accès au fichier courant
$50	File is open Le fichier est déjà ouvert
$52	Unsupported volume type Le Volume ProDOS courant est incompatible
$58	Not a Block Device Le device attendu n'est pas un Block Device
$5A	Block number out of range Le format de la bit map est incorrect

RENAME ($03)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Pathname	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Représente l'ancien chemin d'accès au fichier
$06-$09	New_Pathname	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Représente le nouveau chemin d'accès au fichier

• La commande RENAME renomme le fichier dont le nom est spécifié dans Pathname, en lui affectant comme nouveau nom, le nom contenu dans New_Pathname. Elle a fait son apparition à partir de la version 2.0.

• Avec la commande RENAME, les chemins déclarés doivent être les mêmes et faire partie d'une table des matières commune.

Code d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties
$2B	Disk write protected Disque protégé en écriture
$40	Invalid pathname syntax Syntaxe ou chemin incorrect
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$45	Volume not found Volume non trouvé
$46	File not found Fichier non trouvé
$47	Duplicate pathname Nom du chemin existant déjà
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible
$4B	Unsupported (or incorrect) Storage_Type L'organisation logique du fichier est incorrecte
$4E	Access not allowed Erreur d'accès au fichier courant
$50	File is open Le fichier est déjà ouvert
$52	Unsupported volume type Le Volume ProDOS courant est incompatible
$57	Duplicate volume Le nom du Volume existe déjà
$58	Not a Block Device Le device attendu n'est pas un Block Device

CHANGE_PATH ($04)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Pathname	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Représente le chemin d'accès actuel au fichier
$06-$09	New_Pathname	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Représente le nouveau chemin d'accès au fichier

• La commande CHANGE_PATH remplit une fonction nouvelle sous ProDOS16, permettant de transférer à l'intérieur d'un Volume commun une entrée nom de fichier, d'un Directory dans un autre. Le contenu du fichier n'est pas déplacé par cette commande, mais uniquement l'entrée de son nom. Les champs Pathname et New_Pathname représentent soit un chemin partiel, soit un chemin complet, et sont communs au même Volume.

• CHANGE_PATH, à l'encontre de RENAME, permet de déplacer un nom de fichier dans une table des matières différente, mais faisant partie du même Volume. Les données ne sont pas modifiées, ni déplacées.

• Si l'écriture des chemins Pathname et New_Pathname est identique, abstraction faite du nom du fichier, le résultat final sera le même que celui observé à travers la commande RENAME.

Code d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties
$2B	Disk write protected Disque protégé en écriture
$40	Invalid pathname syntax Syntaxe ou chemin incorrect
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$45	Volume not found Volume non trouvé
$46	File not found Fichier non trouvé
$47	Duplicate pathname Nom du chemin existant déjà
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible
$4B	Unsupported (or incorrect) Storage_Type L'organisation logique du fichier est incorrecte
$4E	Access not allowed Erreur d'accès au fichier courant
$50	File is open Le fichier est déjà ouvert
$52	Unsupported volume type Le Volume ProDOS courant est incompatible
$57	Duplicate volume Le nom du Volume existe déjà
$58	Not a Block Device Le Device attendu n'est pas un Block Device

SET_FILE_INFO ($05)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Pathname	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans le tampon mémoire le Pathname valide
$06-$07	Access	Valeur. 2 bytes ($0000-$00E3)
$08-$09	File_Name	Valeur. 2 bytes ($0000-$00FF)
$0A-$0D	Aux_Type	Valeur. 4 bytes ($0000 0000-$0000 FFFF)
$0E-$0F	Null Field	Valeur. Champ nul de deux bytes Ce champ est ignoré par ProDOS

$10-$11	Create_Date	Valeur. 2 bytes au format défini par convention
$12-$13	Create_Time	Valeur. 2 bytes au format défini par convention
$14-$15	Mod_Date	Valeur. 2 bytes au format défini par convention
$16-$17	Mod_Time	Valeur. 2 bytes au format défini par convention

• La commande SET_FILE_INFO permet la modification sélective de certains attributs copiés dans chaque entrée d'un nom de fichier. Cette commande peut être transmise indifféremment sur un fichier ouvert ou fermé.

• Si l'attribut Access est modifié lorsque le fichier est ouvert, il ne sera reconnu comme tel qu'à la prochaine ouverture du fichier par OPEN. Aucune donnée résidente en mémoire ne sera affectée par SET_FILE_INFO.

• Si la commande SET_FILE_INFO est associée à un fichier ouvert, elle restera dans l'incapacité d'appeler la routine de gestion de la table FCB qui traite les données du bloc. En effet, FCB se trouve implantée dans la carte langage et débute aux adresses $F6FC; elle est par principe difficilement adressable. Il sera plus sage d'utiliser SET_FILE_INFO avec un fichier fermé.

• Sous le système ProDOS16, les champs Create_Date et Create_Time sont ignorés.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties
$2B	Disk write protected Disque protégé en écriture
$40	Invalid pathname syntax Syntaxe ou chemin incorrect
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$45	Volume not found Volume non trouvé
$46	File not found Fichier non trouvé
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible
$4B	Unsupported (or incorrect) Storage_Type L'organisation logique du fichier est incorrecte
$4E	Access not allowed Erreur d'accès au fichier courant
$52	Unsupported volume type Le Volume ProDOS courant est incompatible
$53	Parameter out of range Champ incorrect dans la table des paramètres
$58	Not a Block Device Le device attendu n'est pas un Block Device

GET_FILE_INFO ($06)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Pathname	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans le tampon mémoire le Pathname valide
$06-$07	Access	Résultat. 2 bytes ($0000-$00E3)
$08-$09	File_Type	Résultat. 2 bytes ($0000-$00FF)
$0A-$0D	Aux_Type	Résultat. 4 bytes ($0000 0000-$0000 FFFF) – Si la commande est transmise à un fichier de données, ce champ contiendra une caractéristique complémentaire, relative au fichier courant (adresse du programme Basic, etc)
	or	
	Total Blocks	Résultat. 4 bytes ($0000 0000-$00FF FFFF) – Si la commande est transmise au Volume Directory, ce champ contiendra le nombre total de blocs pouvant être gérés par ce volume.
$0E-$0F	Storage_Type	Résultat. 2 bytes ($0000-$000D) Les valeurs $0E et $0F ne sont pas valides (attributs Key Directory et Key SubDirectory)
$10-$11	Create_Date	Résultat. 2 bytes (format défini par convention)
$12-$13	Create_Time	Résultat. 2 bytes (format défini par convention)
$14-$15	Mod_Date	Résultat. 2 bytes (format défini par convention)
$16-$17	Mod_Time	Résultat. 2 bytes (format défini par convention)
$18-$1B	Blocks_Used	Résultat. 4 bytes ($0000 0000-$FFFF FFFF) Ce champ contient le nombre total de blocs logiques alloués à un fichier de données. Cette valeur correspond à celle de Block_Used dans l'entrée du fichier, positions relatives $13-$14. – Si la commande est transmise au Volume, la valeur retournée par ce champ correspondra au nombre total des blocs logiques occupés par tous les fichiers sauvegardés dans le Volume en question.

• La commande GET_FILE_INFO lit les attributs dans l'entrée d'un fichier pointé par la variable Pathname, et les retourne dans les champs marqués Résultat.

• La commande peut être transmise indifféremment à un fichier ouvert ou fermé.

• Lorsque GET_FILE_INFO est transmis à un fichier Volume, le MLI retourne dans le champ Total_Blocks la capacité totale du volume en nombre de blocs logiques (Volume = disque)

• Si la commande est transmise à un fichier de données, le champ Aux_Type retourne un attribut complémentaire spécifique au fichier pointé par la variable Pathname : adresse d'implantation d'un fichier Basic, longueur d'un fichier texte, etc.

Capacité de sauvegarde d'un Volume (disque)

La capacité de sauvegarde d'un disque se trouve stockée aux positions relatives $29-$2A (41-42) de l'entrée de la table des matières du Volume Directory. Le pouvoir de sauvegarde d'un disque (Volume) est déterminé au moment de son formatage : pour un disque standard du type 5,25 pouces, elle est de $0118 blocs, soit 280 blocs par défaut.

Nombre total de blocs occupés par les fichiers

Ce nombre est calculé par une routine interne au MLI en faisant la somme des blocs attribués à chaque fichier sauvegardé sur le disque. A cet effet, deux bytes sont réservés dans chaque entrée nom de fichier d'une table des matières, ce qui permet de stocker le nombre de blocs qui lui est alloué. Blocks_Used représente l'attribut blocs occupés, et se trouve aux positions relatives $13-$14 (19-20) de son entrée correspondante.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties
$40	Invalid pathname syntax Syntaxe ou chemin incorrect
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$45	Volume not found Volume non trouvé
$46	File not found Fichier non trouvé
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible
$4B	Unsupported (or incorrect) Storage_Type L'organisation logique du fichier est incorrecte
$52	Unsupported volume type Le Volume ProDOS courant est incompatible
$53	Parameter out of range Champ incorrect dans la table des paramètres
$58	Not a Block Device Le device attendu n'est pas un Block Device

GET_ENTRY ($07)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Null_Field	Valeur. Champ nul de 4 bytes Ce champ n'est pas utilisé par ProDOS
$06-$07	Access	Résultat. 2 bytes ($0000-$00E3)
$08-$09	File_Type	Résultat. 2 bytes ($0000-$00FF)
$0A-$0D	Aux_Type	Résultat. 4 bytes ($0000 0000-$0000 FFFF)
$0E-$0F	Storage_Type	Résultat. 2 bytes ($0000-$000D) Les valeurs $0E et $0F ne sont pas valides (attributs Key Directory et Key SubDirectory)
$10-$11	Create_Date	Résultat. 2 bytes au format défini par convention
$12-$13	Create_Time	Résultat. 2 bytes au format défini par convention
$14-$15	Mod_Date	Résultat. 2 bytes au format défini par convention
$16-$17	Mod_Time	Résultat. 2 bytes au format défini par convention
$18-$1B	Blocks_Used	Résultat. 4 bytes ($0000 0000-$FFFF FFFF) Contient le nombre total de blocs alloués au fichier.
$1C-$1F	EOF	Résultat. 4 bytes ($0000 0000-$FFFF FFFF) Nombre total de bytes pouvant être lus dans le fichier courant.
$20-$21	File_Entry	Résultat. 2 bytes ($0000-$FFFF) Identifie la position du fichier à l'intérieur de la table des matières d'un Directory.
$22-$23	Ref_Num	Résultat. 2 bytes ($0000-$00FF)
$24-$27	Entry_String	Résultat. 4 bytes ($0000 0000-$00FF FFFF) Pointe la chaîne de caractères courante
$28-$29	File_Name_Offset	Résultat. 2 bytes ($0000-$00FF) Pointeur offset du nom du fichier à l'intérieur de la chaîne de caractères pointée par le champ Entry_String.
$2A-$2B	File_Name_Len	Résultat. 2 bytes ($0000-$00FF) Longueur du nom du fichier qui se trouve dans la chaîne de caractères pointée par Entry_String.

• Cette commande est un utilitaire du système, permettant une recherche des tables des matières à travers le Volume, et la construction en mémoire d'un catalogue Directory. GET_ENTRY sera indispensable en présence d'une version future de ProDOS16, pour recenser dans un catalogue certains types de fichiers nouveaux. Avant de transmettre la commande, le fichier table des matières devra être ouvert par la commande OPEN.

• GET_ENTRY est très proche de la commande GET_FILE_INFO. Elle retourne, en plus des informations connues, des codes ASCII dans une chaîne de caractères formatée. Le système affiche par la suite, des informations relatives au Volume dans un format similaire à celui de la commande CATALOG sous Basic.System. La chaîne de caractères est limitée à 255 codes ASCII valides, retours-chariot inclus.

• La lecture des données utilisées pour formater la chaîne de caractères est réalisée à travers le Volume en ligne. L'affichage par défaut se fait par le Character Device courant, écran vidéo par exemple. Le bit 7 de chaque code ASCII sera mis à zéro avant l'affichage des données.

• GET_ENTRY, dans sa recherche à travers la table des matières, exploite la particularité de la position relative d'une entrée de nom de fichier dans le bloc auquel il fait partie. Les fichiers effacés (delétés) sont ignorés par la commande et non comptabilisés par la suite.

• Un champ File_Entry qui débute avec la valeur $0000 représente le Directory Header, c'est-à-dire les noms et les attributs des différents fichiers comptabilisés à travers la table des matières. Par la suite, les informations qui suivent sont celles obtenues par la représentation du contenu de la table des matières 'scannée', formatées et renvoyées à l'écran. Le format affiché sera similaire à celui obtenu par la commande CATALOG transmise sous Basic.System.

• Un champ File_Entry qui termine avec la valeur $FFFF représente la fin de la chaîne de caractères pointée par Entry_String. Ce type de chaîne de caractères se compose essentiellement des informations relatives à l'occupation du disque courant (BLOCKS FREE: et BLOCKS USED:).

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$43	Invalid file reference number Numéro de référence du fichier incorrect (Ref_Num)
$45	Volume not found Volume non trouvé
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible
$51	Directory structure damaged Structure de la table des matières non conforme
$52	Unsupported volume type Le Volume ProDOS courant est incompatible

VOLUME ($08)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Dev_Name	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans le tampon mémoire, la chaîne de caractères du nom du Device (.D1, .D2, etc.)

$06-$09	Vol_Name	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans le tampon mémoire le début de la chaîne de caractères qui représente le nom du Volume courant.
$0A-$0D	Total_Blocks	Résultat. 4 bytes ($0000 0000-$0000 FFFF) Total de blocs que peut gérer le Volume.
$0E-$11	Free_Blocks	Résultat. 4 bytes ($0000 0000-$00FF FFFF) Total de blocs libres dans le Volume.
$12-$13	File_Sys_Id	Résultat. 2 bytes ($0000-$00FF) Le byte de poids faible du mot (Word) identifie le système d'exploitation auquel appartient le fichier ou le Volume courant. ProDOS8 et ProDOS16 utilisent les mêmes identificateurs, sauf en ce qui concerne les fichiers du type $B3, $B4, $B5 et $B6, ces derniers étant réservés exclusivement à ProDOS16.

• La commande VOLUME permet, en décarant le nom du Device courant (Dev_Name), de connaître le nom du Volume représenté par le Device (Vol_Name), le nombre total de blocs logiques du Volume (Total_Blocks), le nombre de blocs libres dans le Volume (Free_Blocks) et l'attribut d'identification du Volume (File_Sys_Id).

• Le nom du Volume retourné est précédé du slash (/);

• La liste des Volumes en ligne est générée, ce qui est l'équivalent de la commande ON_LINE sous ProDOS8 avec Unit_Number = 0. La commande VOLUME renvoie les noms successifs des Devices en ligne, format: .D1, .D2, etc. Si aucun Volume n'est en ligne, ProDOS16 renvoie le code d'erreur $11 (Invalid Device name).

Valeurs de File_Sys_Id valides

00 = réservé
01 = ProDOS/SOS File
02 = DOS 3.3
03 = DOS 3.2, DOS 3.1
04 = Apple *II* Pascal
05 = Macintosh
06 = Macintosh (HFS)
07 = LISA
08 = Apple *II* CP/M
09-255 réservés à des applications futures

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$10	Device non found Device non trouvé en ligne
$27	I/O error Erreur d'entrée/sortie
$2E	Disk switched Disques inversés: les commandes ne peuvent aboutir
$45	Volume not found Volume non trouvé
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible
$52	Unsupported volume type Le Volume ProDOS courant est incompatible
$55	VCB table full Table VCB saturée
$57	Duplicate volume Le nom du Volume existe déjà
$58	Not a Block Device Le device attendu n'est pas un Block Device

SET_PREFIX ($09)

$00-$01	Caller_Id	Valeur. deux bytes ($0000-$00FF)
$02-$03	Préfix_Num	Valeur. deux bytes ($0000-$0003) Fixe l'un des quatre attributs possibles d'un préfixe, suivi par le slash (0/, 1/, 2/ et 3/)
$04-$07	Prefix	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans le tampon mémoire la chaîne de caractères qui représente le Directory valide.

• La commande SET_PREFIX alloue un attribut valide à un préfixe déclaré par le champ Prefix qui pointe le nom du Directory courant. Le préfixe sera référencé par la suite à l'aide d'un numéro de 0 à 3, suivi par le slash. Trois valeurs par défaut sont possibles:

0/ désigne le préfixe ayant donné naissance au système: c'est le nom du Volume à partir duquel ProDOS16 a été booté.

1/ désigne le SubDirectory dans lequel se trouve l'application: c'est le chemin d'accès dans le SubDirectory de la prochaine application à exécuter.

2/ désigne le SubDirectory dans lequel se trouve le fichier 'Library': c'est le chemin d'accès dans le SubDirectory, du fichier 'Library' qui sera utilisé par l'application en cours d'exécution.

• Une application permet de remplacer le chemin d'accès courant par un autre préfixe (y compris les numéros 0/, 1/, 2/ et 3/). Le Pathname sera un chemin partiel ou complet.

• Un champ Prefix qui pointe dans le tampon mémoire une chaîne de caractères avec le premier byte nul, désigne un Pathname nul. La chaîne de caractères d'un préfixe nul comporte le premier byte de valeur zéro, les by-

tes suivants étant par définition réservés aux codes ASCII du nom du chemin.

• Lorsque le système ProDOS16 est booté en mémoire, le préfixe par défaut est celui du nom du Volume qui a initialisé le système.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$10	Device non found Device non trouvé en ligne
$27	I/O error Erreur d'entrées/sorties
$40	Invalid pathname syntax Syntaxe ou chemin incorrect
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$45	Volume not found Volume non trouvé
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible
$4B	Unsupported (or incorrect) Storage_Type L'organisation logique du fichier est incorrecte
$52	Unsupported volume type Le Volume ProDOS courant est incompatible
$53	Parameter out of range Champ incorrect dans la table des paramètres
$58	Not a Block Device Le Device attendu n'est pas un Block Device

GET_PREFIX ($0A)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Préfix_Num	Valeur. 2 bytes ($0000-$0003) Représente le préfixe courant
$04-$07	Prefix	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans la zone tampon la chaîne de caractères qui représente le préfixe requis.

• La commande GET_PREFIX retourne l'attribut du préfixe système (Prefix_Num). Le champ Prefix pointe la chaîne de caractères du préfixe désigné.

• Si le préfixe est nul, le champ Prefix pointe dans la zone tampon le byte longueur de valeur nulle.

• Si le préfixe est valide, byte de longueur différent de zéro, le nom du préfixe est retourné entre deux slash.

Exemple
/USERS.DISK/ ou /USERS.DISK/UTILITY/

• La zone tampon, utilisée comme buffer auxiliaire, permet au système de gérer des données relatives au Pathname : la longueur et le nom du chemin, barre oblique (/) incluse, y sont placés.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$53	Parameter out of range Champ incorrect dans la table des paramètres

CLEAR_BACKUP_BIT ($0B)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Pathname	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans le tampon mémoire la chaîne de caractères qui contient le Pathname valide.

• La commande CLEAR_BACKUP_BIT permet de garder une certaine similitude avec le système ProDOS8, qui utilise quant à lui le Backup bit à travers la Global Page, adresse $BF95 (Bubit). Une valeur $20 (bit 5 = 1) signifie que le fichier a été modifié depuis sa dernière copie; une valeur $00 (bit 5 = 0) signifie que le fichier n'a pas subi de modifications depuis sa dernière sauvegarde.

• Sous ProDOS16, le Backup bit se trouve intégré dans le byte Access (bit 5) et conditionne le fichier : si le bit 5 = 1, le fichier a été modifié, tandis qu'une valeur du bit 5 = 0 signifie qu'aucune modification n'est intervenue depuis sa dernière sauvegarde.

• ProDOS16 positionne automatiquement le bit 5 (bit 5 = 1) à la prochaine écriture dans le fichier; la commande CLEAR_BACKUP_BIT permet de le remettre à zéro.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$40	Invalid pathname syntax Syntaxe ou chemin incorrect
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$45	Volume not found Volume non trouvé
$46	File not found Fichier non trouvé
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible

$52	Unsupported volume type Le Volume ProDOS courant est incompatible
$58	Not a Block Device Le Device attendu n'est pas un Block Device

WRITE_PROTECT ($0C)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Vol_Name	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans le tampon mémoire la chaîne de caractères qui contient le nom du Volume ou le Device valide.
	Dev_Name	

• La commande WRITE_PROTECT est transmise pour doter un volume d'un statut 'write-protect'. Le terme 'write-protect' affecte une protection sélective contre l'écriture, au Volume ou Device pointé par le champ Vol_Name ou Dev_Name.

• Le nom du Volume sera précisé au préalable soit par un préfixe (précédé d'un slash), soit par un device qui contient le volume courant.

• Toute tentative d'écriture contre un Volume affecté d'un statut 'write-protect', retourne le code d'erreur $2B: Write Protecteḍ (protégé contre l'écriture).

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$10	Device not found Device non trouvé en ligne
$28	No Device connected Pas de Device connecté à l'UC
$2B	Disk write protected Disque protégé en écriture
$45	Volume not found Volume non trouvé

COMMANDES RELATIVES A LA GESTION DE FICHIERS

Le Technical Manuel les appelle 'File Access Calls' : elles regroupent la famille des commandes MLI traitant plus particulièrement les données d'un fichier en général. Ces routines servent à la lecture ou à l'écriture des données, entre un périphérique et le micro-ordinateur. On pourrait citer en exemple un lecteur de disques et la mémoire de l'Apple. Cette méthode entraîne un certain nombre de contrôles et modifie les paramètres en conséquence.

Liste des commandes de gestion

$10	OPEN	Ouvre un fichier texte et lui alloue un tampon mémoire de 1024 bytes.
$11	NEWLINE	Autorise le mode lecture à travers Newline.
$12	READ	Transfère des données d'un disque dans le tampon mémoire alloué par OPEN.
$13	WRITE	Transfère des données du tampon mémoire sur le disque.
$14	CLOSE	Marque la fin de tout accès à un fichier.
$15	FLUSH	Vide sur le disque les données encore présentes dans le tampon mémoire.
$16	SET_MARK	Définit la position courante dans un fichier (MARK).
$17	GET_MARK	Retourne la valeur de la position courante dans un fichier (MARK).
$18	SET_EOF	Définit une position logique qui correspond à la taille du fichier (EOF = En Of File).
$19	GET_EOF	Retourne la valeur de la taille du fichier.
$1A	SET_LEVEL	Définit le niveau d'ouverture du fichier (Level).
$1B	GET_LEVEL	Retourne le niveau d'ouverture du fichier (Level).

Commandes détaillées

OPEN ($10)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Ref_Num	Résultat. 2 bytes ($0000-$00FF) C'est le numéro de référence alloué à l'ouverture du fichier.
$04-$07	Pathname	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans le tampon mémoire la chaîne de caractères qui représente le Pathname.
$08-$0B	I/O_Buffer	Résultat. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans la zone mémoire le buffer alloué aux données.

• La commande OPEN prépare un fichier à la lecture ou à l'écriture. Le nombre de fichiers pouvant être ouverts simultanément n'est limité que par la taille de la mémoire mise en présence (ProDOS16 version 2.0).

• Le nom du fichier et ses attributs sont copiés dans l'entrée du File Control Block (FCB).

• Lorsque OPEN est déclaré, ProDOS16 alloue au fichier nommé une zone mémoire de 1024 octets utilisée comme tampon (buffer) mémoire transitoire. Ce tampon restera actif jusqu'à la déclaration de CLOSE : les données encore présentes dans le tampon seront sauvegardées sur le disque, le tampon sera libéré et le fichier fermé.

• I/O_Buffer se scinde en deux parties distinctes : les 512 premiers bytes sont utilisés pour placer les données du fichier, tandis que les 512 bytes suivants gèrent le répertoire des blocs alloués.

• La position courante dans le fichier ouvert est initialisée à la valeur zéro, et le champ Ref_Num contient au retour la valeur de Reference Number. Ce paramètre est un identificateur attribué à chaque fichier ouvert, et se trouve copié dans le FCB, à la position relative $00 de chaque entrée d'un fichier.

• Après la commande OPEN, le MLI copie dans la table FCB le numéro de référence qui lui aura été alloué, et retourne cette valeur dans le champ Ref_Num. Par la suite, ce code de référence devra être le même pour toutes les commandes relatives au traitement de ce même fichier.

Codes d'erreur possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties
$40	Invalid pathname syntax Syntaxe ou chemin incorrect
$42	FCB table full Table FCB saturée
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$45	Volume not found Volume non trouvé
$46	File not found Fichier non trouvé
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible
$4B	Unsupported (or incorrect) Storage_Type L'organisation logique du fichier est incorrecte
$4E	Access not allowed Erreur d'accès au fichier courant
$50	File is open Le fichier est déjà ouvert
$52	Unsupported volume type Le Volume ProDOS courant est incompatible
$54	Out of Memory Structure mémoire incompatible
$57	Duplicate volume Le nom du Volume existe déjà

NEWLINE ($11)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Ref_Num	Résultat. 2 bytes ($0000-$00FF) Identificateur alloué par OPEN
$04-$05	Enable_Mask	Valeur. 2 bytes ($0000-$00FF) Masque utilisé pour effectuer un ET logique (AND) avec le caractère courant
$06-$07	Newline_Char	Valeur. 2 bytes ($0000-$00FF) Newline caractère

• La commande NEWLINE permet d'activer ou de désactiver le mode de lecture Newline en présence d'un fichier ouvert par OPEN. Chaque caractère lu est transféré dans la zone mémoire réservée, pointée par le champ Data_Buffer. Par la suite, un ET logique du byte de poids faible du champ Enable_Mask est transmis à chaque caractère lu, et le résultat comparé au contenu du champ Newline_Char. Si le résultat est celui attendu, la lecture prend fin.

• Lorsque le mode Newline est désactivé, la lecture dans un fichier prend fin lorsque le nombre de bytes fixés par le paramètre Request_Count est atteint, ou lorsque la fin du fichier est rencontrée (EOF).

• Lorsque le mode Newline est actif, la lecture dans un fichier prend fin lorsque le caractère défini par le champ Newline_Char est rencontré.

• Un champ Enable_Mask de valeur $00 donne le même résultat qu'un mode Newline désactivé.

• NEWLINE est utilisée pour définir par programmation, un byte de masque (Enable_Mask), ce qui permet de lire par la suite un caractère de fin dans un fichier texte.

• Avec Newline passif, la recherche d'un caractère ou d'une position à l'intérieur d'un fichier est réalisée avec le champ Enable_Mask de valeur $00 et Newline_Char équivalent à un retour-chariot. Le code ASCII de valeur $0D représente le caractère par défaut de la fin d'un enregistrement ou d'un champ de données (retour-chariot).

• Avec Newline actif, un caractère défini par le programmeur sera recherché à travers le fichier courant. Sa valeur, déterminée par un code ASCII, sera représentée par le masque du champ Enable_Mask et influencera de ce fait le paramètre Newline_Char.

Opération avec Enable_Mask actif

Le caractère lu est placé dans le tampon pointé par le champ Data_Buffer, puis un ET Logique (AND) est effectué sur ce caractère par l'intermédiaire du masque Enable_Mask. Ensuite, le résultat est comparé à Newline_Char. Si le résultat est nul, le caractère courant correspond au caractère comparé, et détermine ainsi la fin des données.

Exemple

En attribuant les valeurs $7F à Enable_Mask (masque) et $0D à Newline-_Char, tout caractère testé, ayant comme valeur $0D ou $8D sera reconnu comme caractère de fin des données (retour-chariot). Ce procédé ne modifie en rien la valeur ou le code ASCII du caractère lu.

Opération avec Enable_Mask passif

Le système utilise ses propres paramètres: ces valeurs sont établies par défaut et deux pointeurs permettent une utilisation aisée (EOF et MARK). EOF pointe un byte logique dans le fichier texte, tandis que MARK pointe un byte physique, avec $0D comme valeur du champ Newline_Char.

Valeurs du masque Enable_Mask

Le champ Enable_Mask pourra prendre toute valeur de $00 à $FF inclus, établie selon la circonstance et le but recherché du programme utilisateur: une valeur $7F force à 0 le bit 7 du caractère ASCII considéré; une valeur $FF rend tous les bits significatifs tandis que la valeur $00 rend le masque passif.

Codes d'erreurs possibles

$06	Invalid caller identification Identificateur non valide (commande MLI)
$43	Invalid reference number. Fichier: numéro de référence incorrect

READ ($12)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Ref_Num	Valeur. 2 bytes ($0000-$00FF) Numéro de référence alloué par OPEN
$04-$07	Data_Buffer	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans la zone mémoire l'emplacement où seront copiées les données lues à partir du disque par exemple.
$08-$0B	Request_Count	Valeur. 4 bytes ($0000 0000-$00FF FFFF) Nombre de bytes à transférer.
$0C-$0F	Transfer_Count	Résultat. 4 bytes ($0000 0000-$00FF FFFF) Nombre de bytes ayant été réellement transférés.

• La commande READ effectue la lecture à partir d'un fichier référencé par Ref_Num. Le nombre de bytes fixés par le champ Request_Count sont lus à partir de la position courante, puis transférés dans la zone mémoire pointée par Data_Buffer. Au retour de la commande, le nombre de bytes réellement lus se trouve dans le champ Transfer_Count. Ce nombre de bytes correspond aux données réellement lues.

Rappel de certaines règles fondamentales

La fin d'un enregistrement, ou champ de données, ne détermine pas forcément la fin du fichier courant. Plusieurs champs, ou enregistrements successifs, peuvent faire partie d'un même et unique fichier. La fin de chaque champ de données se trouve matérialisée à l'intérieur d'un fichier par le code ASCII $0D, qui représente en fait un retour chariot (CR). Deux cas de figure peuvent se présenter avec la commande READ:

- le caractère de fin de données a été défini au préalable;
- le caractère n'a pas été défini.

Cela implique que le mode de lecture Newline soit activé dans le premier cas et désactivé dans le second.

Mode de lecture Newline activé

La commande NEWLINE effectue à partir d'un disque le transfert vers le tampon mémoire d'un nombre de bytes définis par Request_Count. La lecture débute à la position courante (Mark). Le fichier est identifié par son numéro de référence (Ref_Num) attribué au moment de son ouverture. Le tampon mémoire où seront placées les données est pointé par Data_Buffer.

Mode de lecture Newline désactivé

Si Newline_Char est rencontré avant que le nombre de bytes fixés par Request_Count soit lu, le champ Transfer_Count prendra comme valeur le nombre de bytes réellement transférés, Newline bytes inclus.

Remarque

L'erreur **End of file encountered** (fin du fichier rencontrée, code $4C), signale qu'aucun byte n'a été transféré; le paramètre Transfer_Count aura dans ce cas la valeur $00.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties
$43	Invalid file reference number Numéro de référence du fichier incorrect (Ref_Num)
$4C	End Of File encountered (out of data) Fin du fichier rencontrée
$4E	Access not allowed Erreur d'accès au fichier courant

WRITE ($13)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Ref_Num	Valeur. 2 bytes ($0000-$00FF) Numéro de référence alloué par OPEN

$04-$07	Data_Buffer	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans la zone mémoire l'emplacement où se trouvent les données lues à partir d'un disque par exemple.
$08-$0B	Request_Count	Valeur. 4 bytes ($0000 0000-$00FF FFFF) Nombre de bytes à transférer
$0C-$0F	Transfer_Count	Résultat. 4 bytes ($0000 0000-$00FF FFFF) Nombre de bytes ayant été réellement transférés

• La commande WRITE effectue le transfert des données du tampon mémoire pointé par Data_Pointer sur un support de sauvegarde du type Block Device. Le fichier traité est celui référencé par Ref_Num. Le nombre de bytes à sauvegarder à partir de la position courante dans le fichier est fixé par le champ Request_Count. Le nombre de bytes réellement sauvegardés est retrouvé dans le champ Transfer_Count.

• La position à l'intérieur du fichier sera mise à jour. Sa nouvelle valeur sera égale à la somme de la position initiale et du contenu du champ Transfer_Count. Les autres paramètres affectés au fichier seront mis à jour par la suite : si le fichier comporte plus de 512 bytes de données, un bloc index lui sera alloué; le pointeur EOF sera mis à jour, ce qui déterminera la taille du fichier (nombre de blocs). Par la suite, l'instruction Basic CATALOG affiche sous la rubrique SUBTYPE la longueur de l'enregistrement du fichier (texte du type aléatoire), ce qui équivaut au paramètre R£.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties
$2B	Disk write protected Disque protégé en écriture
$43	Invalid file reference number Numéro de référence du fichier incorrect (Ref_Num)
$48	Volume full Disquette pleine ou dépassement du pointeur EOF
$4C	End Of File encountered (out of data) Fin du fichier rencontrée
$4E	Access not allowed Erreur d'accès au fichier courant
$5A	Block number out of range Le format de la bit map est incorrect

CLOSE ($14)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Ref_Num	Valeur. 2 bytes ($0000-$00FF) Numéro de référence attribué par OPEN

• La commande CLOSE ferme le ou les fichiers ouverts précédemment par la commande OPEN. Le File Control Block (FCB) est mis à jour et le contenu de l'entrée correspondante au fichier, fermé, puis purgé de son contenu. Le tampon mémoire alloué par OPEN, pointé par le champ I/O_Buffer, est vidé de son contenu. L'entrée du nom du fichier dans sa table des matières correspondante sera mise à jour. Le numéro de référence contenu dans le champ Ref_Num sera libéré et disponible pour une autre application.

• Le champ Ref_Num référence le fichier ouvert. Si un paramètre Ref_Num de valeur $0000 est déclaré avec la commande CLOSE, tous les fichiers ouverts à cet instant et dotés d'un Level égal ou supérieur au dernier Level déclaré, seront fermés.

Exemple

Supposant d'une part trois fichiers ouverts, avec comme Level successifs 0, 1 et 2, et d'autre part, le dernier Level déclaré de valeur 1, la commande CLOSE dotée d'un Ref_Num $0000 fermera tous les fichiers, sauf celui doté du paramètre Level 0.

• Le champ Level pourra être modifié avec la commande SET_LEVEL, tandis que GET_LEVEL permet de lire son contenu.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties
$2B	Disk write protected Disque protégé en écriture
$43	Invalid file reference number Numéro de référence du fichier incorrect (Ref_Num)
$5A	Block number out of range Le format de la bit map est incorrect

FLUSH ($15)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Ref_Num	Valeur. 2 bytes ($0000-$00FF) Numéro alloué par la commande OPEN

• La commande FLUSH sauvegarde sur le disque le contenu du tampon fichier pointé par I/O_Buffer. Par la suite, l'entrée correspondante de la table des matières sera mise à jour.

• Si le champ Ref_Num est de valeur $0000, tous les fichiers ouverts à cet instant seront traités par la commande FLUSH (voir CLOSE pour de plus amples détails).

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties
$2B	Disk write protected Disque protégé en écriture
$43	Invalid file reference number Numéro de référence du fichier incorrect (Ref_Num)
$48	Volume full Disquette pleine ou dépassement du pointeur EOF
$5A	Block number out of range Le format de la bit map est incorrect

SET_MARK ($16)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Ref_Num	Valeur. 2 bytes ($0000-$00FF) Numéro alloué par la commande OPEN
$04-$07	Position	Valeur. 4 bytes ($0000 0000-$00FF FFFF) Fixe dans un fichier, la position relative à partir de laquelle débutera le traitement de la commande WRITE ou READ.

• La commande SET_MARK met à jour la position courante dans un fichier (MARK) dont l'emplacement sera spécifié par le champ Position. Cependant, la valeur de Position ne pourra pas excéder celle de EOF (End Of File).

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties
$43	Invalid file reference number Numéro de référence du fichier incorrect (Ref_Num)
$4D	Position out of range Position courante dans le fichier incorrecte
$5A	Block number out of range Le format de la bit map est incorrect

GET_MARK ($17)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Ref_Num	Valeur. 2 bytes ($0000-$00FF) Numéro alloué par la commande OPEN
$04-$07	Position	Résultat. 4 bytes ($0000 0000-$00FF FFFF) Retourne la position courante à l'intérieur d'un fichier traité par la commande READ ou WRITE.

• La commande GET_MARK retourne dans le champ Position, la position courante à l'intérieur d'un fichier ouvert. C'est la position relative, prise en considération par les commandes READ et WRITE. Après OPEN, le pointeur est mis à $00 et ne pourra en aucun cas déborder la valeur de EOF.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$43	Invalid file reference number Numéro de référence du fichier incorrect (Ref,_Num)

SET_EOF ($18)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Ref_Num	Valeur. 2 bytes ($0000-$00FF) Numéro alloué par la commande OPEN
$04-$07	EOF	Valeur. 4 bytes ($0000 0000-$00FF FFFF) Fixe le nombre total de bytes pouvant être lus à partir d'un fichier. C'est par défaut la position logique dans un fichier.

• La commande SET_EOF fixe à l'intérieur d'un fichier référencé par Ref_Num, une position logique de fin (EOF) qui n'est autre que le total de bytes pouvant être lus.

• Une modification du champ EOF entraîne automatiquement une mise à jour de la bit map représentant le répertoire des blocs occupés.

- si EOF est diminué, la bit map sera diminuée en conséquence;
- si EOF est augmenté, la bit map sera augmentée en conséquence;
- si EOF reste identique, la bit map ne sera pas modifiée.

• Si le fichier traité est protégé en écriture (Write enable, Access byte avec le bit 1 = 0), aucune modification ne pourra être réalisée au niveau du champ EOF.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties

$43	Invalid file reference number Numéro de référence du fichier incorrect (Ref,_Num)
$4D	Position out of range Position courante dans le fichier incorrecte
$4E	Access not allowed Erreur d'accès au fichier courant
$5A	Block number out of range Le format de la bit map est incorrect

GET_EOF ($19)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Ref_Num	Valeur. 2 bytes ($0000-,ds00FF) Numéro alloué par la commande OPEN
$04-$07	EOF	Résultat. 4 bytes ($0000 0000-$00FF FFFF) Retourne le nombre total de bytes pouvant être lus à partir d'un fichier. C'est par défaut la position logique dans un fichier.

• La commande GET_EOF retourne dans le champ EOF le nombre total de bytes pouvant être lus à partir d'un fichier.

• EOF se trouve positionné sur la fin logique d'un fichier et détermine de ce fait sa longueur en nombre de bytes.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$43	Invalid file reference number Numéro de référence du fichier incorrect (Ref,_Num)

SET_LEVEL ($1A)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Level	Valeur. 2 bytes ($0000-$00FF) Détermine le niveau des commandes CLOSE et FLUSH.

• La commande SET_LEVEL modifie la valeur courante du champ Level, ce qui détermine par la suite le nouveau niveau des commandes CLOSE et FLUSH. L'entrée du File Control Block est mise à jour, position relative $1B de l'entrée réservée au fichier (Level).

• La commande OPEN alloue à chaque fichier ouvert un byte caractéristique représenté par le paramètre Ref_Num, utilisé par la suite par toutes les commandes relatives au traitement des fichiers en général.

• La commande SET_LEVEL autorise au champ Level toute valeur de $0000 à $00FF, assignant ainsi à toutes les commandes OPEN à venir le même niveau pour tous les fichiers ouverts.

• Un champ Ref_Num de valeur $0000, transmis à travers les commandes CLOSE et FLUSH, traite tous les fichiers ouverts avec un Level de niveau égal ou supérieur au dernier Level déclaré.

• GET_LEVEL permet de connaître la valeur courante du champ Level.

• Les commandes SET_LEVEL et GET_LEVEL ont été rendues nécessaires par le fait que ProDOS16 n'utilise pas la Global Page et ignore totalement l'adresse $BF94 qui contient normalement le paramètre Level (ProDOS8).

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$59	Invalid Level Niveau d'ouverture du fichier incorrect (Level)

GET_LEVEL ($1B)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Level	Résultat. 2 bytes ($0000-$00FF) Retourne la valeur courante du niveau.

• La commande GET_LEVEL retourne la valeur courante du champ Level, valeur utilisée par les commandes CLOSE et FLUSH pour déterminer le niveau de fermeture des fichiers.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)

COMMANDES RELATIVES AUX DEVICES

Le Technical Manual les appelle 'Device Calls' : elles regroupent la famille des commandes MLI traitant plus particulièrement le matériel de l'environnement de l'Apple *IIGS* , et plus particulièrement les Blocks Devices et les Character Devices.

Liste des commandes relatives aux Blocks Devices

$20	GET_DEV_NUM	Retourne dans le champ Dev_Num le numéro du device nommé lors de l'appel de la commande.

$21	GET_DIB	Retourne dans le champ DIB le pointeur de la table Device Information Block (DIB).
$22	READ_BLOCK	Transfère un bloc de 512 bytes à partir du device courant dans le tampon pointé par Data_Buffer.
$23	WRITE_BLOCK	Transfère à partir du tampon Data_Buffer un bloc de 512 bytes sur le Device courant.

Commandes détaillées

GET_DEVICE_NUM ($20)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Dev_Name	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans la zone mémoire la chaîne de caractères qui désigne le nom du Device
$06-$08	Dev_[illegible]	Résultat. 2 bytes ($0000-$00FF) C'est le numéro de référence utilisé par les différentes commandes relatives aux Devices en général

• La commande GET_DEV_NUM retourne dans le champ Dev_Num le numéro du Device courant utilisé par la suite par les commandes MLI.

• Le périphérique traité pourra être du type Block Device ou Character Device.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$10	Device not found Device non trouvé en ligne

GET_DIB ($21)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Dev_Num	Valeur. 2 bytes ($0000-$00FF) C'est le numéro de référence utilisé par les différentes commandes relatives aux Devices et retourné par la commande GET_DEV_NUM
$04-$07	DIB	Résultat. 4 bytes ($0000 0000-$00FF FFFF) Retourne l'adresse de la table des informations relatives aux Devices en général.

• La commande GET_DIB retourne dans le champ DIB l'adresse de la table des informations relatives au Device désigné par le paramètre Dev_Num.

• Chaque device, qu'il soit du type Block Device ou Character Device, possède sa propre entrée dans la table des informations (DIB).

• GET_DIB renseigne le système ProDOS16 sur le contenu de la table des informations, mais en aucun cas ne modifie un quelconque paramètre interne.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$11	Invalid Device name or number Nom ou numéro du Device incorrect

READ_BLOCK ($22)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Dev_Num	Valeur. 2 bytes ($0000-$00FF) C'est le numéro de référence utilisé par les différentes commandes relatives aux Devices et retourné par la commande GET_DEV_NUM
$04-$07	Data_Buffer	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans la zone mémoire l'adresse à partir de laquelle seront copiées les données du fichier.
$08-$0B	Block_Num	Valeur. 4 bytes ($0000 0000-$00FF FFFF) Numéro du bloc lu par la commande READ.

• La commande READ_BLOCK transfère un bloc de données, lues à partir d'un disque référencé par le champ Dev_Num, vers un tampon mémoire pointé par Data_Buffer.

• Le tampon mémoire représenté par Data_Buffer sert à transférer les données. Il aura au minimum une taille de 512 bytes, sa taille maximale étant limitée par la mémoire disponible et le Device courant.

• Le champ Block_Num représente le numéro du bloc courant lu à partir du device désigné par Dev_Num.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)

$11	Invalid Device name or number Nom ou numéro du Device incorrect
$27	I/O error Erreur d'entrées/sorties
$28	No Device connected Pas de Device connecté à l'UC

WRITE_BLOCK ($23)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Dev_Num	Valeur. 2 bytes ($0000-$00FF) C'est le numéro de référence utilisé par les différentes commandes relatives aux Devices et retourné par la commande GET_DEV_NUM
$04-$07	Data_Buffer	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans la zone mémoire l'adresse à partir de laquelle se trouvent les données du fichier.
$08-$0B	Block_Num	Valeur. 4 bytes ($0000 0000-$00FF FFFF) Numéro du bloc écrit par la commande WRITE

• La commande WRITE_BLOCK sauvegarde à chaque fois un bloc de données, du tampon mémoire pointé par Data_Buffer sur un disque référencé par le champ Dev_Num.

• Le numéro du bloc en cours de sauvegarde est représenté par Block-_Num et désigne un bloc logique.

• Chaque bloc logique représente une taille par défaut de 512 bytes, ce qui correspond à deux secteurs physiques d'un disque standard.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$11	Invalid Device name or number Nom ou numéro du Device incorrect
$27	I/O error Erreur d'entrées/sorties
$28	No Device connected Pas de Device connecté à l'UC
$2B	Disk write protected Disque protégé en écriture

COMMANDES RELATIVES A L'ENVIRONNEMENT

Le Technical Manual les appelle 'Environment Calls' : elles regroupent la famille des commandes MLI traitant plus particulièrement l'environnement logiciel et matériel de l'Apple *II*GS , et sont, de ce fait liées directement à la configuration présente.

Liste des commandes relatives à l'environnement

$25	START_PRODOS	Alloue un numéro d'identification à la commande courante.
$26	END_PRODOS	Annule le numéro d'identification alloué précédemment à la commande courante.
$27	GET_PATHNAME	Retourne le chemin complet de l'application courante.
$28	GET_BOOT_VOL	Retourne le nom du Volume ayant booté le système ProDOS16.
$29	QUIT	Termine la présente application et détermine la nouvelle procédure.
$2A	GET_VERSION	Retourne le numéro de la version du système courant.
$2B	SAVE_STATE	Alloue un numéro à l'environnement courant composé du matériel et du système d'exploitation.
$2C	RESTORE_STATE	Retourne le numéro d'état alloué au matériel et au système d'exploitation référencé par la variable Save_Num.

Commandes détaillées

START_PRODOS ($25)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)

• La commande START_PRODOS précèdera chaque appel du MLI, c'est-à-dire qu'elle sera transmise en premier avant chaque commande ProDOS16.

• Au retour, le champ Caller_Id contiendra l'identificateur de la commande qui sera transmise par la suite.

• Le champ Caller_Id, commun à toutes les tables des paramètres d'une commande ProDOS16, permet de situer la commande référencée.

Pas d'erreur possible.

END_PRODOS ($26)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)

• La commande END_PRODOS libère l'identificateur de la commande dont le champ Caller_Id spécifie le numéro.

• Cette disposition particulière signifie à ProDOS16 que le traitement de la commande référencée par Caller_Id est terminé.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)

GET_PATHNAME ($27)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Data_Buffer	Résultat. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans la zone mémoire l'emplacement de la chaîne de caractères qui représente le Pathname complet.

• La commande GET_PATHNAME retourne à travers le champ Data_Buffer le chemin d'accès complet de l'application en cours d'exécution.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$27	I/O error Erreur d'entrées/sorties
$40	Invalid pathname syntax Syntaxe ou chemin incorrect
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$45	Volume not found Volume non trouvé
$46	File not found Fichier non trouvé
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible
$4B	Unsupported (or incorrect) Storage_Type L'organisation logique du fichier est incorrecte

GET_BOOT_VOL ($28)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$05	Data_Buffer	Résultat. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans la zone mémoire l'emplacement de la chaîne de caractères qui représente le nom du Volume ayant booté le système.

• La commande GET_BOOT_VOL permet au programme utilisateur d'identifier le nom du Volume à partir duquel a été initialisé le système d'exploitation, en l'occurrence ProDOS16.

• Le champ Data_Buffer, au retour du MLI, pointe la chaîne de caractères qui représente le nom du Volume ayant booté le système ProDOS16.

• Il est à noter que ProDOS16 peut être initialisé lors d'un boot à froid (ColdStart) ou par la relance du système machine déjà sous tension (Warmstart). Dans le premier cas, la disquette système se trouvera dans le device sollicité en premier, tandis que la commande dash (-) pourra être utilisée dans le second cas. Si le système ProDOS16 est chargé à l'adresse $00/2000, un JMP to $00/2000 lancera l'exécution.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)

QUIT ($29)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$08	Null_Field	Valeur. 8 bytes nuls ($0000 0000 0000 0000)

• La commande QUIT termine l'application courante.

• Si un fichier QUIT est trouvé dans le SubDirectory approprié du Volume ayant transmis le boot au système, il sera chargé en mémoire et exécuté.

• Si aucun fichier QUIT n'est trouvé, ProDOS16 recherche à travers le dispatcher le nom de la prochaine application.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$46	File not found Fichier non trouvé
$54	Out of memory Structure mémoire incompatible

GET_VERSION ($2A)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Version	Résultat. 2 bytes ($0000-$FFFF) Retourne le numéro de la version du système courant.

• La commande GET-VERSION retourne dans le champ Version le numéro de la version courante du système ProDOS16. Le terme version sous-entend les numéros de la version et de la sous-version.

• Le byte de poids fort (High Byte) représente le numéro de la version, où les bits E à 8 sont de valeur 1 pour un système ProDOS16 version 1.0.

• Le byte de poids faible (Low Byte) représente le numéro de la sous-version, où les bits 7 à 0 sont de valeur 0 pour un système ProDOS16 version 1.0.

• Le bit F (High Byte) compose le bit significatif du mot contenu dans le champ Version, où la valeur 1 (B = 1) est attribuée pour une version finale, et la valeur 0 (B = 0) pour une version prototype (bêta).

SAVE_STATE ($2B)

$00-$01	Caller_Id	Valeur.2 bytes ($0000-$00FF)
$02-$03	Save_Num	Valeur. 2 bytes ($0000-$00FF) Numéro sous lequel a été sauvegardé le statut de l'environnement (matériel et système).

• La commande SAVE_STATE affecte à travers le champ Save_Num un numéro à l'environnement, personnalisant par la suite l'état de la machine et du système d'exploitation mis en présence au moment même où la commande a été transmise.

• Cette commande, transmise pendant le déroulement d'une application, permet de sauvegarder temporairement un certain nombre d'informations:

- toutes les entrées du File Control Block (FCB), relatives aux fichiers ouverts, restent préservées;
- toutes les entrées du Volume Control Block (VCB);
- la table des vecteurs d'interruption;
- le statut de tous les commutateurs logiques utilisés.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$54	Out of memory Structure mémoire incompatible

RESTORE_STATE ($2C)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00ff)
$02-$03	Save_Num	Valeur. 2 bytes ($0000-$00FF) Numéro sous lequel a été sauvegardé le statut de l'environnement (matériel et système).

• La commande RESTORE_STATE, rétablit l'état de la machine et du système, dont une sauvegarde a été réalisée sous le numéro contenu dans le champ Save_Num.

• Tous les statuts sont restitués et les valeurs restaurées, ce qui permet de poursuivre une application avec les mêmes paramètres sauvegardés au moment où la commande SAVE_STATE aura été transmise.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$53	Parameter out of range Champ incorrect dans la table des paramètres

COMMANDES DE CONTROLE DES INTERRUPTIONS

Le Technical Manual les appelle 'Interrupt Control Calls': elles regroupent la famille des commandes MLI, traitant plus particulièrement les interruptions du materiel et du logiciel.

Liste des commandes de contrôle des interruptions

$30	ALLOC_INTERRUPT	Installe une routine de service
$31	DEALLOC_INTERRUPT	Annule la routine de service spécifiée dans le champ Int_Num
$32	SET_INT_MODE	Drapeau d'identification permettant par la suite une gestion correcte de l'interruption en cours.

Commandes détaillées

ALLOC_INTERRUPT ($30)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Int_Num	Résultat. 2 bytes ($0000-$00FF) Retourne le numéro de référence sous lequel ProDOS16 gère la routine de service affectée à l'interruption (Int_Num = numéro de priorité)
$04-$07	Int_Code	Pointeur. 4 bytes ($0000 0000-$00FF FFFF) Pointe dans la table du handler, l'adresse de la routine de service destinée à gérer l'interruption courante.

• La commande ALLOC_INTERRUPT permet de mettre en place une routine de service destinée à gérer par la suite une interruption. ProDOS16 construit une table des vecteurs d'interruption (handler), permettant de gérer jusqu'à 16 routines de service.

• Le Handler soutient une table des vecteurs d'interruption élaborée, rendant possible l'interception d'une interruption issue du matériel (hardware).

• La routine de service destinée à gérer l'interruption en cours, devra être mise en place suivant certaines conventions établies et sous le contrôle du programme utilisateur.

• Le champ Int_Num retourne dans le byte de poids faible le numéro de priorité sous lequel la routine de service sera gérée par le handler. La routine de service mise en place en premier possède le plus haut degré de priorité. Le paramètre Int_Num sera déclaré au moment de purger une routine de service.

• Le champ Int_Code sera chargé avec l'adresse qui pointe dans la table du handler l'adresse d'implantation de la routine de service.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)
$25	Interrupt vector table full Table des routines de service pleine (interruption)
$53	Parameter out of range Champ incorrect dans la table des paramètres

DEALLOC_INTERRUPT ($31)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Int_Num	Résultat. 2 bytes ($0000-$00FF) Désigne le numéro de référence de la routine de service à purger de la table du handler.

• La commande DEALLOC_INTERRUPT permet de purger de la table du handler une routine de service dont le numéro est contenu dans le champ Int_Num. Il sera nécessaire de neutraliser au préalable le matériel pouvant éventuellement engendrer une interruption de ce type par la suite.

• La table du handler sera mise à jour et le pointeur de l'adresse de la routine de service annulé. Les priorités du handler seront réactualisées.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Indentificateur non valide (commande MLI)
$53	Parameter out of range Champ incorrect dans la table des paramètres

SET_INT_MODE ($32)

$00-$01	Caller_Id	Valeur. 2 bytes ($0000-$00FF)
$02-$03	Int_Mode	Valeur. 2 bytes ($0000 ou $80000) Désigne le numéro de référence de la routine de service à purger de la table du handler.

• La commande SET_INT_MODE permet de conditionner l'interruption à travers un drapeau contenu dans le champ Int_Mode.

• ProDOS16, lorsqu'une interruption prend naissance, dépile successivement chaque adresse de la table du handler et compare la validité du contenu. Si aucune routine de service n'est valide, une recherche est transmise à travers le vecteur IRQ. Si le vecteur IRQ est à l'état haut, le déroulement des opérations sera laissé au programme utilisateur, et la main rendue à ProDOS16.

• Le champ Int_Mode, codé sur 2 bytes, est uniquement utilisé par la commande SET_INT_MODE, permettant d'attribuer un drapeau (Flag) de priorité à une interruption.

• Si le drapeau contenu dans le champ Int_Mode est nul, ProDOS16 exécute alors un retour de routine d'interruption à travers l'instruction machine RTI (ReTurn from Interrupt). L'application en cours d'exécution pourra alors continuer normalement.

• Si le drapeau contenu dans le champ Int_Mode est positionné, ProDOS16 recherche à travers le SYSTEM DEATH le message correspondant à l'erreur fatale rencontrée, l'affiche à l'écran et suspend l'exécution en cours.

Codes d'erreurs possibles

$00	No error Pas d'erreur
$06	Invalid caller identification Identificateur non valide (commande MLI)

MESSAGES D'ERREURS SOUS ProDOS16

Erreurs non fatales

• Erreurs d'ordre général

$00	No error Pas d'erreur
$01	Invalid Call number Numéro non valide d'une commande MLI
$05	Call pointer out of bounds Pointeur non valide assigné à une commande
$06	Invalid caller identification Identificateur non valide (commande MLI)

- **Erreurs relatives aux Devices**

$10	Device not found Device non trouvé en ligne
$11	Invalid Device name or number Nom ou numéro du Device incorrect
$20	Invalid request code Requête non valide
$25	Interrupt vector table full Table des routines de services pleine (interruption)
$26	Invalid operation Opération transmise non valide
$27	I/O error Erreur d'entrées/sorties
$28	No Device connected Pas de Device connecté à l'unité centrale
$2B	Disk write protected Disque protégé en écriture
$2E	Disk switched Disques inversés : les commandes ne peuvent aboutir
$30-$3F	Device spécific errors Erreurs définies au préalable et spécifiques au Device

- **Erreurs relatives aux fichiers en général**

$40	Invalid pathname syntax Syntaxe ou chemin incorrect
$42	FCB table full Table FCB saturée
$43	Invalid file reference number Numéro de référence du fichier incorrect (Ref_Num)
$44	Path not found Nom du SubDirectory dans le chemin introuvable
$45	Volume not found Volume non trouvé
$46	File not found Fichier non trouvé
$47	Duplicate pathname Nom du chemin existant déjà
$48	Volume full Disquette pleine ou dépassement du pointeur EOF
$49	Volume directory full Volume saturé (51 noms de fichiers stockés)
$4A	Version error (incompatible file format) La version courante de ProDOS est incompatible
$4B	Unsupported (or incorrect) Storage_Type L'organisation logique du fichier est incorrecte
$4C	End Of File encountered (out of data) Fin du fichier rencontrée
$4D	Position out of range Position courante dans le fichier incorrecte
$4E	Access not allowed Erreur d'accès au fichier courant
$50	File is open Le fichier est déjà ouvert
$51	Directory structure damaged Structure de la table des matières non conforme
$52	Unsupported volume type Le Volume ProDOS courant est incompatible
$53	Parameter out of range Champ incorrect dans la table des paramètres
$54	Out of memory Structure mémoire incompatible
$55	VCB table full Table VCB saturée
$57	Duplicate volume Le nom du volume existe déjà
$58	Not a Block Device Le device attendu n'est pas un Block Device
$59	Invalid Level Niveau d'ouverture du fichier incorrect (Level)
$5A	Block number out of range Le format de la bit map est incorrect

Erreurs non fatales

$01	Unclaimed interrupt Pas de routine de service disponible (interruption)
$0A	VCB unusable La structure de la table VCB est endommagée
$08	FCB unusable La structure de la table FCB est endommagée
$0C	Block zero allocated illegal Le bloc zéro a été alloué par erreur
$0D	Interrupt occured while I/O shadowing off Erreur rencontrée lors du processus shadowing

CONSEILS DE LECTURE

Pour maîtriser le système de base de Apple *II*GS

• **Boîte à outils de l'Apple *II*GS**
Jean-Pierre Curcio (Editions du P.S.I.)

Cet ouvrage s'adresse à tous ceux qui souhaitent programmer des applications sur le *II*GS : ce n'est pas un cours sur l'assembleur ou le C, mais une étude de la boîte à outils intégrée au système. Deux exemples concrets, un miniPaint et un programme d'interface illustrent l'utilisation des différents managers.

• **Clefs pour Apple *II*GS** – 2ème édition
Nicole Bréaud-Pouliquen (Editions du P.S.I.)

Ce mémento adresse aux programmeurs en assembleur, C et Basic de l'Apple *II*GS : il offre une syntèse des spécificités du matériel et des logiciels de développement : architecture interne, brochage, jeu d'instructions du 65816, mémoires, ressources graphiques, entrées/sorties. Le système APW est décrit en détail; l'ensemble des outils du bureau est répertorié, fonction par fonction.

A paraître :

• **Programmation Système de l'Apple *II***
Marcel Cottini (Editions du P.S.I.)

Destiné aux programmeurs chevronnés sur Apple *II*e et *II*c, il est le livre indispensable, contribuant à acquérir une bonne maîtrise avant de s'attaquer au GS. En effet, il contient les outils nécessaires à la bonne compréhension des mécanismes rencontrés en mode émulation sur l'Apple *II*GS.

• **Système ProDOS de l'Apple *II***
Marcel Cottini (Editions du P.S.I.)

Pour programmeurs avertis, cet ouvrage présente l'organisation complète du système ProDOS8 qui est le système d'exploitation des applications destiné à tourner avec les microprocesseurs 6502 et 65c02 (Apple *II*e et *II*c). C'est le complément du système d'exploitation ProDOS16.

• **L'assembleur de l'Apple *II*GS**
Nicole Bréaud-Pouliquen (Editions du P.S.I.)

Une initiation claire et complète à l'assembleur 65816, illustré par de nombreux programmes-exemples.

Votre avis nous intéresse

Pour nous permettre de faire de meilleurs livres, adressez-nous vos critiques sur le présent ouvrage.

— *Titre de votre livre :* .....

— *Ce livre vous donne-t-il toute satisfaction ?*

.....

.....

.....

— *Y a-t-il un aspect du problème que vous auriez aimé voir abordé ?*

.....

.....

.....

Si vous souhaitez des éclaircissements techniques, écrivez-nous, ou laissez un message à notre service Minitel (3615 OI PSI), nous ne manquerons pas de vous répondre directement.

Avez-vous déjà acquis des livres P.S.I. ?

lesquels ? .....

.....

qu'en pensez-vous ? .....

.....

.....

Nom ..... **Prénom** ..... **Age** .....

Adresse .....

Profession .....

Centre d'intérêt .....

Je désire recevoir un catalogue gratuit et être informé régulièrement des nouveaux livres publiés.

☐ oui ☐ non

Veuillez renvoyer cette page, dûment remplie, aux

Editions PSI
5, place du Colonel Fabien
75491 PARIS CEDEX 10