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

Autres ouvrages relatifs à l'Apple II

Maîtrise du Basic

- l'Apple et ses fichiers - Jacques Boisgontier
- Basic plus, 80 routines sur Apple II - Michel Martin
- Programmation système de l'Apple II - Marcel Cottini

Assembleur

- Introduction à Prodos sur Apple II - Francis Versheure
- Clefs pour Apple II - Nicole Bréaud-Pouliquen
- Clefs pour Apple IIe 65C02 et Apple IIc - Nicole Bréaud-Pouliquen
- Assembleur de l'Apple II - Nicole Bréaud-Pouliquen et Daniel-Jean David

Ouvrages contenant des programmes en assembleur sous D.O.S. 3.3 ou ProDOS

- La programmation des jeux d'arcades sur Apple II - Jean-Luc Fischer
- Apple, modems et serveurs - Alain Mariatte
- Création et animations graphiques sur Apple II - Gilles Fouchard et Jean-Yves Corre (livre + disquette)

Pour tout problème rencontré dans les ouvrages P.S.I.
vous pouvez nous contacter au numéro ci-dessous :

Numéro Vert/Appel Gratuit en France

05 21 22 01

(Composer tous les chiffres, même en région parisienne)

ISBN: 2-86595-312-2

SYSTEME PRODOS DE L'APPLE II

MARCEL COTTINI

ÉDITIONS DU P.S.I.
1986

SOMMAIRE

PREFACE

ProDOS est le nouveau système d'exploitation de type professionnel, adopté définitivement par Apple Cupertino et présenté au monde des programmeurs comme l'avenir de la gestion des fichiers à grande capacité.

C'est un système d'exploitation musclé, écrit et mis au point par la société Apple Computer. ProDOS signifie dans la langue d'outre-Manche : Professional Disk Operating System. Ce système fut initialement élaboré et mis au point pour du matériel professionnel et permet de gérer un disque dur du type Profile, d'une capacité maximale de 32 mégaoctets. On trouve dans le passé un programme système appelé SOS, qui est le système d'exploitation de l'Apple III, permettant par ailleurs la gestion d'autres périphériques, comme l'imprimante par exemple. Les deux systèmes - PRODOS et SOS- permettent en outre une certaine compatibilité entre la famille Apple II et Apple III, puisqu'ils possèdent tous deux la même structure pour le traitement des fichiers.

Il semble que la société Apple ait voulu marquer un coup, et tourner ainsi définitivement la page concernant le langage "Integer", dont la structure est totalement rejetée par ProDOS. Dans une circulaire adressée aux programmeurs, Apple leur conseille fortement d'axer l'écriture et la conception de leurs nouvelles créations sur ce nouveau système d'exploitation.

Certes, le DOS 3.3 n'est pas encore mort, mais son blason a fini de briller et son heure de gloire s'estompe peu à peu. Il faut admettre cependant qu'il a fait couler beaucoup d'encre et nombre d'amateurs en herbe auront fouillé ses entrailles à la recherche d'une adresse ou d'une astuce oubliée. Mais, comme on dit, il faut suivre le progrès, et ce progrès-là s'appelle ProDOS.

Ce système permet, contrairement au DOS 3.3, la gestion d'un lecteur de disque rigide du type Profile, et, en même temps, la récupération de fichiers de l'ancien système, grâce à un utilitaire de conversion.

ProDOS est opérationnel sur la série des Apple II avec l'AppleSoft en mémoire morte (ROM) et 64 Ko de mémoire vive (RAM).

A partir de la sortie de l'Apple IIc et du nouveau IIe, ProDOS est livré comme le système d'exploitation standard. Pour l'achat d'un Apple IIc, ce sont six disquettes qui vous sont fournies gratuitement ; Apple fait des frais !

Ceux qui par le passé ont programmés sous DOS 3.3 se feront un plaisir d'adopter le nouveau système d'exploitation ProDOS. Ces deux systèmes permettent, par ailleurs, une certaine compatibilité entre programmes. Cependant, les adresses des POKE et PEEK sont à revoir, et même à proscrire, dans la majorité des cas.

Par contre, les adeptes du langage machine devront revoir totalement leur méthode de programmation, le principe ayant radicalement changé. L'espace mémoire d'implantation des deux systèmes est totalement différent, et ProDOS n'accepte plus les appels de l'ancien DOS - JSR ou JMP aux adresses du DOS 3.3.

"La maîtrise de ProDOS sur Apple II" est un livre d'aide aux programmeurs, il brosse un tableau complet de la structure des systèmes BASIC et PRODOS. C'est un outil de travail pour effectuer la transition d'un système à l'autre, avec tous les atouts de votre côté. Un dictionnaire des mots usuels attribués à DOS 3.3 et ProDOS complète l'ouvrage.

Le chapitre 1 donne une vue d'ensemble des commandes : nouvelles, modifiées, améliorées ou disparues. Les divers tampons alloués à ProDOS sont commentés : File Control Block, Volume Control Block, Volume Bit-Map, etc. Le processus de boot est détaillé dans ses moindres aspects.

Le chapitre 2 traite plus spécialement la syntaxe des commandes relatives aux fichiers, en particulier les fichiers texte.

Le chapitre 3 traite l'organisation interne et l'environnement sous ProDOS. Toute la face cachée est passée en revue: octets de contrôle lors du "boot", espace mémoire utile, adresses réservées, la routine Garbage Collection, les vecteurs de la page 3, la Mystery routine, listings commentés de la page globale de PRODOS et du Basic System, explications sur la routine Drivers - RWTB - etc.

Les chapitres 4 et 5 commentent plus spécialement les commandes MLI avec leurs paramètres : un sujet vaste, traité dans le moindre détail.

Le chapitre 6 détaille l'organisation d'une disquette ProDOS, et donne la structure d'une entrée Volume-Directory, Volume-Subdirectory et fichier programme.

Le chapitre 7 traite les interruptions du microprocesseur, avec le moyen de les récupérer: NMI, IRQ et RESET. Des listings complets et commentés sont dévoilés pour la première fois : routines de traitement des interruptions - IRQ-Handler ; routine GET.TIME ; etc.

Les annexes contiennent une foule de renseignements pour le programmeur : tables des codes d'erreurs du système Basic et du MLI, table des fichiers redéfinissables, les paramètres optionnels, les commandes ProDOS, un programme de gestion de la date ProDOS, etc.

CHAPITRE 1

FAMILIARISATION AVEC LE SYSTEME D'EXPLOITATION ProDOS

APPROCHE DE ProDOS

Evolution des systèmes dans le temps

DOS 3	juin	1978
DOS 3.1	juillet	1978
DOS 3.2	février	1979
DOS 3.2.1	juillet	1979
DOS 3.3	juillet	1980
ProDOS 1.0	janvier	1983
ProDOS 1.0.1	janvier	1984
ProDOS 1.0.2	février	1984

Des versions de test circulent depuis le début de l'année 1983, mais la version officielle n'a fait son apparition que vers 1984.

Le lecteur de disquettes

Le lecteur de disquettes est un maillon essentiel de la chaîne d'un micro-ordinateur, et permet une sauvegarde et une recherche rapides de données sur un support magnétique. Il en existe trois catégories :

- le disque dur du type Profile, pour une exploitation professionnelle ;

- le lecteur de disquettes du type Disk II, qui utilise comme support des disquettes souples, au format 5 pouces 1/4. C'est de loin le plus répandu dans le milieu informatique grand public. D'un prix relativement abordable, avec une capacité formatée de 140 Ko, c'est le modèle standard de la société Apple ;

- un nouveau lecteur de disquettes vient de faire son apparition sous le nom de Unidisk. Il s'agit d'une unité de disquettes de 3 pouces 1/2, qui est déjà le format du Macintosh. La capacité formatée est de 800 Ko et plusieurs de ces Unidisk peuvent être connectés à un Apple II. Il est commercialisé avec un utilitaire offrant un affichage avec icônes pilotées par la souris.

C'est le modèle Disk II qui sera retenu pour nos futurs commentaires.
Un lecteur de disquettes est un périphérique qui peut lire et écrire des informations codées sur un support magnétique. Le principe en est très simple : un disque tourne rapidement en face d'une tête de lecture/ écriture, qui elle-même peut se déplacer dans le sens transversal de la disquette. Néanmoins la tête de lecture est limitée dans sa course par la surface efficace du support magnétique. Les informations sauvegardées peuvent ainsi être modifiées, effacées ou réenregistrées à volonté, et la fonction du lecteur de disquettes est identique à celle d'un lecteur de cassettes.

La disquette est un support circulaire en matière plastique souple, recouvert de particules magnétiques qui seront sensibilisées quand la tête de lecture passera au-dessus d'elles. Le format standard pour le modèle du Disk II est de 5 pouces 1/4. Pour être protégée mécaniquement, la disquette est placée dans une enveloppe qui laisse apparaître une fenêtre de forme ovale. Lors des opérations d'écriture et de lecture, le lecteur fait tourner le disque à l'intérieur de son enveloppe de protection, et lit ou écrit les informations au niveau de la fenêtre.

Les avantages essentiels d'un lecteur de disquettes sont sa souplesse et sa facilité d'emploi. L'accès aux informations est très rapide, tandis que la cassette oblige l'utilisateur à suivre le déroulement séquentiel de la bande, c'est-à-dire à commencer au début et terminer à la fin.

C'est le système d'exploitation qui gère le ou les lecteurs; il se trouve la plupart du temps enregistré sur la disquette même, cette dernière prend alors le nom de disquette système. Elle contient les informations nécessaires pour permettre une gestion efficace du lecteur par le micro-ordinateur.

Système d'exploitation ProDOS

Un système d'exploitation est un ensemble de petits programmes 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 systèmes (Operating System) et permettant de gérer un lecteur de disquettes à l'aide de commandes évoluées. Lors des 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 la disquette. Ce rôle est confié au système d'exploitation. Ces définitions se rapportent plus particulièrement aux DOS de l'ancienne génération (DOS 3.3 pour ne citer que lui).

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 rigide, Profile. Très vite on s'est rendu compte que ce système offrait

de nombreuses possibilités, et le pas a vite été franchi d'une adaptation à la série des Apple II. C'est ainsi qu'est né ProDOS, avec ses avantages, mais aussi ses inconvénients, dont nous verrons le détail au fur et à mesure.

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. Pour PRODOS, la mémoire se trouve amputée de 12 Ko ; l'Apple IIe ou IIc répartit d'une façon astucieuse ; cette occupation : les 12 Ko en haut de la mémoire sont attribués à PRODOS par commutation alternée du système et de la ROM. Cet espace est normalement occupé par la ROM AppleSoft et par le Moniteur.
En réalité, le fichier PRODOS occupe 16 Ko, dont 4 Ko sont obtenus par la commutation de la Bank Switched Memory - bank 1 et 2.

ProDOS, développé par la société Apple Computer, est le nouveau système standard de la société Apple, pour les micros de la série Apple II. Il a fait son apparition officielle début 1984 avec la version 1.0; une nouvelle version, revue et corrigée, circule depuis début 1985. J'ai pu tester les versions 1.0 ; 1.0.1 et 1.0.2, qui apportent chacune une amélioration par rapport à la précédente. Malheureusement, il existe encore un certain nombre de "bugs" dans la dernière version, mais les points d'entrées du MLI semblent être définitivement adoptés.

La dénomination ProDOS recouvre en fait l'association d'un système d'exploitation et d'un programme interface.

ProDOS se compose :

- d'un fichier PRODOS qui se loge dans la carte langage, et regroupe tous les sous-programmes en langage machine - MLI -. Il se trouve sauvegardé sous la forme d'un programme du type SYS, avec comme nom de fichier: PRODOS ou KERNEL ;

- d'un fichier également du type SYS, sous la forme XXX.SYSTEM, et qui se loge à l'ancien emplacement du DOS 3.3.

Chaque disquette de démarrage standard ProDOS comporte au moins deux fichiers communs, écrits en langage machine : PRODOS et XXX.SYSTEM.

Remarque : XXX.SYSTEM représente un format, dont 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 général, c'est BASIC.SYSTEM que vous rencontrerez sur la plupart des disquettes conventionnelles.

FICHIER PRODOS

C'est un fichier écrit en langage machine, qui 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 de la bank 2), 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.

PRODOS fut écrit au départ pour fonctionner sur un Apple II de 48 Ko de mémoire, en se plaçant sous l'adresse $C000, et sur un IIe de 64 Ko, en se plaçant sur la carte langage. Cette dernière version a été retenue pour la commercialisation. Elle se trouve sur la disquette sous la forme d'un fichier du type SYS, qui ne peut être que "booté". PRODOS se loge sur les 16 Ko de la carte langage d'un Apple II+ ou en haut de la mémoire sur un IIe et un IIc, et fixe HIMEM à $BF00.

Lorsque PRODOS est chargé en mémoire, les commandes exécutées sont des commandes PRODOS. Lorsque celui-ci reçoit une commande qu'il ne peut pas interpréter, il la transmet à l'AppleSoft qui teste sa validité avant de l'utiliser. Si, pour une raison quelconque, l'ordre n'aboutit pas, un message d'erreur sera renvoyé par le système Basic.

FICHIER BASIC.SYSTEM

C'est un fichier écrit en langage machine, qui est l'interface entre un programme AppleSoft et le système ProDOS. Il permet de passer la main à Basic pour exécuter des commandes adressées au langage AppleSoft. Il est logé 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 : la programmation sous ProDOS est sensiblement la même qu'avec le DOS 3.3, mais la conversion des fichiers programmes peut poser certains problèmes, si l'on ne tient pas compte de leur différence fondamentale. En effet la syntaxe n'est pas toujours la même, et certaines restrictions sont à observer.

Commandes Basic avec ProDOS

- Le caractère CTRL-D doit être le premier caractère de la ligne. Il n'est plus possible de placer plusieurs commandes PRODOS sur une seule ligne. En effet, PRODOS contrôle chaque instruction à travers la routine TRACE lorsqu'il rencontre un PRINT, et vérifie si le premier caractère est un CTRL-D. Il est impératif de ne plus faire précéder un CTRL-D par un retour-chariot (CR) en définissant : D$ = CHR$(13) + CHR$(4).

- Lorsque l'on recopie les caractères d'une ligne par l'intermédiaire de la flèche droite, les caractères de contrôle ne sont pas reconnus par PRODOS.

- Il est conseillé d'utiliser avec prudence l'instruction HIMEM, car au moment de l'ouverture d'un fichier, PRODOS alloue 1024 octets pour la zone tampon de chacun de ses fichiers. Par ailleurs, si HIMEM est déclaré, il faut le faire par multiples de 256, car la mémoire est gérée par pages de 256 octets.

- Les instructions IN# et PR# en mode immédiat, ou dans un programme, seront précédées par un CTRL-D pour des commandes PRODOS, sinon ces commandes seront purement et simplement ignorées.

- TRACE et NOTRACE fonctionnent correctement avec AppleSoft et PRODOS.

- PRODOS s'occupe de libérer la place occupée par un programme AppleSoft lors de l'utilisation des instructions HGR ou HGR2. De même, il rend à nouveau cette zone opérationnelle lorsque TEXT est déclaré.

- Les PEEK et POKE du DOS 3.3 ne fonctionnent plus sous PRODOS, étant donné que ces adresses ne correspondent plus avec le nouveau système. En règle générale, il faudra éliminer tous les POKE et PEEK se rapportant à des adresses spécifiques de l'ancien système d'exploitation, lors de la conversion des programmes d'un système vers un autre.

LES COMMANDES DE ProDOS

Les commandes disparues

Six commandes du DOS 3.3 ont disparu :

- INIT, qui permettait d'initialiser une disquette Slave. Dorénavant, il faudra charger et exécuter un programme de formatage, ou encore utiliser la disquette utilitaire comportant le fichier FILER.

- Les instructions INT et FP ne sont plus disponibles, et il semble que le langage Integer soit définitivement banni du catalogue de la société Apple.

- MAXFILES, qui servait à déclarer et à ouvrir un nouveau fichier, est géré automatiquement par ProDOS, et celui-ci tolère huit fichiers ouverts simultanément.

- Les commandes MON et NOMON sont totalement ignorées.

Les commandes nouvelles ou modifiées

Ce sont surtout des instructions permettant de manipuler le catalogue ou les fichiers de la disquette.

Il est à retenir que :

- CATALOG garde son sens habituel et affiche les fichiers qui dépendent du chemin (Pathname) ou préfixe (Prefix) utilisé. Il y figure notamment la date et l'heure de la dernière modification, la capacité libre et occupée du Volume : les tailles sont exprimées en blocs. Chaque bloc correspond à deux secteurs, ou encore à 512 octets. La commande CATALOG affiche une version plus complète du contenu du catalogue sur 80 colonnes.

- CAT fournit une version plus sobre du contenu de la disquette - catalogue -, et l'affichage se fait uniquement sur 40 colonnes.

- PREFIX permet de définir ou de modifier le préfixe qu'il faudra ajouter avant de désigner un nom de fichier. Il peut être redéfini, et trouve son avantage lorsqu'une configuration se compose de plusieurs lecteurs de disquettes. La commande PREFIX évite de fournir à chaque fois le nom complet du chemin, pour la recherche d'un fichier quelconque. Il suffira d'indiquer par cette instruction la partie que PRODOS devra ajouter automatiquement avant le nom du fichier.

- CREATE permet de créer une nouvelle table des matières - Subdirectory -, ou un nouveau fichier programme.

- Le Dash, représenté par le signe -, permet de lancer un programme, et peut remplacer avantageusement les instructions : RUN, BRUN ou EXEC.

- FLUSH est une instruction similaire à CLOSE, qui permet de vider la zone tampon du fichier actuel, et d'effectuer une sauvegarde de ces données sur la disquette.

- FRE est une commande de nettoyage mémoire beaucoup plus rapide que celle d'AppleSoft ; néanmoins, elle semble absente dans les versions récentes.

- STORE et RESTORE permettent respectivement de sauvegarder sur disque, puis de rappeler en mémoire centrale, les variables d'un programme AppleSoft.

- Enfin, ProDOS permet l'adjonction de nouvelles commandes à son répertoire, à l'inverse du DOS 3.3 qui nécessitait souvent des prouesses de programmation par l'ajout de "patchs".

Les commandes améliorées

Les autres instructions sont semblables à celles du DOS 3.3, mais comportent néanmoins quelques améliorations : il devient ainsi possible de sauvegarder un programme binaire en précisant son adresse de fin, ou de charger en mémoire centrale un programme AppleSoft, puis de l'exécuter à partir d'une ligne donnée, etc.

Exemples :

BSAVE IMAGE, A$2000, E$3FFF sauvegarde un fichier binaire du nom IMAGE, à l'adresse $2000 (page 1), dont l'adresse de fin est $3FFF.

RUN FICHIER à 500 charge un programme du type AppleSoft en mémoire centrale, et l'exécute à partir de la ligne 500. Si cette ligne n'existe pas, alors la ligne suivante sera prise en compte.

Les instructions APPEND et CHAIN fonctionnent correctement, tandis que IN# et PR# peuvent préciser l'adresse d'une routine d'entrée ou de sortie.

CHAPITRE 2

ORGANISATION D'UNE DISQUETTE SOUS ProDOS

ProDOS, stocké sur un support disquette, n'occupe pas de piste précise. Il est sauvegardé sous la forme de deux fichiers du type SYS, uniquement exécutables, et possède toute une gamme de commandes nouvelles.

Fichiers ProDOS :

- un fichier système Basic, généralement sauvegardé sous le nom de BASIC.SYSTEM : il sert d'interface au système ProDOS ;
- un fichier système ProDOS, généralement sauvegardé sous le nom de ProDOS ou KERNEL : il contient les routines MLI écrites en langage machine.

Le vocabulaire s'est considérablement enrichi. Reportez-vous au dictionnaire des mots usuels placé à la fin de cet ouvrage, pour vous aider à mieux comprendre certaines fonctions ou commandes nouvelles.

Des instructions et des commandes ont disparu, d'autres ont été améliorées ou, tout simplement, rajoutées. Il faudra dès le départ adopter une règle très stricte: faire abstraction de tous les termes propres à l'ancien système d'exploitation : Bit-Map, Track Sector List, IOB, etc., et se méfier des analogies trop rapides entre les deux systèmes.

Une configuration standard avec ses maillons essentiels est décrite dans ce chapitre, et devrait permettre de bien situer ProDOS parmi les périphériques en ligne.

Remarque : certains programmeurs voudront "patcher" le fichier ProDOS : il existe trois façons de le faire :

- après chargement vers son emplacement final, il sera modifié ;
- ProDOS sera chargé comme fichier binaire à l'adresse $2000. Il restera à faire les modifications voulues, puis par un Move, à le placer dans la carte langage ;
- ProDOS peut aussi se modifier sur la disquette, à l'aide d'un utilitaire - Diskfixer, par exemple.

COMPOSITION D'UNE CONFIGURATION TYPE

Configuration de base

Le micro-ordinateur : c'est le maillon essentiel d'un ensemble informatique : il intègre le microprocesseur, ayant comme rôle essentiel le traitement des informations qui lui sont soumises. Le processeur reçoit un certain nombre de données, qui sont des valeurs codées suivant des normes définies au préalable. Il existe deux sortes de langages: le langage évolué, qui utilise des instructions synthétiques et très puissantes, Basic AppleSoft par exemple, et le langage de bas niveau, qui possède des instructions très élémentaires, langage machine. Le microprocesseur équipant l'Apple IIc (65C02) traite des mots de 8 bits de longueur. Le bit étant le plus petit élément que la machine peut traiter, avec comme valeur 1 ou 0. Un octet ou byte est une donnée codée sur 8 bits.

Le clavier se trouve intégré dans la gamme des Apple II et comporte des touches nécessaires pour entamer le dialogue avec les différents périphériques en ligne.

L'écran vidéo : organe de visualisation, souvent appelé Moniteur vidéo. C'est un périphérique qui affiche en clair les données venant soit d'un clavier, soit d'un lecteur de disquettes.

Le lecteur de disquettes : il lit et écrit des données sur un support magnétique, pour effectuer une sauvegarde de mémoire de masse. Avec la génération des Apple II, c'est le lecteur du type Disk II qui est utilisé, avec des disquettes souples de 5 pouces 1/4.

L'imprimante : imprime sur papier les informations envoyées par le micro-ordinateur. Plusieurs types et modèles existent: série, parallèle, à marguerite, à marteau, etc.

L'interface : c'est un intermédiaire entre divers composants d'un système informatique ou d'une configuration donnée. L'Apple IIc possède un certain nombre d'interfaces internes, destinées à raccorder un périphérique: souris ou deuxième lecteur de disquettes.

Le périphérique : c'est en général un accessoire développé par Apple et qui étend les possibilités de l'unité centrale.

LE ROLE DU SYSTEME D'EXPLOITATION

Définition

Le but d'un ensemble informatique est de permettre de traiter des informations à partir d'une mémoire centrale - RAM - à l'aide de sous-programmes résidents - ROM - ou à partir d'une mémoire de masse disponible par un lecteur de disquettes. 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 nouveau système d'exploitation, permettant de gérer des fichiers à partir de lecteurs de disquettes ou d'un disque dur du type Profile. Il a été mis au point par Apple Computer. Les unités de lecteurs sont utilisées comme mémoire auxiliaire.

Le lecteur de disquettes

Fonctions élémentaires du lecteur :

- mettre le disque ou la disquette en rotation ;
- déplacer la tête de lecture/ écriture au-dessus de la disquette ;
- envoyer des signaux déchiffrables par le système et le lecteur ;
- lire des données sur la disquette ;
- écrire des données sur la disquette ;
- faire des choix ;
- gérer des ordres et des erreurs.

A un niveau plus élévé, ProDOS doit pouvoir classer les fichiers dans un certain ordre sur la disquette 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 systèmes, permettant une gestion de la disquette 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 d'une carte interface: c'est une carte contrôleur, conçue pour commander deux drives. La première unité est mise en place dans le slot 6. Cette disposition est rendue nécessaire par un critère purement logiciel, car certains programmes recherchent le slot 6 comme périphérique en ligne.

Dans l'Apple IIc, le lecteur intégré est reconnu par le système comme étant raccordé au slot 6, la ROM de l'interface occupe les adresses de $C600 à $C6FF. Pour utiliser ProDOS, il faut d'abord le placer ou le charger en mémoire. Cette opération est effectuée par le boot de la disquette.

La disquette

Boot d'une disquette :

- Placer la disquette Utilitaire Système dans l'unité 1, branchée au connecteur 6, par exemple.

- Taper à partir du clavier : IN#6 ou PR#6.

- Le sous-programme, implanté à l'adresse $C600 et logé dans la ROM de l'interface du lecteur, est exécuté.

- Le chargement de ProDOS s'effectue à partir de la disquette. 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 0, secteurs 0, 1, 2 et 3. A cet emplacement se trouve le Loader, qui a comme rôle de reconnaître la configuration, puis de charger les systèmes appropriés. Le Loader est d'abord placé à l'adresse $0801, puis exécuté. Si c'est un Apple II, les fichiers PRODOS et BASIC.SYSTEM sont recherchés sur la disquette, puis chargés successivement à leur emplacement mémoire. Certains programmes qui se trouvent alors en mémoire peuvent être endommagés, car les fichiers ProDOS utilisent plusieurs zones de la mémoire avant de se loger à leurs adresses définitives.

- Le chapitre 2, sous la rubrique Processus de boot d'une disquette ProDOS, donne toutes les instructions nécessaires.

LE LECTEUR DE DISQUETTES

Fonctions du drive

Le lecteur de disquettes, encore appelé drive, est un périphérique destiné à lire et à écrire des informations sur une surface magnétique appelée disquette.

Cet appareil électromagnétique permet :

- d'écrire des informations sur une surface magnétique traitée spécialement à cet effet ;
- d'effacer des données physiques au niveau de la disquette ;
- de stocker, dans un temps donné, des informations diverses ;
- de rechercher des données de façon sélective, en les désignant par un nom ;
- de lire des informations sur ledit support magnétique.

Fonctions et organisation de la disquette

Une disquette est un support magnétique, permettant de stocker en permanence des informations. Dans la version standard de l'Apple, les disquettes sont d'un format de 5 pouces 1/4. Mais récemment, une nouvelle génération de drives a fait son apparition avec un format de 3 pouces 1/2 et une capacité formatée de 800 Ko. Nous traiterons uniquement le modèle Disk II, format 5 pouces 1/4, qui est de loin le plus connu.

La disquette est le type de support le plus populaire. Elle se présente sous la forme d'un disque circulaire, en matière plastique vinyle, enveloppé d'un plastique rigide. Cette enveloppe rigide a pour fonction de protéger le disque pendant sa manipulation. Le disque peut tourner librement dans l'enveloppe qui contient un produit antiabrasif, incorporé en usine au moment de sa fabrication. La surface d'une disquette se divise en pistes et secteurs.

PISTES

Les pistes, encore appelées tracks, facilitent la recherche des données sur le support. Pour accéder à un quelconque octet de la disquette, ProDOS, au moment du formatage, divise la surface en 35 pistes physiques, formant un ensemble de 280 blocs logiques. Ces pistes ne sont pas des sillons visibles, elles sont matérialisées par des marques représentant des informations codées. Elles sont concentriques et la piste intérieure a un diamètre inférieur à la piste extérieure. La piste 0 se trouve à l'extérieur de la disquette, la piste 34 est celle qui se trouve le plus près du centre. Chaque piste peut être lue séparément par la tête de lecture. Celle-ci se déplace concentriquement et peut occuper 70 positions, c'est-à-dire qu'elle peut lire 35 positions de pistes et 35 positions intermédiaires. Ces positions intermédiaires sont appelées des demi-pistes, et sont localisées entre chaque piste. La tête de lecture est entraînée par un moteur spécial, appelé moteur pas-à-pas - le moteur du Disk II a un pas de 70.

SECTEURS

Les secteurs, encore appelés sectors, permettent la division d'un support disque en une partie plus petite que la piste. ProDOS divise chaque piste en 16 secteurs, et chaque secteur contient 256 octets. Néanmoins, la notion de piste et secteur est remplacée par la notion de bloc. Un bloc logique est formé par 2 secteurs physiques. Un bloc est égal à 512 bytes. Une disquette a une capacité de 280 blocs. L'accès à un octet, par les commandes READ.BLOCK ou WRITE.BLOCK, est plus rapide que par le passé. Cela est le résultat combiné d'une meilleure gestion des secteurs - Skew - et du traitement des données par une structure de blocs - conversion interne au système. Chaque bloc peut être lu séparément. Le format actuel du microprocesseur, 6502 ou 65C02, de la série des Apple II est de 8 bits par byte, chaque bloc a une capacité de 512 bytes, on peut donc stocker 512 X 8 = 4 096 bits dans un bloc.

Le Disk II est le lecteur de disquettes standard d'Apple ; nous le prendrons comme référence dans nos commentaires. Il possède une capacité de stockage avant formatage de 140 000 caractères environ ; après formatage, cette valeur tombe à 120.000 caractères, ce qui représente environ 40 pages de texte au format 21 X 27.

LE FORMATAGE D'UNE DISQUETTE

Généralités sur le formatage

Avant de pouvoir utiliser une disquette vierge, il est nécessaire de la formater. Formater, ou encore initialiser, une disquette consiste à diviser la surface magnétique de la disquette en sections, secteurs, où l'information peut être stockée et lue. Les disquettes en vente dans le commerce ne sont pas formatées d'origine, parce que chaque type d'ordinateur utilise son propre format. Sous DOS 3.3, cette opération est effectuée par une routine interne, appelée par la commande INIT. ProDOS ne permet plus l'utilisation de cette commodité et nécessite le programme FILER pour formater un Volume. Il est aussi possible d'étendre les possibilités de ProDOS, en lui associant une commande externe qui appellera, par exemple, un programme de formatage.

L'utilitaire FILER, fourni par Apple, introduit certaines restrictions : il interdit l'initialisation d'une disquette pendant l'exécution d'un programme, puisqu'il faut charger le programme Filer, puis, après utilisation, recharger le programme BASIC.SYSTEM comme interpréteur. Le Filer est un ensemble de formatage, relativement long à charger et présente un grand défaut : il est très dépendant du réglage de la vitesse des lecteurs de disquettes. Là où le DOS 3.3 effectuait encore l'initialisation, le FILER peut retourner un message d'erreur, estimant que la vitesse de rotation du lecteur est trop grande par exemple. Il existe dans le commerce des utilitaires qui contournent cet obstacle, en tenant compte des deux modes de formatage : physique et logique.

ProDOS, comme le DOS 3.3, utilise le même formatage physique 16 secteurs, mais par contre, le formatage logique est propre à chaque système, ce qui nécessite une grande prudence avant toute tentative d'incursion à ce niveau.

Divers modes de formatage

FORMATAGE PHYSIQUE

Le formatage, ou initialisation, est une phase relativemant complexe. Une disquette du standard Apple Disk II se divise en 35 pistes et la tête de lecture peut se mouvoir au-dessus de chacune d'elles. Chaque lecteur de disquettes est réglé sur une vitesse de rotation de 300 tours/minute, mais cette vitesse n'est en somme que relative. Il est évident que le nombre de tours/minute (RPM) diffère sensiblement d'un lecteur à l'autre, pour accumuler en certaines circonstances des écarts de vitesse assez importants. Il est très fréquent de trouver sur le marché des disquettes formatées avec des écarts variant de + ou - 5 %. C'est à cet effet que des méthodes d'encryptage/ décryptage ont été mises au point, afin de pouvoir placer les 33 000 bits (en réalité environ 50 000 bits, dus aux nibbles bruts) d'une piste, sans erreur de codage.

A l'origine, la disquette d'un Disk II avait une capacité de 80 Ko (DOS 3.2) pour le format 5 pouces 1/4, car les bits étaient enregistrés par la méthode de synchronisation. Ensuite, ce fut le temps du renouveau, avec l'introduction sur le marché du DOS 3.3, qui permit de porter la capacité de formatage à 140 Ko, avec un découpage de piste en 16 secteurs. Dès lors, certaines contraintes, insurmontables au départ, purent être contournées à l'aide de logiciels.

Règles fondamentales du formatage physique :

- Un octet ne peut être lu ou écrit que s'il respecte un certain nombre de paramètres : bit fort à 1, pas plus de 2 bits nuls consécutifs ; un rapide calcul fait alors apparaître la nécessité de disposer de 342 nibbles pour coder 256 bytes.

- Chaque secteur d'une piste se divise en une zone de données et une zone d'adresses.

- La zone adresses comporte un préfixe, marque D5 AA 96, et des informations sur le numéro de Volume, ignoré par ProDOS, de piste et de secteur de la disquette. Par ailleurs, le système effectue un Checksum ou comptage des nibbles du champ d'adresses. Un suffixe clôture le tout, marque DE AA EB. Chaque contrôle du champ d'adresses sera codé sur 2 nibbles.

- La zone des données se compose : d'un préfixe, D5 AA AD, les 342 nibbles représentant les 256 octets de données du secteur, d'un Checksum ou comptage des nibbles du champ des données, codé sur un nibble et enfin d'un suffixe - DE AA EB.

- Chaque zone est précédée par un certain nombre de bits de synchronisation qui sont codés d'une façon très spéciale : c'est une forme de bytes synchro à dix bits, dont les huit premiers sont à 1 et les deux suivants à 0. Ils sont là pour synchroniser sur le premier bit d'un octet, et non pas sur une position quelconque dans un octet.

Représentation d'un byte de synchro :

```
Bits :  9 8 7 6 5 4 3 2 1 0
        1 1 1 1 1 1 1 1 0 0
                              = #$FF
                              Bits 9-2 à 1
                              Bits 1-0 à 0
```

Les bits 1-0 permettent la division correcte des octets parmi les milliers de bits qui se présentent à chaque lecture. Ces bits à 00 disparaissent lorsque les données sont implantées en mémoire centrale; elles sont filtrées par ProDOS.

Exemple :

Voici une suite de nibbles qui pourrait s'afficher sur votre écran, par l'intermédiaire d'un utilitaire du genre Diskfixer ou autre. Cette opération de lecture a été effectuée avec une disquette formatée sous DOS 3.3, dont le contenu et la valeur n'ont aucune importance.

```
41A0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF D5 AA 96
41B0- FF FE AA AB AF AE FA FB DE AA EB AC F7 FF FF FF
```

La zone adresses - D5 AA 96 - est précédée par une série de huit à quatorze FF pour avertir le système de la venue d'un flot de données.
Le Prologue est le marqueur de tête du champ d'adresses, dont le but est de signaler à ProDOS que le champ d'adresses commence, pour que débute l'analyse des nibbles.

Le formatage d'une disquette dépend directement du nombre de bytes de synchronisation. Pour adapter la lecture ou l'écriture des données sur une disquette, il suffit de mettre en place un certain nombre de bytes synchro entre deux secteurs, pour que tout devienne cohérent.

Nibbles de synchronisation

• Zone des adresses :

D5 AA 96	Address Field Header. Préfixe de la zone : à partir d'ici débute un champ d'adresses.
xx yy	Volume Number. Numéro de Volume de la disquette : 2 nibbles. Cette donnée est ignorée par ProDOS.
xx yy	Track Number. Numéro de la piste : 2 nibbles. Cette adresse de champ se trouve sur la piste xx.
xx yy	Sector Number. Numéro du secteur : 2 nibbles. Le secteur qui suit a le numéro yy.
zz zz	Checksum. Comptage du champ d'adresses : 2 nibbles. Compte les nibbles du champ d'adresses des (données).
DE AA EB	Address File Trailer. Suffixe de la zone : fin de la zone du champ d'adresses.

• Zone des données :

D5 AA AD	Data Field Header. Préfixe de la zone : à partir d'ici commence un champ de données.
zz	Checksum. Comptage du champ des données : 1 nibble.
DE AA EB	Data Field Trailer. Suffixe de la zone : à partir d'ici se termine le champ de données.

Remarque : 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.

Les routines READ.BLOCK et WRITE.BLOCK sont chargées de lire et d'écrire les données sur les 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 qui 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 bruts, pour laisser paraître les seuls bytes réels.

FORMATAGE LOGIQUE

C'est l'organisation interne des pistes et secteurs, 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 la disquette. Un bloc est une unité de mesure, spécifique au système d'exploitation ProDOS, et regroupe 2 secteurs physiques de la disquette.

Organisation de ProDOS :

- Mise à zéro binaire de tous les blocs - $000-$117 ou 0-279.

- Les blocs 0 à 1 contiennent le programme chargeur du système, le Loader. Si la disquette est destinée à être utilisée pour démarrer le système, il faudra y copier les fichiers PRODOS et BASIC.SYSTEM, avec éventuellement un programme STARTUP. Les deux premiers blocs diffèrent selon la version de ProDOS.

- Les blocs 2 à 5 contiennent le Volume-Directory, encore appelé table des matières. Ils sont chaînés entre eux par des pointeurs avant et arrière.

- 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 représenter 4096 blocs d'une disquette. Une Bit-Map d'une taille d'un bloc suffit pour gérer les 280 blocs d'une disquette standard de 5 pouces 1/4.

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

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 secteurs. Les secteurs sur une disquette sont formatés en pistes physiques, dans l'ordre $00-$0F ; 0-15 = 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.

• DOS 3.3 :

00 01 02 03 04 05 06 07 08
00 0D 0B 09 07 05 03 01 0E

• 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.

Accès au disque

L'accès au disque détermine le mode d'emploi de lecture 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. 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 - Big-Mac par exemple - permettent de gagner du temps dans ce mode de programmation, en regroupant les bytes synchros dans des macro-instructions de temporisation.

ORGANISATION D'UNE DISQUETTE FORMATEE SOUS ProDOS

Allocation des blocs

Après le formatage d'une disquette par le programme utilitaire FILER, celle-ci présente un certain nombre de blocs occupés et de blocs libres. Le tableau qui suit reflète l'image type d'une disquette après formatage sous ProDOS.

Blocs alloués par ProDOS

```
Blocs 0-1      <----------Loader---------->
Blocs 2-5      <----------Volume-Directory ---------->
```



```
Bloc 6         <----------Bit-Map---------->
Blocs 7-279    <----------Volume-Subdirectory---------->
               <----------Fichiers systèmes---------->
               <----------Utilisateur---------->
```

Remarque : le bloc $02 est généralement désigné sous l'appellation Key-Block et les blocs $03 à $05 sont appelés Non Key-Block.

Capacité physique d'une disquette

Capacité de stockage d'une disquette :

Pistes	35	(0-34)
Secteurs par piste	16	(0-15)
Secteurs par disquette	560	
Bytes par secteur	256	
Bytes par piste	4 096	
Bytes par disquette	143 360	

Capacité logique d'une disquette

Allocation des blocs :

280 blocs par disquette : de 0 à 279 ($000-$117)
8 blocs par piste : de 0 à 7 ($00-$07)
1 bloc = 2 secteurs ;
512 bytes par bloc : de 0 à 511 ($000-$1FF)

Un bloc se divise en deux parties: partie gauche et partie droite. Les 256 premiers bytes d'un bloc sont ceux de gauche, les 256 bytes suivants, ceux de droite.

Capacité après formatage

Une disquette est opérationnelle après le formatage. A la fin de cette opération, le système effectue un certain nombre de manipulations : écriture du Loader sur les 2 premiers blocs, constitution de la table des matières, ou Volume-Directory, sur les blocs 2 à 5 et de la table d'occupation, la Bit-Map, sur le bloc 6 ; la capacité de stockage aura sensiblement diminué.

Capacité d'une disquette après formatage :

Blocs disponibles	273 blocs
Bytes disponibles	139 776 bytes

A ce total, il faudra cependant retrancher le nombre de blocs nécessaires pour la sauvegarde des fichiers PRODOS et BASIC.SYSTEM, indispensables pour démarrer le système.

PRODOS vers. 1.0.2	31 blocs
BASIC.SYSTEM vers. 1.0.2	21 blocs
total = 52 blocs ou 26 624 bytes	
bytes réellement disponibles	139 776 - 26 624 = 113 152 bytes.

DISQUETTE ProDOS

- Volume Bit-Map et System Bit-Map

ProDOS utilise deux Bit-Map :

- la System Bit-Map - SBM - de la page globale de PRODOS, aux adresses $BF59–$BF6F ;
- la Volume Bit-Map - VBM - sur la disquette, bloc 6.

System Bit-Map - RAM 48 Ko

Pour le traitement des fichiers texte, ProDOS gère automatiquement les pages allouées aux tampons mémoire par la commande OPEN. Le programmeur peut, avant l'implantation d'un programme en mémoire centrale, solliciter par exemple une routine qui pourra fournir la première page disponible.

Implantation d'un programme en mémoire :

- il sera au préalable chargé au bas de la zone mémoire non encore utilisée par le système ;

- à ce programme sera associé un sous-programme Move, qui aura une double fonction : modifier la System Bit-Map et le pointeur de HIMEM, puis placer le programme à son adresse définitive.
  ProDOS suit les mêmes règles de chargement lors du boot d'une disquette.

Volume Bit-Map

C'est la table d'allocation des blocs d'une disquette - bloc 6. Une disquette 35 pistes 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 : 35*8 = 280 blocs.

La table occupe 512 bytes (1 bloc) et peut gérer à la limite : 8*512 = 4096 blocs, mais seuls 280 blocs sont utilisés par les versions actuelles de ProDOS.

Chaque bit représente un bloc : si le bit est à 1, le bloc est libre, si par contre il 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
.....
byte $22 = piste $22 = blocs $110-$117

Représentation d'un byte :

Byte $00 :
blocs $000-$007 01234567
00000001 = 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 Bit-Map

Le Dump de la Volume Bit-Map est celui d'une disquette venant d'être formatée.

Blocs	0-1	Routine Loader
Bloc	2	Volume-Directory - Key-Block -
Blocs	3-5	Volume-Directory - Non Key-Block -
Bloc	6	Volume Bit-Map
Blocs	7-279	Libres

CAT affiche les paramètres suivants :

```
/VOLUME

NAME          TYPE    BLOCKS   MODIFIED

BLOCKS FREE : 273     BLOCKS USED : 7
```

où /VOLUME est le nom du Volume de la disquette ;
NAME, TYPE, BLOCKS et MODIFIED sont des rubriques vides pour le moment ;
7 blocs sont occupés ;
273 blocs sont libres.

En faisant la conversion en format physique, nous obtenons : 14 secteurs occupés, et 546 secteurs libres. Le total faisant évidemment 560 secteurs, ce qui correspond à la capacité d'une disquette standard 35 pistes.

Dump de la Bit-Map d'une disquette 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
```

Le premier byte de la Bit-Map affiche 01 et désigne un bloc libre : le bloc 7 de la disquette. Les trente-quatre bytes suivants sont tous positionnés à la valeur FF - 255 : les blocs 8 à 279 sont libres (8*34 = 272 blocs). Le total des blocs libres est : 272 + 1 = 273 blocs, ce qui correspond dans notre exemple précédent à l'affichage de la commande CAT.

- Dump de la Volume Bit-Map

Représentation du byte $00 :

Bits : 01234567
00000001 = 01
= bloc 7 libre

Le 7e bit représente le bloc 7 de la disquette.

Représentation du byte suivant - $01 :

Bits : 89ABCDEF
111 111 11 = FF
= blocs 8 à 15 libres.

Les blocs $008-$00F sont libres.

PRODOS, SOS ET APPLE PASCAL

Analogie et différence

Comme le système SOS de l'Apple III, ProDOS a la même structure pour traiter les données des fichiers, ce qui lui confère la possibilité d'émulation entre l'Apple II et l'Apple III et permet ainsi d'effectuer des échanges de fichiers. Cet avantage est dû à ce que certaines instructions de ProDOS font partie du système d'exploitation SOS, qui permet par ailleurs une gestion plus étendue des périphériques - imprimante, etc.

Le Loader - programme de chargement sauvegardé sur les quatre premiers secteurs de la disquette - est formé par les blocs 0 et 1. Ce programme comporte le code de boot du système d'exploitation qui se trouve sur la disquette, et selon qu'il a affaire à un Apple II ou III, il fera la différence. Le code de recherche pour le fichier SOS est #$0C, tandis que celui du fichier PRODOS est #$FF. On trouvera la liste complète des codes et types de fichiers en annexe IV.

Lorsqu'une disquette est "bootée" à partir d'un Apple II ou III, le programme contenant le code de boot est chargé à l'adresse $0800, puis exécuté. D'abord, le système vérifie, par l'intermédiaire de la ROM, sur quel modèle d'Apple le boot a été effectué. Si c'est un Apple II, alors le fichier ProDOS sera recherché ; par contre, si c'est un Apple III, le fichier SOS.KERNEL sera exécuté. Si dans les deux cas le fichier reste introuvable, un message d'erreur sera retourné, et le système se plantera. Cette structure particulière autorise le traitement des fichiers sur la série des Apple II et III, en offrant la possibilité d'échanges d'informations entre eux.

SOS et ProDOS utilisent la même structure du Directory, ce qui se traduit par une compatibilité totale entre ces deux modèles de machine.
Les utilitaires du type Filer, Conversion, Editor/ Assembler et ProDOS Basic System Programm, peuvent également être utilisés par les deux systèmes.

ProDOS a été spécialement étudié et mis au point pour gérer et commander un disque dur, mais traite parfaitement les lecteurs de disquettes, et permet ainsi toute une gamme de manipulations supplémentaires par rapport au DOS 3.3. Nous verrons le détail des instructions dans un prochain chapitre.

ProDOS permet une certaine compatibilité avec le système Apple Pascal, dont la lecture des secteurs se fait d'une façon très proche : ils sont lus dans le même ordre pour les deux systèmes (Skewing).

Format d'une disquette ProDOS

Chaque disquette formatée sous ProDOS avec un lecteur de disquettes standard comporte 35 pistes, dont chacune se divise en 16 secteurs. La disquette a une capacité de stockage qui correspond à 280 blocs. Chaque bloc est formé par 2 secteurs, et chaque secteur a une capacité de stockage de 256 octets ou bytes.

Il faut distinguer deux formats: le format physique et le format logique. Le premier est celui qui traite la disquette au niveau des pistes/ secteurs, tandis que le second est utilisé par ProDOS pour des calculs internes au système.

Les 280 blocs de la disquette sont numérotés dans l'ordre croissant des pistes :

- Blocs 0 à 7 pour la piste 0.
- Blocs 8 à 15 pour la piste 1.
- et ainsi de suite.

Contenu global d'une disquette ProDOS

Les deux premiers blocs de la disquette contiennent le Loader, programme chargeur, exécuté par la ROM du contrôleur, dont le rôle est de charger le système d'exploitation, en fonction de la configuration rencontrée. Un code machine vérifie si c'est un Apple II ou III, puis continue le processus en cours.

Les blocs 2 à 5 contiennent le Volume-Directory - catalogue - de la disquette. C'est la table des matières principale, qui se présente comme une liste chaînée avant et arrière. Les 4 premiers bytes de chaque bloc de cette liste contiennent les pointeurs respectifs.
Chaque bloc peut contenir 13 noms de fichiers, programmes ou Subdirectory - sous-catalogue -, et les 4 blocs ont une capacité totale de 52 noms. Mais il ne vous sera possible que de sauvegarder 51 noms de fichiers différents, car l'opération formatage stocke le nom du Volume en en-tête du bloc 5.
La première entrée d'un bloc-Volume - Directory ou Subdirectory - est appelée en-tête ; elle se compose d'une suite de 39 bytes qui caractérisent la table des matières : son nom, si elle est principale ou auxiliaire, date et heure de création, byte d'accès, etc. Les autres noms, viennent à la suite de l'en-tête - à partir du byte #$2B - et désignent soit un nom de fichier programme, soit un nom de Volume-Subdirectory.

Le bloc 6 correspond à la Bit-Map - répertoire des blocs - et renseigne le système sur l'occupation des blocs de la disquette ; c'est la table d'occupation des blocs de la disquette. Chaque bit d'un bloc correspondant indique si celui-ci est libre ou occupé. Lors de l'utilisation d'un disque dur, la Bit-Map peut comporter jusqu'à 16 blocs répertoires, formant ainsi une table d'occupation chaînée. Si l'on s'amuse par curiosité à calculer la capacité de gestion éventuelle d'un disque dur, on s'aperçoit qu'elle est de l'orde de 33 554 432 bytes (16 * 512 * 8 = 65 536 blocs). Comme un disque ProDOS peut contenir au maximum 65 536 blocs, il faudra deux octets pour coder le pointeur de chaque numéro d'un bloc.

Avec un lecteur de disquettes standard, seuls les 35 premiers octets, représentant 280 blocs, sont utilisés, mais il est possible de modifier le système afin d'étendre ses possibilités. La capacité standard d'une disquette n'a pas changé, et permet donc un stockage de 140 Ko (280*512 = 143 360 bytes).

Répertoire des pistes/secteurs

- 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 ProDOS, certains termes sont modifiés, et l'organisation de la disquette est très différente. C'est la Track Blocks List qui désigne la table d'allocation des blocs du fichier. Tout un vocabulaire nouveau a fait son apparition: en annexe se trouve un dictionnaire des mots usuels de ProDOS qu'il sera nécessaire d'apprendre.

ALLOCATION DES BLOCS

- Blocs Index

- Un fichier programme ou Directory - Volume-Directory ou Volume-Subdirectory -, 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), qui pointe vers les blocs d'un répertoire auxiliaire (Blocs Index) d'une capacité maximale de 128 blocs. Chacun de ces blocs du répertoire auxiliaire peut à nouveau pointer 256 blocs de données.

Nous développerons, par la suite, une analyse des emplacements et vecteurs précités pour un support disquette de format standard.

Le RAM-Disk

Lorsque ProDOS détecte une carte 80 colonnes "étendue", sur un Apple IIe ou s'il est booté sur un IIc, il crée un disque virtuel dans l'espace mémoire auxiliaire 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.

APPEL DU PSEUDO-DISK

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 données resteront sauvegardées tant que l'Apple restera sous tension ; dans le cas contraire, tous les programmes et fichiers seront définitivement perdus. Un boot à froid vous fera perdre le bénéfice des programmes contenus dans le Directory de la RAM.

VOLUME DU PSEUDO-DISK

Le disque ainsi créé en mémoire auxiliaire se compose de 128 blocs de 512 octets, qui auront la configuration 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 - 127	Fichiers Directory et fichiers programmes.

LE BOOT D'UNE DISQUETTE ProDOS

Le boot sous 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 meilleurecompréhension de l'ensemble pour la suite des investigations avec le système ProDOS.

Boot sous DOS 3.3 - RAM de 48 Ko

1- BOOT	0	ROM carte contrôleur $C600	Charge BOOT 1 en RAM.
2- BOOT	1	Disquette : Piste 0, secteur 0 RAM : $0800-$0900	Charge BOOT 2 et lui-même.
3- BOOT	2	Disquette : Piste 0, secteurs 1-9 Master : $3700-$4000 Slave : $B700-$C000	Contient RWTS. Charge le SED (et le translateur).

4- SED	Disquette : Piste 2, secteurs 4-0 Piste 1, secteurs F-0 Piste 0, secteurs F-C Master : $1D00-$3600 Slave : $9D00-$B600	Système d'exploitation de la disquette.
Relocator	Disquette Master : Piste 0, secteurs A-B RAM: $1B00-$1D00	Réinstalle le SED à sa place définitive : $ 9D00-$C000

Le boot sous ProDOS

Tout change, ou presque, dans ce processus par rapport au boot du DOS 3.3 : l'emplacement du système d'exploitaion sur la disquette, la phase d'implantation en mémoire, et l'occupation de la mémoire vive, RAM.

Processus de boot d'une disquette ProDOS

1 - BOOT 0. L'ensemble des opérations implique qu'une disquette ProDOS soit introduite dans un lecteur de disquettes, relié au slot 6 d'une configuration standard Apple. Deux cas peuvent alors se présenter: ou votre Apple est sous tension, ou il est hors tension. Dans le premier cas, il vous suffira d'allumer votre micro, tandis que dans le second cas, il vous faudra taper à partir de votre clavier l'une des commandes suivantes : PR#6, 6 CTRL-P ou C600G. Le processeur commence alors le processus de boot en exécutant la routine contenue dans la ROM de la carte du contrôleur. Dans notre cas, il s'agit du slot 6, et la routine concernée débute en $600. Le sous-programme ignore tout au départ, du contenu de la disquette : il charge en premier une partie du Loader - programme de chargement - qui se trouve sur la piste $00, secteur $00. Son but est de continuer le boot de la disquette, en ayant au préalable déterminé la marche à suivre. Au niveau de la phase 1, le Loader ne représente que la moitié du bloc 0, et sera logé à partir de l'adresse $0800.

2 - 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 0 - piste $00, secteur $01 - et finalement le bloc 1 - piste $00, secteurs $02,$03. C'est la phase 2 du Loader qui se termine.

3 - BOOT 2. Les blocs 0 et 1 sauvegardés sur la piste 0, secteurs 0 à 3, forment ainsi 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 de la disquette. 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 la bank 1 de la carte langage et ceci à partir de l'adresse $D000. Ensuite est installé un programme de Reboot dans la bank 2 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.

4 - Ensuite, ProDOS cherche sur la disquette le premier fichier avec le suffixe SYSTEM, qui est en général le BASIC.SYSTEM. Celui-ci est également chargé à l'adresse $2000, puis déplacé à son adresse définitive se situant à partir de $9A00.

5 - Une fois que le fichier BASIC.SYSTEM est exécuté, celui-ci cherche, dans le catalogue de la disquette, un programme HELLO, sauvegardé généralement sous le nom de STARTUP. Ce fichier, du type AppleSoft ou Binaire, est exécuté dès qu'il est trouvé, sinon le système Basic passe la main à AppleSoft.

Remarque : si, pour une raison de programmation, votre disquette ne nécessite pas de fichier STARTUP, il est quand même conseillé de placer un court programme AppleSoft sous ce nom ; dans le cas contraire, la page zéro ne serait pas correctement initialisée. Pour le vérifier, "bootez" une disquette sans programme STARTUP, rentrez ensuite dans le Moniteur par un CALL-151, puis tapez FC58G, qui est l'appel de l'instruction HOME. Votre système se plantera, car le registre d'état PS, statut byte, n'a pas été initialisé au préalable à l'adresse $0048 de la page zéro.

ORGANISATION INTERNE

Situation après le boot de la disquette

Lorsque le cycle du démarrage de la disquette ProDOS s'est effectué normalement, deux fichiers se trouvent alors en mémoire : PRODOS et BASIC.SYSTEM (éventuellement le programme STARTUP). Il vous est désormais possible de "rebooter" en appelant le fichier PRODOS par l'intermédiaire de n'importe quel slot de votre configuration, et en respectant le chemin (Pathname).

Méthode de recherche du chemin sous PRODOS :

PREFIX/VOLUME/NOM ou CATALOG/VOLUME/NOM

L'opération Reboot est simplifiée par l'adjonction du Dash ou "-" qui, associé au fichier PRODOS (-PRODOS), équivaut à une instruction se situant entre un boot à froid et un boot à chaud. Cela s'explique par le fait que le Loader stocké sur la disquette de démarrage n'est plus nécessaire pour le chargement du système. Pour rappel, le Loader est un sous-programme exécuté par le contrôleur, et dont le rôle est de poursuivre le boot, en vérifiant au préalable sur quel modèle Apple se déroule l'opération.

En règle générale, il est possible d'effectuer un boot à froid et à chaud par l'intermédiaire du drive 1, et uniquement un boot à chaud à partir du drive 2, cela à partir de n'importe quel slot valide.

Avec l'ordre -ProDOS, ce sont les fichiers ProDOS et BASIC.SYSTEM qui seront réinstallés en mémoire, tandis qu'avec l'ordre -BASIC.SYSTEM, ce sera uniquement le fichier BASIC.SYSTEM. L'instruction Dash a l'avantage de pouvoir effectuer le chargement, puis le lancement de l'exécution des types de fichiers Basic, Binaire et Exec.

Avec l'ordre -BASIC.SYSTEM, la réinstallation de ce fichier en mémoire ne détruit pas les 64 Ko de RAM-Disk sur la carte d'extension d'un Apple IIe ou sur la mémoire auxiliaire d'un IIc. Par contre, l'ordre -PRODOS ou PR#6 détruit les pages éventuelles, étant donné que cette zone est utilisée pour l'implantation du système. C'est un inconvénient fondamental de ProDOS, car il est en pratique très fréquent d'effectuer un boot à froid, que celui-ci soit volontaire ou involontaire. Il faudra rester très prudent dans l'utilisation du RAM-Disk.

Recherche des slots et drives

Chemin normal :

D'après le manuel de référence, lorsque le système se trouve correctement implanté en mémoire, la recherche d'un programme sur un support de disque se fait par rapport aux Volumes en ligne. L'ordre de recherche des Volumes, par l'intermédiaire des périphériques connectés, se déroule suivant une séquence établie. C'est ainsi que le système consulte à chaque fois tous les drives (Devices), et cela à partir de celui qui a "booté" la disquette.

Le premier Volume référencé est celui d'où la disquette a été "bootée". La lecture des autres Volumes se fera dans un ordre décroissant, à partir de la valeur la plus élevée des slots occupés par des drives. Pour chaque slot, la recherche du Volume sera effectuée dans l'ordre drive 1, drive 2.

Par exemple, la structure pourrait se composer de deux lecteurs du type DISK II dans le slot 6, deux DISK II dans le slot 5, et un disque dur du type Profile dans le slot 7. L'ordre de recherche des différents Volumes se ferait ainsi :

Slot 6, drive 1
Slot 6, drive 2
Slot 7
Slot 5, drive 1
Slot 5, drive 2

ONLINE ($C5) attribue, suivant l'argument de Unit Number, un espace mémoire pour la sauvegarde des données mémorisées lors de la recherche des Volumes en ligne. Si l'argument a comme valeur le numéro du périphérique, alors #$10 (16) bytes seront réservés ; si par contre l'argument est nul, tous les périphériques seront recensés, et #$100 (256) bytes réservés.

La taille du tampon mémoire sera fixée suivant l'argument déclaré. Ensuite, le système compare l'espace alloué ($EE6C) à l'occupation de la System Bit-Map se trouvant aux adresses $BF58-$BF6F. Si l'espace mémoire se trouve déjà occupé, le code d'erreur #$56 sera renvoyé ; BAD BUFFER ADDRESS.

Si l'argument de Unit Number est déclaré, le système recherche l'entrée correspondante dans la table Volume Control Block implantée aux adresses $F200-$F2FF. Dans le cas contraire, une entrée libre (routine en $DA69) servira pour la sauvegarde du bloc 0002 (Volume-Directory Key-Block). Par la suite, le Volume-Directory sera lu à partir de la VCB par la routine placée en $DDE1 - voir le chapitre 3 qui traite de la VCB.

Exécution du boot d'une disquette

Il est possible de suivre pas à pas le boot d'une disquette ProDOS, par une astuce qui sera développée par la suite. Il suffit de démarrer une disquette à partir du slot 6 par exemple, et qui comporte les fichiers suivants : PRODOS, BASIC.SYSTEM et STARTUP éventuellement.

Une fois ProDOS implanté en mémoire, rentrez en mode Basic pour pouvoir par la suite lister, à partir de la mémoire centrale, les situations suivantes :
- les différentes phases du programme Loader ;
- le fichier PRODOS implanté à l'adresse $2000, avant son transfert vers son adresse définitive.

Chaque phase du boot a été décomposée pour être le plus explicite possible. L'adresse $1500 a été choisie en fonction de son emplacement privilégié dans l'environnement mémoire (zone inutilisée par ProDOS). Nous y placerons le sous-programme de la carte contrôleur DISK II, pour effectuer un boot à chaud à partir du Moniteur, par la commande 1500G.

PISTE 0, SECTEUR 0, LISTEZ A PARTIR DE $0801

- Loader phase 1

CALL -151	Appel du Moniteur.
1500<C600.C6FFM	Déplace le sous-programme contrôleur à l'adresse $1500.
15F8:00	BReaK pour arrêter le processus de boot.
1500G	Exécute le programme du contrôleur implanté à partir de l'adresse $1500.

Ensuite, listez à partir de l'adresse $0801.
Cette astuce permet, par ailleurs, de contourner certains problèmes de compatibilité entre le IIe et le IIc, si l'on a pris soin de ne pas modifier l'adresse $15F8 comme décrit plus haut.

Compatibilité IIe - IIc :

L'Apple IIc n'est pas compatible à 100 % avec le IIe, et cela pour deux raisons :

- différence entre les microprocesseurs: 6502 pour le IIe et 65C02 pour le IIc. Le 65C02 comporte un plus grand nombre d'instructions de base ;

- certaines routines de la ROM AppleSoft et du Moniteur ont été modifiées dans le IIc, pour des raisons de commodité et d'évolution de la technologie du matériel en présence.

C'est ainsi que certains programmes conçus pour le IIe ne se laissent plus "booter" sur un IIc (Computer Embush, par exemple), car un octet significatif de la ROM est testé. Il apparaît alors sur l'écran un message d'erreur qui plante le système. Ce problème peut facilement se résoudre en effectuant un boot à chaud de la disquette en question, après avoir au préalable déplacé le sous-programme du contrôleur vers une zone neutre et non utilisée par le système ($1500 pour notre exemple).

Ce procédé de boot peut être utilisé en mode immédiat comme précédemment, ou en mode programme, à partir d'un support disquette par exemple. Dans ce dernier cas, il faudra sauvegarder le sous-programme résident de la ROM (contrôleur) par l'instruction : BSAVE BOOT, A$C600, L$FF (si la carte interface se trouve dans le slot 6).

Pour rendre ce procédé opérationnel à loisir, il faudra lui adjoindre un petit programme en langage Basic, qui chargera au préalable le sous-programme de boot, puis l'exécutera par un CALL. La disquette devra comporter deux programmes :

- un programme Basic sous le nom de PROGRAMME DE BOOT UNIVERSEL par exemple

- un programme du type binaire sauvegardé sous le nom de BOOT, dont le contenu sera le programme de la ROM du contrôleur. Ce programme sera placé à l'adresse $1500, en raison de la neutralité du système envers cette zone mémoire.

Programme de Boot universel

```
10  PRINT CHR$(4) "BLOAD BOOT, A$1500" : REM Charge le programme binaire du
    nom BOOT à l'adresse $1500. Doit se trouver sur la même disquette contenant le
    programme Basic.
20  HTAB 10 : VTAB 5 : PRINT "Mettez en place la disquette à BOOTER," : PRINT
    "tapez ensuite une touche pour continuer." ;
30  GET A$
40  CALL 5376 : REM Exécute le sous-programme implanté à l'adresse $1500.
```

Le programme Basic, une fois exécuté, implante à l'adresse $1500 le sous-programme Binaire du nom de BOOT qui devra se trouver sur la même disquette. A ce moment-là, le système attend que vous mettiez en place la disquette à "booter", puis poursuivra le déroulement. CALL 5376 appelle le sous-programme BOOT logé à l'adresse $1500 (5376), et l'exécute. La disquette contenue dans le lecteur connecté au slot 6 sera "bootée".

PISTE 0, SECTEURS 0-1, LISTEZ A PARTIR DE $801

- Loader phase 2

```
CALL -151              Appel du Moniteur.
1500<C600.C6FFM        Déplace le sous-programme contrôleur à l'adresse $1500.
15F8 : A9 00 8D 31 08
       4C 01 08        Mise en place d'un sous-programme d'interruption du
                       processus de boot :
                       15F8- A9 00      LDA #$00
                       15FA- 8D 31 08   STA $0831
                       15FD- 4C 01 08   JMP $0801
1500G                  Exécute le programme qui débute à l'adresse $1500.
```

LOADER PHASE 3, A LISTEZ A PARTIR DE $0800

PRODOS à lister à partir de $2000.

```
CALL -151              Appel du Moniteur.
1500<C600.C6FFM        Déplace le sous-programme contrôleur à l'adresse $1500.
15F8 : A9 00 8D FC 08
       4C 01 08        Mise en place du sous-programme d'interruption, pour
                       permettre la récupération des programmes :
                       15F8- A9 00      LDA #$00
                       15FA- 8D FC 08   STA $08FC
                       15FD- 4C 01 08   JMP $0801
1500G                  Exécute le sous-programme implanté à l'adresse $1500.
```

LA ROM - INTERFACE CONTROLEUR

Détection de la carte contrôleur

Interface du drive :

Lors du boot d'une disquette ProDOS avec la ROM Autostart, le système effectue une lecture de certains bytes significatifs de la ROM interface du drive. C'est pourquoi ces bytes devront être identiques d'une interface à l'autre, et ceci pour des raisons de compatibilité du matériel.

Bytes d'identification :

Cs01- 20
Cs03- 00
Cs05- 03
Cs07- 3C

La lettre s après le C désigne le slot actuel, et peut prendre toute valeur comprise entre 1 et 7 inclus.

Lors d'un démarrage par l'intermédiaire de la ROM Autostart, le système effectue une lecture aux adresses respectives des quatre bytes d'identification, puis effectue un saut (JMP) à l'adresse Cs00. Le système reconnaît ainsi les interfaces des périphériques en ligne.

Sous DOS 3.3, la valeur de ces différents bytes était identique, mais le processus de reconnaissance inutilisé. Par ailleurs, les adresses de $CsFB à $CsFF contenaient des 00, et n'étaient pas exploitées avant la parution, sur le marché, de ProDOS. Celles-ci sont maintenant utilisées par le système pour mémoriser certaines caractéristiques particulières : c'est un "must" ProDOS.

Bytes significatifs après CsFB :

Les adresses $CsFC,$CsFD doivent contenir le nombre maximal de blocs que peut gérer la disquette sous la forme : octet poids faible, octet poids fort.

```
      $CsFC- LB HB
où          LB désigne le byte de poids faible (LByte),
            HB désigne le byte de poids fort (HByte).
```

L'adresse $CsFE contient le Statut byte, ou byte caractéristique du système, dont la valeur de chaque bit est significative.

Signification des différents bits :

Bit 7 Le support de sauvegarde du périphérique en ligne est amovible. Par exemple, une disquette, en opposition à un disque dur du type Profile.
Bit 6 Possibilité d'interruption du périphérique en ligne.
Bit 5 Nombre de lecteurs de disquettes reliés à la configuration (0-2). Ce bit est lié au bit 4.
Bit 4 Nombre de lecteurs de disquettes reliés à la configuration. Ce bit est lié au bit 5. Par exemple, le contenu des bits 4 et 5 est 10, signifiant ainsi que 2 drives sont connectés à un slot.
Bit 3 Possibilité de formatage.
Bit 2 Possibilité d'écriture.
Bit 1 Possibilité de lecture.
Bit 0 Possibilité de lecture du Statut (caractéristique).

Chaque propriété particulière est définie en donnant sa valeur respective à chaque bit concerné. La valeur définitive, une fois chaque bit significatif codé, est la valeur équivalente du byte contenu à l'adresse $CsFE.

L'adresse $CsFF doit contenir l'adresse de poids faible du point d'entrée de la ROM interface du contrôleur, et aura toujours comme valeur #$00 pour n'importe quel numéro de slot.

Exemples :

$C600 slot 6 LB = 00
$C500 slot 5 LB = 00
$C400 slot 4 LB = 00
$C300 slot 3 LB = 00
$C200 slot 2 LB = 00
$C100 slot 1 LB = 00

Espace libre après le boot

Sous DOS 3.3, certaines pages de la mémoire sont détruites lors du boot d'une disquette. En particulier, certains vecteurs de la page zéro sont modifiés, tandis que la page 1 graphique haute résolution est utilisée temporairement pour hisser le DOS vers son adresse haute.

Lors du boot d'une disquette ProDOS, le scénario diffère un peu, et d'autres pages sont exploitées. Connaître leurs emplacements est une nécessité pour le programmeur. Lorsque les fichiers PRODOS et BASIC.SYSTEM sont chargés seuls, sans le programme STARTUP, les espaces mémoire suivants restent intacts :

$0100 - $01D7
$0200 - $027F
$0291 - $02FF
$0356 - $036B
$03D6 - $03EF
$0A86 - $0BFF
$1800 - $1DFF
$5C00 - $95FF
$9600 - $99FF
$BD00 - $BDFF

Remarque : sans le fichier BASIC.SYSTEM, l'espace $5C00-$BEFF reste intact : les adresses $9600-$BEFF étant la zone d'implantation du système Basic. L'adresse $9600 est le début d'un tampon extérieur. L'adresse $BD00 est le début d'un tampon interne.

Par ailleurs, l'espace compris entre les adresses $5C00-$95FF pourra trouver une utilisation judicieuse lors de la programmation. En effet, la place est suffisante pour charger un programme binaire sous DOS 3.3 et l'emplacement restera le même après le boot à chaud d'une disquette ProDOS. Cela vous permettra la réalisation et la mise au point d'un programme particulier, nécessitant un fonctionnement sous les systèmes d'exploitation ProDOS et DOS 3.3.

CHAPITRE 3

ORGANISATION INTERNE DE ProDOS TABLES DES MATIERES VOLUME-DIRECTORY

ORGANISATION DU FICHIER PRODOS

- Version 1.0.2

Préambule

Comme le fichier PRODOS n'occupe pas de place bien précise sur la disquette, il sera fait référence à une position relative par rapport à un bloc de départ ayant comme valeur 0. L'adresse de départ - $2000 - est celle où le fichier PRODOS est implanté par le Loader, au moment du boot de la disquette, et juste avant l'appel de la routine Move.

Structure de PRODOS

ROUTINE MOVE - RELOCATOR

Sous-programme pour transférer PRODOS dans la carte langage.
Blocs 0-4 : adresses $2000-$29FF

RAM DISK-DRIVER

Bloc 5 : adresses $2A00-$2BFF.
- RAM Disk-Driver (64 Ko auxiliaires $0200-$03FF).

Bloc 6 : partie gauche ; adresses $2C00-$2C7E.
- RAM Disk-Driver ($FF00-$FF7E Bank 1).

Bloc 6 : partie gauche ; adresses $2C7F-$2CFF.
- Bytes nuls.

LE MLI (MACHINE LANGUAGE INTERFACE)

Bloc 6 : partie droite ; adresses $2D00-$2DFF.
- Début du MLI ($D000-$D0FF Bank 1).

Blocs 7-21: adresses $2E00-$4BFF.
- MLI ($D100-$EEFF Bank 1).

Bloc 22 : partie gauche ; adresses $4C00-$4CFF.
- Fin du MLI ($EF00-$EFFF Bank 1).

Bloc 22 : partie droite ; adresses $4D00-$4D05.
- MLI ($F000-$F005 Bank 1).

Bloc 22 : partie droite ; adresses $4D06-$4DFF.
- Bytes nuls.

LA GLOBAL PAGE DE PRODOS

Bloc 23 : partie gauche ; adresses $4E00-$4EFF.
- Page globale pour appel de PRODOS dans la carte langage ($BF00-$BFFF).

Bloc 23 : partie droite ; adresses $4F00-$4FFF.
- Page Globale BASIC.SYSTEM ($BF00-$BFFF).

LES INTERRUPTIONS

Bloc 24 : partie gauche ; adresses $5000-$5069.
- Routine Horloge ($F142-$F1AB).

Bloc 24 : partie gauche ; adresses $506A-$507C.
- Date ($F1AC-$F20E).

Bloc 24 : partie gauche ; adresses $507D-$509A.
- Bytes nuls.

Bloc 24 : partie gauche ; adresses $509B-$50EB.
- Interrupt ($FF9B-$FFEB).

Bloc 24 : partie gauche ; adresses $50EC-$50F9.
- Bytes nuls.

Bloc 24 : partie gauche ; adresses $50FA-$50FF.
- NMI, IRQ et Reset ($FFFA-$FFFF).

Bloc 24 : partie droite ; adresses $5100-$51FF.
- Bytes nuls.

LE DISK-DRIVER

Bloc 25 : partie gauche ; adresses $5200-$5395.
- Disk-Driver ($F800-$F995).

Bloc 25 : partie gauche ; adresses $5396-$53FF.
- Données du Disk-Driver ($F996-$F9FF).

Bloc 25 : partie droite ; adresses $5400-$54FF.
- Table des nibbles ($FA00-$FAFF).

Bloc 26 : partie gauche ; adresses $5500-$5572.
- Bytes nuls.

Bloc 26 : partie droite ; adresses $5573-$5584.
- Données Disk-Driver ($FB74-$FB84).

Bloc 26 : partie droite ; adresses $5585-$55FF.
- Disk-Driver ($FB85-$FBFF).

Bloc 27 : adresses $5600-$57FF.
- Disk-Driver ($FC00-$FDFF).

Bloc 28 : partie gauche ; adresses $5800-$58BD.
- Disk-driver ($FE00-$FEBD).

Bloc 28 : partie gauche ; adresses $58BE-$58FF.
- Bytes nuls ($FEBE-$FEFF).

LA ROUTINE REBOOT

Bloc 28 : partie droite ; adresses $5900-$59FF.
- Début de Reboot ($D100-$D1FF Bank 2).

Bloc 29 : adresses $5A00-$5BFF.
- Routine Reboot ($D200-$D3FF Bank 2).

TABLE DES MATIERES PRINCIPALE

Le Volume-Directory

- Entrée du nom d'un Volume

Ce type de Directory se trouve toujours à l'emplacement des blocs 0002 à 0005. A l'encontre des Subdirectory, le Volume-Directory ne permet pas sa duplication.
Le Volume-Directory occupe 4 blocs sur une disquette souple, de format 5 pouces 1/4, et peut contenir jusqu'à 51 noms de fichiers.

Les quatre premiers bytes de chaque bloc d'un Directory sont réservés pour les pointeurs arrière et avant - Backward et Forward.
Le pointeur avant - Forward Pointer - désigne, dans le Directory, le numéro du bloc qui précède son propre bloc. Le pointeur arrière - Backward Pointer - désigne, dans le Directory, le numéro du bloc qui suit son propre bloc.

Dans le Key-Block, bloc 0002, le pointeur arrière de valeur 0000 signifie que c'est le premier bloc du Directory.
Dans le bloc 0005, Non Key-Block, le pointeur avant de valeur 0000 signifie que c'est le dernier bloc du Directory.

VOLUME-DIRECTORY - KEY-BLOCK

Header Volume Name - Block 00 02

Le premier nom qui figure dans la table des matières principale - Volume-Directory - est celui du Volume sauvegardé lors de l'initialisation de la disquette par le programme FILER. Le Volume-Directory contient un certain nombre de paramètres particuliers.

Position	*Contenu*
$00-$01	Pointeur arrière - LByte, Hbyte : 00 00 ; $0000 signifie que c'est le premier bloc.
$02-$03	Pointeur avant - LByte, HByte : 03 00; $0003 pointe le bloc 3 du Directory.
$04	Storage Type/Name Length. 4 bits de poids fort qui caractérisent la sauvegarde d'un fichier. 4 bits de poids faible qui désignent la longueur du nom du Volume. Exemple : $F6 ou F = Fichier Volume et 6 = 6 caractères.
$05-$13	Nom du fichier - USERS.DISK par exemple -.
$14-$1B	Bytes réservés pour une application future.
$1C-$1F	Date et heure de la sauvegarde d'un nom du Volume-Directory.
$20-$21	Versions de ProDOS.
$22	Access Byte. Byte d'accès au fichier. Exemple : $C3 = Lecture/écriture permise.
$23	Nombre de bytes alloués à chaque entrée du Volume-Directory : 39 bytes pour chaque entrée Volume.
$24	Nombre d'entrées possibles par bloc :13 entrées par bloc.
$25-$26	Total des noms de fichiers actuellement sauvegardés dans la table des matières ou Volume-Directory : maximum 51 fichiers.
$27-$28	Pointeur de la Bit-Map - 06 00 = 0006.
$29-$2A	Capacité totale de la disquette - Blocs : 18 01 = $0118 = 280 blocs.

SIGNIFICATION DES BYTES DE POSITION

- Bytes $00-$01: Pointeur arrière - LByte,HByte.

  Pointe le bloc qui le précède. Comme l'entrée du nom du Volume se trouve au début de la table des matières, ce pointeur est nul - 00 00.

- Bytes $02-$03 : Pointeur avant (LByte,HByte).

  Pointe le bloc suivant. Comme c'est le début de la table des matières, le bloc suivant est : 03 00 = 0003.

- Byte $04 : Storage Type/ Name Length.

  Le byte se divise en 2 nibbles de 4 bits chacun : les 4 bits de poids fort caractérisent la sauvegarde du fichier, et les 4 bits de poids faible la longueur du nom du fichier.

  a) Storage Type - 4 bits de poids fort.

  ProDOS gère plusieurs index pour caractériser la taille d'un fichier au moment de la sauvegarde de ses données dans le catalogue.

  Index 00 : Nul, pas de nom ou fichier delété.

  Index 01 : Seedling. Fichier du plus petit format, comportant entre $0000 et $0200 bytes. Il n'occupe qu'un bloc de données sur la disquette et n'a pas de bloc index. Les données sont directement pointées par le fichier : pointeur sauvegardé aux bytes $11-$12 d'une entrée fichier, dans une table des matières.

  Index 02 : Sapling. Fichier d'une taille supérieure à un bloc, avec une capacité de $0201 à $20000 bytes. Ce type de fichier peut comporter jusqu'à 256 blocs de données avec un seul bloc index comme répertoire.

  Index 03 : Tree. Fichier dont la taille nécessite plus d'un bloc index. La capacité d'un tel fichier de données est de $020001 à $1000000 bytes. Sa structure nécessite alors un Master Block qui peut pointer de 1 à 128 blocs index, dont chacun peut pointer à nouveau 256 blocs de données. Un rapide calcul montre qu'un tel fichier peut contenir 16 777 215 bytes soit 16 mégabytes.

  Index 0D : Subdirectory File. Paramètre attribué au nom du fichier qui se trouve dans un Directory et pointe un Subdirectory.

  Index 0E : Volume-Subdirectory. Paramètre attribué au nom du Volume qui se trouve au début d'un Subdirectory.

  Index 0F : Volume-Directory. Paramètre attribué au nom du volume qui se trouve au début du Directory. C'est le nom de Volume d'une disquette, attribué lors du formatage.

  b) Longueur du nom :

  - Fichier programme ou Subdirectory

  Ce sont les 4 bits de poids faible qui représentent la longueur du nom, dont la syntaxe ne devra pas dépasser 15 caractères - 4 bits = maximum 24 = 16.

- Bytes $05-$13 : File Name.

  Un maximum de 15 caractères désigne un nom de fichier programme ou Subdirectory.

  Exemple:

  Si USERS.DISK est le nom du Volume-Directory, le byte Storage Type/ Name Length sera représenté par : FA où F est l'indice du Volume-Directory et A la longueur - L =10. Les noms des fichiers sont sauvegardés dans la table des matières avec le bit 7 à 0.

- Bytes $14-$1B : Bytes inutilisés.

  Ce vecteur contient actuellement 8 bytes nuls : il est réservé pour une utilisation future (Citation du *Technical Manual*).

- Bytes $1C-$1F : Date et heure du formatage.

  La date et l'heure de la création du Volume-Directory - formatage Volume - sont sauvegardées à cet emplacement, avec une carte du type Thunderclock.

- Bytes $20-$21 : Version de ProDOS.

  ProDOS version et version mini. Seul le byte $21 est testé par la commande OPEN : si sa valeur est différente de #$00 pour les versions ProDOS 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.

  Ce byte détermine si un nom de fichier peut-être : re-nommé, éffacé, lu, ou si ses données peuvent être copiées. La valeur binaire des bits détermine le byte d'accés.

  Représentation binaire :

  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, autorise l'instruction DELETE.

  Bit 6 : Rename bit. Si ce bit est à 1, autorise l'instruction RENAME.

  Bit 7 : Backup bit. Ce bit est positionné à 1 dans la page globale, à l'adresse $BF95. Il est utilisé par des commandes CREATE, RENAME, CLOSE, WRITE et SET.FILE.INFO pour permettre la copie des données. A l'appel du MLI, à partir de l'adresse $E9D9, un LDA $BF95, EOR #$20, teste le bit 5 - #$20 représente l'image binaire du bit 5. Au retour MLI, l'adresse $BF95 ets remise à zéro par un LDA #$00, STA $BF95, à partir de l'adresse $D078.

Test du bit 5 :

Bits :	76543210	
	00100000	= #$20

L'instruction machine EOR effectue un test sur le bit 5 - Backup bit. Par ailleurs, l'adresse $E9D9 est interne à la routine MLI SET.FILE.INFO.

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 (vérrouillé)
	0	0	1	0	0	0	1	-	= #$21 = SYS

- Byte $23 : Entry Lenght.

  Longueur attribuée, par entrée, pour la sauvegarde des paramètres à l'intérieur de cette table des matières. (39 bytes pour l'entrée du nom du Volume.)

- Byte $24 : Total Block Entry.

  ProDOS autorise treize entrées de noms de fichiers par bloc (Directory-Block).

- Bytes $25-$26 : File Count.

  Nombre de noms de fichiers actuellement sauvegardés dans le Volume-Directory. C'est le nombre d'entrées réellements occupées dans cette table des matières.

- Bytes $27-$28 : Bit-Map Pointer.

  Pointeur du premier bloc de la Volume Bit-Map - table d'occupation des blocs de la disquette. Ce pointeur a comme valeur par défaut : 06 00 = 0006.
  Théoriquement, ProDOS peut gérer une VBM de 4 096 blocs; dans ce cas, les blocs devront se suivre et être chaînés entre eux.

- Bytes $29-$2A : TOTAL BLOCKS.

  C'est la capacité, en nombre de blocs, d'une disquette ou support de sauvegarde : 280 blocs - $0118 - pour une disquette standard 35 pistes.
  Théoriquement, ProDOS peut gérer jusqu'à 65 5365 blocs de données ($10000).

  Remarque : le byte suivant est la première entrée d'un nom de fichier. Chaque bloc peut contenir 13 noms de fichiers, sauf le premier - Key-Block - qui contient d'origine le nom du Volume (sauvegardé lors du formatage).

Volume-Directory

Position relative des entrées : configuration d'une entrée

Dans le paragraphe précédent, nous avons vu le détail d'une entrée d'un nom de Volume dans une table des matières d'un Volume-Directory. C'est la première entrée du Key Block - bloc 0002. Ce paragraphe détaille la suite de la table des matières qui permet la sauvegarde de 51 noms de fichiers. Le bloc 0002, ou Key-Block, peut contenir 12 noms de fichiers - 13 moins celui du Volume - et les 3 blocs suivants, 13 noms chacun, ce qui fait un total de : (13*3) + 12 = 51 noms de fichiers.

Structure type d'un bloc d'entrées :

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

Le byte 512 est inutilisé par les 4 blocs du Volume-Directory et contient la valeur $00.

Chaque bloc d'une table des matières se divise en un certain nombre d'entrées de noms de fichiers, dont chacune peut être désignée par un pointeur d'une valeur relative : pointeur de début, $00 ; pointeur de fin, $26 ; total, #$27 ou 39 bytes par entrée.

Position relative des entrées dans un bloc

Points d'entrée des noms de fichiers dans le Directory-Block

Numéro d'ordre des entrées	*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	$0EE	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

DEUXIEME BLOC DU DIRECTORY

- Block 00 03

Bytes	*Contenu*
$00-$01	Pointeur arrière - LByte, HByte. Pointe le bloc précédent : 02 00 ; $0002 = bloc 02.
$02-$03	Pointeur avant - LByte, HByte. Pointe le bloc suivant : 04 00 ; $0004 = bloc 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.

TROISIEME BLOC DU DIRECTORY

- Block 00 04

Bytes	*Contenu*
$00-$01	Pointeur arrière - LByte, HByte. Pointe le bloc précédent : 03 00 ; $0003 = bloc 03.
$02-$03	Pointeur avant - LByte, HByte. Pointe le bloc suivant : 05 00 ; $0005 = bloc 05.
$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.

QUATRIEME BLOC DU DIRECTORY

- Block 00 05

Bytes	*Contenu*
$00-$01	Pointeur arrière - LByte, HByte. Pointe le bloc précédent : 04 00 ; $0004 = bloc 04.
$02-$03	Pointeur avant - LByte, HByte. Pointe le bloc suivant. Ce pointeur est nul : dernier bloc.
$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

Entrée d'un nom de fichier

Remarque : la position des paramètres de chaque entrée d'un nom de fichier d'une table des matières est donnée en valeur relative par rapport à son entrée. Le pointeur débute avec $00 et se termine par $26 - #$27 ou 39 bytes pour chaque fichier. Chaque bloc peut contenir 13 noms de fichiers avec ses caractéristiques, le byte 512 étant toujours inutilisé. Une table des matières principale peut contenir un total de 52 noms de fichiers, le premier nom étant réservé à celui du Volume de la disquette.

Les explications concernant les paramètres déjà cités pour l'entrée du nom du Volume ne seront plus réitérées.

STRUCTURE D'UNE ENTREE D'UN NOM DE FICHIER

Entrée d'un nom de fichier
Position relative $00-$26

Position	*Contenu*
$00	Storage Type/Name Length. 4 bits de poids fort ayant pour valeur la caractéristique de sauvegarde du fichier. 4 bits de poids faible qui désignent la longueur du nom du fichier.
$01-$0F	Nom du fichier programme - 15 caractères maximum.
$10	File Type. Type du fichier - $0B = BIN.
$11-$12	Pointe le premier bloc du fichier. 12 00 = 0012 ; pointe le bloc $0012.
$13-$14	Blocks Used. Nombre total de blocs occupés par le fichier. 45 00 = 0045 ; Blocs occupés = $0045
$15-$17	EOF Pointer. Longueur d'un fichier texte. Fin d'un fichier binaire. 55 00 00 = $000055 bytes de longueur.
$18-$1B	Date & Time. Date et heure de création du fichier. Format : 00 00 00 00 (2 bytes pour la date et 2 bytes pour l'heure, avec une carte horloge).
$1C-$1D	Versions de ProDOS.
$1E	Access Byte. Paramètre d'accès au fichier.
$1F-$20	Aux.Info. Information auxiliaire : adresse d'implantation du programme par exemple.
$21-$24	Date & Time. Date et heure de la modification. Format : 00 00 00 00 (2 bytes pour la date et 2 bytes pour l'heure, avec une carte horloge).
$25-$26	Pointe le Key-Block d'un Directory, dans lequel se trouve sauvegardé le nom du fichier.

SIGNIFICATION PARTICULIERE DES BYTES

- Byte $00 : Storage Type/Name Length.

  L'instruction DELETE met ce byte à zéro; les bytes $13-$14 - Blocks Used - sont remis à zéro pour marquer l'occupation comme nulle et les autres données dans l'entrée restent intactes. Cette structure de la table des matières diffère du DOS 3.3, qui permet un undelete du nom d'un fichier avec l'aide d'un utilitaire approprié. ProDOS laisse très peu de chance pour une telle récupération de fichier, 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 : Name.

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

- Bytes $10 : File Type.

  Emplacement réservé pour le type du fichier dont le nom est sauvegardé aux bytes $01-$0F. En annexe 4 se trouve la table complète des types de fichiers redéfinissables sous PRODOS et SOS. File Type ne doit pas être confondu avec Storage Type. ProDOS gère les types de fichiers suivants : $00, $04, $06, $0F, $C0-$EF, $F0, $F1-$F8, $F9-$FF.

Remarques sur les fichiers programmes :

- Les fichiers Intéger 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 parmi la table des types de fichiers (Technical Manual).

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

- TXT désigne un fichier du type TEXTE. ProDOS sauvegarde les données différemment que 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, fait automatiquement la conversion entre les deux systèmes.

Les routines COUT ($FDED) et RDKEY ($FD0C) ne permettent plus un accès aux fichiers texte.

- Les fichiers binaires sont représentés par BIN. Sous DOS 3.3, la sauvegarde d'un programme binaire nécessite l'adjonction de deux paramètres : son adresse d'implantation et sa longueur (A$Adresse,L$Longueur). Lors d'une sauvegarde sur disquette, ces valeurs figurent en même temps que les données du programme, plus précisément au début du premier secteur alloué : deux bytes pour l'adresse et deux bytes pour la longueur.
  Sous ProDOS, l'adresse d'implantation se trouve aux bytes relatifs $1F-$20 - Aux.Info - et la longueur aux bytes relatifs $15-$17 - EOF Pointer. Ces paramètres ne sont plus enregistrés sur le premier secteur des données du programme.

- Le fichier BAS est l'équivalent du fichier A - AppleSoft sous DOS 3.3. Comme avec les fichiers BIN, la longueur n'est plus sauvegardée sur le premier secteur des données du programme, mais stockée aux bytes relatifs $15-$17 - EOF Pointer. Fait nouveau sous ProDOS : l'adresse d'implantation d'un programme AppleSoft se trouve stockée aux bytes relatifs $1F-$20 - Aux.Info.

- Le fichier VAR est nouveau : il est créé par l'instruction STORE et sauvegarde les variables d'un programme en mémoire - Variables + Noms. L'instruction RESTORE rappelle en mémoire les variables et leurs noms sauvegardés dans un fichier du type VAR. L'avantage essentiel de cette instruction 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 -, 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 l'instruction : 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-directory -) et permet d'augmenter en nombre la capacité de sauvegarde des fichiers programmes ou Subdirectory.

• Bytes $11-$12 : Key Pointer.

Pointe le bloc des données du fichier lorsque celui-ci ne comporte pas plus de 512 bytes, ou un bloc index dans le cas contraire. Si la taille du fichier est supérieure à #$20000 bytes - 131 072 -, c'est un Master Block qui sera pointé ; ce dernier peut alors 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 Subdirectory.

• Bytes $15-$17 : EOF - End Of File

Trois bytes sont disponibles pour représenter la longueur d'un fichier : LByte, MByte, HByte. Cette structure permet de pointer des nombres 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 sera en base 10, à condition d'avoir gardé les mêmes unités.
Le pointeur EOF désigne un byte logique, non matérialisé sur la disquette. Il renseigne sur la longueur réelle d'un fichier. Le chapitre 6 traite le sujet plus en détail. Avec un Subdirectory, la valeur des blocs est multipliée par #$200 - 512.

• Bytes $18-$1B : Create Date et Time.

Sauvegarde de la date et de l'heure à la création d'un fichier, à condition qu'une carte horloge soit en place dans un des slots.

• Bytes $1C-$1D : PRODOS versions.

Versions de ProDOS et mini-version; seul le byte $1D est testé par la commande OPEN.

• Byte $1E : Access-Byte.

Identique à l'entrée d'un nom du Volume.

• Bytes $1F-$20 : Aux.Info.

Information auxiliaire: représente une adresse d'implantation d'un fichier binaire par exemple. Certains types de fichiers n'utilisent pas ces deux bytes.
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, l'adresse d'implantation en mémoire centrale est représentée (LByte, HByte).

• Bytes $21-$24 : Modified Date et Time.

La date et l'heure sont sauvegardées, ici, par les commandes MLI de CLOSE et WRITE, associées à des instructions de fichiers.

• Bytes $25-$26 : Header Pointer.

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

TABLE DES MATIERES AUXILIAIRE VOLUME-SUBDIRECTORY

Définition et termes utilisés

Appelé Subdirectory, sous-catalogue ou encore table des matières auxiliaire, ce type de fichier ne peut être créé que par la commande CREATE. Si une disquette a comme nom de Volume VOLUME, la marche à suivre pour créer un Subdirectory est la suivante :

PREFIX/VOLUME	Fixe le préfixe /VOLUME.
CREATE/VOLUME/SUBDIRECTORY	Crée le Subdirectory, ayant comme nom SUBDIRECTORY, et sauvegarde son nom dans le Volume-Directory. La déclaration de VOLUME dans notre cas est facultative.
PREFIX/VOLUME/SUBDIRECTORY	Fixe, comme nouveau Directory, la table des matières du nom de SUBDIRECTORY. La commande CAT affiche alors le Subdirectory.

Le nom du fichier Subdirectory - SUBDIRECTORY - se trouve sauvegardé dans le Volume-Directory - VOLUME : il pointe le Volume-Subdirectory – SUBDIRECTORY –, celui-ci occupe au départ un bloc - Key-Block Subdirectory.
Il est possible de sauvegarder, dans ce nouveau Volume, des fichiers programmes ou Subdirectory. Le fichier SUBDIRECTORY - du type DIR - sauvegardé dans le Volume-Directory, a comme Storage byte $DC, tandis que dans le Volume-Subdirectory, le nom du Volume SUBDIRECTORY a comme valeur $EC. $D et $E représentent Storage Type et $C Name Length.

STRUCTURE D'UN VOLUME-SUBDIRECTORY

Entrée d'un nom de Volume du Subdirectory
Header-Entry - Key-Block

Les 4 premiers bytes sont réservés aux pointeurs arrière et avant, sur le même principe que le 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 de la première entrée d'une table des matières auxiliaire - Volume-Subdirectory.

A l'encontre du Volume-Directory, le Volume-Subdirectory n'a qu'un nombre limité de blocs. Lorsque le Subdirectory est créé par CREATE - BASIC.SYSTEM + commande MLI -, le système lui attribue un seul bloc. S'il arrive par la suite que plus de 13 noms de fichiers doivent y être sauvegardés, un deuxième bloc sera chaîné au premier (nom du Volume-Subdirectory compris).

Etant donné la structure de chaînage d'un Subdirectory, les blocs ne peuvent en aucun cas se suivre dans le sens ascendant.

Bytes	*Contenu*
$00-$01	Pointeur arrière - 00 00 = $0000.
$02-$03	Pointeur avant - 00 00 = $0000.
$04	Storage Type/Name Length. 4 bits de poids fort qui caractérisent le type de sauvegarde. 4 bits de poids faible qui désignent la longueur du nom du Volume-Subdirectory.
$05-$13	Nom du Volume-Subdirectory - SUBDIRECTORY dans notre exemple précédent. Maximum 15 caractères.
$14-$1B	Le nom de "uHUSTON!" est sauvegardé à cet endroit, mais il n'a aucune signification précise.
$1C-$1F	Create Date & Time. Date et heure de création du Volume-Subdirectory.
$20-$21	ProDOS versions.
$22	Access Byte. Paramètre d'accès au fichier - $C3.
$23	Longueur d'une entrée dans la table des matières - 39 ou #$27 bytes.
$24	Nombre d'entrées possibles par bloc (512/39 = 13 ou #$0D).
$25-$26	Nombre de fichiers actuellement sauvegardés dans le Volume-Subdirectory - 01 00 = #$0001 fichier.
$27-$28	Parent Pointer. Numéro du bloc de la table des matières où est sauvegardé le nom du fichier qui pointe ce Volume-Subdirectory. - 02 00 = $0002 = bloc 2, si le nom du fichier se trouve sauvegardédans le bloc 2.
$29	Parent Entry. Numéro de l'entrée dans le bloc parent.
$2A	Longueur d'une entrée dans la table des matières du parent, c'est-à-dire le Directory dans lequel se trouve le nom du fichier Subdirectory.

SIGNIFICATION PARTICULIERE DES BYTES

• Byte $04 : Storage Type/Name Length.

Détermine la caractéristique du nom du fichier lors de sa sauvegarde. Pour un nom de Volume-Subdirectory, la valeur de Storage Type - 4 bits de poids fort - est de $E.

• Bytes $14-$1B : Sigle "uHUSTON!".

Dans un Volume-Directory, ces 8 bytes sont inutilisés, leurs valeurs étant nulles. Le nom "uHUSTON!" se trouve aussi à l'adresse $EFA7 - carte langage - dans une table qui contient entre autres, une réplique provisoire du Subdirectory.

Dans un Subdirectory - Header ou première entrée du Key-Block - le byte $14 est testé ; il doit avoir un minimum de 5 bits à 1, sinon un message d'erreur sera renvoyé : ERR#4E INCOMPATIBLE FILE FORMAT. Le concepteur a sûrement une idée derrière la tête, au sujet de la routine qui effectue la vérification ; quelle surprise nous réserve-t-il ?

- Bytes $27-$28 : Parent Pointer.

  Pointe le bloc du Directory, où se trouve le nom du fichier qui a donné naissance à ce Subdirectory. Si le pointeur a comme valeur $02, le nom du fichier qui a donné naissance au Volume-Subdirectory se trouve stocké dans le bloc 2 du Volume-Directory.

- Byte $29 : Parent Entry Number.

  Le byte désigne le numéro de l'entrée du nom du fichier Subdirectory dans la table des matières qui le contient (bloc parent). Si par exemple ce byte a la valeur $03, le nom SUBDIRECTORY se trouve alors à la troisième position dans la table des matières qui le contient.

- Byte $2A : Parent Entry Length.

  Indique le nombre de bytes alloués pour caractériser l'entrée dans la table des matières qui contient le nom du fichier. Cette valeur attribuée est #$27 - 39 - bytes.

BLOCS INDEX

Généralités

Sous DOS 3.3, le bloc index - Index Block - s'appelle Track Sector List.

1 bloc = 2 secteurs = 512 bytes.

Une disquette ProDOS comporte des blocs de données d'une taille de 512 bytes.

Désignation d'un bloc de données :

- la table des matières principale
  - Volume-Directory ;
- la table des matières auxiliaire
  - Volume-Subdirectory ;
- un fichier de données.

Blocs répertoires

- Index Blocks

Il existe deux types de blocs index :

- Master Block : c'est un bloc répertoire principal, qui peut pointer 128 blocs répertoires auxiliaires - Index Blocks - dont chacun peut à nouveau pointer 256 blocs de données. Ce type de bloc pointe toujours un bloc index, mais jamais un bloc de données d'un programme.
  Un fichier avec un Storage Type de valeur #$3 contient un Master Block qui pointe un ou plusieurs blocs index. Cette limitation de blocs index est due au maximum de 24 bits alloués au pointeur EOF - longueur d'un fichier. La grandeur d'un tel pointeur est : 128*256*512 = 16777226 = 16 mégabytes, où 128 représente le maximum de blocs index, 256 le maximum de blocs de données et 512 le nombre de bytes par bloc.
  Un Master Block ne se différencie pas visuellement - Dump par exemple - d'un bloc index, sa structure interne reste la même.

- Index Block : c'est un bloc répertoire auxiliaire encore appelé bloc index ; il se compose d'un bloc de 512 bytes et pointe un maximum de 256 blocs de données. Ce type de bloc pointe toujours un bloc de données.
  Un fichier, avec un Storage Type de valeur #$2, contient un seul bloc index, tandis qu'avec un Storage Type #$3, la capacité est de 128 blocs index.

Ces deux types d'index - Master Block et Bloc Index - ont la même structure de sauvegarde interne des pointeurs : la table répertoire se divise en deux parties, ayant chacune une capacité de #$100 bytes - 1 secteur par demi-bloc index. Le byte de poids faible du numéro d'un bloc se trouve stocké dans la première partie du bloc index - côté gauche -, et le byte de poids fort, dans la seconde partie du bloc index - côté droit.

Exemple :

Posons le problème suivant : à la position relative $0000 d'un bloc index se trouve stocké LByte de valeur #$11, et à la position relative $0100 se trouve HByte de valeur #$01 ; dans ce cas, le pointeur a comme valeur 11 01 = $0111 - soit 273. La position relative $0100 est le premier byte de la deuxième partie du bloc index - côté droit - par rapport au début de la table répertoire - byte relatif $0000. Pour les numéros des blocs compris entre $0000 et $00FF - 0-255 -, le byte de poids fort est nul - HByte = #$00.

Différentes configurations du Directory

DISQUETTE VENANT D'ETRE FORMATEE

Structure du Directory :

blocs 0-1	Loader.
blocs 2-5	Volume-Directory.
bloc 6	Volume Bit-Map.
blocs 7-279	Libres.

SAUVEGARDE D'UN FICHIER DU TYPE SEEDLING

Lorsqu'un nom de fichier programme est sauvegardé sur la disquette par la commande CREATE, le système lui alloue d'office un bloc. Le nom sera stocké en deuxième position dans la table des matières du Volume-Directory, à partir du byte #$2B - 43 - et pointe le bloc 0007. C'est le premier bloc de données alloué au fichier, vide pour l'instant.

Création d'un nom de fichier par CREATE :

CREATE/VOLUME/PROGRAMME,TBAS

CREATE	Commande ProDOS : elle permet de créer un fichier vide de données.
/VOLUME/	Nom du Volume-Directory.
PROGRAMME	Nom du fichier.
TBAS	Détermine le type de fichier : BAS = Basic.

Structure du Directory après la création du fichier :

blocs 0-1	Loader.
blocs 2-5	Volume-Directory.
bloc 6	Volume Bit-Map.
bloc 7	Réservé pour les 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 nom de fichier pointe plus de 512 bytes de données - 1 bloc -, le système crée un bloc index qui pointe à son tour des blocs de données - 256 blocs de données au maximum. 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 d'un fichier Sapling de taille minimale :

blocs 0-1	Loader.
blocs 2-5	Volume-Directory.
bloc 6	Volume Bit-Map.
bloc 7	Premier bloc de données.
bloc 8	Bloc index.
bloc 9	Deuxième bloc de données.
blocs10-279	Libres.

Allocation des blocs :

1 bloc index.
2 blocs de données.
Total : 3 blocs.

SAUVEGARDE D'UN FICHIER DU TYPE TREE

Dès que la capacité d'un fichier dépasse les 128 Ko, il nécessite 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 mégabytes - 16 777 716 bytes.

Exemple de création d'un fichier Tree :

Le programme Basic qui suit permet de créer sur une disquette un fichier texte avec la structure suivante :
- 1 Master Block ;
- 2 blocs index ;
- 257 blocs de données.

L'écriture sur la disquette 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 Bloc et 2 blocs index
50    FOR X = 1000000 TO 1016384 : PRINT X
60    NEXT X
70    PRINT CHR$ (4) ; "CLOSE FICHIER"
```

Structure du Directory après la création d'un fichier Tree de capacité minimale :

blocs 0-1	Loader.
Blocs 2-5	Volume-Directory.
bloc 6	Volume Bit-Map.
bloc 7	Premier bloc de données.
bloc 8	Premier bloc index.
bloc 9	Deuxième bloc de données.
bloc 10	Troisième bloc de données.
	.....

.....

bloc 263	256e bloc de données.
bloc 264	1 Master Block - Key-Block.
bloc 265	Deuxième bloc index.
bloc 266	257e bloc de données.
blocs 267-279	Libres.

Allocation des blocs :

1 bloc index principal - Master Blocks.
2 blocs index auxiliaire - Index Blocks.
257 blocs de données.
Total : 260 blocs.

LES TAMPONS ALLOUES A ProDOS

FCB, VCB et VBM

ProDOS sauvegarde en mémoire différentes données, auxquelles les fichiers et les périphériques pourront se référer par la suite. Ce sont des caractéristiques placées dans des tables bien structurées, où chaque élément d'une donnée a une signification précise. C'est en sorte un pense-bête structuré, qui regroupe plusieurs tables ou pointeurs de la mémoire - Memory Layout.

STRUCTURE ET ADRESSES DES TABLES

- Memory Layout pour un Apple II - 64 Ko

$F000-$F0FF	$100 (256) bytes pour la sauvegarde des variables des sous-programmes MLI.
$F100-$F1FF actuel (Pathname).	$100 (256) bytes pour la sauvegarde du préfixe (PREFIX) et du chemin
$F200-$F2FF	$100 (256) bytes pour la sauvegarde du Volume Control Block (VCB) : 8 entrées maximum, à raison de $20 (32) bytes par entrée.
$F300-$F3FF	$100 (256) bytes pour la sauvegarde du File Control Block (FCB) : 8 entrées maximum, à raison de $20 (32) bytes par entrée.
$F400-$F5FF	$200 (512) bytes pour la sauvegarde de la Volume Bit-Map (VBM) actuelle.
$F600-$F7FF	$200 (512) bytes pour la sauvegarde des Volume-Directory et Volume-Subdirectory actuels.

FCB - File Control Block

File Control Block - FCB - est un espace mémoire réservé pour les entrées des fichiers de données, ou de Directory ; il se trouve aux adresses $F300-$F3FF.

Cette structure est uniquement implantée en mémoire vive - RAM - et n'a rien de commun avec les entrées d'un fichier de données ou de Directory d'une disquette.
Lorsqu'un fichier est ouvert par la commande OPEN - qu'il s'agisse d'un fichier de données, de Directory ou de Subdirectory - le système sauvegarde certaines caractéristiques de ce fichier, à un emplacement mémoire appelé File Control Block ou FCB. Divers renseignements sont alors disponibles pour ProDOS : numéro du bloc d'entrée du Volume-Directory, numéro du premier bloc de données, pointeur à l'intérieur d'un fichier - Mark -, pointeur de la fin d'un fichier - EOF -, byte de référence du périphérique en ligne - Unit Number -, etc.

Un fichier de données ProDOS, contrairement au DOS 3.3, est toujours créé par la commande MLI CREATE. Le FCB a une capacité maximale de 8 entrées, ce qui correspond à une zone mémoire ayant une possibilité de sauvegarde pour les caractéristiques de 8 fichiers.
Un maximum de 8 fichiers peuvent être ouverts simultanément sous ProDOS, d'où le nombre maximum de 8 entrées autorisées.
Chaque entrée ampute la mémoire de $#20 (32) bytes, ce qui totalise #$100 (256) bytes pour l'espace mémoire alloué à FCB.
Si l'on a une suite de plusieurs entrées dans la table FCB, elles s'échelonnent d'une manière graduelle et progressive, c'est-à-dire que la première entrée débute à l'adresse $F300, la deuxième entrée à l'adresse $F320, etc.
FCB permet à la routine GET.FILE de se documenter sur certaines particularités d'un fichier.

ENTREE D'UN FILE CONTROL BLOCK

L'entrée FCB traitée par la suite n'est fournie qu'à titre d'exemple et, de ce fait, les adresses ne contiennent pas de valeur numérique. Pos donne la position relative du byte à l'intérieur de l'entrée FCB ; Adres donne l'adresse réelle de la première entrée, et Signification renseigne brièvement sur le sens à donner.

Pos. Adres.	*Signification*
$00 F300 -	File Reference Number. Un byte : utilisé comme référence interne du MLI, indique le numéro d'ordre dans les entrées de FCB.
$01 F301 -	Unit Number - DSSS0000. Un byte : renseigne le MLI sur le périphérique en ligne.
$02 F302 -	Key-Block Number. Deux bytes qui indiquent le numéro du bloc - Key-Block - du Directory où se trouve sauvegardé le nom d'un fichier nommé DATA par exemple.
$04 F30 4 -	Directory-Block. Deux bytes qui indiquent le numéro du bloc où se trouve réellement sauvegardée l'entrée du fichier au nom de DATA cité comme exemple.
$06 F306 -	Entry Number. Un byte : DATA 2 est la deuxième entrée à l'intérieur de ce même bloc - DATA 2 étant un nom de fichier cité comme exemple.
$07 F307 -	Storage Type. Un byte : il caractérise le genre du fichier DATA ; ce n'est en fait qu'une partie de Storage Type, et plus précisément 4 bits qui sont utilisés. Le byte pouvant prendre les valeurs suivantes : $01, $02, 03, $0D, $0E ou $0F.

$08 F308 - Statut byte. Un byte : il représente l'image binaire de plusieurs données (Index bloc modifié, bloc de données modifié, fichier modifié depuis que la commande OPEN a été déclarée, etc.).
$09 F309 - Access byte. Un byte : il caractérise la façon dont un fichier sera abordé par la suite : écriture autorisée, lecture seule, etc.
$0A F30A - Newline Character. Un byte : il est déterminé par la fonction NEWLINE, et désigne le dernier caractère à lire dans un fichier.
$0B F30B - Buffer Index. Un byte : numéro d'ordre du tampon dont le pointeur se trouve sauvegardé dans la page globale de PRODOS.
$0C F30C - Data Pointer. Deux bytes - LByte, HByte ; pointent le premier bloc des données du fichier DATA.
$0E F30E - Index Block Number. Deux bytes - LByte, HByte ; cette paire de bytes est nulle si le fichier DATA ne possède pas de répertoire des blocs - fichier de moins de 512 bytes de longueur - sinon, ces deux bytes indiquent le nombre de blocs index.
$10 F310 - First Block. Deux bytes ; c'est le numéro du premier bloc qui se trouve en mémoire.
$12 F312 - Pointeur MARK. Trois bytes - LByte, MByte, HByte ; pointent la position relative dans un fichier.
$15 F315 - Pointeur EOF. Trois bytes - LByte, MByte, HByte ; désignent la longueur d'un fichier texte, qui correspond à la position absolue dans un fichier.
$18 F318 - Blocks Used. Deux bytes - LByte, HByte ; désignent le nombre de blocs occupés par le fichier.
$1A F31A - Un byte inutilisé.
$1B F31B - Level. Un byte : utilisé par les commandes CLOSE et FLUSH ; c'est la copie de l'adresse $BF94 de la page globale de PRODOS.
$1C F31C - Flag Byte - Drapeau. Un byte : prend la valeur #$00, pour un fichier fermé, et #$80, pour un fichier ouvert ou modifié.
$1D F31D - Deux bytes inutilisés.
$1F F31F - Newline Enable Mask. Un byte de masque, utilisé par NEWLINE ($C9).

SIGNIFICATION DETAILLEE DES BYTES

• Byte $00 - Reference Number.

Ce byte est généré par le MLI, et n'a qu'une fonction interne à celui-ci. La première entrée de FCB, se situant à l'adresse $F300, aura toujours comme numéro de référence 01, tandis que l'entrée suivante, se situant à l'adresse $F320, aura comme numéro 2, et ainsi de suite. Comme le MLI ne possède pas de tampon mémoire interne où ce dernier puisse sauvegarder les noms des fichiers programmes, le numéro de référence joue un rôle primordial pour l'environnement externe. A chaque fois qu'un fichier ouvert sera sollicité, il sera fait référence à ce numéro, qui aura été attribué au préalable par la commande OPEN. Ce numéro sera retourné au sous-programme appelant comme paramètre identificateur du fichier traité.

• Bytes $02-$06 - Directory et Key-Blocks.

Si le nom d'un fichier ne se trouve pas dans le Volume-Directory - Key-Block = bloc $0002 -, les bytes relatifs $02-$03 seront de valeur différente au contenu des bytes

relatifs $04-$05. Le Key-Block - bloc numéro $0002 - du Volume-Directory est réservé pour une éventuelle extension de la structure du Directory. Les bytes $02-$05 sont utilisés pour permettre au système de retrouver, à l'intérieur des blocs alloués à un Directory, un nom de fichier, lorsque les commandes CLOSE et FLUSH sont utilisées.

• Byte $07 - Storage type.

C'est une partie du byte utilisé par FCB : en l'occurrence, les 4 bits de poids faible représentatifs pour le type de fichier. Les 4 bits de poids fort restent toujours à zéro.

Représentation des bits :

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

où les 4 bits représentés par x peuvent prendre toute valeur comprise entre $0-$F inclus.

• Byte $08 - Statut byte.

C'est la représentation binaire de certaines caractéristiques spécifiques à un fichier.

Représentation du Statut byte :

```
Bits :  7 6 5 4 3 2 1 0
        x x 0 x x x x x   x = Valeur 1 ou 0
        Bit 5 = toujours 0
```

Bit 7 à 1 — Un bloc index se trouve encore en mémoire, et devra être sauvegardé avant qu'un autre bloc index soit chargé.
Bit 6 à 1 — Un bloc de données se trouve encore en mémoire, et devra être sauvegardé avant de charger un autre fichier de données.
Bit 5 — Inutilisé, et, de ce fait, se trouve toujours à 0.
Bit 4 à 1 — Un fichier ouvert par la commande OPEN a été modifié. Est utilisé comme test de fonction pour contrôler la disponibilité du drive - DriveReady. Idem pour la commande FLUSH. Lorsque les bits 6, 5 et 4 sont nuls, ainsi que le byte relatif $1C de l'entrée FCB - Flag Byte -, une sauvegarde de l'entrée du Directory ne sera pas nécessaire. Cette situation est présente avec la commande READ.
Bit 3 à 1 — Positionné lorsque le byte Storage Type utilise un fichier du type supérieur au paramètre $3. Ce paramètre n'est pas testé par le MLI.
Bit 2 à 1 — Positionné lorsque le byte Storage Type traite un fichier du type $3 - la taille maximale du type $2 est dépassée.
Bit 1 à 1 — Positionné lorsque le Storage Type traite un fichier du type $2, ou a nécessité un bloc index supplémentaire.
Bit 0 à 1 — Positionné lorsqu'un nouveau bloc de données a été nécessaire.

Remarque : le byte Storage Type représente un type de fichier caractérisé par sa taille en nombre de bytes - données. Il prend plusieurs significations suivant que celui-ci comporte une taille définie par le concepteur - Voir le chapitre 3 qui traite Storage Type plus en détail.

Règle fondamentale :

Type 1 = ou > type 2 : 1 bloc index + 1 nouveau bloc de données.
Type 2 = ou > type 3 : 1 bloc principal + 1 bloc index auxiliaire + 1 nouveau bloc de données.

- Byte $09 - Access Byte.

Une copie en partie du byte Access de l'entrée fichier de la disquette : les bits Write et Read sont seuls utilisés. L'accès à un fichier est autorisé lorsque le bit concerné est positionné à 1.

Représentation du byte :

Bits :	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	W	R	W = Write ; R = Read
	0	0	0	0	0	0	1	1	= $03 Lecture/écriture
	0	0	0	0	0	0	0	1	= $01 Lecture
	0	0	0	0	0	0	1	0	= $02 écriture

- Byte $0A et $1F - Newline.

Ces bytes sont déterminés par la commande NEWLINE, associée à un fichier ouvert. Le BASIC.SYSTEM fixe comme valeur #$7F à Enable Mask, et #$0D à Newline Character.

- Byte $0B - Buffer Index.

La page globale de PRODOS comporte une zone réservée pour la sauvegarde de 8 pointeurs de tampons, alloués à des fichiers ouverts. Chaque fichier ouvert nécessite un tampon mémoire de 1024 octets, et OPEN lui attribue en même temps un numéro de référence, qui le dissociera par rapport à un autre fichier ouvert - paramètre du programme appelant.

L'espace mémoire alloué à la table paramétrique des pointeurs tampons se situe aux adresses $BF70-$BF7F.

L'index tampon (Buffer) de l'entrée FCB - byte relatif $0B - décrit ici équivaut à la valeur référence attribuée à un fichier ouvert, multipliée par deux. Pour trouver la valeur référence, il faudra diviser par 2 la valeur de Buffer Index.

- Bytes $0C-$0D - Data Pointer.

C'est le numéro du premier bloc des données du fichier; il dépend du byte Storage Type, et peut pointer : soit un bloc principal - Master Index Block -, soit un bloc auxiliaire- Index Block -, soit un bloc de données - Data Block -, soit un bloc d'entrée d'un Subdirectory - Key-Block.

- Bytes $12-$14 - Pointeur Mark.

C'est la position relative pointée à l'intérieur d'un fichier, et qui correspond à un byte physique. Le système lui attribue 3 bytes, ce qui nécessite quelques explications :

- un volume peut avoir une taille maximale de 32 mégabytes ;
- un fichier peut avoir une taille maximale de 16 mégabytes ;
- pour la numérotation des blocs, 16 bits sont réservés, ce qui autorise un nombre de blocs maximum de : 256*256 = 65 536 blocs de 512 bytes chacun ;
- pour le pointeur de la position interne du fichier - Mark -, 24 bits sont réservés, ce qui permet de pointer : 256*256 = 65 536 blocs de 256 bytes ;
- le calcul du numéro de bloc déterminé par la position du pointeur Mark s'effectue de la manière suivante : Position/2 = Index à l'intérieur d'un bloc Index. Exemple : si le pointeur Mark affiche 00 66 xx, le résultat sera le #$33^{e} bloc à l'intérieur du bloc Index.

- Bytes $15-$17 - Pointeur EOF.

C'est un pointeur qui désigne un byte logique à l'intérieur d'un fichier ; c'est la position absolue codée par 3 bytes. EOF pointe la fin d'un fichier, directement derrière le byte ayant comme valeur #$0D - Return - de son dernier enregistrement. C'est un byte fictif et non visible sur la disquette, utilisé pour marquer la longueur totale des enregistrements d'un fichier.

- Bytes $18-$19 - Blocks Used.

C'est la somme de tous les blocs occupés sur la disquette ; elle regroupe les blocs index et les blocs de données. C'est le MLI qui effectue la somme de ces différents blocs occupés.

- Byte $1B - Level.

Ce byte est une copie de l'adresse $BF94 de la page globale de PRODOS, et utilisé par le système Basic. Il désigne le niveau de la commande CLOSE ou FLUSH, pour la fermeture des fichiers à partir du numéro stocké dans Level.

- Byte $1C - Flag.

C'est un byte utilisé par le MLI comme drapeau pour la commande FLUSH. Lorsqu'un fichier est abordé avec la commande WRITE ou SET.EOF, le système effectue toujours, au retour de ces sous-programmes, une écriture au niveau de la disquette, en ayant au préalable testé le byte du Statut - byte relatif $08 de FCB.

• Byte $1F - Newline Byte.

C'est un byte utilisé par la commande NEWLINE, pour effectuer un AND - ET Logique - pour tester un caractère - Newline Character.
(Voir le chapitre 6 qui traite plus en détail les paramètres et commandes du MLI.)

VCB - Volume Control Block

Volume Control Block désigne les entrées des périphériques - drives - en ligne, et se trouve aux adresses $F200-$F2FF. Comme FCB, cette structure existe uniquement dans la mémoire de l'Apple, et n'a rien de commun avec les entrées des fichiers de données, ou Directory de la disquette.

Un maximum de 8 entrées sont disponibles, et chacune occupe #$20 - 32 - bytes, ce qui permet au système de mémoriser 8 lecteurs de disquettes.
L'occupation de l'espace mémoire alloué au VCB est dégressive, c'est-à-dire que la première entrée se fera à l'adresse $F2E0, la deuxième à l'adresse $F2C0, etc.

La raison d'être de VCB est essentiellement la notion de Volume des disquettes mises en place dans les drives en ligne. C'est ainsi que le nom du volume d'une disquette, placé en en-tête d'un Volume-Directory, sera copié aux bytes relatifs $01-$0F (1-15) de l'entrée VCB, tandis que le numéro de bloc du Key-Block sera copié en $16 (22).

La commande PREFIX ne modifie en rien cette entrée ; par contre, SET.PREFIX, associé à une commande d'un Subdirectory, obligera le MLI à utiliser certaines des informations de l'entrée VCB : numéro des blocs de la VBM, capacité totale en blocs, etc.

RECHERCHE DETAILLEE D'UN VOLUME NOUVEAU

C'est ainsi que la commande PREFIX/VOLUME/SUBDIR, où VOLUME désigne une disquette venant d'être mise en place dans le drive 1 du slot 6 - encore inconnue du MLI- déclenchera le processus suivant :

- Le chemin - 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 FCB - disquette nouvellement introduite dans le lecteur. C'est alors que le système consulte l'un après l'autre tous les paramètres - Device Table - de la page globale : il recherche en premier une entrée libre dans VCB, puis lit le paramètre Unit Number, qui représente l'image binaire du byte d'un périphérique en ligne - Voir commentaire du listing de la page globale de PRODOS au chapitre 5.

- Le nom du Volume-Directory trouvé - nom du Volume - sera comparé avec toutes les entrées de 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 inconnue, la commande n'aboutissait pas, une des erreurs suivantes serait renvoyée : VOLUME CONTROL BLOCK TABLE FULL, ou VOLUME NOT FOUND. Cette dernière erreur pouvant survenir, si le nom VOLUME n'était pas trouvé en en-tête du 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 Volume principal).

- 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 - R sera placé en $F1FF. Le byte Storage Type sera copié à l'adresse $F100 et dans PREFIX - adresse $BF91 -, qui se situe à la page globale de PRODOS. Exemple de Storage Type : $F6 = VOLUME, où F représente un fichier Directory, et 6 le nombre de caractères du mot VOLUME.

- Ensuite ProDOS sauvegarde le numéro du Subdirectory-Block - Key-Block - SUBDIR, aux adresses $F060-$F061 - et non en VCB comme on aurait pu s'y attendre.

- Le processus de la routine de recherche SET.PREFIX prend fin, et le retour se fera - via la page globale de PRODOS - au sous-programme appelant.

ENTREE D'UN VOLUME CONTROLE BLOCK

Pos. Adres.	*Signification*
$00 F2E0 -	Name Length. Un byte : désigne la longueur du nom d'un fichier.
$01 F2E1 -	Name. Nom du Volume-Directory, avec un maximum de 15 caractères.
$10 F2F0 -	Unit Number - DSSS0000. Un byte qui renseigne sur le périphérique en ligne - Drive et slot.
$11 F2F1 -	Flag Byte - Drapeau. Un byte qui informe que le fichier est ouvert ou fermé : dans le premier cas le byte prendra comme valeur #$80, et dans le second cas #$00.
$12 F2F2 -	Total Blocks. Deux bytes qui désignent le nombre total des blocs alloués à la disquette ; c'est la capacité maximale de sauvegarde - $0118 = 280 blocs pour un disque 35 pistes standard.
$14 F2F4 -	Free Blocks. Deux bytes qui renseignent sur le nombre de blocs disponibles de la disquette.
$16 F2F6 -	Blocks Number Key-Blocks. Deux bytes qui renseignent sur la position du Key-Block - Volume-Directory - ($0002 normalement).

$18 F2F8 -	Deux bytes inutilisés.
$1A F2FA -	VBM Block. Deux bytes - LByte, HByte - qui pointent le premier bloc de la Bit-Map - normalement bloc $0006.
$1C F2FC -	VBM Extent Number. Un byte qui renseigne sur le numéro – valeur relative du bloc de la Bit-Map. 00 = premier bloc de la VBM. VBM Extent est une configuration engendrée par un Volume qui nécessite plus d'un bloc pour sa Bit-Map.
$1D F2FD -	Un byte inutilisé.
$1E F2FE -	File Open. Un byte qui désigne le nombre de fichiers ouverts à cet instant.
$1F F2FF -	Un byte inutilisé.

VBM - Volume Bit-Map

A ne pas confondre avec la System Bit-Map implantée dans la page globale de PRODOS, aux adresses $BF58-$BF6F. L'espace mémoire $F400-$F5FF - VBM - gère l'occupation des blocs de la disquette. Chaque bloc concerné de la disquette sera représenté par un bit : il sera positionné à 1 lorsque le bloc qui lui est attribué sera libre, et à 0 lorsqu'il sera occupé. C'est, en somme, une réplique de la Bit-Map d'une disquette, tenue à jour par le système.

Contenu des premiers blocs d'une disquette :

Bloc	$00	Loader pour ProDOS - Piste $00, secteurs $00-$02.
Bloc	$01	Loader pour SOS.
Blocs	$02-$05	Volume-Directory - 4 blocs.
Bloc	$06	Volume Bit-Map - VBM.
Bloc	$07	Libre.

Un bloc est formé de 512 bytes, et à l'intérieur de la VBM, chaque byte représente 8 blocs. C'est ainsi qu'une VBM peut gérer (#$200*8) #$1000 blocs de données, ce qui équivaut à 2 mégabytes. Pour une disquette standard 35 pistes, cela suffit largement, mais pour un disque dur du type Profile, ayant une capacité de 5 mégabytes, il faudra une VBM étendue - 3 blocs VBM seront alors nécessaires, ce qui impliquera l'occupation des blocs $07 et $08.

Ce sont les commandes CREATE/ WRITE qui marquent les blocs occupés, et les commandes DESTROY/ SET.EOF qui libèrent les blocs.

Le Volume-Directory contient un pointeur - Parent Pointer - qui désigne le premier bloc VBM - lequel est d'ailleurs l'unique bloc Bit-Map pour une disquette standard.

DUMP de la Bit Map implantée dans la VBM :

```
$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 : 01234567
Bits :  01234567
        00000001    = $01
                    = 1 bloc libre - bloc 7
```

C'est le 7e bit qui représente le bloc 7 de la disquette.

Représentation du byte suivant - $01 :

```
Blocs :  89ABCDEF
Bits :   89ABCDEF
         111 11 111  = $FF
                     = 8 blocs libres (blocs 8-15)
```

SAUVEGARDE SOUS ProDOS

Capacité de stockage

Au départ, ProDOS fut mis au point pour un disque dur du type Profile, pour la gestion des gros fichiers. Par la suite, il fut réécrit pour le IIe et le IIc avec une capacité mémoire de 64 Ko minimum. Sa configuration est telle qu'il permet la gestion de périphériques ayant une capacité de sauvegarde de 32 mégaoctets, dont un fichier seul peut avoir une longueur de 16 mégaoctets. 1 octet = un caractère ; 1 Ko = 1024 octets ; 1 mégaoctet =

1024 * 1024 = 1 048 576 octets ; 16 mégaoctets = 16 777 216 octets. On constate que ce système ouvre de nouvelles perspectives, en 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. Certaines de ces applications portent déjà leurs fruits : le IIe nouveau modèle, équipé du processeur 65C02, où l'on peut désormais raccorder un lecteur d'une capacité formatée de 800 Ko. On est bien loin du format DOS 3.3, dont la routine RWTS fut écrite au départ pour des lecteurs de disquettes standard du type Disk II, avec une capacité formatée de 124 Ko. Dans le cas de ProDOS, l'étude fut au préalable réalisée pour un disque dur (32 mégaoctets maximum).

Blocs et secteurs

Sous DOS 3.3, la routine RWTS peut effectuer des manipulations au niveau du secteur, avec un minimum de 256 bytes - 1 secteur = 256 bytes. Cette pratique permet un certain nombre de manipulations et d'astuces dont le but essentiel est de pouvoir effectuer des manœuvres d'écritures et de lectures sur une disquette.

Sous ProDOS, cette notion est lègèrement modifiée, et c'est avec des blocs qu'il faudra dorénavant travailler. Un bloc se caractérise par une capacité de 512 - $0200 - bytes -1 bloc = 512 bytes.

OSSATURE DE ProDOS

Le système ProDOS se divise en deux parties essentielles : le Command Dispatcher and Block File et le Disk Driver.

1 - Le Command Dispatcher and Block File Manager : c'est l'équivalent du File Manager du DOS 3.3. Cette partie gère l'allocation des fichiers sur une disquette, et traite des blocs de données, donc 512 octets à chaque fois.

2 - Le DisK-Driver : il traite les disquettes au niveau des pistes/ secteurs. C'est la routine RWTB - Read Write Track Block -, qui est désormais incapable de formater un disque ; elle est rebaptisée Disk-Driver pour la circonstance. C'est la même routine que celle qui est utilisée dans les systèmes Pascal et CP/M, ce qui leur confère des similitudes frappantes avec PRODOS, dues essentiellement à une gestion identique des blocs. Avec l'utilisation d'un disque dur, cette routine DisK-Driver comporte un module différent.

ProDOS traite les données d'une disquette sous forme de blocs, dont chacun peut être comparé à une unité logique se composant de deux secteurs physiques. Une disquette formatée en 35 pistes comporte donc : 35 * 16 / 2 = 280 blocs.

NOTION DE VOLUME, DIRECTORY ET FICHIER

Le Volume-Directory

- Notion de Volume

La notion de Volume sous DOS 3.3 était plutôt une chose abstraite, et l'on pouvait tout au plus lui affecter un numéro pour l'opération formatage. Dans la majorité des cas, cette attribution d'un numéro s'effectuait par défaut, en donnant comme valeur 254 au Volume de la disquette. Cette manipulation s'opérait par l'intermédiaire d'une routine interne au système d'exploitation, qui débute à l'adresse $A54F - INIT Command. A l'adresse $A55E était alors chargé le nombre 254 par un LDA #254 qui correspond à la valeur par défaut du numéro du Volume.
Lorsque l'instruction CATALOG était déclarée avec une disquette en ligne et formatée sous DOS 3.3, le renseignement suivant s'affichait en haut et à gauche de l'écran :

DISK VOLUME 254

où 254 désigne le numéro de Volume de la disquette.

ProDOS ignore l'affectation du numéro d'un Volume, et traite une disquette d'après un nom - Volume-Directory - qui est celui de la table des matières principale. Chaque disquette ProDOS reçoit en cours du formatage un nom qui correspond au Volume-Directory.

Pour le traitement d'un fichier, le système effectue une recherche sélective d'après le nom du Volume et le nom du fichier. C'est en somme une structure très proche du système Apple Pascal, dont les habitués n'auront pas manqué de remarquer les similitudes. Ce moyen de recherche a un avantage capital : il permet à ProDOS de mémoriser les lecteurs en ligne, et de leur attribuer respectivement le nom de chaque Volume. Cela confère au système un moyen très élaboré pour la reconnaissance du chemin - Pathname - lors d'une recherche de fichier.

Les Volume-Directory et Volume-Subdirectory sont identifiés par un nom ; le choix de leur écriture est subordonné à certaines règles et restrictions.

LE NOM D'UN VOLUME

- Directory ou Subdirectory

Pour nommer un Volume, il faut :

- commencer par une barre oblique - Slash -, suivi par une lettre ;
- composer des lettres, chiffres et points :
  - lettres entre A et Z,
  - chiffres entre 0 et 9,
  - le point "." ;

- que le premier caractère soit une lettre ;
- ne pas comprendre d'espace ou de caractère de ponctuation autre que le point qui sert au format et à la présentation du nom ;
- ne pas dépasser 15 caractères au total, y compris le point, mais barre oblique exclue
- que les minuscules soient automatiquement converties en majuscules.

Voici quelques exemples valides :

/USERS.DISK
/UTILITIES
/COURRIER.18.09
/FACTURATION

et quelques exemples non autorisés :

/EXEMPLE.TROP.LONG	Plus de 15 caractères.
/BIG MAG	Un espace après BIG.
/5.UTILITAIRES	Débute par un chiffre.
/.COURRIER	Débute par un point.

Si vous adressez par erreur un nom de Volume non conforme, l'Apple vous retourne un message d'erreur, car le système contrôle à tout moment sa validité.

FORMATAGE D'UN VOLUME

Comme tout support de sauvegarde, une disquette ProDOS nécessite, avant toute sauvegarde ou lecture de données, une préparation de sa surface physique. Cette opération, appelée tantôt formatage, tantôt initialisation, divise la disquette en un certain nombre de pistes et de secteurs, puis y inscrit des données et des repères, qui sont des informations codées. Ces indices seront utilisés ultérieurement par le système pour permettre l'accès de la tête de lecture au-dessus des pistes et secteurs, à la recherche des données enregistrées.

Au niveau du logiciel, ProDOS crée un Directory, sorte de table des matières, avec un nom de Volume, pour y stocker différents fichiers ou programmes. Un Directory pourra à nouveau appeler un ou plusieurs autres Directory, qui prennent alors le nom de Subdirectory.

Pour un disque dur Profile, cette opération n'est généralement effectuée qu'une fois, lors de la mise en route de l'ensemble. Le formatage efface toutes les données inscrites sur le support considéré, et tout fichier ou programme initial serait irrémédiablement perdu.

Pour la série IIe et IIc, équipés de lecteurs de disquettes standard, l'opération formatage est réalisée par l'intermédiaire d'un logiciel fourni à l'achat du micro-ordinateur. La

disquette d'accompagnement comporte un certain nombre de programmes qui sont des aides pour l'utilisateur. Le menu principal se compose des programmes suivants :

? - AIDE : DES EXPLICATIONS
F - GESTIONNAIRE DE FICHIERS
C - CONVERSION DOS < - > PRODOS
P - OCCUPATION DES PORTS
D - DATE ET HEURE
B - BASIC APPLESOFT

En choisissant l'option F qui appelle le programme FILER, il vous sera possible de gérer globalement les fichiers sur une disquette.

POSSIBILITE DU FILER

Menu du FILER :

- File Commands.

  Ce sont les commandes concernant les fichiers.

- Volume Commands.

  Ce sont les commandes concernant les Volumes.

- Configuration Defaults.

  C'est un utilitaire de configuration de votre système.

- Quit. Pour quitter.

Ces différentes commandes permettent, entre autres :

- le formatage d'un Volume,
- la copie d'un Volume,
- la comparaison de deux Volumes,
- la création d'une disquette de boot,
- de lister les Volumes en ligne,
- la modification du nom d'un Volume,
- la détection des blocs défectueux,
- l'allocation des blocs d'un Volume.

Lorsqu'une disquette ProDOS vient d'être initialisée, elle comporte un nom de Volume stocké en en-tête du Volume-Directory, par exemple USERS.DISK, et un catalogue - Directory - pouvant contenir 51 noms de fichiers divers. Dans le cas d'une disquette de démarrage classique, le Directory contient au moins 3 fichiers : PRODOS, BASIC.SYSTEM et STARTUP.

Caractéristiques générales de PRODOS

- PRODOS permet de formater des Volumes ayant une capacité de 32 mégabytes.

- Un fichier peut avoir une capacité de 16 mégabytes.

- Organisation hiérarchisée des fichiers sur la disquette.

- Table des matières principale - Volume-Directory - pouvant avoir une capacité de 51 fichiers. Cette table des matières principale peut contenir à son tour une table des matières auxiliaire - Subdirectory -, ou des fichiers de données, ou encore des programmes.

- Table des matières auxiliaire, de taille illimitée, ne dépendant que du format du disque : ce sont les Subdirectory, et les Sub-Subdirectory, etc.

- Un chemin complet, partiel ou un préfixe, doit comporter au maximum 64 caractères.

- Préfixe et chemin partiel s'ajoutent au moment de la recherche d'un Subdirectory.

- D'origine, 7 types de fichiers standard sont définis par PRODOS ; en plus, il vous sera possible de définir 8 autres types de fichiers - $F1 à $F8. En annexe 4, on trouvera la table complète des différents types de fichiers redéfinissables sous les systèmes SOS et PRODOS.

- Possibilité d'ouvrir à partir d'un programme, et, à un instant donné, jusqu'à 8 fichiers différents.

- Possibilité d'accès à des blocs de données et à des fichiers Directory.

- Amélioration des instructions comme : APPEND, BLOAD, BRUN, BSAVE, CAT ou CATALOG, CHAIN, etc.

- Une instruction - FLUSH - permet de vider le tampon d'un fichier sur la disquette, sans le fermer.

- Des instructions comme STORE et RESTORE permettent de stocker des variables et leurs noms sur la disquette.

- Architecture structurée, composée d'une interface avec des périphériques normalisés - Thunderclock par exemple.

- Gestion de 4 interruptions.

- Plus grande rapidité de transfert des données disquette/ mémoire ou vice versa.

- Disque virtuel avec un IIc. L'Apple IIe nécessite alors une extension mémoire de 64 Ko (Carte 80 colonnes étendue par exemple).

Volume-Directory ou Subdirectory

- Notion de catalogue

Ce qui différencie une disquette ProDOS de tout autre système, c'est l'organisation hiérarchisée ou arborescente de la structure du catalogue - Volume-Directory ou Volume-Subdirectory.

Il existe une analogie frappante entre la structure interne d'un livre et le catalogue d'une disquette ProDOS. Un livre comporte un titre qui définit la pensée de l'auteur ; en l'ouvrant, on découvre une table des matières, puis suivent les chapitres, qui se divisent en sous-chapitres, qui à leur tour se subdivisent en rubriques, et ainsi de suite.

Une disquette ProDOS comporte un nom de Volume comparable au titre du livre. Dans le catalogue, on distingue la table des matières, qui peut être principale et auxiliaire. La ressemblance est grande avec le Directory de la disquette, qui peut se composer d'un Volume-Directory - table des matières principale - et d'un ou de plusieurs, Volume-Subdirectory - table des matières auxiliaire. En poussant plus loin la comparaison, on se rendrait compte que cette similitude subsiste jusque dans les moindres détails. Le texte du livre serait alors notre fichier sur la disquette.

ANALOGIE LIVRE/DISQUETTE SOUS PRODOS

Livre	*Disquette*	*Noms*
Titre	Volume-Directory	USERS.DISK
Table des matières principale	Directory	- FILER - PRODOS - BASIC.SYSTEM - STARTUP
Chapitres		
Titre rubrique 1	Subdirectory	UTILITAIRES
Titre rubrique 2	Subdirectory	JEUX
Rubrique 1	Volume-Subdirectory	UTILITAIRES
Textes		- BIG.MAG - EDITEUR - EXERCICER - APPLEWRITER - VISICALC
Rubrique 2	Volume-Subdirectory	JEUX
Textes		- SPACE PANIC - SABOTAGE

NOTION DE VOLUME-SUBDIRECTORY

Le Volume-Directory peut contenir 51 noms de fichiers, mais ProDOS peut gérer jusqu'à 32 mégabytes de données. Il serait bon de pouvoir disposer d'une capacité plus importante, surtout avec un disque dur. Cela ne présente aucune difficulté ; pour cela, il

suffira de créer une nouvelle table des matières : le Volume-Subdirectory. Cette fonction de création est réservée à la commande CREATE, laquelle est d'ailleurs le seul moyen d'ouvrir un nouveau fichier Directory.

Un Volume-Directory peut contenir des fichiers programmes et des fichiers Directory. Un fichier Directory, lors de sa création, ouvre un Volume-Subdirectory ; le système lui alloue automatiquement un bloc, pointé par le nom sauvegardé dans le Directory qui l'a créé. Le Directory contient le fichier Subdirectory, et son Volume-Subdirectory est le Volume "parent" du Directory ; ils sont liés entre-eux par des pointeurs. (Voir au chapitre 3 le détail d'une entrée Volume-Directory, Volume-Subdirectory et fichier).

ORGANISATION HIERARCHISEE

Il est possible de regrouper différents fichiers, d'après une suite logique et fonctionnelle, ou d'après leur appartenance à une famille de programmes. Par exemple, il vous sera permis de créer un Volume-Subirectory du nom de UTILITAIRES, pour y sauvegarder vos programmes utilitaires, et un autre Volume-Subdirectory JEUX pour vos programmes de jeux. Cette structure vous permettra la sauvegarde de vos programmes utilitaires suivant le chemin /VOLUME/UTILITAIRES/FICHIER.UTIL, et tous les programmes jeux, suivant le chemin /VOLUME/JEUX/FICHIER.JEUX. Ce raisonnement nous amène tout droit vers la notion de chemin - Pathname - et préfixe - PREFIX.

Explication des mots employés :

/VOLUME est le nom du Volume de la disquette.
/UTILITAIRES est le nom du Volume-Subdirectory qui pourra contenir tous vos fichiers utilitaires.
/JEUX est le nom du Volume-Subdirectory qui pourra contenir tous vos fichiers jeux.
FICHIER.UTIL est un nom quelconque de fichier utilitaire.
FICHIER.JEUX est un nom quelconque de fichier jeux.

Terminologie des mots usuels

- **Pathname**

C'est une suite de noms de Directory et Subdirectory juxtaposés, et qui désignent le chemin à parcourir pour arriver au fichier que l'on désire traiter. Ce chemin peut être partiel ou complet. Un chemin complet commence par le nom du /VOLUME, tandis qu'un chemin partiel sera complété par le préfixe.

- **PREFIX**

Le préfixe est un chemin par défaut, qu'il est nécessaire d'ajouter à celui qu'on veut nommer. Ses différentes possibilités permettent une simplification dans l'élaboration des chemins : la frappe de la totalité du chemin n'est pas toujours nécessaire.

Exemples :

Chemin complet :

/USERS.DISK/UTILITAIRES/FICHIER

Chemin partiel :

/UTILITAIRES/FICHIER
UTILITAIRES

Préfixe :

/USERS.DISK/UTILITAIRES
/USERS.DISK/JEUX
UTILITAIRES
JEUX

Un chemin complet, partiel ou préfixe, peut être constitué d'un maximum de 64 caractères, mais lors d'une association d'un chemin partiel et d'un préfixe, le système tolèrera par défaut 128 caractères.

Notion de joker

Le joker - Wildcard - est toléré avec l'utilisation du programme FILER qui fait partie des utilitaires du IIe. Avec le IIc, le FILER est remplacé par une version à sélection visuelle des fichiers, à partir d'une liste de commandes affichées à l'écran. Ce sont les flèches haut et bas qui permettent le déplacement dans le tableau des fichiers ; la flèche droite fait la sélection, et la flèche gauche l'annule.

Les jokers sont au nombre de deux :

- Le signe égal "=" remplace n'importe quel caractère ou suite de caractères.
- Le signe du point d'interrogation "?" agit d'une façon similaire mais demandera en plus confirmation avant de traiter le fichier concerné. La touche ESC arrête la séquence engagée, et annule l'ordre en cours.

Volume et Pathname

Commandes de gestion des Directory et Chemins (Volume and Pathname) :

Ces commandes permettent :

- de lister le contenu d'un Volume,
- de purger certains fichiers pour libérer de la place,
- de changer le nom d'un fichier ou d'un Volume,
- de verrouiller ou de déverrouiller un fichier,
- de créer un sous-catalogue - Subdirectory.

DETAIL DES COMMANDES PRODOS

- Volume et Pathname

• CAT

Cette commande permet de visualiser le Directory d'une disquette, c'est-à-dire sa table des matières, en format 40 colonnes. Le nom de la table des matières apparaît en tête de la liste des fichiers, en haut et à gauche, sous la forme : /VOLUME, où VOLUME est le nom du Volume. L'affichage divise l'écran en quatre colonnes, qui portent les indications : NAME, TYPE, BLOCKS, MODIFIED. Le bas de l'écran renseigne sur la capacité de sauvegarde du disque, et affiche : BLOCKS FREE : et BLOCKS USED : .

Syntaxe : CAT chemin,S#,D#
Directory, catalogue simple.

Exemples :

CAT
CAT/USERS.DISK
CAT,S6,D2

Signification des indications affichées :

*	Tout à fait à gauche de l'écran, un astérix (*) précise si le fichier est verrouillé.
NAME	Précise le nom d'un fichier programme, ou Subdirectory.
TYPE	Spécifie le type de fichier sauvegardé - DIR, TXT, VAR, BIN, REL, etc.
BLOCKS	Signale le nombre de blocs de 512 bytes utilisés par le fichier en question.
MODIFIED	Affiche la date et l'heure de la dernière modification ou enregistrement effectué. Se fait automatiquement avec une carte d'extension de la gestion date et heure, Thunderclock.
BLOCKS FREE	Affiche le nombre de blocs de 512 bytes encore libres. Le système effectue une soustraction de la capacité de stockage de la disquette par le nombre de blocs occupés par les fichiers.
BLOCKS USED	Affiche le nombre de blocs occupés sur la disquette. C'est la somme de tous les blocs marqués occupés par les différents fichiers sauvegardés.

• CATALOG

Commande similaire à CAT, qui permet de visualiser le Directory, ou table des matières, avec un format de 80 colonnes (Carte 80 colonnes pour l'Apple IIe). Si la carte 80 colonnes fait défaut, l'affichage se fera sur deux lignes, et dans un format peu lisible. En plus des renseignements affichés par la commande CAT, les colonnes CREATED, ENDFILE et SUBTYPE complètent les rubriques. Au bas de l'écran nous retrouvons BLOCKS FREE et BLOCKS USED avec en plus TOTAL BLOCKS.

Syntaxe : CATALOG chemin,S#,D#
Catalogue étendu.

Exemple :

CATALOG/USERS.DISK

Renseignements supplémentaires :

CREATED	Indique la date et l'heure de création du fichier.
ENDFILE	Si cette valeur est différente de 0, elle indique la taille maximale possible du fichier. Ce n'est pas forcément la taille réelle de ce fichier.
SUBTYPE	Une lettre A est suivie d'une valeur en base 16 - HEX -, qui renseigne sur l'adresse normale de chargement d'un fichier binaire. Une lettre R précède la valeur de la longueur du dernier enregistrement d'un fichier à accès aléatoire.
BLOCKS FREE	Indique le nombre de blocs de 512 bytes encore libres.
BLOCKS USED	Indique le nombre de blocs occupés par les fichiers.
TOTAL BLOCKS	C'est le nombre total de blocs que ProDOS peut gérer avec une disquette formatée. Cette valeur est sauvegardée à l'emplacement du premier bloc Directory où se trouve l'entrée du Volume-Directory, aux positions $29-$2A, au moment du formatage de la disquette - 280 blocs pour un Disk II.

• PREFIX

Cette commande permet de connaître le préfixe, ou de le déclarer pour la recherche éventuelle d'un fichier. Déclarée seule, cette instruction affiche le préfixe actuel ; suivie d'un suffixe, elle fixe un nouveau préfixe. La taille maximale d'un préfixe est de 64 caractères.
PREFIX évite de fournir à chaque fois le chemin complet d'une commande adressée à un fichier sauvegardé dans un Subdirectory. Il suffit, pour cela, d'indiquer au moyen de PREFIX la partie que PRODOS devra ajouter avant le nom du fichier concerné.

Syntaxe : PREFIX chemin,S#,D#

Exemples :

PREFIX affiche le préfixe par exemple : /USERS.DISK/

PREFIX/USERS.DISK fixe le préfixe au Volume /USERS.DISK

PREFIX/USERS.DISK/UTILITAIRES fixe le préfixe au sous-catalogue UTILITAIRES du Volume /USERS.DISK

PREFIX,S6,D2 fixe le préfixe au nom du Volume se trouvant dans le slot 6, drive 2.

- **CREATE**

C'est la seule commande permettant de créer un Volume-Subdirectory : elle ouvre un nouveau Volume, lui alloue un bloc, puis sauvegarde dans le Directory "parent" le nom du fichier Subdirectory. CREATE est nécessaire pour aller au-delà des 51 noms de fichiers autorisés au départ, restriction imposée essentiellement par la taille du Volume-Directory - blocs $02-$05 soit 4 blocs au total.

Syntaxe : CREATE chemin,Ttype,S#,D#
L'option Ttype est un paramètre qui permet de créer d'autres types de fichiers. Le "T" sera suivi de trois lettres désignant le type de fichier - SYS, DIR, BAS, etc. (L'annexe 4 donne le détail de tous les types de fichiers redéfinissables.)

Exemples :

CREATE/USERS.DISK/PROGRAMMES,TDIR crée un fichier DIR - Directory - du nom de PROGRAMMES sur la disquette ayant comme nom de Volume USERS.DISK.

CREATE APPLEWRITER01,TBIN : si le préfixe est /USERS.DISK/PROGRAMMES, crée un fichier du type BIN (Binaire), ayant comme nom APPLEWRITER01.

CREATE/USERS.DISK : non autorisé, car /USERS.DISK est un nom de Volume, et non un Subdirectory.

- **RENAME**

Cette commande permet de changer le nom d'un fichier quelconque, et de quelque nature qu'il soit : fichier programme ou Volume-Subdirectory. Le fichier devra être au préalable déverrouillé. Le nom du Volume-Directory n'est pas modifiable par la commande RENAME : il faudra éventuellement utiliser le programme FILER, en optant pour VOLUME COMMAND qui vous permettra de renommer le Volume.

Syntaxe : RENAME nomfich, nouveau, S#, D#
Où nomfich représente le chemin partiel ou complet. Le nom d'un fichier est en fait le dernier stade d'un chemin spécifié. Si nomfich détermine un chemin, alors nouveau devra également nommer le même chemin, pour donner en dernier lieu le nouveau nom au fichier. Le Directory ou Subdirectory, dans lequel se trouve le fichier à renommer, devra être le même pour les deux chemins cités par nomfich et nouveau.

Exemples :

RENAME/USERS.DISK//UTILITAIRES/NOMFICH,
/USERS.DISK/UTILITAIRES/NOUVEAU

Où le nom du fichier NOMFICH du Subdirectory UTILITAIRES sera changé en NOUVEAU.

- **DELETE**

Cette commande permet de supprimer un fichier programme ou Subdirectory : ils ne devront pas être verrouillés, et dans le cas d'un Volume-Subdirectory, celui-ci devra être vide. Avant d'utiliser cette commande, il faudra s'assurer de son opportunité, on risque sinon de perdre irrémédiablement des données.

Syntaxe : DELETE chemin,S#,D#

Exemples :

DELETE /USERS.DISK/UTILITAIRES/NOMFICH
DELETE /UTILITAIRES/NOMFICH
DELETE NOMFICH
DELETE UTILITAIRES

- **LOCK**

Cette commande permet de verrouiller tout type de fichier, et de le protéger ainsi contre le risque de suppression accidentelle, tout en le laissant libre d'accès. Un fichier texte verrouillé peut être lu, et un programme verrouillé peut être chargé en mémoire centrale, listé et modifié pour une mise au point éventuelle. L'utilisation des commandes CAT et CATALOG permet de lister les fichiers ou les programmes, afin de détecter ceux qui sont verrouillés : ils sont précédés par *. Un fichier Subdirectory, tout en étant verrouillé, n'est pas protégé contre l'écriture : il vous sera permis de sauvegarder des fichiers dans ce Volume-Subdirectory. Il faudra, pour supprimer un fichier Subdirectory, que son Volume "parent" soit vide.

Syntaxe : LOCK chemin,S#,D#
Le chemin est obligatoire, et désigne le fichier ou programme à verrouiller.

Exemples :

LOCK /USERS.DISK/UTILITAIRES/NOMFICH
LOCK /UTILITAIRES/NOMFICH
LOCK NOMFICH

- **UNLOCK**

Cette commande, inverse de LOCK, permet de déverrouiller les fichiers programmes ou Subdirectory verrouillés par LOCK.

Syntaxe : UNLOCK chemin, S#, D#

Exemples :

UNLOCK /USERS.DISK/UTILITAIRES/NOMFICH
UNLOCK /UTILITAIRES/NOMFICH
UNLOCK NOMFICH

CHAPITRE 4

LES FICHIERS DE TEXTE... ET LES AUTRES, SOUS ProDOS

GENERALITES SUR LES FICHIERS

Types de fichiers PRODOS
Directory, Subdirectory et programmes

Une disquette ProDOS peut contenir plusieurs types de fichiers: les fichiers tables des matières principales, ou tables des matières auxiliaires (DIR); et les autres, qui regroupent les fichiers programmes (TXT, BAS, VAR, BIN, REL, SYS et ceux du type $F#). Pour le détail des codes et types de fichiers divers pouvant être déclarés sous SOS et PRODOS, consulter l'annexe 4.

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 uniquement charger en mémoire un programme machine.
BSAVE Pour sauvegarder sur la disquette 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# et IN# Pour sélectionner un périphérique d'entrée/ sortie en ligne, autre que l'écran et le clavier.

Types de fichiers standard :

DIR Désigne une table des matières, ou sous-catalogue : Directory ou Subdirectory.
TXT Fichier de données, utilisable 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.

Les catégories de fichiers

Les fichiers se regroupent en deux catégories.

Fichiers Directory :

Ce sont des noms de fichiers qui pointent vers d'autres Volumes. Ils peuvent être de deux types : fichiers tables des matières principales et fichiers tables des matières auxiliaires.
Une table des matières principale - Volume-Directory - peut contenir des noms de fichiers programmes et des noms de fichiers tables des matières auxiliaires - Volume-Subdirectory. Chaque table auxiliaire, ou Volume-Subdirectory, peut à son tour contenir des fichiers programmes ou tables des matières.
Le Volume-Directory est créé lors de l'opération formatage, et c'est la première table des matières d'une disquette ProDOS, d'où le nom de table des matières principale.

Fichiers de données :

Ce sont les autres types de fichiers. Ils ont tous une particularité : ils pointent des 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

ProDOS ignore le bit de poids fort, et change d'office le jeu de caractères : c'est ainsi que tous les caractères 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 orthographiques.
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.

- Chemin complet - Pathname -

Un chemin complet se compose d'un ou de plusieurs noms de fichiers, précédés par le nom du Volume-Directory. Cette succession de noms désigne un fichier sauvegardédans une table des matières, qui peut être du type Directory ou Subdirectory. Ce type de chemin est toujours précédé par le "/" qui est un slash. La longueur maximale d'un tel chemin est de 128 caractères.

Exemple :

/USERS.DISK/UTILITAIRES/NOMFICH

où /USERS.DISK est le Volume-Directory.
/UTILITAIRES/ est le Volume-Subdirectory.
NOMFICH est le nom d'un programme.

- Chemin partiel

Une autre méthode pour la recherche d'un fichier emprunte le chemin partiel. Le chemin partiel est une portion du chemin complet, qui caractérise le nom du fichier recherché. Cette pratique ne débute pas par le "/" suivi d'un nom de Volume, et se contente uniquement du nom d'un fichier. La longueur maximale d'un tel chemin est de 64 caractères.

Exemple valide :

/USERS.DISK/UTILITAIRES/

c'est le préfixe déclaré ;

/USERS.DISK est le nom du Volume ;
/UTILITAIRES/ est le nom du Subdirectory.

Puis le nom du fichier pourra être déclaré seul :

NOMFICH

où NOMFICH est le nom d'un programme.

Non autorisés :

2.EXEMPLE	Débute par un chiffre.
EXEMPLE.TROP.LONG	Trop long.
COURRIER 2	Un blanc non autorisé.

- Le préfixe - PREFIX

ProDOS additionne de lui-même le préfixe (PREFIX) au début d'un chemin partiel. Le préfixe est un chemin valide déclaré au préalable par la commande MLI PREFIX.
ProDOS effectue une recherche suivant le nom du Volume déclaré, et teste ainsi tous les périphériques en ligne.
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. L'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.

Exemples :

PREFIX/USERS.DISK/
PREFIX/USERS.DISK

Création d'un fichier

CREATE est la seule commande MLI permettant de créer une nouvelle table des matières - Volume-Subdirectory. Elle permet aussi la création d'un nom de fichier d'un type quelconque, et lui alloue alors un bloc par défaut. Ce bloc, encore vide, sera réservé pour la sauvegarde des données futures.

Le nom du fichier ainsi créé pourra être modifié par RENAME, et effacé par DELETE si le fichier n'est pas verrouillé. Si c'est un fichier Directory, le Volume pointé ne devra pas contenir de programmes pour permettre son effacement par DELETE, mais RENAME pourra le renommer.

Identité d'un fichier

Chaque fichier se caractérise par un certain nombre de paramètres que le système lui attribue, à un moment ou à un autre de son traitement. Les principales caractéristiques sont les suivantes :

- Le 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.

- Le byte d'accès

Il est matérialisé par l'astérix "*", dont la valeur détermine si le fichier pourra être effacé ou re-nommé, 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.

- Le type de fichier

Le byte qui le caractérise détermine son format physique lors de la sauvegarde sur la disquette, et n'affecte pas son contenu.

- Le byte caractéristique

Ce byte détermine si le fichier est une table des matières principale ou auxiliaire, ou encore simplement un nom de fichier programme : Volume-Directory, Volume-Subdirectory ou programme.

- Date et heure

Possibilité de gestion de la date et de l'heure, si la configuration est pourvue d'une carte d'extension horloge du type Thunderclock ou similaire.
Format de la date : Année/Mois/Jour
Format de l'heure : Heure/Minute

LE FICHIER TEXTE

Généralités

La structure d'un fichier texte (TXT) est utilisée pour traiter des données, qui par la suite seront susceptibles d'être modifiées, ou pour écrire à des emplacements précis d'un tel fichier. Tout fichier ouvert devra être fermé avant de quitter le programme, sinon les données risquent d'être perdues.
Un fichier texte peut être aléatoire, séquentiel ou exec.

Les fichiers texte sont répertoriés dans le Directory comme tout autre fichier, mais le byte $1E (ou 30 en décimal, position relative à l'entrée de sa table des matières) renseigne sur l'état du fichier : verrouillé - LOCK - ou déverrouillé - 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 byte d'accès :

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

bit à 1 Autorise la fonction.
bit à 0 N'autorise pas la fonction.

Signification des différents bits :

bit 7 Destroy. Il détermine si le nom du fichier peut être effacé ou non par la commande MLI DESTROY (DELETE en Basic). Avec #$C3 comme valeur équivalente du byte d'accès, le fichier pourra être effacé.

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

bit 5 Backup. Ce bit informe le système qu'une donnée peut être copiée ou 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 : ils sont toujours positionnés à 0.

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

bit 0 Read. 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 effectués avec le bit 7 à 0, sous forme d'une suite de codes ASCII, et d'une longueur déterminée d'avance. RETURN est le code ASCII #$0D, et NUL est le code ASCII #$00. NUL n'est pas apparent après un enregistrement normal, mais seulement dans le dernier enregistrement.

LES 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.

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 65535.
- 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.

LES FICHIERS TEXTE ALEATOIRES

Les fichiers aléatoires sont structurés en 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 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 tous 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, il sera nécessaire d'indiquer la longueur de l'enregistrement. Cette longueur sera stockée dans la table des matières. La longueur par défaut sera de1 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 longueur de 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.

LES FICHIERS EXEC

Les fichiers EXEC sont des fichiers de commande du type séquentiel, contenant 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 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 l'instruction OPEN qui appelle le sous-programme MLI. Pour ouvrir un fichier texte, il faudra spécifier au préalable : le chemin et l'adresse du tampon I/O.

- Le chemin - Pathname

  Si le fichier est du genre séquentiel, seul le chemin comportant le nom du fichier, le connecteur et le lecteur de disquettes 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.

- L'adresse mémoire - I/O buffer

  Cette adresse du tampon 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.

Au total, 8 fichiers peuvent être ouverts simultanément. 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

L'instruction CLOSE ferme tout fichier texte ouvert au moment où elle est donnée.
Lorsqu'un fichier est ouvert, ProDOS se réserve en mémoire un tampon de 1 024 octets pour y placer des données avant d'effectuer des transferts périodiques sur la disquette. La commande CLOSE permet donc de forcer le vidage de ce tampon de travail sur la disquette, 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 vous désirez fermer un fichier particulier, CLOSE devra être suivi du paramètre ou chemin particulier à ce fichier.

CLOSE peut être déclaré en mode immédiat ou par programme, et n'est pas 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 commande sauvegarde sur la disquette les données encore contenues dans le tampon d'un fichier ouvert, sans pour autant le fermer.
Si plusieurs fichiers sont ouverts et que la commande FLUSH est utilisée seule, alors tous les tampons des fichiers actuels seront sauvegardés sur la disquette.
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 certaines données :

- 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 sur la disquette 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 sauter à partir de la position courante.

Remarque : il est impératif de fermer un fichier texte par la commande CLOSE, avant d'éteindre le micro-ordinateur, 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 1 024 octets pour effectuer le transfert des données entre la disquette et la mémoire.
La commande FLUSH permet d'effectuer un vidage instantané du tampon actif, en sauvegardant les données sur la disquette.

Structure de sauvegarde d'un fichier

Un fichier Directory a la plupart du temps une occupation moindre par rapport à un fichier programme, et de surcroît d'un accès généralement séquentiel : PRODOS utilise une forme simplifiée de sauvegarde sur la disquette. Par contre, les deux types de fichiers pratiquent de la même manière la disposition et l'allocation du nombre de bytes par bloc. Les données sont en effet stockées par blocs de 512 bytes sur la disquette.

Si un fichier texte comporte 512 bytes ou moins, alors le nom stocké dans le Directory pointe vers un bloc qui contient les données de ce fichier. Si, par contre, celui-ci comporte plus de 512 bytes, alors le nom (fichier) pointe un bloc contenant une liste des pistes et secteurs où sont sauvegardées les données du programme. Cette liste s'appelle la Blocks List ou Index Blocks. Si le bloc index comporte plusieurs blocs, ceux-ci seront chaînés entre eux par des pointeurs avant et arrière, permettant au système d'effectuer une recherche de données.

LES COMMANDES RELATIVES A UN FICHIER TEXTE

Commandes et syntaxes

Pour gérer les fichiers de données du type texte - TXT -, il faudra utiliser les instructions suivantes :

OPEN	Pour ouvrir et créer un fichier.
CLOSE	Pour fermer un fichier ouvert.
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 du fichier, sur la disquette.
POSITION	Pour désigner un emplacement particulier à l'intérieur d'un fichier et pouvoir, par la suite, y lire ou y écrire des données.

• **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#
NOM représente le nom du fichier ouvert.
L# représente la longueur du fichier.
S# et D# représentent 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 en question. 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
NOM est 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 ouverts. La conséquence directe en sera la sauvegarde sur une disquette 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 la disquette et libère de la place en mémoire.

- **Instruction FLUSH**

Cette instruction est exécutable en mode immédiat ou à partir d'un programme. Elle force le vidage du tampon mémoire, et sauvegarde les données sur une disquette.

Syntaxe : FLUSH NOM
NOM est le nom du fichier.

Lorsque ProDOS travaille avec un fichier texte, il utilise une technique de sauvegarde particulière. En effet, les instructions PRINT placent les données d'un fichier dans un tampon mémoire, pour ne pas solliciter le drive à chaque octet, pendant l'accès lecture ou écriture. C'est pourquoi, lors de la déclaration de l'instruction OPEN, ProDOS alloue un tampon d'une taille identique - 1 024 octets - à chaque fichier texte. Au fur et à mesure que ce tampon se remplit, le système effectue des sauvegardes ponctuelles sur la disquette. Si, pour une raison particulière, vous désirez effectuer une sauvegarde des données du tampon, sans pour autant fermer le fichier, c'est la commande FLUSH qu'il faudra utiliser.

- **Instruction OPEN**

Cette instruction ne peut être déclarée qu'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
NOM est le nom du fichier.
S# et D# désignent le slot et le drive.
L# est la longeur d'un fichier aléatoire.
Ttype désigne le type de fichier.

- **Instruction POSITION**

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

Syntaxe : POSITION NOM,F#,B#
F# désigne un chiffre qui permet de sauter un nombre de champs d'enregistrements représentés par #.
B# désigne le nombre d'octets à sauter, représentés par #. Uniquement utile avec un fichier séquentiel.

- **Instruction READ**

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

Syntaxe : READ NOM,R#,F#,B#
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'enregistrements à sauter.
B# désigne le nombre d'octets à sauter.

- **Instruction WRITE**

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

Syntaxe : WRITE NOM,R#,F#,B#
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'enregistrements à sauter.
B# désigne le nombre d'octets à sauter.

POINTEURS D'UN FICHIER TEXTE

Pointeurs EOF et MARK

Lorsqu'un fichier texte est ouvert par la commande OPEN et que des données y sont écrites, un 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. D'après le *Technical Manual*, ces pointeurs se nomment : EOF (End Of File) et MARK (position actuelle du caractère considéré) et désignent respectivement un pointeur logique et un pointeur physique à l'intérieur d'un fichier.

POINTEUR MARK

Il trouve son utilisation avec les commandes MLI SET.MARK et GET.MARK.
Le pointeur est sauvegardé en mémoire dans la File Control Block, aux positions relatives #$12-#$14 de l'entrée de la table des matières qui correspond au fichier traité. La 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 une entrée.

Mark est un pointeur ou position actuelle à l'intérieur d'un fichier texte, utilisé lors d'un accès sur une disquette, et déterminé par un paramètre associé à une commande PRODOS. La valeur du pointeur est calculée par PRODOS à partir d'une position absolue dans un fichier texte - #$000000 - et désigne le caractère actuel à lire ou à enregistrer. Ce pointeur sera désigné par la suite sous le nom de Position lorsqu'il s'agira de traiter les commandes MLI. Avec l'utilisation des instructions READ et WRITE, le pointeur MARK est la somme équivalente de l'association des paramètres R#, F# et B# à partir d'une position absolue dans le fichier.

POINTEUR EOF

Trouve son utilisation avec les commandes MLI SET.EOF et GET.EOF.
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 une disquette, EOF est enregistré dans la table des matières du fichier correspondant, à la position relative #$15-#$17 de son entrée. Il occupe 3 bytes et désigne la longueur d'un fichier texte.

EOF est un pointeur de fin de données, qui matérialise la longueur d'un fichier pouvant prendre toute valeur comprise entre $000001 et $FFFFFF (LByte, MByte, HByte). Par EOF, on entend la position d'un byte pointé dans un fichier, juste à la suite du dernier caractère de fin, qui est un retour-chariot, valeur par défaut (code #$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 non matérialisé sur la disquette lors d'un enregistrement.

Le pointeur EOF est une longueur relative, qui a son origine à partir du début du fichier texte, et commence avec la valeur 1.

CONCLUSION

Ces pointeurs sont 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 sauter. Ce sont des paramètres associés aux instructions WRITE et READ, qui permettent une extension de la gestion des fichiers.

Exemples :

- Dans un fichier aléatoire, il vous sera possible de préciser le numéro de l'enregistrement pour l'écriture ou la lecture, par le paramètre R#.

- Le paramètre B# servira à sauter un nombre de bytes déterminé par l'indice #, à partir de la position courante.

- Le paramètre F# permettra de sauter un certain nombre de champs d'enregistrements ; ProDOS effectue alors le comptage des retours-chariot (RETURN = code #$0D) dont le nombre limite sera fixé par l'indice #. Un retour-chariot délimite en somme une ligne de texte d'un enregistrement effectué sur la disquette, dont la longueur équivaut à un champ de données.

Particularité des pointeurs

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

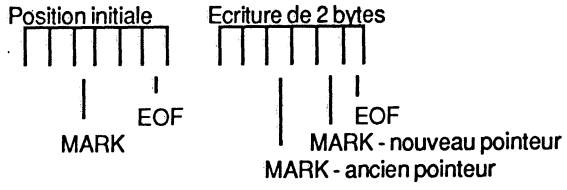
Lorqu'un byte est écrit ou lu dans un enregistrement, la marque se déplacera d'une unité pour pointer la position byte +1, qui est l'emplacement du prochain byte à écrire ou à lire. Néanmoins, la position extrême de lecture/écriture dans un fichier ne pourra excéder sa taille fixée par le pointeur de fin de fichier (EOF).

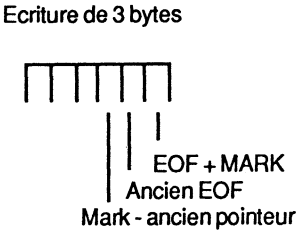
Lors d'une opération d'enregistrement, la marque courante et la position de fin du fichier seront déplacées, vers l'avant, du nombre de bytes écrits à cet instant déterminé. Le nombre de données ajoutées fixera la nouvelle taille du fichier (EOF) et déterminera automatiquement la position de la marque courante du pointeur. ProDOS affectera au fichier une nouvelle marque de fin (EOF) et mettra à jour le pointeur de la table des matières - Directory - de la disquette.

La taille d'un fichier est sauvegardée dans la table des matières, lors du premier enregistrement, par la commande OPEN suivie de la longueur choisie (paramètre L# pour les fichiers aléatoires). Si le paramètre longueur n'est pas précisé lors d'un enregistrement ultérieur, la longueur totale du fichier ne pourra pas excéder la taille spécifiée lors de sa création. Quand une nouvelle longueur est déclarée lors de la réouverture du même fichier, ce sera cette valeur que l'on utilisera pendant les opérations à venir, mais la valeur sauvegardée dans la table des matières lors de sa création ne sera

pas modifiée. 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 par des bytes nuls. Ces types de fichiers sont appelés fichiers "creux", car ils possèdent des zones de bytes non significatifs (Bytes nuls - Ctrl-à). Le *Technical Manual* appelle les fichiers creux des Sparse File.

Schématisation des pointeurs :





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. 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 disquette. Celle-ci pourra être hiérarchisée, ou du type éparse et comportera dans ce dernier cas des enregistrements indésirables.

STRUCTURE HIERARCHISEE

Dans le cas des fichiers de structure hiérarchisée, ProDOS utilise pour la sauvegarde de ceux-ci un nombre de blocs en fonction de leur capacité en caractères enregistrés :

- si un fichier a besoin de moins de 512 bytes, il n'utilisera qu'un seul bloc ;

- s'il nécessite plus de 512 bytes, il lui faudra 2 blocs pour ses données, et un bloc pour la 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. Le bloc répertoire se composera alors d'un Master Block et d'un ou plusieurs Blocs Index.

Chaque bloc répertoire est un index principal (Master Block) qui pourra se diviser en index ou répertoires auxiliaire (blocs index).

STRUCTURE D'UN FICHIER CREUX

Un fichier aléatoire est structuré en enregistrements, qui sont des sauvegardes successives de données. Chaque enregistrement contient le même nombre de données, défini en bytes (caractères), valeur lui ayant été attribuée lors de son ouverture.
Un enregistrement peut contenir un ou plusieurs champs de données.
Le total des informations placées dans un enregistrement est appelé la longueur de l'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 un tel type de fichier, il faudra inclure un paramètre lors de son ouverture : le paramètre longueur - L# - où # désigne la longueur en bytes de chaque enregistrement.

Paramètre Longueur

```
10PRINT 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 à l'intérieur d'un fichier aléatoire et réduire ainsi sa taille, on devra copier les enregistrements à préserver dans un nouveau fichier à accès aléatoire.

Les enregistrements indésirables se trouvant ainsi prisonniers dans le contenu global d'un fichier aléatoire s'appellent enregistrements creux ou Sparse File.

Exemple de fichier creux :

Au départ, le fichier créé contient 1 000 enregistrements, avec comme paramètre longueur : L# = 100. Si l'on place dans ce fichier aléatoire, qu'on vient d'ouvrir, des données dans l'enregistrement 1 000, et que, pour une raison déterminée, les enregistrements se trouvant 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'une disquette.

L'inconvénient des fichiers aléatoires est qu'ils enregistrent des codes nuls (CTRL-0) 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'une disquette peut être très rapidement envahie par des blocs contenant des bytes inutiles (bytes nuls). Il faudra alors formater ces types de fichiers, pour pouvoir réutiliser les zones éparses, où se trouvent les codes nuls. Cette opération de préformatage 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 préformatage

```
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 seront ajustées selon la longueur et le nombre des enregistrements du fichier à préformater.

Origine d'un Sparse File

Les fichiers aléatoires, dits "creux", peuvent être engendrés de trois manières différentes :

1 - Lorsqu'un fichier ouvert ne comporte qu'un champ dans chaque enregistrement ; par exemple :

```
R0 : MARCELr
R1 : LUCr
R2 : RODOLPHEr
```

où r est égal au retour-chariot, code #$0D.
Les enregistrements plus courts seront complétés par des bytes nuls (Ctrl-à) après le retour-chariot.

Si le paramètre L# = 9, l'enregistrement sur la disquette apparaîtra 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
```


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

```
R0 : M A R C E L r L U C r R O D O L P H E r
```

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

Si le paramètre L# = 25, l'enregistrement sur la disquette apparaîtra ainsi:

```
R0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 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
```

3 - Lorsque, dans un ancien fichier, on effectue une nouvelle sauvegarde d'enregistrements, sans tenir compte de la valeur des emplacements chronologiques du paramètre R#. Par exemple, si vous déclarez 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 : la table des matières (Directory) contient un 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 4 bits de poids fort, tandis que les 4 bits de poids faible sont affectés à Name Length qui désigne la longueur du nom d'un fichier.
Les 4 bits de poids fort peuvent avoir comme valeur 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 permettant d'écrire ou de lire un certain nombre de bytes, par l'intermédiaire de deux pointeurs qui se déplacent à l'intérieur de celui-ci : MARK et EOF.
Les fichiers, comme nous l'avons vu, regroupent deux grandes familles : les fichiers Directory et les fichiers programmes

FICHIER DIRECTORY

Un Volume-Subdirectory est pointé par un nom de fichier Directory, pouvant à nouveau contenir des noms de fichiers Directory, ou tout simplement des programmes. En début de chaque table des matières, se trouve sauvegardé le nom du Volume-Directory. Celui-ci comporte en préfixe un byte caractéristique : les 4 bits de poids fort indiquent si c'est un Volume-Directory ($F), un Volume-Subdirectory ($E), un fichier de Volume-Subdirectory ($D), ou tout simplement un fichier de données ($1, $2 ou $3). Les 4 bits de poids faible désignent la longueur du nom d'un fichier. Le byte est sauvegardé à la position relative $00 de chaque entrée d'une table des matières.

Le *Technical Manual* nomme les 4 bits de poids fort Storage Type, et les 4 bits de poids faible Name Length. Nous garderons par la suite ces dénominations respectives, et désignerons le byte par Byte caractéristique.

Exemple :

Dans un Volume-Directory, le nom de Volume USERS.DISK sera précédé du byte caractéristique référencé ainsi :

F pour désigner que c'est un Volume-directory ;
B pour représenter la longueur du nom.
10 en décimal pour notre exemple ;
FB = Byte caractéristique.
(Voir le chapitre 3 pour plus de détails.)

Si un Volume-Directory contient un ou plusieurs fichiers qui pointent un ou plusieurs Volume-Subdirectory, alors ces différents fichiers Directory seront chaînés dans le sens avant et arrière par des pointeurs. Cette technique est essentielle afin de pouvoir pointer à chaque fois le bon chemin lors de la recherche d'un fichier programme dans un Volume-Subdirectory. Un fichier qui pointe un Volume-Subdirectory est le parent de ce dernier : c'est là qu'il a pris naissance.

Exemple :

Volume-Directory ----->
<----- Volume-Subdirectory

Un Volume-Subdirectory prend une structure de Volume, avec son nom sauvegardé au début de la table des matières auxiliaire. Le système alloue un bloc par défaut lors de toute création d'un nouveau Volume sur une disquette.

FICHIER PROGRAMME

Un fichier programme désigne tout autre type de fichier, par exemple : type Binaire, AppleSoft, Texte, Exec, etc.
Tous ces différents types de fichiers peuvent être lus ou écrits à partir d'une table des matières du Volume d'une disquette.

Ce sont les fichiers qui permettent l'exécution de différents programmes, et cela dans un langage donné. Les fichiers que ProDOS peut gérer 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. Ce sujet a été largement développé précédemment, et ne sera plus traité dans ce paragraphe.
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.

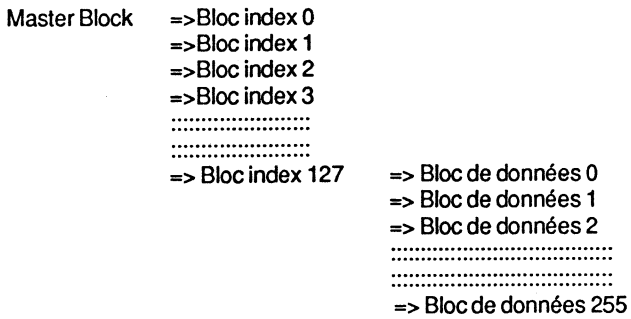
Notion de blocs index :

Un bloc index principal (Master Block) peut pointer vers des blocs index auxiliaires (Index Blocks) d'une taille maximale de 128 blocs. Chaque bloc index peut à nouveau pointer 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.
Les blocs sont alloués par ProDOS dans une suite consécutive de secteurs, lorsque la configuration de la disquette le permet, sinon ils seront chaînés entre eux, et les pointeurs sauvegardés dans la table index.

Allocation de blocs consécutifs :

- la disquette doit être 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 même nom du programme figurant sur la disquette.

Allocation maximale des blocs à un fichier programme



CHAPITRE 5

ESPACE MEMOIRE ET VECTEURS DE ProDOS LISTING DE LA PAGE GLOBALE DU BASIC.SYSTEM LISTING DE LA PAGE GLOBALE DE PRODOS

UTILISATION DE LA MEMOIRE VIVE - RAM

Analogie de la mémoire sous DOS et ProDOS

Certains DOS, comme DIVERSI-DOS, se laissent aisément loger dans la carte langage - Bank 2. Si l'on associe ce système d'exploitation à une carte d'extension de 64 Ko, celle-ci pourra être utilisée comme RAM DISK, ou encore pseudo-disque.
L'espace mémoire se partage ainsi :

$0800-$BDFF Libre au programmeur.
$D000-$DFFF Bank 1 de la carte langage libre.

Cette disposition laisse au programmeur une zone de travail de 50 688 octets, avec un espace homogène de 46 952 octets.

Lorsque ProDOS est "booté" sur un Apple IIe avec une carte d'extention de 64 Ko, ou sur un Apple IIc, il se loge automatiquement dans la carte langage : la Bank 1 sera totalement occupée, et la Bank 2 le sera partiellement. Ultérieurement sera installé un RAM Disk-Driver, ainsi que le fichier BASIC.SYSTEM, interface entre l'AppleSoft en ROM, et ProDOS qui est le système d'exploitation, logé en haut de la mémoire.

Le fichier BASIC.SYSTEM sera placé aux adresses $9600-$BFFF, avec un tampon ouvert - MAXFILES = 1. Pour chaque nouveau tampon, le système se réserve $400 octets (1024), valeur égale environ au double d'un tampon I/O sous DOS 3.3. Dans la pratique, trois tampons ouverts est chose courante, ce qui fixe HIMEM à $8E00 et délimite l'espace mémoire libre à la valeur $0800-$8DFF. Cette affirmation suppose que se trouvent en mémoire centrale les fichiers PRODOS + BASIC.SYSTEM. L'espace mémoire disponible s'élève maintenant à 34 304 octets, ce qui réduit la zone des programmes, d'environ 16 Ko. C'est un des handicaps majeur de ProDOS, qui, en raison de cette disposition, ne permet plus l'implantation du langage Integer. Cependant, comme l'utilisation de ce langage est essentiellement orientée vers les jeux, cette restriction n'est nullement ressentie par les programmeurs comme un handicap réel.

Occupation mémoire de ProDOS - RAM

ProDOS et BASIC.SYSTEM
Occupation de la mémoire vive - RAM

Espace mémoire vive commutée :

$D000 - bank 1 et bank 2 -, et $E000 à $FFFF.

Occupation de la RAM 64 Ko :

$F800-$FFFF	Disk Driver - Sous-programmes de commande et de contrôle du lecteur.

BanK 1 : Reboot

$D100-$D3FF	Routine de reboot.

BanK 2 : MLI

$D000-$EFFF	PRODOS - Programme.
$F000-$F7FF	Données de PRODOS.
$BF00-$BFFF	PRODOS page globale. Variables du système.
$BE00-$BEFF	BASIC.SYSTEM page globale. Variables globales du Basic.
$9A00-$BDFF	BASIC.SYSTEM.
$9600-$99FF	1 tampon de PRODOS.
$0800-$95FF	Espace libre pour la programmation.
$0400-$07FF	Page 1 texte. Ecran 40 colonnes.
$0300-$03FF	Vecteurs de la page 3.
$0200-$02FF	Tampon clavier.
$0100-$01FF	Pile du microprocesseur.
$0000-$00FF	Page zéro.

Occupation de la carte auxiliaire - 64 Ko :

$0C00-$FFFF	RAM Disk. Disque virtuel.
$0800-$0BFF	Espace libre.
$0400-$07FF	Ecran 80 colonnes. Deuxième page d'écran.
$0200-$03FF	RAM Disk-Driver.
$0000-$01FF	Espace libre.

Occupation de la ROM :

$F800-$FFFF	Moniteur.
$D000-$F7FF	AppleSoft.
$C000-$CFFF	Gestion des périphériques - Slots.

Commutation de la RAM :

$D100-$DFFF	PRODOS MLI.
$D000-$D0FF	Libre.

Transfert des données

Un des points forts de ProDOS réside dans sa gestion plus rapide des fichiers par rapport à d'autres DOS, en particulier que DOS 3.3, qui se caractérise par sa lenteur.
Le tableau ci-dessous donne un aperçu du temps en secondes pour le traitement d'un fichier binaire de 32 Ko.

	DOS 3.3	ZDOS 2.0	ProDOS 1.0
BSAVE	46 s.	39 s.	18 s.
BLOAD	33 s.	08 s.	05 s.

Encore faudra-t-il avoir à l'esprit que le temps d'accès aux pistes/secteurs n'est pas régulier et que ces chiffres ne sont donnés qu'à titre indicatif.

Comme le système Apple Pascal, ProDOS traite comme base des blocs de 512 octets, ce qui correspond à 2 x 256 octets, ou 2 x 2 secteurs sous DOS 3.3. Cette structure lui confère une rapidité supérieure à 20 % environ, lors d'accès aux pistes/secteurs, à condition que le fichier traité soit au moins égal ou supérieur à 1 bloc c'est-à-dire à 512 octets. Par contre, lorsque ProDOS traite des fichiers d'une capacité inférieure à 1 bloc, il devient moins performant que le DOS 3.3. Cette situation se rencontre avec un fichier aléatoire : si le système traite un enregistrement d'une longueur supérieure à 256 octets, ProDOS sera plus rapide ; si l'enregistrement est inférieur à 256 octets, ce sera DOS le plus rapide. Cette analogie nous permet de formuler la règle suivante : plus un fichier à traiter est grand, plus rapide sera le temps d'accès à celui-ci, temps relatif évidemment. Il faut néanmoins se souvenir que cette règle n'est que relative.

RAM DISK-DRIVER

Généralités sur la carte 80 colonnes

Actuellement, deux structures de boot, sous ProDOS, s'offrent à nous : configuration du type Apple IIe ou Apple IIc. Dans le premier cas, il sera nécessaire d'adjoindre une carte d'extension 80 colonnes étendue, tandis que l'Apple IIc est complet et prêt à dialoguer avec ProDOS.

La carte étendue 80 colonnes intègre une RAM de 64 Ko qui se placent en parallèle sur la mémoire principale et peuvent être commutés par des commutateurs logiques. Lorsque la carte 80 colonnes est activée, les espaces mémoire, $0400-$07FF, des pages principales et auxiliaires sont commutés en même temps. Une partie des 64 Ko de

la mémoire auxiliaire sera utilisée : $0400-$07FF. Cette manipulation sera effectuée en allant lire ou écrire à une adresse spéciale utilisée comme commutateur logique. (Voir en annexe 9 la liste des commutateurs, et un bref rappel d'utilisation.)

Pour un Apple IIe, la carte 80 colonnes étendue sera enfichée dans le slot 3 marqué auxiliaire, et se comportera comme une extension en mémoire morte, ROM, pour dialoguer avec le microprocesseur. Les routines seront implantées dans l'espace mémoire aux adresses $C300-$C3FF et $C800-$CFFF.

L'Apple IIc intègre cette extension sur sa carte mère, et la ROM a été totalement réécrite pour l'intégration du périphérique de la souris. Par ailleurs, le microprocesseur est du type 65C02 (Apple IIe/ 6502) et les slots d'extension ont totalement disparu. Mais la plupart des points d'entrées sont restés compatibles avec les autres séries.

Commutateurs - points d'entrées

- ALTZP $0000-$01FF et $D000-$FFFF

$C008	MAIN MEMORY. Mémoire principale : regroupe la page zéro, la pile, et la ROM ou la carte langage.
$C009	AUX MEMORY. Mémoire auxiliaire : regroupe la page zéro, la pile, et la ROM ou l'espace mémoire $D000-$FFFF.
$C018	READ ALTZP SWITCH. Le bit 7 est à 1 si la mémoire auxiliaire est commutée - AUX MEMORY ON.

- RAMRD/RAMWRT - $0200-$BFFF

$C002	Read Main Memory.
$C003	Read Aux Memory.
$C004	Write Main Memory.
$C005	Write Aux Memory.
$C013	Read RAMRD Switch. - $C002-$C003 - Le bit 7 sera positionné à 1 pour lire la mémoire auxiliaire.
$C014	Read RAMWRT Switch. - $C004-$C005 - Le bit 7 sera positionné à 1 pour écrire dans la mémoire auxiliaire.

Ces différents commutateurs permettent de déterminer, pour l'espace mémoire $0200-$BFFF, quelle bank - MAIN ou AUX - le processeur pourra lire. Par ailleurs, il sera possible de commuter une bank pour écrire des données soit dans la mémoire principale, soit dans la mémoire auxiliaire. Vous pourrez ainsi copier d'une bank dans une autre, ou par exemple lire dans MAIN et écrire dans AUX, ou inversement.

Exemple de structure :

```
        LDY #$00
ENCORE  LDA $2000,Y
        STA $2000,Y
        INY
        BNE ENCORE
```


- RAM-Disk RWTB :

$C000	80STORE "off". Affichage vidéo : toujours de Main Memory.
$C001	80STORE "on". Affichage vidéo : d'une bank sélectionnée. - MAIN Memory ou AUX Memory -
$C018	Read 80STORE Switch. Le bit 7 sera positionné à 1, si 80STORE "on".

Avant d'appeler la routine RWTB - $0200 -, le commutateur 80STORE est mis sur "off" pour empêcher un scintillement de l'affichage vidéo pendant l'attaque des routines du RAM-Disk. Avant de quitter la routine RWTB, le commutateur sera rebasculé sur la position initiale.

- Commutation de 80STORE en RWTB :

```
0200- AD 18    C0  LDA  $C018      Lecture de 80STORE,
0203- 48           PHA             et sauvegarde.
0204- 8D 00    C0  STA  $C000      Déconnecte 80STORE

      Affichage Main Memory
0207- ::::::::::::::::::::::::::::::::::::::::::::::::
      ::::::::::::::::::::::::::::::::::::::::::::::::
03E7- 8D 01    C0  STA  $C001      Commute 80STORE "on".
```

Mise en place de la RAM-Disk

Lorsque le système ProDOS est "booté" sur un Apple IIe avec 64 Ko d'extension, ou sur un Apple IIc, il initialise la RAM-Disk RWTB qui se place, d'une part, aux adresses $FF00-$FF80 et, d'autre part, aux adresses $0200-$03FD (sur la carte auxiliaire).
Lorsque PRODOS est mis en place par la commande -PRODOS, les 64 Ko de la mémoire auxiliaire se partagent ainsi :

$0000-$01FF	Page zéro et pile : espace inutilisé dans son ensemble, à part quelques adresses comme $3D-$43 et le sommet de la pile à partir de $01CA.
$0200-$03FF	RAM-Disk RWTB.
$0400-$07FF	Espace mémoire pour l'affichage vidéo 80 colonnes. C'est la seconde moitié de la page texte 80 colonnes.
$0800-$0BFF	Inutilisés avec un Apple IIe. Avec un Apple IIc, espace mémoire pour des sauvegardes intermédiaires, et clavier.
$0C00-$0DFF	Volume Bit-Map (table d'occupation bloc #$03) du disque virtuel et tampon des blocs avant le transfert des données dans la carte langage.
$0E00-$0FFF	Volume-Directory. Blocs pour la gestion du catalogue disque virtuel (12 noms de fichiers au total - bloc #$02).

$1000-$19FF Blocs #$09, #$1A, #$2B, #$3C et #$4D :
Bloc #$09 : $1000
Bloc #$1A : $1200
Bloc #$2B : $1400
Bloc #$3C : $1600
Bloc #$4D : $1800

$1A00-$1FFF Blocs #$5D, #$5E et #$5F :
Bloc #$5D : $1A00
Bloc #$5E : $1C00
Bloc #$5F : $1E00

$2000-$3FFF Bloc #$08 : $2000
Bloc #$0A : $2200
::::::::::::::::
Bloc #$18 : $3E00

$4000-$5FFF Bloc #$19 : $4000
Bloc #$1B : $4200
::::::::::::::::
Bloc #$29 : $5E00

$6000-$7FFF Bloc #$2A : $6000
Bloc #$2C : $6200
::::::::::::::::
Bloc #$3A : $7E00

$8000-$9FFF Bloc #$3B : $8000
Bloc #$3D : $8200
::::::::::::::::
Bloc #$4B : $9E00

$A000-$BFFF Bloc #$4C : $A000
Bloc #$4E : $A200
::::::::::::::::
Bloc #$5C : $BE00

Bank 1 de la carte langage - AUX Memory "on" -
$D000-$DFFF Bloc #$60 : $D000
Bloc #$61 : $D200
Bloc #$62 : $D400
::::::::::::::::

Bank 2 de la carte langage - AUX Memory "on" -
$D000-$DFFF Bloc #$68 : $D000
Bloc #$69 : $D200
Bloc #$6A : $D400
::::::::::::::::

$E000-$FFFF Bloc #$70 : $E000
Bloc #$71 : $E200
Bloc #$72 : $E400

Interface du RAM-Disk RWTB

Les paramètres sauvegardés aux adresses $0042-$0047 de la page zéro sont utilisés par le MLI pour la commande Read Write Tracks Blocks : Unit Number, adresse de transfert et numéros de blocs. Cette opération est réalisée via DO DISK I/O, en appelant l'adresse $FF00. Cet espace mémoire $FF00-$FF8C est utilisé comme interface pour la RWTB de la RAM-Disk.

Adresses commentées :

$FF00 Les valeurs stockées aux adresses $003C-$0047 sont sauvegardées, ainsi que celles qui le sont aux adresses $03ED-$03EE. Le contenu de $003C-$0043 sera utilisé à l'intérieur du RWTB comme pointeurs. Les adresses $03ED-$03EE contiennent l'adresse de retour de la routine Bankswitch.

$FF16 Si la commande est nulle, adresse $0042 = #$00, un saut sera effectué à l'adresse $FF44 - EXIT - ; la RAM-Disk est toujours en service, mais protégée contre l'écriture - WRITE PROTECTED.

$FF1A La commande devra avoir comme valeur #$00 à #$03 :
00 = déplace la tête de lecture
01 = lecture
02 = écriture
03 = formatage.
Si la valeur est différente : I/O ERROR sera retourné.

$FF1E Après le OU exclusif OR #$03, le résultat sera le suivant :
formatage #$03 = #$00
écriture #$02 = #$01
lecture #$01 = #$02.

$FF24 Teste la valeur maximale de la capacité en blocs. Si le numéro du bloc vérifié est supérieur à #$7F, un message d'erreur sera retourné: I/O ERROR.

$FF2C Les adresses $03ED-$03EE contiennent l'adresse de retour : $0200.

$FF33 Commute l'espace mémoire auxiliaire $0200-$BFFF en lecture/ écriture, puis saute à l'adresse de retour dans AUX Memory. Les zones mémoire-page zéro, pile et carte langage - restent commutées sur MAIN Memory.

$FF3B Sortie par erreur.
ERR# = #$27 : I/O ERROR. Provient d'un dépassement de blocs ou d'une commande incorrecte contenue dans l'adresse $0042. Continue l'exécution à l'adresse $FF47.

$FF44 Commande aboutie : sortie avec ERR# = #$00. Entrée sollicitée par l'adresse $FF22 avec une commande = #$00, ou par RWTB de AUX Memory via XFER adresse $C3B0.

$FF47 EXIT.
Restaure les adresses de la page zéro : $003C-$0047 ainsi que $03ED-$03EE. Commute les espaces mémoire sur MAIN Memory, et la bank 1 de la carte langage en mode lecture et écriture. Retour par un RTS à l'adresse $D0DA - DO DISK I/O - avec annulation de la retenue par CLC, et avec un #$00 chargé dans l'accumulateur.

$FF62-$FF7E WRITE TRANSFER.
Cette boucle sera engendrée par RWTB dans AUX Memory, via XFER - $C3B0 - lorsqu'il faudra écrire de MAIN Memory vers AUX Memory - Ecriture dans la RAM-Disk.
Les pointeurs seront déterminés à l'intérieur de AUX RWTB : adresse destination du transfert, adresse des blocs, adresse tampon $0C00-$0D00, etc.
#$200 bytes seront copiés de MAIN vers AUX, puis un saut effectué à l'adresse $02D8 dans AUX via XFER - $C3B0 ; sollicité par l'adresse $FF33.

$FF7F-$FF80 Adresses intermédiaires pour la sauvegarde du contenu de $03ED-$03EE.

$FF81-$FF8C Adresses intermédiaires pour la sauvegarde des contenus de $003C-$0047.

LA PAGE ZERO

Adresses réservées pour ProDOS

Utilisation de la page zéro :

Le système d'exploitation PRODOS logé dans la carte langage n'utilise que peu d'adresses de la page zéro ($00-$FF), en particulier pour le MLI (Machine Language Interface) et la routine Disk-Driver.

Adresses modifiées dans la page zéro :

$00-$03 Adresses utilisées pour le reboot.
$3A-$3F Adresses utilisées par les routines Disk-Driver appelées par le MLI.
$40-$4F Adresses utilisées par PRODOS comme palier intermédiaire.

Remarque : les adresses $3A, $3E, $40-$43, $45, $46 et $4F sont modifiées par le système, tandis que les autres sont restaurées en fin d'utilisation de la routine. En annexe 1 se trouve un tableau complet des adresses de la page zéro utilisées par : le Moniteur, AppleSoft, ProDOS (MLI et le Disk-Driver).

Paramètres d'appel au Disk-Driver

Adresses réservées pour les périphériques :
Avant que PRODOS sollicite ses périphériques, il mémorise 6 adresses de la page zéro, dont les renseignements seront utilisés par la suite (adresses $42 à $47).

$42 Commandes pour le Disk-Driver :
0 pour déplacer la tête de lecture (Seek).
1 pour lire des données.
2 pour écrire des données.
3 formatage.

$43 Numéros de slot et de drive : 5 bits significatifs.

Bits : 76543210
DSSSxxxx D = drive
S = slot
x = inutilisés.
Détail :
Bit 7 : 0 = drive 1
1 = drive 2
Bits 6-4 Numéro de slot : (110 = slot 6)
(111 = slot 7)
Bits 3-0 Non significatifs (000)

Exemple :

Pour la mémorisation du slot 7, drive 2, le codage bit par bit du byte contenu à l'adresse $43 serait :

Bits : 76543210
11110000 = #$F0

$44 Pointeur du tampon (Low Byte).
$45 Pointeur du tampon (High Byte).

Ces deux adresses contiennent le pointeur d'un tampon d'une capacité de 512 octets, avant le transfert des données.

Exemple :

44- LB HB Avec LB = LByte et HB = HByte
00 20 Pointe l'adresse $2000 pour le transfert de 512 octets.

$46 Numéro du bloc (Low Byte).
$47 Numéro du bloc (High Byte).

Ces deux adresses désignent le numéro du bloc considéré, dont la capacité est de 512 octets.

Exemple :

46- LB HB Avec LB = LByte et HB = HByte
70 00 Désigne le bloc $0070 (112).

Codes d'erreurs

Comme lors de l'appel de la routine RWTS sous DOS 3.3, ProDOS demande la mise en place de paramètres, avant l'appel de la routine Driver, pour gérer ses périphériques. Lorsque celle-ci est exécutée, le système effectue un contrôle des Drivers sollicités, et copie dans la Device-Bit table à la page globale de PRODOS, aux adresses $BF32-$BF3F, les valeurs respectives trouvées. Si pour une raison quelconque l'ordre n'aboutissait pas, le code d'erreur se trouverait dans l'accumulateur.

Drivers errors :

03	Pas de périphérique connecté.	(No Device Connected)
04	Protégé à l'écriture.	(Write Protected)
08	Erreur d'entrée/sortie.	(I/O Error)

Adresses libres avec BASIC.SYSTEM

Si PRODOS charge le fichier BASIC.SYSTEM, il restera très peu d'adresses libres dans la page zéro. Par ailleurs, ces adresses disponibles seront sensiblement les mêmes que sous DOS 3.3.

$06-$07	Inutilisées.
$08	Utilisée par un Apple IIe.
$09	Inutilisée.
$19-$1E	Inutilisées.
$1F	Utilisée par un Apple IIe.
$CE-$CF	Inutilisées.
$D6-$D7	Inutilisées.
$E3	Inutilisée.
$EB-$EF	Inutilisées.
$FA-$FF	Inutilisées.

TAMPON D'ENTREE - TAMPON CLAVIER

Utilisation du tampon clavier sous ProDOS

Sous le système PRODOS, le tampon clavier - ou tampon d'entrée - se situe aux adresses $0200-$02FF. Il n'est plus exploité dans les mêmes conditions qu'avec le DOS 3.3. Certains programmes Basic utilisent ce tampon pour y placer des commandes via le Moniteur.

Le tampon se divise en deux parties :

- la première partie ($0200-$027F) est utilisée pour la gestion des blocs de la disquette. Le système exploite ce vecteur pour placer le résultat des calculs intermédiaires effectués sur les blocs alloués à la disquette ;
- la seconde partie, à partir de l'adresse $0280, est utilisée pour la sauvegarde des données rentrées par le clavier.

GETLN et le tampon clavier

Cette disposition particulière du tampon d'entrée autorise le traitement par la routine GETLN qui gère certaines variables ou données rentrées au clavier, de la même manière que sous DOS 3.3. En association avec un lecteur de disquettes, la première partie ($200-$27F) est réservée pour la sauvegarde des calculs de blocs, et toute donnée risque d'être écrasée.

Il est évident que tous les programmes utilisés avec le DOS 3.3 et qui exploitent la zone complète du tampon clavier, $200-$2FF sont à proscrire sous ProDOS, en particulier la routine de S.H.LAM. Celle-ci utilise une technique permettant de placer dans une chaîne de caractères des commandes à destination du Moniteur : un court programme AppleSoft place une routine en langage machine dans la mémoire centrale. Cette routine effectue alors les commandes indispensables pour placer une chaîne de caractères dans le tampon clavier, comme si elle était tapée à partir du Moniteur.

Parmi toutes les restrictions concernant le tampon clavier, il faudra encore classer la mise en place des sous-programmes utilisés pour le démarrage ultérieur d'autres programmes.

LES VECTEURS DE LA PAGE 3 SOUS ProDOS

La Page 3

ProDOS utilise partiellement la page 3. La version 1.1.1 a été légèrement modifiée par rapport aux versions 1.0 et 1.0.2. En particulier, les adresses $03D6-$03E9 contiennent maintenant une partie d'une nouvelle et mystérieuse routine et $03EA-$30EB le pointeur du début de l'autre partie de la mystérieuse routine. La routine est recopiée par le Relocator, en même temps que le MLI, dans la carte langage, aux adresses $FD3B-$FD64.

Le reste des adresses $03D0... $03FF est occupé par les pointeurs classiques, et ne présente pas de différence notoire avec les versions initiales.

Plusieurs spéculations sont possibles au sujet de l'opportunité de la routine mystérieuse :

- Aucune action n'est introduite une fois la routine relogée. La question reste posée : la ROM des futurs Apple serait-elle modifiée en conséquence dans l'avenir ?
- Pour l'instant, le IIc, dernier-né de la gamme Apple II, ne possède pas de liaison à cette routine.
- Par ailleurs, le vecteur $03EA-$03EB pourra par la suite être utilisé par une routine du Moniteur.
- La routine qui débute en $FD3B appelle par la suite la deuxième partie ($03D6-$03E9), dont le rôle est de POKEr les adresses $42-$47, de la page zéro mémoire principale, aux mêmes adresses dans la mémoire auxiliaire.
- Les adresses $0042-$0047 de la page zéro seront utilisées pour la copie des paramètres de la commande Read Write Tracks Blocks. Mais la routine de la RAM-Disk devra être revue et corrigée. Elle pourra, par exemple, utiliser une carte d'extension mémoire au-delà de 64 Ko, pour atteindre les 256 Ko.
- Apple garde le silence au sujet de cette routine.

Vecteurs de la page 3 - ProDOS Vers. 1.1.1

$03D0	JMP	$BE00	Démarrage à chaud - Warmstart.
$03D3	JMP	$BE00	Démarrage à chaud - Warmstart.
$03D6 à $03E9			Routine mystérieuse. (Voir listing à la suite.)
$03EA	3B		LByte.
$03EB	FD		HByte; pointeur $FD3B.
$03F0	59	FA	JMP $FA59 En cas de BReaK.
$03F2	00	BE	JMP $BE00 RESET vecteur - Warmstart.
$03F4	1B		Octet significatif (PWRUP).
$03F5	4C	03 BE	JMP $BE03 Adresse en cas du &.
$03F8	4C	00 BE	JMP $BE00 CTRL-Y - Monitor Warmstart.
$03FB	4C	59 FF	JMP $FF59 Interruption non masquée.
$03FE	EB	BF	JMP $BFEB Interrupt Entry.

Mystery Routine - Page 3

03D6-	8D	08	C0	STA	$C008	CLRALTZP. Sélectionne la page zéro et la pile de la mémoire principale.
03D9-	B5	42		LDA	$42,X	Charge les différents paramètres de sauvegarde.
03DB-	8D	09	C0	STA	$C009	SETALTZP. Sélectionne la page zéro et la pile de la mémoire auxiliaire.
03DE-	95	42		STA	$42,X	Transfert des données de sauvegarde en mémoireauxiliaire : $42.. $47.
03E0-	CA			DEX		Décrémente le registre X.
03E1-	10	F3		BPL	$03D6	Traite les adresses de $42.. $47.
03E3-	A9	28		LDA	#$28	Utilisé comme code d'erreur.
03E5-	38			SEC		
03E6-	8D	08	C0	STA	$C008	CLRALTZP. Retour à la mémoire principale.
03E9-	60			RTS		Retour au sous-programme appelant.

Mystery Routine - $FD3B-$FD64

FD3B-	08			PHP		Sauvegarde du registre d'état PS au sommet de la pile,
FD3C-	68			PLA		et dépile vers l'accumulateur.
FD3D-	48			PHA		Empile à nouveau.
FD3E-	B8			CLV		Annulation de l'indicateur de débordement.
FD3F-	29	04		AND	#$04	Test du bit 2 du registre d'état - Inhibition.
FD41-	F0	03		BEQ	$FD46	Si interruption IRQ autorisée, annulation du bit de débordement pour le retour de la routine.
FD43-	2C	64	FD	BIT	$FD64	= #$60, positionne le bit 6 du registre d'état. Débordement.
FD46-	78			SEI		
FD47-	AD	83	C0	LDA	$C083	Commutation de la carte langage, en lecture,
FD4A-	AD	83	C0	LDA	$C083	et en écriture - Bank 2.
FD4D-	38			SEC		
FD4E-	A2	05		LDX	#$05	Charge le registre X comme compteur: 6 valeurs.
FD50-	20	D6	03	JSR	$03D6	Saut à la page 3 pour la copie des adresses $42-$47de la mémoire principale vers la mémoire auxiliaire.
FD53-	8D	0F	BF	STA	$BF0F	Sauvegarde dans SYSERROR de la valeur contenue dans l'accumulateur, au retour de la routine de la page 3 - #$ 28.
FD56-	AD	8B	C0	LDA	$C08B	Bank 1 de la carte langage,
FD59-	AD	8B	C0	LDA	$C08B	et écriture autorisée.
FD5C-	28			PLP		Restaure le registre d'état PS.
FD5D-	18			CLC		
FD5E-	AD	0F	BF	LDA	$BF0F	Charge la valeur de SYSERROR - #$28.
FD61-	F0	01		BEQ	$FD64	
FD63-	38			SEC		Retour - RTS - avec code erreur #$28 : NO DEVICE CONNECTED.
FD64-	60			RTS		Retour au sous-programme appelant.

GENERALITES SUR LES VECTEURS DE LA PAGE 3

Le boot à chaud et à froid

Le démarrage à chaud sous ProDOS, par comparaison au DOS 3.3, se situe techniquement entre un démarrage à chaud et à froid.

Sous DOS 3.3, le démarrage à froid, effectué par la commande 3D3G formulée à partir du Moniteur, annule toutes les variables et purge par la même occasion tout programme AppleSoft se trouvant en mémoire centrale. Avec un démarrage à chaud - commande 3D0G -, les variables et programmes restent en place, et peuvent être réutilisés par la suite.

Sous ProDOS, un démarrage à chaud préserve tout programme AppleSoft implanté en mémoire centrale, mais purge ses variables. C'est la conséquence logique du JMP $D665 qui se trouve copié à partir de l'adresse $AC35 appelée lors d'un démarrage à chaud. Le vecteur $D665 est un des points d'entrée de la routine AppleSoft CLEAR. Il n'existe plus la même notion de démarrage à froid et à chaud 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.

Les touches CTRL-Y et RESET

La combinaison des touches CTRL-Y à partir du Moniteur effectue un démarrage à chaud, et efface ainsi toute variable de la mémoire ; il en va de même avec la combinaison des touches CTRL-RESET, qui en plus gèle le prompt jusqu'à ce qu'une autre touche soit activée.

Processus de démarrage à chaud sous ProDOS :

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

```
03D0- 4C 00 BE  JMP $BE00
03D3- 4C 00 BE  JMP $BE00
```

b) En $BE00 se trouve un saut à l'adresse $AC35 du BASIC.SYSTEM (Warmstart) :

```
BE00- 4C 35 AC  JMP $AC35
```

c) En $AC35 se trouve un saut à l'adresse $D665 de la ROM AppleSoft :

```
AC35- 20 65 D6  JSR $D665
```

d) A partir de l'adresse $D665, l'interpréteur effectue une purge des variables se trouvant dans la zone des variables par l'instruction CLEAR.

Le & ou l'Ampersand

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

Lorsque le & 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
```

Le & reste redéfinissable comme sous le DOS 3.3, en modifiant au préalable le pointeur aux adresses $03F6,$03F7.

Exemple :

```
03F5- 4C 58 FC  JMP $FC58
```

En tapant à partir du clavier : & + RETURN, le système effectue un saut à l'adresse $03F5. Celle-ci contient la valeur "4C", qui est le code de l'instruction "JMP" ou saut inconditionnel ; l'interpréteur branche à l'adresse qui le suit, en l'occurrence $FC58. Le sous-programme débutant à l'adresse $FC58 de la ROM Moniteur efface l'écran dans la limite de la fenêtre de texte. C'est l'équivalent du traditionnel HOME en Basic.

Saut indirect à $03FE

Lorsque ProDOS se trouve en mémoire et qu'une routine d'interruption est active, le système effectue un saut au vecteur $03FE de la page 3, qui contient un pointeur : normalement EB BF - pointeur $BFEB. Cette adresse a pour objet de commuter la Bank 1 de la carte langage, puis d'effectuer un saut à l'adresse $D13A qui affiche le message suivant :

```
INSERT SYSTEM DISK AND RESTART  -ERR xx
```

où xx représente le code de l'erreur rencontrée. Il ne reste plus qu'à "rebooter", car le système boucle sur l'adresse $D239 :

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

Sous-programme implanté à partir de $BFEB :

```
BFEB- AD  8B  C0      LDA  $C08B
BFEE- AD  8B  C0      LDA  $C08B
BFF1- 4C  3A  D1      JMP  $D13A
```

Vecteurs d'entrées/sorties

Sous ProDOS, les commandes PR#, IN# et FF59G déconnectent les vecteurs d'entrées/sorties, et le traditionnel CALL 1002 - 3EAG - reste sans effet. Heureusement, il existe une parade en effectuant un CALL 39447 - 9A17G à partir du Moniteur. Ce procédé 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 suffira simplement d'y placer un saut vers l'adresse $9A17.

Revectorisation des E/S :

03EA : 4C 17 9A JMP $9A17

Avec 4C comme code du JMP et 17,9A comme octets respectifs du poids faible et poids fort de l'adresse $9A17. Après la modification introduite, il suffira de reconnecter le BASIC.SYSTEM par l'ancien CALL 1002 - 3EAG à partir du Moniteur.

Sous-programme à partir de $BE00 :

BE00- 4C 35 AC JMP $AC35 Démarrage à chaud - Warmstart.

Sous-programme à partir de $AC35 :

AC35- 20 65 D6	JSR $DD65	AppleSoft CLEAR.
AC38- 20 17 9A	JSR $9A17	Reconnecte le système
AC3B- A9 00	LDA #$00	Charge X,
AC3D- 85 24	STA $24	et annule HTAB.
AC3F- 4C 3F D4	JMP $D43F	ROM AppleSoft - Warmstart.

Remarque : sous DOS 3.3, l'entrée du démarrage à chaud se fait à l'adresse $D43C, et contient un JSR $DAFB. A cet emplacement de la ROM AppleSoft se trouve le sous-programme d'affichage du retour-chariot : cela se traduit à l'écran par un saut de ligne. PRODOS n'utilise plus ce saut de ligne avant l'ordre transmis, et se connecte généralement à l'entrée directe des différentes routines de la ROM.

LE BASIC.SYSTEM - INTERFACE ProDOS

Garbage Collection

L'espace mémoire $0800-$95FF est normalement libre et permet de loger les programmes - AppleSoft, machine, etc. Le premier tampon de gestion fichier se trouve à partir de l'adresse $9600 : il occupe 1 024 octets et sa dernière adresse se trouve à $99FF. Lors du traitement d'un fichier sous ProDOS, les 512 premiers octets du tampon ($9600-$97FF) sont utilisés comme bloc de données, et les 512 octets suivants comme

index des blocs gérés ($9800-$99FF). La commande CATALOG utilise la première partie du tampon, 512 octets, comme mémoire intermédiaire avant l'affichage sur l'écran des données du Directory. Au bas de la zone mémoire du système Basic se trouve une zone flottante ou dynamique, car les vecteurs alloués à HIMEM et aux différents tampons de gestion des fichiers (maximum 8) ne sont pas figés.

OCCUPATION TYPE DE LA ZONE DYNAMIQUE SOUS ProDOS

- Après le boot, les fichiers PRODOS et BASIC.SYSTEM se trouvent dans leurs espaces mémoire respectifs, et HIMEM est initialisé à $9600. Le fichier BASIC.SYSTEM est logé à partir de l'adresse $9A00, et le premier tampon alloué débute à $9600 -$9600-$99FF.

- Les commandes CATALOG, LOAD, RUN, BLOAD ou BRUN n'ont aucune influence sur la configuration précédente, et n'altèrent en rien ses vecteurs.

- Lorsqu'un fichier texte est ouvert par la commande OPEN, un deuxième tampon sera réservé à partir de $9200, et HIMEM initialisé à cette même adresse. Après la commande CLOSE - qui ferme le fichier texte - les données du tampon utilisé seront sauvegardées sur la disquette, HIMEM sera replacé à son adresse initiale, et le tampon libéré. Comme l'espace mémoire des chaînes de caractères débute à partir de HIMEM, BASIC.SYTEM est contraint d'effectuer lui-même la purge des chaînes dont les variables sont devenues inutiles. Cette opération correspond à l'ancienne routine Garbage Collection de la ROM AppleSoft. Avant l'exécution de WRITE, l'espace mémoire des chaînes de caractères sera déplacé de $0400 bytes vers le bas, pour être à nouveau restauré lorsque la commande CLOSE sera rencontrée. Cela se traduira par un nouveau déplacement de $0400 bytes, mais cette fois vers le haut de la mémoire. Ce changement par rapport au DOS 3.3 entraîne un certain nombre de contraintes : il vous appartiendra d'assurer une exécution correcte des programmes logés à des adresses susceptibles d'être influencées par le mécanisme de Garbage Collection.

CONTRAINTES OCCASIONNEES AUX PROGRAMMES PAR LA ROUTINE GARBAGE COLLECTION

- Certains programmes compilés par des utilitaires du genre TASC, SPEEDSTAR, EXPEDITER, et j'en passe, ne sont plus fiables, car le système Basic ne peut plus gérer les pointeurs des programmes après compilation.

- HIMEM ne devra plus être déclaré à partir d'un programme, sauf si la maîtrise totale peut être assurée : chaque nouveau fichier ouvert ou refermé déplace automatiquement le pointeur de HIMEM.

- Il existe 3 formes de Garbage Collection :
  1- celle de l'interpréteur AppleSoft ;
  2- celle du compilateur ;
  3- et celle intégrée au BASIC.SYSTEM.
  Un compilateur ne peut interpréter que sa propre routine de Garbage Collection, tandis que le système Basic traitera la sienne et celle de l'interpréteur AppleSoft.

- Lorsque l'espace mémoire, situé en deçà du BASIC.SYSTEM, vient à être saturé par des chaînes de caractères actuelles et que cette zone est inférieure à 1 Ko (1 024 octets), il ne sera plus possible d'y sauvegarder de nouvelles chaînes. La raison en est que plus aucun tampon - ou Buffer - de 1 024 octets n'est disponible. A cette situation viendra s'ajouter le fait que le bas de la zone des chaînes correspond avec la limite d'une page à ne pas franchir (Page Boundary). En réalité, ce sont 1 280 octets (1 024 + 256) qui seront réservés comme tampon juste pour pouvoir gérer un fichier texte.

Remarque : l'allocation d'un minimum de 1 280 octets, lors d'une saturation de la zone des chaînes de caractères, présente un inconvénient de taille et le concepteur du BASIC.SYSTEM s'est laissé prendre à un piège fatal. En effet, lorsqu'un programme AppleSoft ouvre un fichier texte et qu'au même moment la configuration se trouve confrontée à un espace mémoire insuffisant (< 1 024 octets), le système se plantera. Il est à souhaiter que ce "bug" sera rapidement éliminé, faute de quoi on aurait des surprises : un programme comportant un grand nombre de variables chaînes de caractères, associé à un fichier texte, pourra perturber le système.

La page globale du BASIC.SYSTEM

La page $BE (190) débute à partir de l'adresse $BE00 pour occuper l'espace mémoire jusqu'à $BEFF (48640-48895). C'est la "Global Page" du BASIC.SYSTEM, qui englobe les adresses $9A00-$BEFF ; à ne pas confondre avec la "Global Page" de PRODOS ($BF ou 191), qui se situe aux adresses $BF00-$BFFF.

Dans cette page particulière, dont la taille est limitée à 256 bytes sont placées les variables et les données du système Basic.

Deux grandes familles sont regroupées dans cette page :

- les commandes du système (Command-Handler) ;
- les vecteurs d'entrées/sorties (I/O).

Listing commenté de la "Basic Global Page" :

• ENTRY

BE00-	4C 35 AC	JMP	$AC35	BY.ENTRY. Démarrage à chaud du système. Entrée de PRODOS et d'AppleSoft.
BE03-	4C B4 A6	JMP	$A6B4	DOSCMD. Point d'entrée des commandes du système : Command-Handler 1.
BE06-	4C 9E BE	JMP	$BE9E	EXTRNCMD. Point d'entrée des commandes externes au système : Command-Handler 2 ($BE9Econtient un RTS, code 60).
BE09-	4C 22 9B	JMP	$9B22	ERROUT. Gestion de la routine d'erreurs.
BE0C-	4C BF 9F	JMP	$9FBF	PRINTERR. Affiche le message de l'erreur rencontrée. (Le code de l'erreur se trouve dans l'accumulateur.)
BE0F-	00	DFB	0	ERRCODE. Code de l'erreur.

Remarque : le système Basic a deux entrées pour appeler son interface. La première entrée se situe à l'adresse $BE03 - DOSCMD - et traite les commandes standard de ProDOS, dont il reconnaît la syntaxe. La deuxième se situe à l'adresse $BE06 - EXTRNCMD - et peut traiter une ou plusieurs commandes externes au système. Ces commandes externes seront regroupées dans un sous-programme qui sera implanté en mémoire centrale lors de l'installation du système. En temps normal, l'adresse $BE06 contient un JMP vers une instruction RTS, qui est une instruction de retour du microprocesseur - code 60.

• I/O VECTORS (I/O = INPUT/OUTPUT)

Vecteurs d'entrées/sorties :

C'est le répertoire des vecteurs d'entrées/ sorties : lorsqu'une ROM interface d'entrée/ sortie est détectée, la valeur de son pointeur est sauvegardée dans le répertoire débutant à l'adresse $BE10. Tout slot vide est considéré comme non connecté, et contient les deux bytes du pointeur de l'entrée d'une routine, pour afficher : NO DEVICE CONNECTED. - Error Handling $9B20.

Exemple de slot vide :

Si le slot 5 est vide, le pointeur aura comme valeur :

BE1A- 20 9B ; Pointeur = $9B20

BE10-	OUTVECT0	DA	COUT1	Normalement $FDF0 : Vidéo Output.
BE12-	OUTVECT1	DA	DEVERR	C100 par défaut - Slot 1
BE14-	OUTVECT2	DA	DEVERR	C200 par défaut - Slot 2
BE16-	OUTVECT3	DA	DEVERR	C300 par défaut - Slot 3
BE18-	OUTVECT4	DA	DEVERR	C400 par défaut - Slot 4
BE1A-	OUTVECT5	DA	DEVERR	C500 par défaut - Slot 5
BE1C-	OUTVECT6	DA	DEVERR	C600 par défaut - Slot 6
BE1E-	OUTVECT7	DA	DEVERR	C700 par défaut - Slot 7
BE20-	INVECT0	DA	KEYIN	Normalement $FD1B : Keyboard-Input.
BE22-	INVECT1	DA	DEVERR	C100 par défaut - Slot 1
BE24-	INVECT2	DA	DEVERR	C200 par défaut - Slot 2
BE26-	INVECT3	DA	DEVERR	C300 par défaut - Slot 3
BE28-	INVECT4	DA	DEVERR	C400 par défaut - Slot 4
BE2A-	INVECT5	DA	DEVERR	C500 par défaut - Slot 5
BE2C-	INVECT6	DA	DEVERR	C600 par défaut - Slot 6
BE2E-	INVECT7	DA	DEVERR	C700 par défaut - Slot 7

• Traitement des fichiers

BE30-	VECTOUT	DA	COUT1	Normalement $FDF0 : Vidéo Output.
BE32-	VECTIN	DA	KEYIN	Normalement $FD1B : Keyboard-Input.
BE34-	VDOSIO	DA	DOSOUT	Vecteurs sortie de caractères pour le DOS : CSWL $36,$37

BE36-		DA	DOSINP	Vecteur entrée de caractères pour le DOS : KSWL $38,$39.
BE38-	VSYSIO	DA	0	Gestion de sortie.
BE3A-		DA	0	Gestion d'entrée.
BE3C-	DEFSLT	DFB	$06	Valeur par défaut : slot 6
BE3D-	DEFDRV	DFB	$01	Valeur par défaut : drive 1
BE3E-	PREGA	DFB	0	Sauvegarde du registre A.
BE3F-	PREGX	DFB	0	Sauvegarde du registre X.
BE40-	PREGY	DFB	0	Sauvegarde du registre Y.
BE41-	DTRACE	DBF	0	Autorise la fonction TRACE.
BE42-	STATE	DFB	0	Valeur 0 = mode immédiat, > 0 = mode par défaut.
BE43-	EXACTV	DFB	0	Bit 7 à 1 ; fichier EXEC.
BE44-	IFILACTV	DFB	0	Bit 7 à 1 ; écriture fichier
BE45-	OFILACTV	DFB	0	Bit 7 à 1 ; lecture fichier
BE46-	PFXACTV	DFB	0	Bit 7 à 1 ; préfixe actif.
BE47-	DIRFLG	DFB	0	Drapeau d'accès au fichier dans le catalogue.
BE48-	EDIRFLG	DFB	0	Fin du catalogue rencontré
BE49-	STRINGS	DFB	0	Espace libre des chaînes.
BE4A-	TBUFFTR	DFB	0	Compteur utilisé lors d'une écriture.
BE4B-	INPTR	DBF	0	Compteur utilisé par le clavier lors d'une entrée de caractères.
BE4C-	CHRLAST	DFB	0	Détection d'une erreur lors de la sortie du dernier caractère.
BE4D-	OPENCNT	DFB	$00	Nombre de fichiers ouverts.
BE4E-	EXFILE	DFB	$00	Drapeau utilisé lors du traitement d'un fichier EXEC.
BE4F-	CATFLAG	DFB	$00	Accès d'un fichier au catalogue (transfert).
BE50-	XTRNA	DDR	DA $0000	Adresse de retour lors de la déclaration d'une commande externe.
BE52-	XLEN	DFB	$00	Longueur de la commande -1.
BE53-	XCNUM	DFB	0	Commandes externes : 0 si commande externe.

PBITS/FBITS. -*Commandes avec paramètres*

Table des codes paramètres : PBITS/FBITS

PBITS = bytes définis.
FBITS = bytes reconnus.

Si la commande externe ne nécessite aucun paramètre, ou si vous désirez les extraire vous-même, placez un #$00 en PBITS et PBITS+1 - $BE54-$BE55. Le retour à ProDOS se fera par un RTS avec la retenue, Carry, à 0.

Si, au contraire, vous désirez utiliser des paramètres associés à la commande externe, il faut indiquer à ProDOS lesquels sont autorisés en alimentant les bits de PBITS et PBITS+1.

Les adresses $BE54,$BE55 - PBITS,PBITS+1 - sont réservées pour alimenter les bits avec un paramètre codé sur 2*8 bits, pour mener à terme une commande externe avec des paramètres. Ces options ont été développées afin de définir la syntaxe des commandes ProDOS. Pour plus de détail, veuillez consulter le paragraphe de ce chapitre, qui traite les commandes externes avec ou sans paramètres.

PBITS/FBITS - *Codes paramètres*

BE54-	PFIX	EQU	$80	Demande de concaténer le préfixe actuel avec le nom de fichier.
	SLOT	EQU	$40	Seul un numéro de slot est autorisé : PR# ou IN#.
	RRUN	EQU	$20	Commande uniquement valide à partir d'un programme (mode différé).
	FNOPT	EQU	$10	Nom du fichier autorisé.
	CRFLG	EQU	$08	Commande CREATE autorisée.
	T	EQU	$04	Paramètre Ttype autorisé. (Type de fichier.)
	FN2	EQU	$02	Deuxième nom de fichier autorisé - commande RENAME.
	FN1	EQU	$01	Nom du fichier attendu.

PBITS+1/FBITS+1 - *Codes paramètres*

BE55-	AD	EQU	$80	Adresse d'implantation du fichier.
	B	EQU	$40	Byte. C'est le numéro du premier octet du fichier.
	E	EQU	$20	Adresse de fin du fichier.
	L	EQU	$10	Longueur du fichier.
	à	EQU	$08	Numéro de ligne (paramètre à).
	S	EQU	$04	Numéro de slot (1-7).
	D	EQU	$04	Numéro du drive (1-2).
	F	EQU	$02	Champ du fichier. C'est la position où l'on veut écrire une donnée dans un fichier texte.
	R	EQU	$01	Numéro de l'enregistrement dans un fichier texte.
	V	EQU	$00	Numéro du Volume (ProDOS l'ignore).

Remarque : lorsque le système Basic reconnaît la syntaxe d'une de ses instructions, il vérifie les paramètres déclarés d'après PBITS et PBITS+1, et transmet l'ordre donné (paramètres : S#, D#, L#, R#, etc.).
A chaque fois qu'un paramètre est lu, le système le sauvegarde en $BE54,$BE55 - PBITS,PBITS+1 - puis il est analysé ; s'il est reconnu, ProDOS le confirme par un bit de présence - FBITS,FBITS+1 - aux adresses $BE56,$BE57. Ensuite, la valeur effective des paramètres est stockée dans son vecteur respectif, dans l'espace mémoire $BE58-$BE6B.

Les paires d'adresses $BE54,$BE55 et $BE56,$BE57 sont utilisées au fur et à mesure de l'analyse des paramètres. Ensuite, le système copie la valeur de ce paramètre à son emplacement, dans la table qui se trouve aux adresses $BE58-$BE6B, et entame l'analyse suivante.

Exemple :

BLOAD NOMFICHIER, A$2000, L$20, S6, D2

charge les $20 premiers octets du fichier NOMFICHIER qui se trouve dans le drive 2 du slot 6, à l'adresse $2000.

Le paramètre A#, A$2000 sera d'abord traité : son code sera placé par le système aux adresses $BE54,$BE55, puis il sera analysé. Ensuite, un bit de présence sera placé en $BE56,$BE57 et la valeur réelle du paramètre sauvegardé sera, dans notre exemple, placée aux adresses $BE58,$BE59. Le paramètre L# - L$20 - sera analysé à son tour, et son image sauvegardée aux adresses $BE5F,$BE60. Le même processus se déroulera avec les paramètres suivants : S6 et D2.

• PBITS à analyser

BE54 -	PBITS	DA 0	Paramètre déclaré ; LByte.
BE55 -	PBITS+1	DA 0	HByte.

• FBITS trouvés

BE56 -	FBITS	DA 0	Paramètre trouvé ; LByte.
BE57 -	FBITS+1	DA 0	HByte.

• Paramètres déclarés

La table ci-dessous correspond à la copie des paramètres déclarés par une commande Basic. Lorsque le système Basic analyse une commande optionnelle, sa valeur est stockée dans la table des paramètres - $BE58-$BE6B - à un emplacement prévu. Cette valeur restera actuelle tant qu'un autre paramètre de valeur différente n'aura pas été déclaré. Il sera possible de retrouver une ancienne donnée en effectuant une lecture à la bonne adresse, par l'instruction Basic PEEK, ou LDA en langage machine. Certains paramètres utilisent un, deux ou trois bytes.

BE58-	VADDR	DA	0	A# = adresse du programme.
BE5A-	VBYTE	DFB	0,0,0	B# = nombre d'octets.
BE5D-	VENDA	DA	0	E# = adresse de fin du programme.
BE5F-	VLNTH	DA	0	L# = longueur du programme.
BE61-	VSLOT	DFB	0	S# = slot.
BE62-	VDRIV	DFB	0	D# = drive.
BE63-	VFELD	DA	0	F# = champ du fichier.
BE65-	VRECD	DA	0	R# = numéro d'enregistrement.

BE67-	VVOLM	DFB	0	V# = volume ignoré.
BE68-	VLINE	DA	0	à# = numéro de ligne.
BE6A-	VTYPE	DFB	0	T# = type de fichier.
BE6B-	VIOSLT	DFB	0	IN# ou PR#.

• BUFFER VECTORS

BE6C-	VPATH1	DA	TXBUF-1	Adresse sur 2 octets, qui pointe le nom du fichier. Paramètre 1.
BE6E-	VPATH2	DA	TXBUF-2	Paramètre 2. Utilisé par la commande RENAME.

• GOSYSTEM ENTRY

Gestion des erreurs du MLI :

L'entrée "GOSYSTEM" est utilisée pour traiter une erreur avant le retour au sous-programme appelant, lors de l'exécution d'une commande MLI non aboutie. Le numéro du MLI et le registre X sont sauvegardés, puis le système se branche à l'adresse de gestion des codes d'erreur.

A l'entrée, l'accumulateur contient le numéro de la commande MLI. Il est sauvegardé en $BE85. L'adresse de la table des paramètres, LByte, est sauvegardée en $BE86, juste avant l'appel de la routine MLIENTRY, qui est l'entrée MLI - $BF00 par défaut. A l'adresse $BE76, le AND #$1F effectue un ET logique avec l'accumulateur, au cas où ce dernier contiendrait un numéro MLI de $C0 à $D3 - Commande-Handler. Cette commande spéciale est appelée par un "JSR" à partir de $D075, et retourne directement après exécution à l'adresse $D078. Le numéro de la commande est réduit à la valeur $00-$13, AND #$1F, et une table est créée.

Structure de la table :

Bit 7 à 1	Appel de COPY et de VERIFY Pathname - $D27F.
Bit 6 à 1	Appel de SET.FILE DATA - $D3C5.
Bit 5 à 1	Appel de SET.TIME - $BF06.
Bits 4-0	Index de la prochaine table qui contient l'adresse de début des commandes précitées.

BE70-	GOSYSTEM	STA	SYSCALL	Sauvegarde du numéro MLI.
BE73-		STX	CALLX	Sauvegarde du registre X.
BE76-		AND	#$1F	Extrait le bit de poids fort de l'accumulateur,
BE78-		TAX		et l'utilise comme index.
BE79-		LDA	SYSCTBL,X	$B85F, X normalement. Charge le paramètre de la table, LByte,
BE7C-		STA	SYSPARM	et le sauvegarde.

```
BE7F-              LDX  CALLX        $BCA8 normalement. Initialisation
                                     du registre X,
BE82-              JSR  MLIENTRY     et appel du sous-programme MLI
                                     de ProDOS $BF00.
BE85- SYSCALL      DFB  0            Numéro MLI.
BE86- SYSPARM      DA   *            LByte, table des paramètres.
BE88-              BCS  BADCALL      $BE8B normalement. Branche-
                                     ment si une erreur est rencontrée.
BE8A-              RTS               Retour au sous-programme.
```

• BADCALL ENTRY

Gestion des erreurs :

Cette entrée convertit une erreur MLI en traitement équivalent au système Basic. A l'entrée, le code de l'erreur se trouve dans l'accumulateur. Toute erreur non reconnue par le système sera renvoyée sous la forme : I/O ERROR.

Le registre d'index X est restauré après le message d'erreur, et avant le retour au sous-programme appelant. Le bit de retenue (Carry Flag), positionné lors d'une rencontre d'erreur, est mis à 1 avant de quitter la routine.

```
BE8B - BADCALL     LDX #$12          Convertit les erreurs MLI en équivalent du
                                     système Basic. - 19 erreurs reconnues.
BE8D - MLIERR1     CMP MLIERTBL,X    Sinon affiche I/O ERROR.
BE90 -             BEQ MLIERR2       Branchement à l'erreur.
BE92 -             DEX               Décrémente le registre X,
BE93 -             BPL MLIERR1       et continue la recherche.
BE95 -             LDX #$13          Erreur non reconnue, affiche I/O ERROR.
BE97 - MLIERR2     LDA BIERRTBL,X    $BA53,X normalement. Charge le code de
                                     l'erreur dans l'accumulateur.
BE9A -             LDX CALLX         Restaure le registre X ; $BCA8 normalement.
BE9D -             SEC               Force à 1 le bit de retenue positionné lors de
                                     l'erreur.
BE9E - XRETURN     RTS               Retour au sous-programme.
BE9F - BISPARE1    DFB $00
```

• Tables des commandes MLI

Table des paramètres de CREATE :

```
BEA0 - SCREATE     DFB $07           CREATE Parameter Count.
BEA1 -             DA  TXBUF-1       Pointeur du chemin (Pathname).
BEA3 - CRACESS     DFB $C3           Byte d'accès au fichier.
BEA4 - CRFILID     DFB $00
BEA5 - CRAUXID     DA  $0000
BEA7 - CRFKIND     DFB 0
BEA8 -             DA  0             Paramètres de la carte horloge.
BEAA-              DA  0
```


Table des paramètres de DESTROY :

```
BEAC- SDSTROY      DFB $01           DESTROY Parameter Count.
BEAD -             DA  TXBUF-1       Ne pas modifier.
```

Table des paramètres de RENAME :

```
BEAF - SRENAME     DFB $02           RENAME Parameter Count.
BEB0 -             DA  TXBUF-1       Ne pas modifier.
BEB2-              DA  TXBUF2
```

Table des paramètres SET.FILE.INFO ou GET.FILE.INFO :

```
BEB4 - SSGINFO     DFB $00           Parameter Count :
                                     = 7 avec SET.FILE INFO
                                     = A avec GET.FILE.INFO.
BEB5 -             DA  TXBUF-1

BEB7 - FIACESS     DFB $00           Utilisé par LOCK/UNLOCK.
BEB8 - FIFILIO     DFB $00           Type de fichier défini.
BEB9 - FIAUXIO     DA  $0000         Utilisé pour traiter un fichier dont les para-
                                     mètres longueur et adresses sont définis
                                     (L# et A#).
BEBB - FIFKIND     DFB $00           Utilisé pour la recherche d'un fichier du type
                                     "Tree".
BEBC - FIBLOKS     DA  $0000         Utilisé par la commande CAT.
```

SET.TIME Access :

```
BEBE - FIMDATE     DA  $0000         Modification date et heure.
BEC0 -             DA  $0000
BEC2 -             DA  $0000
BEC4 -             DA  $0000
```

Table des paramètres de GET.BUF :

```
BEC6 - SGETBUF     DFB $02           GET.BUF Parameter Count.
BEC7 - SUNITNUM    DFB 0             Unit Number. Drive et slot.

BEC8 - SBUFADR     DFB 0,0,0         Certains appels utilisent 2 bytes. Pour traiter
                                     les fichiers texte, MARK et EOF utilisent
                                     3 bytes.
```

Table des paramètres de OPEN :

```
BECB - SOPEN      DFB  $03       OPEN Parameter Count.
BECC -            DA   TXBUF-1
BECE - OSYSBUF    DA   $0000
BED0 - OREFNUM    DFB  0
BED1 - SNEWLIN    DFB  $03
BED2 - NEWLREF    DFB  $00       Reference Number.
BED3 - NLINENBL   DFB  $7F       Newline Character. Equivalent de 128 pour
                                 mettre le bit 7 à 1. Sert à reconnaître les deux
                                 codes du retour-chariot (0D, 8D).
BED4 -            DFB  $0D       Code retour-chariot avec le bit 7 à 0.
```

Table des paramètres de READ :

```
BED5 - SREAD      DFB  $04       READ Parameter Count.
BED6 - RWREFNUM   DFB  $00       Reference Number.
BED7 - RWDATA     DA   $0000     Pointeur pour la lecture/écriture des
                                 données.
BED9 - RWCOUNT    DA   $0000     Nombre de bytes pour la lecture/écriture de
                                 données.
BEDB - RWTRANS    DA   $0000     Echo du nombre de bytes pour l'opération
                                 lecture/écriture.
```

Table des paramètres de CLOSE :

```
BEDD - SCLOSE     DFB  $01       CLOSE Parameter Count.
BEDE - CFREFNUM   DFB  $00       Reference Number.
BEDF - CSPARE     DFB  $00

BEE0 -            ASC "(C) 1984 APPLE COMPUTER INC."
```

Remarque : le tampon clavier - $0200-$02FF - utilise une partie de son espace pour stocker les données rentrées au clavier, $0280-$02FF, et l'autre partie, $0200-$027F, pour gérer l'allocation des blocs de la disquette. Le bit 7 des caractères ASCII placés dans le tampon clavier est automatiquement mis à 1. La chaîne ainsi rentrée sera acceptée après un retour-chariot (CR) comme dans la routine du Moniteur GETLN.

L'adresse $BE03 est le point d'entrée standard des commandes internes de ProDOS : celles-ci sont analysées grâce aux paramètres déclarés, puis exécutées si elles sont reconnues. Lors de la détection d'une erreur, le code est sauvegardé à l'adresse $BE0F (ERRCODE). Le Basic système gère 19 codes différents et toute erreur non reconnue émettra le message : I/O ERROR.

Lorsqu'une erreur est reconnue, son code se trouve dans l'accumulateur, et Basic effectue un saut à la routine PRINTERR pour l'afficher ($BE0C).

COMMANDES EXTERNES A ProDOS

Généralités

L'entrée EXTRNCMD ($BE06) autorise l'adjonction d'autres commandes que celles qui sont définies par le système, et accroît considérablement les possibilités de ProDOS.

L'entrée standard de toute commande ProDOS se situe à l'adresse $BE03 (DOSCMD). Elle effectue un contrôle de syntaxe de l'ordre transmis.

Lorsqu'un ordre est passé à Basic, le système vérifie si c'est bien une commande ProDOS. Dans le cas contraire, et au retour de la routine, le système effectue un saut à l'entrée EXTRNCMD - adresse $BE06. A ce vecteur se trouvera un JMP (code 4C) qui devra être suivi par le pointeur de la commande externe. En temps normal, les adresses $BE07-$BE08 ont comme valeur "9E BE" qui sont les octets poids faible et poids fort de $BE9E, lequel contient un RTS (code 60). Celui-ci redonne le contrôle à Basic.

Lorsqu'une commande ProDOS est déclarée, la chaîne de caractères sera placée dans le tampon d'entrée du clavier, et ses huit premiers octets copiés dans le tampon pointé par VPATH1, $BE6C,$BE6D. L'oubli du pointeur en VPATH1 fera perdre la chaîne interceptée, et mettra l'utilisateur dans l'impossibilité de disposer d'une commande AppleSoft, avec les conséquences que cela entraîne. Parmi les huit caractères pris en considération, les blancs (caractères "espace") ne seront pas comptabilisés, mais leurs codes ASCII seront quand même copiés dans le tampon clavier.

COMMAND-HANDLER

Trois possibilités s'offrent au système :

1 - C'est une commande ProDOS : elle sera analysée, puis exécutée telle quelle par la routine en $BE03 - Command-Handler 1.
2 - Ce n'est pas une commande ProDOS : au retour de Command-Handler 1, le système se branche à l'adresse $BE06 - Command-Handler 2 - qui devra pointer vers le sous-programme traitant la commande externe.
3 - L'une ou l'autre des commandes précitées comporte des paramètres optionnels. Les bytes de définition - PBITS/PBITS+1 - sont placés, au fur et à mesure de l'exécution de la commande, aux adresses $BE54,$BE55 puis analysés par le Basic système. Cette structure permet de traiter des commandes complexes avec paramètres optionnels.

ANALYSE DES PARAMETRES OPTIONNELS

Toute commande ProDOS, suivie de paramètres optionnels, sera vérifiée par le système. Lorsqu'une commande externe sera déclarée, il faudra indiquer à ProDOS les paramètres autorisés, en alimentant les bits de PBITS et PBIT+1. Pour les commandes internes, ProDOS effectue cette manipulation de lui-même.

Tableau PBITS/PBITS+1

Adresse	$BE54	$BE55
PBITS/PBITS+1	|_|_|_|_|_|_|_|_|	|_|_|_|_|_|_|_|_|
Bits	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00

SIGNIFICATION DES DIFFERENTS BITS

Bits	*Significations*
15	Nécessité de déclarer le préfixe : chemin (Pathname) optionnel.
14	Pas de paramètre déclaré lors du déroulement.
13	Commande unique, durant l'exécution : sans option.
12	Nom du fichier optionnel.
11	Commande CREATE autorisée si le nom du fichier n'existe pas.
10	Type de fichier optionnel (paramètre Ttype).
09	Deuxième nom de fichier attendu (RENAME, etc.).
08	Premier nom de fichier attendu.
07	Paramètre adresse alloué : A#
06	Paramètre byte alloué : B#
05	Paramètre fin d'adresse alloué : E#
04	Paramètre longueur alloué : L#
03	Paramètre numéro de ligne alloué : à#
02	Paramètres slot et drive alloué : S# et D#
01	Paramètre de champ alloué : F#
00	Paramètre du numéro d'enregistrement : R#

Recherche des paramètres par ProDOS

Si la commande ne nécessite pas de paramètres, ou si l'on préfère les manipuler soi-même, il faudra placer un $00 aux adresses $BE54,$BE55, PBITS-PBITS+1, puis, par un RTS, effectuer un retour à ProDOS, sans oublier d'annuler la retenue (Carry). Dans le cas contraire, il faudra indiquer au système les paramètres optionnels autorisés, en approvisionnant PBITS et PBITS+1 par les bits dont le détail est dressé dans la précédente table. Par ailleurs, pour que ProDOS reconnaisse une commande externe, il faudra placer un $00 en $BE53 (XCNUM), et mettre en place aux adresses $BE50,$BE51 (XTRNADDR) le vecteur de retour après l'analyse des paramètres. La longueur diminuée de 1 du nom de la commande devra être placée en $BE52 (XLEN), par exemple 3 pour "HELP".

ProDOS se chargera par la suite d'analyser la syntaxe de votre commande, et d'afficher éventuellement le message de l'erreur rencontrée. Si le processus normal se déroule, ProDOS rappelle le programme grâce au pointeur qui se trouve en $BE50,$BE51. Les paramètres éventuellement trouvés sont signalés par un bit de présence en

$BE56,$BE57 - FBITS, FBITS+1. Ensuite, la valeur réelle de chaque paramètre est sauvegardée à son adresse respective dans une table située à la page globale de Basic, aux adresses $BE58-$BE6B. VPATH1 ($BE6C,$BE6D) pointe le tampon contenant la longueur du nom, codé sur un octet, et les caractères du nom avec le bit 7 à 0. Si le préfixe a été demandé (bit 15 de PBITS), le chemin complet d'accès au fichier sera stocké dans le tampon. Le programme pourra ensuite poursuivre son exécution normale.

Appel de ProDOS en assembleur

La phase des opérations diffère par rapport au DOS 3.3 où il était possible d'envoyer des commandes précédées par CTRL-D, puis d'appeler la routine de sortie de caractères - COUT - de la ROM du Moniteur - $FDED.

Sous ProDOS, cette pratique est devenue caduque : il faudra dorénavant placer la chaîne de caractères dans le tampon clavier, à partir de l'adresse $0200, avec les bits forts à 1 (bits 7), la faire suivre par un retour-chariot (code #$8D), puis appeler DOSCMD à l'adresse $BE03. Il est évident que cette méthode ne peut fonctionner qu'en présence du système Basic. Cette pratique est illustrée par l'exemple suivant :

```
            LDY   #$00        Initialise le registre X.
ENCORE      LDA   COMMANDE    Charge les caractères ASCII de la commande,
            STA   $0200,Y     et les place dans le tampon clavier.
            INY
            CMP   #$8D        Est-ce que la touche RETURN a été sollicitée ?
            BNE   ENCORE      non, continue de charger les caractères.
            JMP   DOSCMD      Saut à l'entrée standard : $BE03.
COMMANDE    ASC   "CAT"       Syntaxe de la commande.
            DFB   $8D         Retour-chariot, qui accepte les caractères ASCII.
```

Il existe un autre moyen d'appeler des commandes à partir d'un programme binaire, en faisant un appel à PRODOS par l'entrée de sa syntaxe générale, sous la forme :

```
JSR  $BF00
DFB  COMMANDE
DA   PARAMETRE
BCS  ERREUR
```

Au retour, la retenue (Carry) est à 1 si la commande n'a pas abouti, avec le code de l'erreur dans l'accumulateur.

La table des paramètres débute toujours par un octet indiquant le nombre de paramètres qui suivent - Parameter Count. (On trouvera en annexe 8 la liste complète des commandes MLI de PRODOS. Le chapitre 6 traite les commandes MLI.)

Règles générales pour une programmation opérationnelle

Sous ProDOS, il sera bon, dès le départ, d'adopter une discipline stricte pour l'utilisation et l'appel de certaines adresses du système, pour ne pas aboutir au même résultat qu'avec le DOS 3.3. En effet, celui-ci était truffé de "patchs", ce qui le rendait souvent inutilisable avec certains programmes vendus dans le commerce. Cette situation a sûrement pesé dans le choix des programmeurs d'Apple Computer et Cupertino, d'adopter en définitive le système d'exploitation ProDOS. Théoriquement, les programmes doivent pouvoir coexister en mémoire, quel que soit le nombre de fichiers ouverts. ProDOS comporte une table des pages libres de la RAM - $BF58-$BF6F.

Règles fondamentales :

- Les routines et sous-programmes courts et d'occupation momentanée se placeront à l'adresse $0300. Ce sont généralement les programmes chargés et exécutés une seule fois.

- Les programmes ne faisant pas appel à ProDOS devront au préalable examiner la table d'occupation de la RAM, puis se placer sur les plus hautes pages disponibles.

- Les programmes devront également marquer sur la table d'occupation de la RAM les pages mémoire dont ils auront besoin pour leur exécution. Cette disposition leur évitera d'être écrasés par les programmes futurs, ou par ProDOS lui-même lorsque celui-ci gère ses tampons et déplace HIMEM.

La table d'occupation de la RAM, qui gère les pages libres de la mémoire, contient 24 octets : chaque bit représente le statut d'une page allant de #$00 à #$BF (0-191). Le bit est à 1 si la page est libre, et à 0 si la page est occupée. Les pages #$00-#$02 sont toujours marquées occupées (#$00), ainsi que les pages #$9A à #$BF (à condition que le BASIC.SYSTEM soit chargé).

Mise en place d'une commande externe

Plusieurs procédés permettent d'implanter un programme en langage machine dans l'environnement existant. Les ordres requis peuvent être transmis à partir d'un programme machine dont le rôle sera d'initialiser les commandes externes. Ce programme machine devra être implanté dans un espace mémoire inutilisé par le système, donc libre par définition, et avant l'exécution d'un programme AppleSoft. Il sera ensuite bon de vérifier si ProDOS est le système actuel, car, dans le cas contraire, la poursuite du programme serait aléatoire. Il faudra vérifier si ProDOS n'a pas ouvert de fichier, à la limite appeler une routine pour une éventuelle réservation des pages mémoire nécessaires à l'exécution du programme. Ensuite, le programme à placer en mémoire pourra se reloger sur les pages allouées précédemment avec la routine MOVE.

Si le programme machine doit être chaîné à ProDOS comme une commande externe, il faudra au préalable sauvegarder l'adresse de Command-Handler 2 qui se trouve en $BE07-$BE08, puis y placer le pointeur de la nouvelle commande. Si le programme de commande externe doit être appelé par STARTUP, il faudra le sauvegarder sur la disquette sous le nom XXX.CODE, où XXX représente le nom du fichier binaire.

LA PAGE GLOBALE DE PRODOS - $BF00-$BFFF

Généralités

La "PRODOS Global Page" débute à la page $BF (191) de l'espace mémoire d'un Apple II, et occupe les adresses $BF00-$BFFF. Elle contient toutes les données et paramètres du système d'exploitation, pour effectuer des appels aux sous-programmes MLI logés dans la carte langage. La page complète est listée par la suite, en utilisant les codes DFB et DA qui sont des directives de l'assembleur BIG-MAC : ils désignent respectivement une valeur courante ou une adresse courante représentée par deux bytes : Low byte et High byte - LByte et HByte.

Listing commenté de la PRODOS Global Page

- Version 1.0.2 (1984)

Remarque : comme plusieurs versions de ProDOS circulent actuellement, certaines adresses diffèrent d'une série à l'autre. Cette situation particulière sera signalée par la mention "non officiel".

• Machine Language Interface

Le MLI est appelé par un JSR des adresses suivantes :

```
$BE82   (BASIC.SYSTEM)
$D141   (bank 2 programme de Reboot)
$D1BF
$D24F
$D26B
$D282
$D292
$D2B7
$D2BE

BF00 -      4C B7 BF      JMP $BFB7   MLIENTRY.
```

• Programme de Reboot

Débuter par :

```
LDA $C08B
LDA $C08B
JMP REBOOT

BF03 -  4C DB EE      JMP $EEDB     REBOOT. - Non officiel
```

• SET.TIME

Routine horloge. Le sous-programme est appelé par un JSR des adresses :

$D060 (Bank 1)
$D270
BF06 - RTS SET.TIME.
BF07 - DFB 42 Avec la carte horloge,
BF08 - DFB F1 contient 4C 42 F1 d'ou JMP $F142.

• SET.SYSERR

Routine gestion d'erreur. Le sous-programme est appelé par un JSR des adresses :

$D0A4 (Bank 1)
$D0D7
$D121
$D278
$DFB4
$EB7C
$EC1C
$EC4D

BF09 - 4C DA D1 JMP $D1DA SET.SYSERR - Non officiel

• SYSTEM DEATH

Le sous-programme est appelé par les adresses :

$D19C (Bank 1)
$D40F
$D414
$E04C
$EC71

BF0C - 4C E6 D1 JMP $D1E6 SYSDEATH. - Non officiel

• SYSTEME ERROR

BF0F - 00 HEX 00 SYSERR. Sauvegarde du code de l'erreur.

• Drive Table

Cette table contient un pointeur - unique - d'entrée de la routine Read Write Tracks Blocks qui débute à l'adresse $F800, pour chaque slot et drive dont une interface sera détectée. Contrairement à la Device Table, les adresses RWTB sont agencées par rapport au numéro du slot. Si la routine incorporée dans le Relocator détecte une interface, elle sauvegarde dans la Drive Table, aux adresses appropriées, le pointeur de la RWTB. Par exemple, lorsqu'une interface Disk II est détectée, le système copie le pointeur $F800 à l'emplacement attribué à la table, tandis que l'adresse de la RAM-Disk sera fixée à $FF00 - RAM de 128 Ko.
Par définition, le slot 3, drive 2 est réservé à la RAM-Disk.

Tout slot libre, ou dont l'interface n'aura pas été reconnue, contiendra le pointeur $D0A2, qui est l'entrée de la routine NO DEVICE.
La "Drive Table" n'est sollicitée à l'intérieur du MLI que par la routine "DO DISK I/O", implantée à l'adresse $D0DA. Sa fonction essentielle est la détection de toute anomalie au niveau d'une interface lecteur de disquettes.
Le paramètre "Unit Number" - format DSSS0000 - sera divisé par 8, ce qui fera apparaître un nouveau format: 000DSSS0 (correspond à D*10 + SSS*2) et donne, comme valeur de base, l'index sauvegardé à partir de $BF10.

La Drive Table, faisant partie de la page globale de PRODOS, contient l'adresse $D0A2 pour chaque interface non reconnue. L'adresse $D0A2 est l'entrée de la routine NO DEVICE qui fixe le code d'erreur "ERR" du MLI à la valeur #$28 : NO DEVICE CONNECTED. Le retour se fera par SET.SYSERR via $D0D7 à la routine qui aura appelé DO DISK I/0 :

D0D7 - 20 09 BF JSR $BF09 SET.SYSERR

Slots 0-7 Drive 1

BF10 - SL0DR1 DA $D0A2 Slot 0, réservé
BF12 - SL1DR1 DA $D0A2 Slot 1, drive 1
BF14 - SL2DR1 DA $D0A2 Slot 2, drive 1
BF16 - SL3DR1 DA $D0A2 Slot 3, drive 1
BF18 - SL4DR1 DA $D0A2 Slot 4, drive 1
BF1A - SL5DR1 DA $D0A2 Slot 5, drive 1
BF1C - SL6DR1 DA $F800 Slot 6, drive 1
Interface Disk II détectée.
BF1E - SL7DR1 DA $D0A2 Slot 7, drive 1

Slot 0-7 drive 2

BF20 - SL0DR2 DA $D0A2 Slot 0 réservé
BF22 - SL1DR2 DA $D0A2 Slot 1, drive 2
BF24 - SL2DR2 DA $D0A2 Slot 2, drive 2
BF26 - SL3DR2 DA $FF00 Slot 3, drive 2
RAM Disk si 64 Ko auxiliaire
BF28 - SL4DR2 DA $D0A2 Slot 4, drive 2
BF2A - SL5DR2 DA $D0A2 Slot 5, drive 2
BF2C - SL6DR2 DA $F800 Slot 6, drive 2
Interface Disk II détectée.
BF2E - SL7DR2 DA $D0A2 Slot 7, drive 2

• Last Device

Dernier périphérique sollicité :
Numéro du drive = drive -1
$00, $01 et $02 = 3 LASTDRV

BF30 - LASTDEV HEX 00 Dernier slot
BF31 - DEVNUM HEX 00 Dernier drive

• Device Table

Device bit matrice :
BITS : 76543210
DSSSTTTT

Drives
0 drive 1
1 drive 2

Slots
110 slot 6
011 slot 3

Type de périphérique
0000 drive Disk II
0100 disque Profile
1111 RAM carte

Exemples :

11100000 S6,D2
01100000 S6,D1
10111111 S3,D2

BF32 - DEVICE1 DFB $00 Valeur des bits en fonction des périphériques en ligne.
BF33 - DEVICE2 DFB $00
BF34 - DEVICE3 DFB $00
BF35 - DEVICE4 DFB $00
BF36 - DEVICE5 DFB $00
BF37 - DEVICE6 DFB $00
BF38 - DEVICE7 DFB $00
BF39 - DEVICE8 DFB $00
BF3A - DEVICE9 DFB $00
BF3B - DEVICE10 DFB $00
BF3C - DEVICE11 DFB $00
BF3D - DEVICE12 DFB $00
BF3E - DEVICE13 DFB $00
BF3F - DEVICE14 DFB $00

• Copyright Apple

BF40 - 43 4F 50 52 2E 20 41 50 ASC 'COPR. APPLE,1983'
BF48 - 50 4C 45 2C 31 39 38 33

• Adresses système

Les adresses $BF50-$BF57 sont réservées au système.

Lors d'une interruption IRQ, avec la carte langage commutée "on" (MLI actif), l'adresse de retour est fixée à la valeur $BF50, par l'intermédiaire de la routine IRQ-handler - $FF9B.

La routine LANGINTEXIT, qui débute en $FFD8, réactualise l'adresse $0045 de la page zéro - RWTB Pointer - en chargeant la valeur qui se trouve à $BF56. Cette adresse est utilisée pour une sauvegarde intermédiaire de l'emplacement mémoire, $0045, lors d'une interruption du type IRQ, quand la carte langage est commutée "on".

L'adresse $BF57 est utilisée pour la sauvegarde d'un drapeau (Flag positionné à 1) pour signifier au système qu'une interruption IRQ est survenue avec la carte langage commutée "on". Conséquence : INTEXIT, adresse $BFD0 sera sollicitée par deux fois, et commute au premier passage par INTBANK1 l'espace mémoire sur le Moniteur - Monitor Interrupt.

BF50 - 8D 8B C0 STA $C08B LANGIRQ - Non officiel
BF53 - 4C D8 FF JMP $FFD8 PRENIBBLE - Non officiel
BF56 - HEX 00 INTACCU - Non officiel
BF57 - HEX 00 INTBANK2 - Non officiel

• Système Bit-Map

Table d'occupation mémoire :

La table d'occupation de la RAM se trouve aux adresses $BF58-$BF6F et traduit l'état (occupé si le bit à 1, ou libre si le bit à 0) de chaque page (256 bytes) de l'espace mémoire, délimité par les adresses $0000-$BFFF. Ce sont au total 24 bytes qui gèrent les 192 premières pages du système, et chaque page nécessite un bit pour le codage

(24*8 = 192 pages).

Représentation par page :

Adresses	Espace mémoire	Pages représentées
$BF58-$BF5F	XXXXXXXX	$00-$3F
$BF40-$BF67	XXXXXXXX	$40-$7F
$BF68-$BF6F	XXXXXXXX	$80-$BF

La configuration d'un byte de la table est assez singulière : les bits sont évalués dans l'ordre inverse de leur représentation. C'est ainsi que le bit 7 du byte se trouvant à l'adresse $BF58 désigne les 256 bytes de la page $00, tandis que le bit 0 du byte de l'adresse $BF6F représente la dernière page, $BF.

Position des bits :

Bit 1 = page occupée.
Bit 0 = page libre.

Exemple :

Valeur du byte de l'adresse $BF58 :

Bits : 01234567
11001111 Zone mémoire de $0000-$07FF codée #$CF.

Les pages $00, $01 et la page texte 1 (pages 4,5,6 et 7)
sont marquées occupées au départ.

```
$BF58- SBITMAP   DFB $CF,$00,$00,$00,$00,$00,$00,$00
                 DFB $00,$00,$00,$00,$00,$00,$00,$00
                 DFB $00,$00,$00,$00,$00,$00,$00,$00
```

Occupation des pages mémoire avec Basic.System en mémoire

Pages	*Bits*	*Adresses*	*Situation des pages*
$00-$07	11001111	$0000	Pages 0,1 et 4 à 7 occupées
$08-$0F	00000000	$0800	Pages libres
$10-$17	00000000	$1000	Pages libres
$18-$1F	00000000	$1800	Pages libres
$20-$27	00000000	$2000	Pages libres
$28-$2F	00000000	$2800	Pages libres
$30-$37	00000000	$3000	Pages libres
$38-$3F	00000000	$3800	Pages libres
$40-$47	00000000	$4000	Pages libres
$48-$4F	00000000	$4800	Pages libres
$50-$57	00000000	$5000	Pages libres
$58-$5F	00000000	$5800	Pages libres
$60-$67	00000000	$6000	Pages libres
$68-$6F	00000000	$6800	Pages libres
$70-$77	00000000	$7000	Pages libres
$78-$7F	00000000	$7800	Pages libres
$80-$87	00000000	$8000	Pages libres
$88-$8F	00000000	$8800	Pages libres
$90-$97	00000000	$9000	Pages libres
$98-$9F	00111111	$9800	Pages 9A à 9F occupées
$A0-$A7	11111111	$A000	Pages A0 à A7 occupées
$A8-$AF	11111111	$A800	Pages A8 à AF occupées
$B0-$B7	11111111	$B000	Pages B0 à B7 occupées
$B8-$BF	11000011	$B800	Pages B8-B9 et BE-BFoccupées

• Buffer table

Table des tampons fichiers :

Cette table contient les pointeurs des tampons alloués aux fichiers ouverts, avec un maximum de 8 fichiers. La "File Control Block" comporte un index (byte à la position

relative #$0B, ou 11 décimal, de la FCB) qui correspond à la valeur de "Reference Number" multipliée par 2.
"Reference Number" est un nombre alloué par le système lors de l'ouverture d'un fichier par la commande OPEN. Il peut prendre toute valeur de 1 à 8, et dépend directement du nombre de fichiers ouverts: 1 pour 1 fichier, 2 pour 2 fichiers, et ainsi de suite.
Index est une valeur qui servira par la suite à déterminer l'adresse de sauvegarde du pointeur attribué à un fichier ouvert. Sa valeur, sauvegardée dans la table FCB, est un nombre de 2 à 16, et sera ajoutée à une adresse de base - $BF6E,$BF6F - qui contiendra le pointeur du tampon alloué.

La commande OPEN, adresse de début $E0B9, charge en $E177 par un "LDA $F309,Y" la valeur de l'index utilisé par la suite pour le calcul de l'adresse - LByte,HByte - stockée dans la "Buffer table".

Exemple :

Si dans la table FCB (début $F300), au byte relatif 11, se trouve la valeur 2 comme index du premier fichier ouvert, celle-ci sera ajoutée respectivement aux adresses $BF6E,$BF6F, dont les valeurs réelles seront $BF70,$BF71. La paire d'adresses $BF70,$BF71 est bien celle qui doit contenir le pointeur du tampon du premier fichier ouvert. Le raisonnement sera identique pour les tampons alloués aux fichiers suivants.

Remarque : l'index de la table FCB est multiplié par 2 dans la routine SET.BUFFER, qui débute à l'adresse $EDD9. A l'adresse $EE16 est effectué un ASL, qui a comme conséquence de multiplier par 2 la valeur de l'accumulateur :

```
EE13 - B9 00 F3   LDA $F300,Y   Charge "Reference Number",
EE16 - 0A         ASL           le multiplie par 2,
EE17 - 99 0B F3   STA $F30B,Y   et le sauvegarde comme index pour le calcul
                                de l'adresse contenue dans la Buffer Table.
```

BUFTBL :

```
BF70 - BUFFER1  DA $0000   Adresse du tampon 1
BF72 - BUFFER2  DA $0000   Adresse du tampon 2
BF74 - BUFFER3  DA $0000   Adresse du tampon 3
BF76 - BUFFER4  DA $0000   Adresse du tampon 4
BF78 - BUFFER5  DA $0000   Adresse du tampon 5
BF7A - BUFFER6  DA $0000   Adresse du tampon 6
BF7C - BUFFER7  DA $0000   Adresse du tampon 7
BF7E - BUFFER8  DA $0000   Adresse du tampon 8
```

• Interrupts

Vecteur des adresses d'interruptions :

```
BF80 - INTRUPT1  DA $0000   Routine d'interruption 1
BF82 - INTRUPT2  DA $0000   Routine d'interruption 2
BF84 - INTRUPT3  DA $0000   Routine d'interruption 3
BF86 - INTRUPT4  DA $0000   Routine d'interruption 4
```

• Registres du processeur

Sauvegarde des registres du microprocesseur :

BF88 -	INTAREG	DFB $00	Registre A - Accumulateur
BF89 -	INTXREG	DFB $00	Registre d'index X
BF8A -	INTYREG	DFB $00	Registre d'index Y
BF8B -	INTSREG	DFB $00	Registre pointeur de pile
BF8C -	INTPREG	DFB $00	Registre d'état PS

• Carte Langage

Bank 1, Bank 2, ROM statut :

BF8D -	INTBANK1	DFB $00	ROM, RAM1 ou RAM2 - $D000 -. - Non officiel.
BF8E -	INTADDR	DA $0000	Adresse de retour au programme en cours d'exécution, lors d'une interruption IRQ.

• System Time

Vecteur horloge : DATE

En l'absence d'une carte horloge compatible Apple, les vecteurs date et heure peuvent être mis en place par un programme écrit en langage machine. L'annexe 10 donne un exemple de ce type de programme, pour la gestion date de ProDOS.

Représentation des bytes :

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 — Année/Mois/Jour

BF90- DATE DA $0000 — Bits 15-9 Année
Bits 08-05 Mois
Bits 04-00 Jour

Vecteur horloge : HEURE

Représentation des bytes :

Bits : F D E C B A 9 8 7 6 5 4 3 2 1 0
x x x H H H H H x x M M M M M M — Heure/Minute
x : bits inutilisés

BF92- TIME DA $0000 — Bits 12-08 Heure
Bits 05-00 Minute

• Level

Niveau du fichier :
C'est la valeur attribuée à un fichier texte, lors de son ouverture par la commande OPEN, et copiée dans la table FCB - byte relatif #$00 de l'entrée correspondante. Avec la commande CLOSE, pour fermer tous les fichiers ouverts, la valeur stockée dans LEVEL sera prise en considération. La commande CLOSE du système ProDOS place par défaut la valeur 7 à l'adresse $BF94.

BF94- LEVEL DFB $00 Niveau du fichier ouvert, utilisé par OPEN, FLUSH et CLOSE.

• Bubit

Le bit 5 est seul utilisé - valeur #$20. Chaque routine MLI utilise cette adresse pour sa sortie (EXIT-$D078) et fixe la valeur du byte à #$00. L'adresse $BF95 est lue par SET.FILE.INFO qui effectue une liaison interne au MLI, pour fixer ou annuler le bit de Backup à l'adresse $F074.
L'adresse $BF95 est utilisée comme emplacement de transition entre le MLI dans la carte langage et la mémoire principale.

BF95- BUBIT DFB $00 Byte de contrôle utilisé par CREATE, RENAME, CLOSE, WRITE et SET FILE.

BF96- SPARE DA $0000 Réservé pour des applications futures.
Contient NOP, NOP.

• Identification de la "machine"

Cette adresse contient l'image binaire de l'identification de la "machine", contenue dans un byte.

Détail des bits :

- le type d'Apple et sa configuration ;
- le slot ;
- le préfixe ;
- situation du MLI ;
- adresse de retour ;
- registre X ;
- registre Y.

Bits d'identification machine :

Bits: 76543210

	Type d'Apple II
00	Apple II
01	Apple II+

10	Apple IIe
11	Apple III Emulation
	Capacité mémoire
00	inutilisée
01	48 Ko
10	64 Ko
11	128 Ko
	Libre
00	en réserve
	Carte 80 Colonnes
1	oui
0	non
	Carte horloge
1	oui
0	non

Exemple d'identification machine :

Bits : 76543210
10110010 = $B2
= Apple IIe, 128 Ko RAM et 80 colonnes.

Exemple d'identification des slots :

Bits : 76543210
01001010 = $4A

= slots 6, 3 et 1 occupés.

Exemple MLI actif :

Bits : 76543210
01010101 = $55

Décalé par l'instruction machine ROR.

BF98 -	MACHID	DBF $00	Identification.
BF99 -	ROMBIT	DBF $00	Bits slot : slot 7 = 7.
BF9A -	PREFIX	DFB $00	Si 0 pas de préfixe défini.
BF9B -	MLIACTV	DFB $00	Bit 7 = 1 si MLI actif.
BF9C -	MLIRTS	DA $0000	Adresse de retour du dernier appel MLI. - Non officiel.
BF9E -	XSAVE	DFB $00	Sauvegarde du registre X.
BF9F -	YSAVE	DFB $00	Sauvegarde du registre Y.

• MLI-EXIT

Avant le branchement à l'adresse $BFA0, le système sauvegarde sur la pile, à l'intérieur du MLI (EXIT-MLI $D078), l'adresse de retour du programme en cours d'exécution. Ensuite le registre d'état PS et le code d'erreur seront placés à leur tour sur le sommet de la pile. L'accumulateur sera chargé avec la valeur de l'adresse $BFF4, BANKSEL, qui contient le drapeau de la mémoire commutée : bank 1, bank 2 ou ROM active. Après le retour à la mémoire commutée, avant l'appel du MLI, le code d'erreur "ERR#" sera récupéré de la pile. Le RTI, qui se trouve à l'adresse $BFB6, sera actif dès que la valeur du registre d'état PS sera dépilée : le retour se faisant au programme en cours d'exécution.

Bytes significatifs des commutateurs :

ROM Moniteur	$D000	= #$6F ; $E000 = #$4C
Bank 1	$D000	= #$D8 ; $E000 = #$20
Bank 2	$D000	= #$EE ; $E000 = #$00

Situation des commutateurs :

	ROM	Bank 1	Bank 2
Etat	READ	RD/WR	RD/WR
Commutateurs	C082	C081	C08B
		C081	C08B

BFA0 -	MLIEXIT	EOR	$E000	Test de la ROM - Non officiel.
BFA3 -	F0 05	BEQ	$BFAA	Bank 1 est, ou a été,commuté.
BFA5 -	8D 82 C0	STA	$C082	ROMon. Moniteur : AppleSoft.
BFA8 -	D0 0B	BNE	$BFB5	Branche toujours suivant la situation de ROMon.
BFAA -	AD F5 BF	LDA	$BFF5	BANKSEL2. Alterne RAM/ROM.
BFAD -	4D 00 D0	EOR	$D000	Bank 1 commutée ?
BFB0 -	F0 03	BEQ	$BFB5	Branche si bank 1 active ;
BFB2 -	AD 83 C0	LDA	$C083	sinon, sélectionne la bank 2.
BFB5 -	68	PLA		Récupère le code d'ERR#.
BFB6 -	40	RTI		Retour au programme en cours.

• MLI ENTRY

Entrée pour effectuer un saut à la routine MLI par la Bank 1 de la carte langage - $D000. La bank 1 est en lecture/ écriture après l'entrée MLIENT1 : elle est sollicitée par MLI = $BF00.

BFB7 -	MLIENT1	SEC		- Non officiel.
BFB8 -	6E 9B BF	ROR	$BF9B	Fixe le bit 7 de MLIACTIV.
BFBB -	AD 00 E0	LDA	$E000	Carte langage/ ROM.

```
BFBE - 8D F4 BF   STA  $BFF4   BANKSEL - $E000 de la bank active.
BFC1 - AD 00 D0   LDA  $D000   Restaure si MLI actif.
BFC4 - 8D F5 BF   STA  $BFF5   BANKSEL1 - $D000.
BFC7- AD 8B C0    LDA  $C08B   Sélectionne la bank 1,
BFCA - AD 8B C0   LDA  $C08B   et autorise l'écriture.
BFCD - 4C 00 D0   JMP  $D000   Saut au MLI.
```

• INTERRUPT EXIT

Entrée sollicitée par un JMP de l'adresse $D1CB (bank 1) et par un JSR de $E1F2. Au retour de l'interruption, le système rétablit la Bank qui était en service initialement.
La routine INTEXIT - $BFD0 - est sollicitée par le sous-programme IRQ-Handler du MLI qui a son entrée à l'adresse $D13A, après l'exécution à terme d'une routine d'interruption.

```
BFD0 - INTEXIT  LDA  $BF8D   INTBANK1. - Non officiel.
BFD3 - F0 0D    BEQ  $BFE2   Bank 1 est, ou a été commutée.
                             Sollicité par un JMP de l'adresse $FFE9.
BFD5 - 30 08    BMI  $BFDF   Bank 2 actuelle, commuter.
BFD7 - 4A       LSR
BFD8 - 90 0D    BCC $BFE7
BFDA - AD 82 C0 LDA $C082    ROMon - lecture autorisée.
BFDD - B0 08    BCS $BFE2    Branchement inconditionnel.
BFDF - AD 83 C0 LDA $C083    BANKSELECT. Sélectionne la bank 2.
BFE2 - A9 01    LDA #$01     Flag - Monitor-IRQ,
BFE4 - 8D 8D BF STA $BF8D    et sauvegarde comme INTBANK1.
BFE7 - AD 88 BF LDA $BF88    INTAREG. Rétablit la valeur de l'accumulateur -
                             IRQ.
BFEA - 40       RTI          Retour au programme en cours d'exécution.
```

• INTERRUPT ENTRY

Cette entrée est pointée par deux bytes du vecteur IRQ, de la page 3 - $03FE-$03FF.
Lors d'une interruption IRQ issue du Moniteur, seul l'accumulateur sera sauvegardé à l'adresse $0045 de la page zéro, avant qu'un saut ne soit effectué à $BFEB.
Lors d'une interruption IRQ issue de la carte langage, INTBANK1 et INTBANK2 seront positionnés - Flag - et le contenu de l'accumulateur sauvegardé à l'adresse $0045 (routine $FF9C) :

```
FF9C - A5 45    LDA  $45
FF9E - 8D 56 BF STA  $BF56
FFA1 - 68       PLA
FFA2 - 85 45    STA  $45
```

```
BFEB - INTENTRY BIT  $C08B   Sélectionne la RAM en lecture - Non officiel -
BFEE - 2C 8B C0 BIT  $C08B   et en écriture.
BFF1 - 4C 3A D1 JMP  $D13A   Saut au sous-programme MLI
                             IRQ-Handler.
```


• BANKSELECT BYTES

Les adresses $BFF4-$BFF5 sont réservées pour la sauvegarde d'un drapeau - Flag - pour mémoriser l'état du commutateur mémoire, lors d'un appel par l'entrée MLIENT1 - $BFB7 - (bank 1, bank 2 ou ROM).

Avec MLIEXIT - $BFA0 - le système reconnaît quelle bank était commutée avant l'appel du MLI, pour effectuer un retour avec la bonne commutation mémoire.

Exemple :

$4C = $E000
$6F = $D000

```
BFF4 - BANKSEL  DFB  $00  Flag - Non officiel.
BFF5 - BANKSEL1 DFB  $00
```

• SYSDEATH

L'adresse $BFF6 est très particulière : avant que le vecteur Reboot ne soit installé dans la bank 2 de la carte langage, l'adresse $BF09 pointe l'entrée SYSDEATH2. Ensuite, le sous-programme Reboot-Handler modifie l'adresse $BF09, après la copie de Reboot par le Relocator.
Lorsque l'entrée $BFF6 est sollicitée, le message suivant s'affiche : INSERT SYSTEM DISK AND RESTART.

```
BFF6 - SYSDEATH2 BIT  $C08B  - Non officiel.
BFF9 - 4C E4 D1  JMP  $D1E4  SYSTEM DEATH sans ERR#.
                             La version 1.0 contient l'adresse $D1DB.
```

• PRODOS version

Les prochaines adresses devraient, d'après le manuel de référence, contenir les caractéristiques de la version de PRODOS. Nous avons pu tester plusieurs versions du système, et à chaque fois les bytes des adresses $BFFC-$BFFF étaient nuls.

```
BFFC - IBAKVER   DFB  $00  MLI version antérieure.
BFFD - IVERSION  DFB  $00  Version actuelle du système.

BFFE - KBAKVER   DFB  $00  Versions compatibles.
BFFF - KVERSION  DFB  $00  Numéro de la version.
```

LA CARTE LANGAGE

Commutateur logique

Un commutateur logique permet de sélectionner, ou de commuter, un endroit bien déterminé de l'espace mémoire. La plupart de ces commutateurs ont trois points d'entrées : l'un pour les connecter, l'autre pour les débrancher et le troisième pour lire leur état. Voir, en annexe 9, les différents points d'entrée des commutateurs logiques avec leurs fonctions.

PRODOS, à l'encontre du BASIC.SYSTEM, se loge dans la carte langage, et occupe ainsi 16 Ko en haut de la mémoire. Comme les adresses $D000-$FFFF ne comptabilisent que 12 Ko, il faut bien que les 4 Ko manquants soient pris quelque part : c'est la "bank 2" qui est utilisée conjointement avec la "bank 1", par simple commutation alternée. L'action des commutateurs permet, entre autres, d'effectuer soit une lecture, soit une écriture, soit encore une lecture et écriture de la carte langage.

Situation typique de la ROM :

ROM : $D000 - $FFFF (AppleSoft + Moniteur)
Bank 1 : $D000 - $FFFF (PRODOS)
Bank 2 : $D000 - $FFFF (à partir de $D100 programme de Reboot)

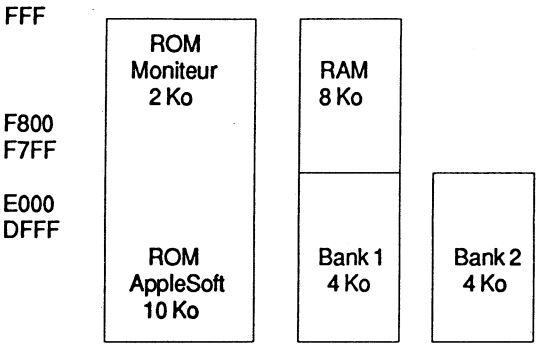
Activation des commutateurs :

$D000 - $DFFF Bank 1 avec 4 Ko
$D000 - $DFFF Bank 2 avec 4 Ko
$E000 - $FFFF Bank 1 et 2 avec 8 Ko

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

Légende : R/W = lecture/écriture
R = lecture
W = écriture

Carte de la mémoire



Les 4 Ko de l'espace mémoire qui se trouvent à $C000-$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. La ROM intégrée dans la configuration est logée dans l'espace $D000-$FFFF, et occupe ainsi 12 Ko qui se placent en parallèle sur la RAM. La ROM regroupe la ROM AppleSoft ($D000-$F7FF) et la ROM du Moniteur ($F800-$FFFF). Mais une zone a été plus particulièrement élaborée par le constructeur : ce sont les 4 Ko se situant aux adresses $D000-$DFFF et qui sont en fait 2 X 4 Ko placés en parallèle. Ces zones sont appelées "bank 1" et "bank 2". 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 oeuvre par un POKE ou un PEEK en langage Basic AppleSoft, et par les instructions machine LDA, STA, BIT, CMP, etc.

Adresses de contrôle :

$C080 (-16256) Sélectionne la RAM en lecture et commute la bank 2. Protège la RAM contre l'écriture.
$C081 (-16255) Sélectionne la ROM en lecture et commute la bank 2. Autorise l'écriture sur la RAM si la commutation est effectuée deux fois (ex. : LDA $C081, LDA $C081).
$C082 (-16254) Sélectionne la ROM en lecture et commute la bank 2. Protège la RAM contre l'écriture.
$C083 (-16253) Sélectionne la RAM en lecture et commute la bank 2. Autorise l'écriture sur la RAM si la commutation est effectuée deux fois.
$C088 (-16248) Sélectionne la RAM en lecture et commute la bank 1. Protège la RAM contre l'écriture.
$C089 (-16247) Sélectionne la ROM en lecture et commute la bank 1. Autorise l'écriture sur la RAM si la commutation est effectuée deux fois.
$C08A (-16246) Sélectionne la ROM en lecture et commute la bank 1. Protège contre l'écriture sur la RAM.
$C08B (-16245) Sélectionne la RAM en lecture et commute la bank 1. Autorise l'écriture sur la RAM si la commutation est effectuée deux fois.

Utilisation des commutateurs à partir d'un programme

A partir du Basic AppleSoft, l'adressage de la carte langage est très délicat, et de ce fait déconseillé, car HIMEM restera toujours positionné comme si la mémoire était celle d'une configuration 48 Ko. D'ailleurs, il existe souvent matière à confusion lorsque certains constructeurs avancent des chiffres de capacité mémoire : à savoir si cette mémoire est utile au programmeur ou au système.

Il faut simplement retenir que, dans la majorité des cas, la commutation de la carte langage s'effectue à partir d'un sous-programme machine, et permet ainsi un adressage aisé. Le DOS 3.3, par exemple, peut facilement se loger dans les 16 Ko commutés (tampons exceptés) et HIMEM positionné à l'adresse $B800. Ce savoir-faire reste quand même du domaine privilégié du programmeur averti.

Organisation de la carte langage, MLI, Tampon, Disk-Driver et Reboot

MACHINE LANGUAGE INTERFACE - MLI

Le MLI occupe l'espace $D000-$EFFF de la bank 1 - carte langage.
L'entrée dans le MLI - adresse $D000 - s'effectue par un JMP à partir de la page globale de PRODOS. Le pointeur de la commande MLI (Commandes $40-$D3) est dépilé, puis sauvegardé aux adresses $40-$41 de la page zéro. Parallèlement, l'adresse de retour est définie (MLI JSR+4), puis sauvegardée en $BF9C-$BF9D. L'adresse $BF0F - SYSERR -, vecteur de sauvegarde du code d'erreur, est mise à zéro, puis par une manipulation complexe, le registre X teste, dans une table située en $EF25-$EF44, la présence du numéro de la commande MLI. En cas d'absence d'un numéro de commande, le système abandonne l'ordre reçu en branchant à l'adresse $D0A7 avec le code d'erreur 1 dans l'accumulateur (Bad System Call Number). Si le numéro de la commande MLI est trouvé, PRODOS vérifie le nombre de paramètres qui accompagnent la commande. Le byte représentatif - Parameter Count - figure dans une table répertoire aux adresses $EF45-$EF64. Si le nombre de paramètres déclarés est différent de celui de la table, le système interrompt la commande, et se branche à l'adresse $D0AB avec le code d'erreur 4 dans SYSERR - adresse $BF0F - (Bad System Call Parameter Count).

Remarque : lorsque la commande MLI $82 est utilisée, et que la carte horloge n'est pas en ligne, le système branche à l'adresse $BF06 qui contient le code #$60 - RTS. Par ailleurs, les paramètres de la zone mémoire réservée pour les données date/heure ($BF90-$BF93) seront ignorés.

Si la routine GET TIME ($82) est absente, et si de surcroît un faux paramètre est déclaré, le système recharge dans l'accumulateur le numéro de la commande MLI, puis effectue une comparaison avec la commande REBOOT - numéro $65. Si la comparaison est

positive, la commande REBOOT sera exécutée en faisant un saut à l'adresse $D05D, qui appelle $BF03. En $BF03 se trouve un "JMP $EEDB" - REBOOT. Si la comparaison n'aboutit pas, un test des bits 7 et 6 de la commande MLI est effectué pour éventuellement reconnaître une commande se situant entre les numéros $40-$41, $80,$81 ou $C3-$D3.

Si aucune erreur n'est rencontrée lors de la formulation d'une commande MLI, l'exécution du module se fera par le vecteur $D078 et l'adresse de retour sera sauvegardée sur le sommet de la pile. La routine sera abandonnée en $D09F par un JMP $BFA0.

TABLE DES COMMANDES MLI

- Version 1.0.2

CMDE	*Entrée*	*Fonction*
$40	$D0F3	ALLOC INTERRUPT
$41	$D124	DEALLOC INTERRUPT
$65	$EEDB	REBOOT
	$D100	REBOOT origine
$80	$D0B2	READ BLOCK
$81	$B0B2	WRITE BLOCK
$82	$D060	GET TIME
$C0	$D4F7	CREATE
$C1	$EB50	DESTROY
$C2	$EA0B	RENAME
$C3	$E9D4	SET FILE.INFO
$C4	$E96F	GET FILE.INFO
$C5	$D42A	ON LINE
$C6	$D32A	SET PREFIX
$C7	$D382	GET PREFIX
$C8	$E09B	OPEN
$C9	$E95D	NEWLINE
$CA	$E1F7	READ
$CB	$E47C	WRITE
$CC	$E677	CLOSE
$CD	$E6E8	FLUSH
$CE	$DE1E	SET MARK
$CF	$DE08	GET MARK
$D0	$E807	SET EOF
$D1	$E94B	GET EOF
$D2	$EEA7	SET BUF
$D3	$EE98	GET BUF

IRQ-HANDLER INTERRUPT

Cette routine est logée aux adresses $D13A-$D1D7. A partir de l'adresse $D1CE se trouvent quatre JMP pour effectuer un saut à la routine d'interruption appropriée :

```
D1CE - 6C 80 BF   JMP  ($BF80)
D1D1 - 6C 82 BF   JMP  ($BF82)
D1D4 - 6C 84 BF   JMP  ($BF84)
D1D7 - 6C 86 BF   JMP  ($BF86)
```

La présence d'une routine d'interruption est testée à partir de $D172. Si aucune des adresses réservées de la page globale, $BF80-$BF87, n'est occupée par un pointeur, un saut est effectué à l'adresse $D19A. Le code ERR# 1 est chargé dans l'accumulateur, puis le système branche à SYSTEM DEATH en $D1E6, *via* la page globale, adresse $BF0C.

L'erreur renvoyée sera du type :

INSERT SYSTEM DISK AND RESTART ERR#xxx

où xxx représente le numéro de l'erreur.

SET SYSERR - $D1DA - sera sollicité par l'adresse $BF09 de la page globale et la sortie de la routine se fera normalement par l'adresse $D078.
Avec une carte horloge du genre "Thunderclock", la routine d'interruption se trouve à l'adresse $F142.

TAMPONS DE PRODOS

A partir de	$EF25	541	bytes
A partir de	$F1AC	1 620	bytes
A partir de	$F996	495	bytes
A partir de	$FEBC	66	bytes
A partir de	$FF7F	28	bytes

Certaines adresses en vrac :

Tables

$EF25 - $EF44	Command Bytes : numéros des commandes MLI.
$EF45 - $EF64	Nombre de paramètres pour chaque commande, dans le même ordre que la table précédente.
$EF65 - $EF8C	Adresses d'entrée des commandes : $C0-$D3.
$EF8D - $EFA0	Indices pour la table $EF65-$EF8C.
$EFA1 - $EFA5	Blocks Used +1 ; EOF + $200.
$EFA6 - $EFAD	uHOUSTON!
$EFB0 - $EFB1	MLI version : première et dernière.

$EFB2 - $EFB7	Access : longueur d'une entrée. Nombre d'entrées par Directory-Block.
$EFB8 - $EFBF	FIND FILE. Storage type $0F non utilisé. Premier bloc d'un fichier, ou blocs occupés.
$EFC0 - $EFC7	Bit masque pour la Subdirectory Bit-Map ou la Volume Bit-Map.
$EFC8 - $EFDE	Table de conversion pour GET FILE INFO.
$EFDF - $F004	INSERT SYSTEM DISK AND RESTART.

Tampons

$F000 - $F0FF	Variables du MLI.
$F100 - $F1FF	Préfixes et chemins - Pathname.
$F200 - $F2FF	Volume Control Block - VCB.
$F300 - $F3FF	File Control Block - FCB.
$F400 - $F5FF	Volume Bit-Map - VBM.
$F600 - $F7FF	Directory ou Subdirectory Table.
$FA00 - $FAFF	Transcription des nibbles - Table.

DRIVERS DU Disk II - RWTB

PRODOS comporte plusieurs drivers (commandes) pour le disque dur profile, le disque virtuel (RAM disk) et le Disk-Driver. La routine Disk-Driver, pour le lecteur de disquettes standard du type Disk II, se trouve implantée aux adresses $F800-$F995 et $FB85-$FEBD. Entre ces deux zones se trouve un tampon de 495 bytes utilisé par le système.

Détail des drivers - RWTB - et détail du tampon $F996-$FB84 :

$F800 ENTRY. Unique entrée de la routine. Les registres A,X,Y et le registre d'état PS seront indéfinis. Le numéro du bloc sera sauvegardé aux adresses $46-$47, et testé par rapport à la capacité de stockage du Disk II. Si cette valeur dépasse $0117 (0-279 = 280 blocs), un saut sera effectué à l'adresse $F836.

$F810 Calcul du numéro de la piste en effectuant cinq décalages à gauche, puis une sauvegarde en $FB56 (5 fois décaler à gauche, équivaut à décaler 3 fois à droite, et se traduit par une division par 8). Dans l'accumulateur se trouve le reste de la division, partie intégrante des bits 5-7. Par un décalage savant, le système détermine le bloc auquel sera ajouté ce reste.

Reste de la division :
- Blocs numéros internes à une piste -

```
0  1  2  3  4  5  6  7
-  -  -  -  -  -  -  -
0(2) 4(6) 8(A) C(E) 1(3) 5(7) 9(B) D(F)
```

Remarque : le premier chiffre désigne le premier secteur physique du bloc ; le chiffre entre parenthèses, le deuxième secteur du bloc.

$F822 — Le numéro du secteur calculé précédemment sera sauvegardé et lu par la suite - $F83A. Si une erreur est rencontrée, un saut à l'adresse $F832 sera effectué. Le numéro du secteur sera augmenté de 2, et l'adresse de base - source - de #$100. Un deuxième appel émanera de l'adresse $F83A pour lire la seconde partie du bloc. L'adresse de base - source RWTB $0045 - sera rétablie à sa valeur initiale.

$F832 — Cette entrée est sollicitée lors d'une rencontre d'erreur. ERR# sera chargé dans l'accumulateur, et retour par RTS à la routine DO DISK I/O, adresse $D0DA, avec la retenue positionnée - Carry Flag. Si le code d'erreur est nul - ERR#00 - la retenue sera nulle et l'accumulateur chargé avec la valeur #$00 :

```
F832- AD 58 FB LDA $FB58
F835- 60       RTS
```

$F83A — READ SECTOR. Le numéro du secteur se trouve dans l'accumulateur. Le compteur du recalibrage de la tête de lecture sera positionné à 1, et le numéro du secteur sauvegardé. Le paramètre Unit Number - adresse $0043 - sera utilisé pour déterminer le numéro du slot (bit 7 ==> numéro du slot * 16), puis sauvegardé à l'adresse $003E. Les constantes du moteur pas à pas seront déterminées au fur et à mesure des besoins.

$F874 — CORRECT DRIVE SELECT. Si l'adresse $0042 contient la valeur #$00 - (pas de lecture/ écriture) - un saut sera effectué à l'adresse $F87E. Aucune position de la tête de lecture ne correspond à cette valeur.

$F890 — CORRECT DRIVE ON. Lors d'une commande nulle ($42 = #$00), saut à l'adresse $F8FD - test de WRITE PROTECTED et RTS. Pour une commande WRITE ($42 = #$2), saut à la routine Prenibble qui se situe à l'adresse $FDF0.

$F89A — TRY TRACK. Le compteur d'erreur (Retry Count = 0, recalibré) sera positionné à la valeur #$40.

$F8C8 — SETTRACK II. Le numéro de piste déclaré sera sauvegardé, et le numéro de piste trouvé sera chargé. Le recalibrage de la tête de lecture sera fixé sur la valeur piste trouvée +8.

$F8D7 — ON CORRECT TRACK. Comparaison des valeurs : secteurs déclarés et secteurs trouvés. Si elles diffèrent, saut à $F8A6, sinon, l'exécution se poursuivra. La commande WRITE branche à l'adresse $F8F4.

$F8F7 — Si un saut provient de l'adresse $F8FD, la retenue sera testée : si le bit 7 est à 1, la routine en cours sera interrompue avec ERR#2B comme code d'erreur - WRITE PROTECTED ; dans le cas contraire, l'exécution se poursuivra.

$F8FD — WRITE PROTECTED TEST. Ce test est introduit pour une valeur nulle de la commande ($42 = #$00). La valeur du numéro de slot * 16 sera chargée, et un test WRITE PROTECTED du drive en ligne sera effectué. Le résultat sera introduit dans la retenue - Carry avec le bit 7 à 1 - et saut à $F8F7.

$F90C — MYSEEK. C'est le point d'entrée de la routine de commande pour déterminer la position absolue de la tête de lecture, avec dans l'accumulateur le numéro de la piste déclarée. Celui-ci sera multiplié par 2, puis sauvegardé. Toutes les phases du moteur pas à pas seront annulées - $FB25. L'ancienne position de la tête de lecture, *via* Unit Number, sera déterminée et reconnue comme numéro de la piste actuelle, puis sauvegardée.

$F933 — SEEKABS. L'entrée est sollicitée par MYSEEK, avec des valeurs définies du numéro de la piste actuelle et de la piste déclarée. Ces valeurs correspondent aux numéros des pistes logiques multipliés par 2. Chaque boucle - Loop - interne à la routine SEEKABS introduit un déplacement de la tête de lecture d'une demi-piste.

$F987 — ACTUAL ARM MOVER. Cette entrée détermine l'adressage des phases du moteur pas à pas - Stepp Motor : appel avec CLC, PHASE OFF et SEC, PHASE ON. La commutation de PHASE et le sens du déplacement de la tête de lecture vers la piste déclarée sont préalablement déterminés.

$F98A — Entrée pour déconnecter la phase - Motor-on ou Motor-off - en cours : si le saut provient de l'adresse $F925, mise hors fonction de toutes les phases en cours. La retenue détermine le mode :

```
$C080/82/84/86 PHASE OFF
$C081/83/85/87 PHASE ON
```

Après la commutation, le registre X sera chargé avec le numéro du slot * 16. Retour - RTS - au sous-programme appelant, ou éventuellement à MYSEEK, à la fin de la routine SEEKABS.

Zone des tampons :

$F996 - $F9FF — Read Translation Table I - RDTRANS1.

$FA00 - $FAFF — Read Translation Table II et Write Translation.

$FB00 - $FB55 — Tampon pour les #$56 nibbles par secteur.

$FB56	Numéro de la piste choisie : bloc numéro/8.
$FB57	Numéro du secteur physique.
$FB58	ERR#. Adresse pour la sauvegarde momentanée du code d'erreur : elle est rechargée avant le retour au MLI.
$FB59	Unit Number. Byte représentatif du dernier périphérique utilisé.
$FB5A	Piste actuelle. Piste au-dessus de laquelle se trouve la tête de lecture.
$FB5B - $FB68	Positions de la tête de lecture pour les différents lecteurs en ligne et traités par la routine RWTB - Read Write Tracks Blocks. Les valeurs sont les numéros des pistes logiques, multipliés par 2. Ordre dans la table : S1,D1 ; S1,D2 ; etc.
$FB69	Retry Counter - #$40.
$FB6A	Recal Counter - #$01.
$FB6B	SEEKABS : nombre de déplacements de la tête de lecture du drive. Delay Index. RDADDR : Re-Seek Counter - #$FC - avec un maximum de quatre Re-Seeks. RDADDR : Scratch ==> Even Odd Decoding
$FB6C	Checksum pour RDADDR.
$FB6D - $FB70	Checksum Sector Track Volume :
$FB6E	Numéro du secteur trouvé.
$FB6F	- RDADDR : numéro de la piste trouvée. - MYSEEK : Scratch pour piste déclarée. - Compteur auxiliaire du déplacement de la tête de lecture.
$FB70	- Compteur des phases Motor-on et Mover Delay. - Compteur du slot actuel.
$FB71	SEEKABS : dernière phase active du moteur pas à pas de la tête de lecture.
$FB72	Piste déclarée * 2. - SEEKABS.
$FB73 - $FB7B	PHASE ON. Delay Table : FB73- 01 30 28 24 20 1E 1D 1C 1C
$FB7C - $FB84	PHASE OFF. Delay Table : FB7C- 70 2C 26 22 1F 1E 1D 1C 1C

Suite de la routine Drivers :

$FB85 - $FB97	ARM MOVER DELAY. Motor-on utilise cette entrée, avec le paramètre de la valeur du temps sauvegardé dans l'accumulateur. Comptabilise la durée - Motor-on Delay en $FB70 - et introduit la notion de temps lors de chaque déplacement de la tête de lecture, juste au moment où le moteur est activé. Retour par RTS au sous-programme appelant.
$FB98 -$FBFC	RDADDR. Lecture d'une adresse d'un champ de données - Address Field. La valeur de Re-seek est fixée à #$4 - #$FC - et chargée dans le registre Y. Ensuite, le registre Y est incrémenté de 1, puis le résultat comparé à #$00 : si la valeur est inférieure, saut à l'adresse $FBA5.
$FBA0	Re-seek est à son tour augmenté de 1 : dès que la valeur sera nulle, le retour - RTS - se fera à la routine appelante via $FBFB, et la retenue, Carry, positionnée.
$FBA5	Lecture d'un byte à partir de la disquette: si la valeur est différente de #$D5, saut à $FB9D.
$FBAF	Le premier byte a comme valeur #$D5, lecture du byte suivant. Si la valeur est différente de #$AA, saut à l'adresse $FBAA - teste #$D5.
$FBB8	Le registre Y sera chargé avec la valeur 3, avant la lecture du troisième byte.
$FBBA	Lecture du troisième byte : si la valeur est différente de #$96, saut à l'adresse $FBAA - teste #$D5.
$FBC3	Les vecteurs d'interruption seront déconnectés. Le rétablissement des Statuts INT - entrées - se fera seulement à partir de la routine qui aura appelé RWTB - $D0DA, par exemple.
$FBC4	Checksum - compteur - sera positionné à #$00.
$FBC9	Huit bytes seront lus, décodés puis sauvegardés dans un ordre de grandeur décroissant - 4 bytes retenus comme résultat.
$FBE3	Le dernier byte - Checksum - devra avoir comme valeur #$00, par comparaison avec le compteur en $FBC4 - Checksum Counter -, sinon retour avec la retenue positionnée - Carry Flag.
$FBE6	Lecture des nibbles de synchronisation - champ d'adresses ou Address Field Trailer ; seuls les bytes #$DE/#$AA seront analysés. Lors de la rencontre d'une erreur, le retour se fera avec la retenue positionnée - Carry Flag.

$FBF7	Retour par un RTS à la routine appelante, avec la retenue à #$00.
$FBF9	Sortie lors d'une erreur : la retenue sera positionnée.
$FBFD	READ DATAFIELD & DENIBBLE. L'entrée se fera avec le numéro du slot multiplié par 16 - slot * 16 - contenu dans le registre X. A cette valeur seront incorporées les adresses de base du champ d'adresses.
$FC31	Un maximum de #$20 essais de lecture est fixé - Retry Count. L'adresse du champ de données (D5 AA AD) sera lue. Si plus de #$20 essais de lectures sont nécessaires, le retour se fera par un RTS, avec la retenue positionnée, et le moteur restera actionné.
$FCD3 - $FCD9	SET ARM POSITION. L'entrée de la routine se fera avec le numéro de la piste multiplié par 2, contenu dans l'accumulateur - Position de la tête de lecture.
$FCDA- $FCF0	MOTOR ON ? Un byte sera lu, puis un temps d'attente de 23 microsecondes observé ; ensuite une comparaison sera effectuée par "CMP $C8CC,X". Si cette valeur change entre-temps, la PHASE du moteur sera considérée comme "on". Retour avec le registre d'état PS positionné. Dans le cas contraire, le code ERR#28 : NO DEVICE CONNECTED sera chargé et 256 essais renouvelés. Si, après les 256 essais, le byte lu n'a toujours pas changé de valeur, le retour se fera à la routine appelante. La retenue sera indéterminée, et l'erreur affichée dans le cas de l'appel par la routine implantée en $F88B.
$FD00	WRITE DATAFIELD. Le branchement à cette entrée se fera après l'exécution de la routine "Prenibble" - $FD0F - qui calcule au préalable les adresses de base (première/deuxième moitié tampon source), et le numéro codé du slot, et qui détermine les vecteurs de la page zéro - bytes $003B-$003D et $003F.
$FD01	Test de WRITE PROTECTED : dans l'affirmative, saut à l'adresse $FDDF.
$FD0C	Le dernier byte - $FB00 - sera sauvegardé à l'adresse $003A de la page zéro.
$FDE6 - $FDEF	WRITE SINGLE BYTE. Cette routine comporte plusieurs entrées sollicitées suivant le temps d'accès - Timing. Le byte sera sauvegardé dans l'accumulateur, et le retour se fera par un RTS à la routine appelante.
$FDF0 - $FE16	PRENIBBLE SET + SET ADDRESS FOR WRITE. Les trois adresses de base seront déterminées - premier/deuxième/troisième tiers - et incorporées dans le byte du code. Les adresses seront fixées en fonction de l'index #$AA - Prenibble fixe cette valeur à l'intérieur de chaque tiers d'adresse.

$FE9B-$FEBD	SAME SLOT ? L'appel de la routine se fera avec le nouveau numéro - Unit Number - dans l'accumulateur. Si ce nouveau numéro de slot est identique à celui qui était utilisé précédemment, retour par un RTS, avec le bit 7 - drive - du paramètre "Unit Number" masqué. Si le numéro du slot est nul - possibilité du RAM-Disk installé - le retour se fera par un RTS. Dans les autres cas, un test de Motor-on sera exécuté avec l'ancienne valeur de "Unit Number" - $FCDC : dans l'affirmative, un temps d'attente sera introduit - $FB85 - et le test renouvelé autant de fois qu'il sera nécessaire pour mettre le Motor-off. La valeur du compteur, sauvegardée à l'adresse $FB70, sera alors indéterminée et dépendra essentiellement du nombre de tests effectués.
$FEBE - $FEFF	Adresses inutilisées.
$FF00 - $FF69	Interface du RAM-Disk RWTB.
$FF6A - $FF9A	Variables du RAM-Disk RWTB.
$FF9B - $FFFF	IRQ/RESET Handler. (Voir le chapitre 7 pour plus de détails.)

UTILISATION DE LA PAGE ZERO

$003A	WRITE. Copie de la routine $FB00, du dernier nibble.
$003B	WRITE #$00. Si le tampon se trouve à la limite d'une page mémoire - Page Boundary. Sinon, byte $00 EOR $01 de la prochaine zone de sauvegarde.
$003C	WRITE. Byte $01 de la prochaine zone de sauvegarde, si le tampon ne débute pas à la même page mémoire.
$003D	WRITE. Dernier byte du tampon pour le Checksum.
$003E	Numéro du slot multiplié par 16.
$003F	WRITE. Index du dernier byte du tampon.
$0042	Commande disque : - 00 moteur activé - 01 lecture - 02 écriture.
$0043	Unit Number. Paramètre de la configuration : Bits : 76543210 DSSSxxxx (D= drive; S = slot ; x = Libre)
$0044 - $0045	Adresse de base destination : LByte, HByte.
$0046 - $0047	Numéro des blocs - de #$0000 à #$0117 - 280 blocs au total.

Processus de la routine Drivers

D'abord, le système vérifie la capacité de sauvegarde du drive et compare le nombre de blocs : ##000 à #$117 = 280 blocs. Un compteur totalise le nombre de blocs à partir de la valeur 0, pour un maximum de 279 (- #$117). Ensuite, une table (ProDOS Skewing

Table) effectue la transcription de l'image des blocs en équivalent pistes/secteurs (blocs = format logique ; pistes et secteurs = format physique). Il est à noter que deux blocs consécutifs se trouveront toujours sur la même piste, cela pour une raison évidente de recherche piste/ secteur. Les valeurs transitoires pistes/ secteurs seront sauvegardées respectivement en $FB56 et $FB57. La routine lecture/ écriture d'un secteur - READ A SECTOR - débute en $F83A : elle sera sollicitée par deux fois, deux secteurs déterminant un bloc. La routine Disk-Driver se laisse facilement modifier pour une éventuelle augmentation du nombre de blocs, à condition de posséder un drive ayant les caractéristiques techniques et de formatage adéquates (supérieur à 35 pistes).

Modifications pour un formatage différent de 35 pistes :

Routine initiale :

```
F809 -  CA      DEX         Décrémente le numéro des blocs.
F80A -  D0 2A   BNE $F836   Fin des 280 blocs.
```

Après modification :

```
F809 -  CLC                 Annule la retenue.
F80A -  90 04   BCC $F80E   Saute la routine de contrôle des blocs.
```

Par ailleurs, le formatage d'une disquette au-delà des 35 pistes standard nécessite un programme spécial, car l'utilitaire FILER ne permet pas cette extension. Il existe un certain nombre de revues spécialisées, qui traitent très bien ce sujet (*POM'S* par exemple).

CHAPITRE 6

MACHINE LANGUAGE INTERFACE LES COMMANDES

LES ROUTINES MLI

Généralités

Le MLI est un ensemble de routines écrites en assembleur - langage machine. Il regroupe les modules de fonctionnement du système d'exploitation PRODOS et se loge dans la carte langage d'un Apple II. La version 1.0.2 est constituée de 26 routines, dont chacune comporte un point d'entrée standard pouvant être appelé par un programme externe.

Que doit pouvoir faire un système d'exploitation? C'est la première question que l'on peut se poser. ProDOS peut se définir comme un DOS ayant atteint sa maturité, grâce à de multiples capacités et grâce au BASIC.SYSTEM qui est son interface. C'est la première fois qu'un système d'exploitation n'a rien à envier à ses homologues, malgré certaines petites imperfections qui subsistent encore. De nouveaux concepts sont introduits, et le langage diffère de celui du DOS 3.3.

Appel du MLI - Version 1.0.2

Le point d'entrée des MLI pour la version 1.0.2 est $D000. La société Apple Computer s'est donnée les moyens d'un changement éventuel du point d'entrée MLI, en utilisant la page globale - $BF00 - comme vecteur intermédiaire. L'entrée se fait par un JMP, saut inconditionnel, et non par l'appel JSR comme un sous-programme. Les registres X et Y sont sauvegardés dans la page globale, et l'adresse de retour du programme appelant est dépilée, puis sauvegardée dans la page zéro - adresses $0040-$0041. Ensuite, le pointeur de retour est augmenté de 4, puis stocké dans la page globale comme adresse de retour du MLI (MLIRTS). SYSERR à l'adresse $BF0F aura comme valeur #$00.

A partir de l'adresse de retour du MLI appelant, un byte de commande sera déterminé par l'instruction AND #$1F effectuée à l'adresse $D027. Le résultat obtenu sera utilisé comme index de recherche par le registre X, avec une valeur comprise entre #$00 et #$1F inclus (00 et 31 inclus). Elle servira d'index pour comparer le numéro de la

commande MLI appelante avec les numéros des commandes de PRODOS, stockés dans une table qui débute à l'adresse $EF25. La table des commandes contient tous les numéros des commandes MLI : si la comparaison n'aboutit pas, un code d'erreur ERR# = #$01 sera transmis au système, et le texte BAD SYSTEM CALL NUMBER sera affiché.

Une deuxième table, parcourue par le même index, contient le nombre de paramètres – Parameter Count – du numéro du MLI trouvé auparavant dans la table des commandes. Une comparaison est effectuée : si celle-ci n'aboutit pas, le code d'erreur ERR# = #$04 sera transmis et le message d'erreur BAD SYSTEM CALL PARAMETER COUNT affiché.
A partir de là, la première commande sera recherchée, analysée, puis interprétée pour être transmise au système.

Table des commandes :

```
EF25 -  D3 00 00 00 40 41 00 00 80 81 82 65 C0
        C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD
        CE CF 00 D0 D1 D2
```

Table des paramètres :

```
EF45 -  02 FF FF FF 02 01 FF FF 03 03 00 04 07
        01 02 07 0A 02 01 01 03 03 04 04 01 01
        02 02 FF 02 02 02
```

Version 1.1.1 :

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 son point d'entrée à l'adresse $DE00. 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

Appel du MLI en assembleur - WRITE BLOCK Routine :

```
APPEL MLI     JSR  MLI       $BF00.
PARAMETRE     DA   PARMS     LByte-HByte.
ERREUR1       BNE  ERROR     Accumulateur avec le code d'erreur.

FIN1          RTS            Retour sans erreur.
ERREUR2       JSR  $FDDA     Affiche le code d'erreur.

FIN2          RTS            Retour après l'erreur.

PARMS         HEX  03        Nombre de paramètres de la table.

SLOT          HEX  60        Slot 6, drive 1.
BUFFER        DA   $2000     $2000-$21FF.
BLOCS         HEX  0100      Bloc $0001.
```


Commentaire de l'appel MLI en assembleur :

1. L'appel d'une commande MLI se fait par un JSR MLI-ENTRY (MLI-CALL) par la page globale, à l'adresse $BF00. Ce type d'appel est préféré à un JMP - saut inconditionnel - car PRODOS sauvegarde l'adresse de retour sur le sommet de la pile, et mémorise le numéro de la commande avec ses paramètres. Les numéros de commandes et paramètres sont gérés par des tables dont PRODOS tient le contrôle : table des commandes en $EF25, table des paramètres en $EF45.

2. A la suite de la commande MLI devra se trouver le numéro MLI - Command Number ($81 dans notre exemple) pour désigner WRITE BLOCK.

3. Suit l'adresse de la table des paramètres dans le format désormais familier : Low Byte, High Byte - LByte-HByte.

4. Le premier RTS à l'entrée FIN1 détermine l'adresse de retour de la commande MLI, dont la position mémoire devra être le sixième byte après la commande JSR MLI. Les 3 bytes après JSR MLI seront ignorés par PRODOS, et le pointeur de pile incrémenté dans le même ordre de grandeur. Les 3 bytes regroupent le numéro et le nombre de paramètres de la commande.

5. Au retour de la routine MLI, le contenu des registres X et Y reste inchangé, et leurs valeurs intermédiaires seront sauvegardées dans la page globale de PRODOS. Si aucune erreur n'est rencontrée, l'accumulateur contient la valeur #$00, et la retenue annulée (Carry Flag). A ce niveau du déroulement, l'appel MLI pourra être abandonné par l'instruction BEQ ou BCC.
   Si une erreur survient pendant la transmission d'une commande MLI, l'accumulateur sera chargé avec le code de l'erreur, et la retenue positionnée (Carry-Flag). Les instructions machine BCS et BNE permettent une gestion de l'erreur, en transmettant celle-ci vers un sous-programme de traitement de l'erreur, Error Handling routine. Dans notre exemple, l'instruction BNE sera sollicitée.

6. La liste des paramètres pourra se trouver à une adresse fixée par le programmeur : elle devra se loger au bas des 48 Ko de la mémoire, ou à la suite du sous-programme d'appel du MLI. Chaque appel MLI devra comporter une liste de paramètres, avec une structure semblable à celle de notre exemple - WRITE BLOCK.

Table des commandes de ProDOS

Formulation des paramètres :

Les paramètres peuvent être de deux types différents : d'abord, des données qui seront transmises par la suite à la routine MLI et préalablement initialisées ; en second lieu, une zone de paramètres, dont PRODOS établira lui-même les valeurs. Ces derniers ne donnent pas lieu à une déclaration préalable.

Structure d'une table de commande PRODOS :

Une table des paramètres peut contenir au maximum 17 bytes. Comme cet espace est limité, la mise en place de pointeurs autorise une extension des possiblités. C'est ainsi que certaines adresses extérieures à la table peuvent être pointées : un ou plusieurs tampons, des zones de données, etc.
Le premier byte désigne toujours le nombre de paramètres contenus dans la table de la commande - Parameter Count. Il faut écarter tout rapprochement avec le nombre de paramètres (Byte-Parameter) de la table : par exemple un pointeur sur deux octets, ou encore une date, etc.
La table d'une commande peut contenir un maximum de 10 paramètres. On pourrait citer comme exemple la commande CREATE ($C0), qui comporte un maximum de 7 paramètres, mais qui nécessite 12 bytes pour les loger (Parameter Count compris).

LES COMMANDES MLI - GROUPES

Les différents groupes MLI :

Les commandes MLI se divisent en trois groupes ou familles : les commandes du Directory, les commandes pour la gestion des fichiers, et les commandes du système.

Les commandes du Directory

Elles constituent les routines de création, d'opération de mise en forme, d'effacement ou de moyens pour renommer un fichier. Ces commandes permettent de modifier la structure et le statut d'un fichier. 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 — Permet de créer un fichier Directory ou programme, vide de donnée. Si CREATE désigne un fichier Directory, son nom sera sauvegardé dans le Volume-Directory qui lui a donné naissance. Un Volume-Subdirectory sera créé avec un bloc alloué. Si, par la suite, plus de 12 noms doivent être sauvegardés dans le Volume-Subdirectory, le moment venu, le système allouera un bloc supplémentaire à la table des matières, et celui-ci sera chaîné avec le bloc précédent. Si CREATE désigne un fichier programme, son nom sera sauvegardé dans le Volume-Directory lui ayant donné naissance. Un bloc vide lui sera automatiquement alloué.

DESTROY — Purge le fichier nommé du Directory, et met à jour la table d'occupation des blocs de la disquette. Le fichier devra être déverrouillé au préalable - instruction UNLOCK. Tout fichier effacé fait perdre le bénéfice de ses données.

Le nom du Volume-Directory pourra être effacé lors du formatage de la disquette.
Le nom d'un fichier Subdirectory pourra être effacé lorsque la table des matières du Subdirectory sera vide, et le nom déverrouillé.

RENAME — Change le nom d'un fichier. Le nom à modifier devra figurer dans la table des matières de la disquette. Le changement du nom d'un fichier étant une opération d'écriture, il ne sera pas possible de l'effectuer sur un fichier verrouillé.

SET.FILE.INFO — Sauvegarde sur la disquette les données concernant la date et l'heure, à condition d'avoir en place une interface du type Thunderclock.

GET.FILE.INFO — Effectue, à partir d'une disquette, la lecture des données enregistrées par la commande SET.FILE.INFO, pour les placer dans un tampon mémoire. Ces paramètres se trouvent stockés sur la disquette, et sont affichés par les commandes CAT et CATALOG.

ON.LINE — Retourne les paramètres slots, drives et noms des Volumes en ligne. Ces paramètres sont stockés dans une table de la page globale de Basic, qui débute à l'adresse $BE58 - voir le chapitre 5 pour plus de détails.

SET.PREFIX — Fixe le nouveau chemin (Pathname) déclaré par le préfixe.
Le préfixe devra désigner un nom de Volume existant dans le Directory ou dans le Subdirectory.

GET.PREFIX — Retourne la valeur du dernier préfixe déclaré.

Les commandes pour la gestion des 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 1 024 bytes, et n'autorise pas plus de 8 ouvertures de fichiers.
Ces commandes regroupent des routines appropriées au traitement des fichiers en général.

Syntaxe des commandes :

OPEN — Prépare le système et alloue un tampon mémoire de 1 024 bytes comme espace de sauvegarde, pour le transfert des données d'une disquette vers la mémoire, ou vice versa. Appelle un sous-programme qui implante une table en mémoire - File Control Block $F300-$F3FF - qui a comme rôle de gérer un certain nombre de renseignements

nécessaires à la gestion du fichier : File Reference Number, Unit Number, Newline Character, etc. (Le chapitre 7 donne plus de détails sur la table FCB.)
Le module OPEN n'est pas exclusivement réservé aux fichiers texte.

NEWLINE — Cette commande effectue la recherche d'un caractère particulier dans un fichier de données : par défaut, sa valeur équivaut à un retour-chariot (code #$0D). C'est la marque de fin d'un enregistrement, souvent comparée à une ligne de caractères ASCII, dont la fin peut correspondre à celle du fichier. Le code #$0D détermine la fin d'un champ de l'enregistrement à l'intérieur du fichier.

READ — Effectue le transfert des données d'une disquette vers une adresse mémoire déterminée par le système, et met à jour la position de fin courante du fichier (MARK).
En principe, le caractère est lu par l'appel de la routine NEWLINE.

WRITE — Sauvegarde des données en effectuant un transfert, de la mémoire tampon spécifiée, vers une disquette par exemple. Met à jour la position courante, MARK, et fixe le pointeur de fin, EOF, du fichier.

CLOSE — Force le vidage du tampon mémoire alloué vers la disquette, et libère de la place en mémoire. Met à jour le Directory de la disquette, si nécessaire. CLOSE, déclaré seul, ferme tous les fichiers ouverts à cet instant.

FLUSH — Transfert des données du tampon mémoire vers la disquette, sans fermer le fichier nommé. Si FLUSH est déclaré seul, tous les tampons des fichiers ouverts seront sauvegardés sur la disquette.

SET.MARK — Change la position courante dans le fichier (MARK). La position courante désigne l'emplacement de valeur absolue à l'intérieur du fichier et pointe le caractère devant être lu ou écrit.

GET.MARK — Retourne la position actuelle à partir d'un fichier. C'est la position absolue du prochain caractère devant être lu ou écrit.

SET.EOF — Change la grandeur logique du fichier, en mettant à jour un pointeur de fin, EOF, qui n'est autre que la longueur en bytes des caractères.

GET.EOF — Retourne la valeur de la taille logique d'un fichier. C'est la longueur en bytes du fichier.

SET.BUF — Affecte un nouveau pointeur à l'emplacement du tampon mémoire, lors de l'ouverture d'un fichier - Input/Output Buffer.

GET.BUF — Retourne la valeur de l'adresse courante du tampon actuel lors de l'ouverture d'un fichier.

Les commandes du système

Elles regroupent un certain nombre de routines utiles à la gestion date/heure, ainsi que les commandes d'écriture/lecture d'un bloc. Lorsqu'une carte horloge se trouve en ligne, c'est une commande système qui assure la mise en place et le retrait du vecteur d'interruption.

Syntaxe des commandes :

GET.TIME — La carte horloge est nécessaire pour la gestion date et heure. Lors du "boot" de la disquette, une routine est implantée à l'adresse $F142, par l'intermédiaire du Relocator. En même temps, ProDOS consulte tous les slots en ligne à la recherche d'une interface du type Thunderclock. Dès que la carte horloge est détectée, un pointeur est mis à l'emplacement du RTS, à l'adresse $BF06 de la page globale - JMP $F142. Le programme SET TIME, à l'adresse $F142, est appelé, puis exécuté. Il lit la date et l'heure dans la carte horloge et mémorise les paramètres trouvés dans la page globale de ProDOS, aux adresses $BF90-$BF93. En fin d'exécution, le RTS à l'adresse $F1AB effectue un retour dans le MLI, via $D078 - MLI-EXIT - uniquement si SET TIME a été sollicité, ou au Command-Handler avec les commandes $C0-$D3 nécessitant la fonction temps (CLOSE, CREATE, etc.).

ALLOC.INTERRUPT — L'adresse d'exécution sera mise en place dans la table des pointeurs des interruptions, à la page globale, $BF80-$BF87. Reference Number sera retourné avec une valeur de 1 à 4, et utilisé par la commande DEALLOC.INTERRUPT.

DEALLOC.INTERRUPT — Dans la table des pointeurs des interruptions de la page globale, l'adresse d'interruption et son numéro de référence - Reference Number - sont recherchés, puis purgés par la routine IRQ-Handler qui débute à l'adresse $D13A.

READ.BLOCK — Lecture de données à partir du bloc - 512 bytes - spécifié d'une disquette, vers l'emplacement du tampon mémoire spécifié. Les données sont stockées dans un tampon destination. La lecture est indépendante du type de fichier.

Avant l'appel de la routine RWTB, la System Bit-Map est testée en vue de déterminer les pages libres : si l'adresse du tampon mémoire alloué se trouve sur une page limite, deux zones mémoire seront testées, sinon ce seront 3 zones mémoire. Si l'adresse de destination est supérieure à $BFFF, le test n'aura aucun sens.

WRITE.BLOCK — Sauvegarde des données dans un bloc de disquette, à partir d'un tampon mémoire déterminé à l'avance. Cette zone s'appelle tampon source. Les mêmes recommandations que pour la commande READ.BLOCK restent vala- bles dans ce cas.

REBOOT — Cette commande est recopiée à partir de la carte langage bank 2 et exécutée. Elle a été introduite ultérieurement dans le MLI. Elle ne possède pas de paramètre ; néanmoins, le Parameter Count indique 4. Permet de "rebooter" un autre programme système par exemple.

LES PARAMETRES DES COMMANDES MLI

Généralités

Chaque commande MLI se compose, d'une part, d'un index, Parameter Count, et, d'autre part, d'une liste de paramètres, en nombre variable, se rapportant à cette commande. Les noms et termes techniques seront donnés en version d'origine, avec leur transcription française. L'interprétation de chaque paramètre d'une commande MLI sera développée et largement commentée dans ce paragraphe. (Le chapitre 6 détaillera chaque commande et renseignera sur le tableau des paramètres.)

Tableau récapitulatif des paramètres MLI

1. **PARAMETER COUNT** - 1 byte.
   Désigne le nombre de paramètres attribués à la commande MLI.

2. **PATHNAME POINTER** - 2 bytes (LByte,HByte).
   Pointeur du chemin ou nom d'un fichier.

3. **NAME**
   Name Length : 1 byte pour la longueur du nom d'un fichier.
   Name : 1-64 caractères pour le chemin - Pathname.

4. **DATA BUFFER POINTER** - 2 bytes (LByte,HByte).
   Pointeur du tampon des données.

5. **DATA BUFFER** - 1 byte minimum.
   Tampon des données.

6. **I/O BUFFER POINTER** - 2 bytes (LByte,HByte).
   Pointeur du tampon d'entrée/sortie.

7. **I/O BUFFER** - 1 024 bytes maximum.
   Tampon des données d'un fichier.

8. **CREATE DATE** - 2 bytes.
   Affecte la date à la création du fichier.

9. **CREATE TIME** - 2 bytes.
   Affecte l'heure à la création du fichier.

10. **MODIFIED DATE** - 2 bytes.
    Modification de la date pour différencier 2 fichiers.

11. **MODIFIED TIME** - 2 bytes.
    Modification de l'heure pour différencier 2 fichiers.

12. **ACCESS** - 1 byte.
    Byte d'accès au fichier.

13. **FILE TYPE** - 1 byte.
    Type du fichier.

14. **STORAGE TYPE** - 1 byte.
    Un nibble qui détermine le genre du fichier.

15. **AUXILIARY TYPE** - 2 bytes.
    Fichier du type auxiliaire.

16. **BLOCKS USED** - 2 bytes (LByte,HByte).
    Blocs occupés.

17. **NULL FIELD** - 3 bytes (avec SET.FILE.INFO).
    3 bytes nuls dans la commande.

18. **EOF** - 3 bytes (LByte,MByte,HByte).
    Marque de la fin logique d'un fichier - Longueur.

19. **POSITION** - 3 bytes (LByte,MByte,HByte).
    Pointe la position actuelle dans un fichier - MARK.

20. **NEWLINE CHARACTER** - 1 byte.
    Nouvelle donnée dans un fichier.

21. **ENABLE MASK** - 1 byte.
    Masque pour déterminer un caractère - AND.

22. **FILE REFERENCE NUMBER** - 1 byte.
Paramètre de référence, retourné, d'un fichier ouvert.

23. **REQUEST COUNT** - 2 bytes (LByte,HByte).
Nombre de bytes à transférer.

24. **TRANSFER COUNT** - 2 bytes (LByte,HByte).
Nombre de bytes transférés réellement.

25. **UNIT NUMBER** - 1 byte.
Image binaire du byte qui désigne le périphérique en ligne.

26. **BLOCKS NUMBER** - 2 bytes (LByte,HByte).
Numéro des blocs.

27. **INTERRUPT CODE POINTER** - 2 bytes (LByte,HByte).
Adresse de la routine d'interruption.

28. **INTERRUPT REFERENCE NUMBER** - 1 byte.
Numéro d'ordre de l'interruption.

Détail des paramètres MLI

• PARAMETER COUNT

C'est un chiffre caractéristique : il désigne le contenu d'une table des commandes MLI, par un nombre précis. Le nombre des paramètres peut varier entre un minimum 0 et un maximum 10, sans pour autant dépasser 17 bytes. La commande GET.TIME ne possède pas de paramètre. Le paramètre est codé sur un byte.

• PATHNAME POINTER

Pointeur de Name désigne le chemin partiel ou complet avec l'utilisation des commandes MLI : CREATE, DESTROY, RENAME SET.FILE.INFO, GET.FILE.INFO et OPEN. Si le nom déclaré débute par un slash ou /, le chemin complet sera traité, sinon, ce sera un chemin partiel, avec comme nom celui d'un fichier, par exemple. L'instruction PREFIX permet de définir le préfixe. Lorsque la commande PREFIX précède un chemin partiel, ProDOS place le préfixe au début pour former le chemin complet. Un chemin complet ou partiel ou un préfixe peuvent contenir un maximum de 64 caractères. Préfixe et chemin peuvent s'ajouter au moment de la recherche d'un fichier, ce qui permet de composer un nouveau chemin d'une longueur de 128 caractères.

Deux bytes sont nécessaires pour caractériser le pointeur : LByte, HByte.

Représentation des 2 bytes :

```
Bits :   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
         0 0 0 1 0 1 1 1 0 0 1 0 0 0 0 0
```

LByte, HByte
17,20
Pointeur = $2017.

• NAME

$00 (00) : position relative dans l'entrée de la table des matières d'un nom de fichier.
- Uniquement pour Name Length.
NAME est l'association du paramètre Name Length et du nom ou chemin d'un fichier.
NAME est utilisé avec le paramètre Name Pointer.
Name Length, codé sur un nibble - 4 bits de poids faible -, désigne la longueur du nom d'un fichier et peut comporter de 1 à 64 caractères ASCII.

La commande RENAME utilise deux pointeurs Name Pointer, ainsi que 2 paramètres Name: un pour l'ancien nom, et l'autre pour le nouveau. Lorsque le chemin débute par un nom de Volume, le slash devra le précéder. Les caractères ASCII qui forment le nom du chemin ont le bit 7 à 0.

Représentation des bytes :

Byte 00 : Name Length - Longueur du chemin.
Bytes 01-64 : Name - Caractères définissant le chemin ou nom du fichier.

Exemple :

09 2F 56 32 2F 4E 4F 4D 2F

09 = 9 bytes de longueur.
2F 56 32 2F 4E 4F 4D 2F = /V2/NOM/.

• DATA BUFFER POINTER

Pointeur de Data Buffer - tampon des données : il est utilisé par les commandes MLI ON.LINE, GET.PREFIX, READ, WRITE, SET.BUF, GET.BUF, READ.BLOCK et WRITE.BLOCK. Il est codé sur 2 bytes : LByte, HByte.

Représentation des 2 bytes :

```
Bits :   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 = LByte
                                           H = HByte
```

• DATA BUFFER

Tampon des données ou programmes : il occupe la partie basse des 48 Ko de la configuration. Le début d'un tampon n'a pas forcément son origine à la limite d'une page mémoire (Page Boundary) et peut tout aussi bien chevaucher deux pages.
Sa taille dépend de la commande déclarée : elle aura une longueur de 16 bytes à condition que le périphérique soit précisé. Si aucune valeur d'unité (slot, drive) n'est déclarée, le paramètre Unit Number sera mis à $00 et les slots en ligne mémorisés ; Data Buffer nécessite alors un tampon de 256 bytes.
Data Buffer Pointer pointe les données stockées dans Data Buffer.
Avec l'utilisation de la commande WRITE, deux cas de figures peuvent se présenter :

1- le système Basic est implanté en mémoire centrale, alors l'espace $0800-$95FF sera disponible ;
2- le système Basic est absent, alors l'espace mémoire $0800-$BFFF sera disponible.

Implantation du nom de Volume dans le tampon :

Deux cas peuvent se présenter : le nom d'un Volume est placé normalement dans le tampon, ou une erreur est rencontrée pendant l'exécution de la commande. Lorsqu'un nom de Volume est copié dans le tampon, sa structure sera la suivante :

```
Position :00 01 02 03 04 05 06 07 08 09 0A 0B 0E 0F
ASCII    :UL N  O  M  V  O  L  U  M  E  00 00 00 00
Erreur   :U0 E  00 00 00 00 00 00 00 00 00 00 00 00
```

Position	Désigne la position, évaluée en base 16, du paramètre et des caractères ASCII placés dans le tampon.
ASCII	Désigne la copie des caractères, en code ASCII, dans le tampon.
Erreur	Représente la valeur des bytes dans le tampon, lors de la rencontre d'une erreur.
Byte UL	Les 4 bits de poids fort représentent le paramètre Unit-Number et les 4 bits de poids faible la longueur du nom d'un Volume.
Byte U0	Les 4 bits de poids fort représentent le paramètre Unit Number et les 4 bits de poids faible la longueur du nom d'un Volume (longueur = 0).
NOMVOLUME	Représente le nom du Volume cité comme exemple, avec 9 bytes de longueur.
E	Représente le code de l'erreur rencontrée.

Représentation des bytes dans un tampon :

Byte $00 : ce byte représente la combinaison binaire du paramètre Unit Number et de la longueur du nom de Volume de la disquette. Les 4 bits de poids fort représentent le slot et le drive, tandis que les 4 bits de poids faible représentent la longueur.

Représentation du byte $00 :

```
Bits :  7 6 5 4 3 2 1 0
        D S S S L L L L     D = drive
                            S = slot
                            L = longueur
```

Exemple :

```
Bits :  7 6 5 4 3 2 1 0
        0 1 1 0 1 0 1 0     = $6A
                            = drive 1, slot 6,
                              longueur 10 bytes
```

Byte $01 : il représente normalement le premier caractère ASCII du nom d'un Volume. Si une erreur est détectée pendant l'exécution de la commande, le byte $00 sera modifié : la longueur sera mise à 0 tout en gardant intacts les 4 bits représentatifs de Unit Number. Le code d'erreur E (E = Erreur) sera copié à la position du byte $01.

Exemple :

```
Bits :  7 6 5 4 3 2 1 0
        D S S S L L L L
        0 1 1 0 0 0 0 0     = $60
                            = drive 1, slot 6,
                              longueur 0.
```

Bytes $01 à $0F : en temps normal, c'est le nom du Volume qui est copié à cet emplacement, mais sans le slash.
Lorsque la dernière sauvegarde sera effectuée dans le tampon, les bytes $00 et $01 seront mis à zéro - 00 - et interprétés par ProDOS comme marque de fin.

• I/O BUFFER POINTER

Utilisé par les commandes OPEN, SET.BUFF et GET.BUFF, ainsi qu'avec toutes les déclarations implicites se rapportant à OPEN - NEWLINE, READ, etc.
Deux bytes sont nécessaires pour coder le pointeur du tampon d'entrée/sortie : LByte, HByte.

Représentation des 2 bytes :

```
Bits :  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 = LByte
                                            H = HByte
```

• I/O BUFFER

I/O Buffer Pointer pointe le tampon I/O Buffer : il nécessite de débuter à la limite d'une page mémoire et se réserve à chaque fois 1 024 bytes - $0400 bytes. Ces pages ne doivent en aucun cas coïncider avec la System Bit-Map. Lorsque le système Basic est actif, le premier tampon d'entrée/ sortie - par défaut - se trouve dans la zone $9600-$99FF. Un tampon supplémentaire sera mis en place avec chaque nouvelle commande OPEN : c'est ainsi que 3 nouveaux tampons seront alloués avec 3 fichiers ouverts.
La routine READ BLOCK n'utilise pas ces tampons d'entrées/ sorties.
Name, Data Buffer et I/O Buffer ne figurent pas dans la table des paramètres d'une commande MLI et sont gérés par des pointeurs internes au système ProDOS.
Un byte est nécessaire par caractère ou adresse mémoire occupée par une donnée et un I/O Buffer se réserve 1 024 octets à sa déclaration, par OPEN, même s'il n'est pas exploité au maximum.

• CREATE DATE

$18-$19 (24-25) : position relative dans l'entrée de la table des matières d'un nom de fichier.
Sauvegarde, sur une disquette, la date du jour, au moment de la création d'un fichier avec la commande CREATE. CATALOG affiche, sous la rubrique CREATED, la date de création du fichier ; tandis que la commande CAT l'ignore.
Deux bytes sont réservés dans chaque entrée table des matières d'un nom de fichier.

Représentation des 2 bytes :

```
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

7 bits pour l'année.
4 bits pour le mois.
5 bits pour le jour.

• CREATE TIME

$1A-$1B (26-27) : position relative dans l'entrée de la table des matières d'un nom de fichier.
Sauvegarde, sur une disquette, l'heure de la création d'un fichier avec la commande CREATE. CATALOG affiche, sous la rubrique CREATED, l'heure de création du fichier ; tandis que la commande CAT l'ignore.
Les paramètres date et heure sont utilisés par les commandes CREATE et GET.FILE.INFO.

Représentation des 2 bytes :

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

H = Heure
M = Minute

Les bits D-Fv sont inutilisés.
Les bits 8-C sont réservés pour les heures.
Les bits 6-7 sont inutilisés.
Les bits 0-5 sont réservés pour les minutes.

• MODIFIED DATE

$21-$22 (33-34) : position relative dans l'entrée de la table des matières d'un nom de fichier.
Cette commande sauvegarde sur une disquette la date actuelle en même temps que l'enregistrement des données d'un fichier, et permet de différencier deux versions différentes. CAT et CATALOG affichent, sous la rubrique MODIFIED, la date de la dernière modification intervenue sur un fichier.
Deux bytes sont nécessaires pour coder la date. La représentation des bits est identique à Create Date.

• MODIFIED TIME

$23-$24 (35-36) : position relative dans l'entrée de la table des matières d'un nom de fichier.
Cette commande sauvegarde sur une disquette l'heure courante en même temps que l'enregistrement des données d'un fichier, et permet de différencier deux versions. CAT et CATALOG affichent, sous la rubrique MODIFIED, l'heure de la dernière sauvegarde d'un fichier.

Deux bytes sont nécessaires pour coder l'heure.

Remarque : si vous possédez la carte horloge, les mises à jour seront effectuées automatiquement. Dans le cas contraire, la date et l'heure ne seront actuelles que si vous avez pris la précaution de mettre à jour le vecteur approprié de la page globale de ProDOS : adresses $BF90-$BF91 pour la date, et $BF92-$BF93 pour l'heure.

Affichage de la commande CATALOG sans carte horloge :

```
MODIFIED          CREATED
<NO DATE> 0:00 <NO DATE>
```

Il est possible de gérer les vecteurs date/heure par l'utilisation d'un programme machine. (Le listing détaillé et commenté d'un tel procédé est fourni en annexe 10.)

• ACCESS

$1E (30) : position relative dans l'entrée de la table des matières d'un nom de fichier.
Paramètre codé sur un byte; il est utilisé par les commandes CREATE, SET.FILE.INFO et GET.FILE.INFO. La représentation binaire du byte détermine l'accès au fichier.

Représentation du byte :

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

D Destroy-bit : définit si DELETE peut effacer le fichier.
N Name : RENAME permet de renommer le fichier.
B Backup : bit significatif stocké à la page globale - adresse $BF95 -, utilisé par des routines MLI pour autoriser la copie des données d'un fichier, bit 5 à 1. Au retour de la routine, le bit 5 sera positionné à 0.

Bits 4-2 inutilisés.
W Write : bit d'autorisation d'écriture.
R Read : bit d'autorisation de lecture.

Exemple :

```
Bits : 7 6 5 4 3 2 1 0
       D E B 0 0 0 W R
       1 1 0 0 0 0 1 1 = $C3
```

Bit = 1, commande autorisée.
Bit = 0, commande non autorisée.

Avec un byte de valeur #$C3, la lecture et l'écriture dans un fichier pourront être réalisées. Les commandes RENAME et DELETE sont autorisées.
Avec un byte de valeur #$E3, le fichier autorise une copie de ses données - bits 7, 6, 5, 1 et 0 positionnés à 1.

• FILE TYPE

$10 (16) : position relative dans l'entrée de la table des matières d'un nom de fichier.
Paramètre utilisé par les commandes CREATE, SET.FILE.INFO et GET.FILE.INFO. Le codage sur un byte caractérise le type de fichier sauvegardé dans le catalogue de la disquette. Exemple : $FC = fichier du type BAS ou Basic.

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		Réservé 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.

Pour la table complète des 256 types de fichiers pour les systèmes PRODOS et SOS, voir l'annexe 4.

• STORAGE TYPE

$00 (00) : position relative dans l'entrée de la table des matières d'un nom de fichier.
Storage Type caractérise un fichier suivant sa taille, si c'est un fichier de données, ou suivant son identité, si c'est un fichier Directory.
Les commandes CREATE et GET.FILE.INFO utilisent un byte caractéristique, dont l'image binaire traduit le genre (fichier de données ou Volume) et la longueur du nom d'un fichier. Le byte se divise en deux nibbles: celui de gauche, appelé Storage Type, détermine la spécificité du fichier, taille ou Volume type ; celui de droite, appelé Name Length, donne la longueur du nom.
Le nibble de gauche, relatif au paramètre Storage Type, peut prendre plusieurs valeurs : #$1, #$2 ou #$3, s'il caractérise un fichier de données ; #$D, #$E ou #$F si, par contre, il désigne un fichier Directory ou Subdirectory.
Si sa valeur est nulle (#$0), le fichier n'est pas défini et peut soit être le résultat d'une commande DELETE, soit désigner un Volume-Directory vide.

Représentation du byte complet :

```
Bits : 7 6 5 4 3 2 1 0
       T T T T N N N N      T = Storage Type
                            N = Name Length
```

Représentation du nibble de gauche :

```
Bits : 7654
       TTTT  T = Storage Type
```

Exemple :

$F8, où F = Volume Directory.
8 = Nom, formé de 8 caractères.

$0 Non défini. Représente un fichier "deleté" ou un fichier catalogue encore vide.

$1 Seedling. C'est la plus petite structure d'un fichier sauvegardé sur une disquette : elle occupe 1 bloc. C'est un fichier de données, pointé directement par le Directory qui contient son nom. Le pointeur est stocké à la position relative $11-$12 (17-18) dans l'entrée de la table des matières du fichier. La capacité de sauvegarde d'un tel fichier est de #$0000 à #$0200 bytes (0-512 bytes) et comporte un bloc de données mais pas de bloc index.

$2 Sapling. C'est un fichier de la taille au-dessus du précédent : il peut avoir une capacité de sauvegarde de $0201 à $20000 bytes (513 à 131 072 bytes). La structure se résume ainsi : un nom de fichier pointe un bloc index qui peut pointer au maximum 256 blocs de données : 256 * 512 = 131 072 bytes ou 128 Ko.

$3 Tree. C'est la capacité maximale d'un fichier dans la hiérarchie des tailles : il peut avoir une capacité de sauvegarde de $020001 à $1000000 bytes - 131 073 à 16 777 216 bytes ou 16 mégabytes. La structure se résume ainsi: un bloc index principal, Master Block, peut pointer au maximum 128 blocs index, (Index Blocks). Chacun des blocs index peut à nouveau pointer au maximum 256 blocs de données.
Capacité maximale : 32 768 blocs de données, pointés par 128 blocs index auxiliaires et 1 bloc index principal.

$D Subdirectory-File : nom de fichier sauvegardé dans un Directory et qui pointe son Subdirectory - Nom d'un fichier Subdirectory.

$E Volume-Subdirectory : nom de Volume sauvegardé en en-tête de la table des matières d'un Volume-Subdirectory - Nom de Volume d'un Subdirectory.

$F Volume-Directory : nom de Volume sauvegardé en en-tête de la table des matières du Volume-Directory - Nom de Volume de la disquette.

• AUXILIARY TYPE

$1F-$20 (31-32) : position relative dans l'entrée de la table des matières d'un nom de fichier.
Utilisé par les commandes CREATE, SET.FILE.INFO et GET.FILE.INFO, ce paramètre est : soit un pointeur, soit une longueur. ProDOS ne l'utilise pas toujours, mais s'il est présent, la commande CATALOG l'affiche sous la rubrique SUBTYPE. Les renseignements ainsi retournés peuvent être de nature différente : par exemple, avec un fichier texte du type aléatoire, la longueur déclarée par le paramètre L# sera sauvegardée à l'emplacement Auxiliary Type, sous la forme LByte, HByte.

Représentation des 2 bytes :

Bits :	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	LByte,HByte
	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	50,01
																	Longueur = 150 bytes

Exemple à partir de Basic :

```
10 PRINT CHR$(4) ;"OPEN ALEATOIRE, L150"
```

Les fichiers du type TXT séquentiels suivent une autre règle : ils ne nécessitent pas la déclaration du paramètre L#, longueur de l'enregistrement, et dans ce cas Auxiliary Type sera nul.

Fichier BIN et BAS : le pointeur désigne l'adresse de début d'implantation du programme en mémoire sous la forme LByte,HByte. En Basic AppleSoft - type BAS - le pointeur sera représenté par 01 08 - adresse $0801 - qui est la valeur par défaut du système.

en place uniquement pour garder une symétrie avec la position des paramètres de la commande GET.FILE.INFO. Les 3 bytes de Null Field occupent la même position, dans la

Représentation des 2 bytes :

Bits :	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	Adresse LByte,HByte
	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	Adresse 00,20
																	Adresse $2000

• BLOCKS USED

$13-$14 (19-20) : Position relative dans l'entrée de la table des matières d'un nom de fichier.
La commande GET.FILE.INFO utilise le paramètre Blocks Used. Deux bytes sont nécessaires pour coder le nombre de blocs occupés par un fichier : blocs index et blocs de données.
Un fichier Seedling occupe 1 bloc.
Un fichier Sapling occupe un minimum de 3 blocs - 1 bloc index + 2 blocs de données - et au maximum 257 blocs - 1 bloc index + 256 blocs de données.
Un fichier Tree nécessite un minimum de 260 blocs - 1 Master Block + 2 Blocs Index + 257 blocs de données - et au maximum 32 897 blocs - 1 Master Block + 128 Blocs Index + 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 2 bytes :

Bits :	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	Blocs LByte,HByte
	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	Blocs 57,00
																	Blocs occupés = 57

• NULL FIELD

Null Field désigne un emplacement nul à l'intérieur de la table des paramètres de la commande SET.FILE.INFO. Cette zone de 3 bytes, sans valeur significative, a été mise

en place uniquement pour garder une symétrie avec la position des paramètres de la commande GET.FILE.INFO. Les 3 bytes de Null Field occupent la même position, dans la table, que les paramètres de Storage Type et de Blocks Used.

• EOF - End Of File

$15-$17 (21-23) : position relative dans l'entrée de la table des matières d'un nom de fichier.
Ce paramètre est utilisé par la commande SET.EOF et GET.EOF. Il nécessite 3 bytes pour coder la longueur d'un fichier de texte.
EOF pointe un emplacement logique - fictif et non matérialisé - à l'intérieur d'un fichier texte et désigne la longueur des données: Low Byte - Middle Byte - High Byte.
La longueur théorique d'un fichier peut se situer de #$0001 à #$FFFF bytes (1 à 65 535 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 + 4 096 + 131 072 = 135 344 bytes

EOF détermine d'une part un pointeur qui désigne 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 :

Prenons le texte APPLE et sauvegardons-le sur une disquette, sous la forme d'un fichier texte. La représentation des bytes physiques et logiques sera la suivante :

A P P L E CR	CR = retour-chariot.
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.

MARK, pointé par le byte $05, est représenté par le byte de valeur #$0D - code du retour-chariot. C'est la position relative dans le texte du fichier, souvent désignée comme position actuelle pour lire ou écrire la donnée courante.

EOF est matérialisé par un byte logique, avec un pointeur de valeur $06 : les données ont une longueur de 6 bytes. Le 6e byte, à partir du début des données, est la position absolue dans le fichier, matérialisée par un byte physique (#$0D = CR). Le pointeur débute par la valeur $00 : il compte les caractères physiques et leur ajoute 1 (la valeur totale est la longueur de l'enregistrement).

Commentaires du texte APPLE :

La première rangée contient les caractères ASCII du texte + le retour-chariot - CR.
La deuxième rangée contient les codes ASCII des caractères de la première rangée, avec les bits 7 à 0.
La troisième rangée affiche la position relative du pointeur à l'intérieur du fichier créé et débute avec une valeur nulle - $00. Pointeur désigné le plus souvent sous le nom de MARK et qui débute avec la valeur #$00.
La quatrième rangée représente la position absolue à l'intérieur du fichier et détermine, en même temps, la longueur totale des données sauvegardées. Le pointeur débute avec la valeur #$01.
Le retour-chariot matérialise la fin de chaque enregistrement à l'intérieur d'un fichier, qui peut en contenir plusieurs à la file. Le code #$0D ne matérialise pas la fin du fichier, mais uniquement celle de l'enregistrement.
EOF serait le prochain byte - byte imaginaire - qui suivrait le retour-chariot du dernier enregistrement.
Si la fin d'un enregistrement sur une disquette correspond avec la fin d'un bloc, 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 de la disquette, le complément des bytes formant ce bloc sera représenté par des zéros (bytes nuls = 00). Le premier 0 après le dernier code ASCII #$0D matérialise la fin du fichier, qui est vu par ProDOS comme un byte logique - EOF.

• POSITION - MARK

Ce paramètre est utilisé par les commandes SET.MARK et GET.MARK. Il est codé sur 3 bytes : Low Byte - Middle Byte - High Byte et représente la position courante du caractère à l'intérieur d'un fichier texte. Cette position ne peut, en aucun cas, excéder la valeur du pointeur EOF.
Le pointeur Position (MARK) désigne la position d'un byte physique à l'intérieur d'un fichier : c'est le caractère actuel en cours de traitement. Lorsqu'une sauvegarde de données est réalisée sur une disquette, c'est le retour-chariot qui détermine Position. Le pointeur débute avec la valeur #$00, et compte les bytes physiques d'un enregistrement pour se positionner sur le caractère à lire ou à écrire.

• NEWLINE CHARACTER

Ce paramètre est utilisé par la commande MLI NEWLINE et nécessite 1 byte pour coder un caractère. Lorsque le byte du paramètre Enable Mask est nul, Newline Character sera passif.

Mise au point :

Pointeur EOF : EOF désigne la longueur totale d'un enregistrement, ou encore la position absolue du dernier byte des données. EOF n'est pas matérialisé sur la disquette, mais pointe un byte logique et imaginaire qui se trouverait à la suite du dernier enregistrement. Le premier caractère du prochain enregistrement se trouvera à l'emplacement désigné actuellement par EOF.

Pointeur MARK : MARK indique la position de fin des données d'un enregistrement dans un fichier texte. Sa position est matérialisée sur la disquette de sauvegarde : par le code #$0D (Return). C'est une marque physique et réelle, visible dans le fichier, qui est le repère ou la marque de la fin d'un champ de données, ou d'un enregistrement. Un fichier peut comporter une ou plusieurs marques de fin, suivant qu'il contient un ou plusieurs champs ou enregistrements de données.

Pointeur actuel : la position courante est calculée en fonction du pointeur MARK et détermine le caractère à lire ou à écrire dans un fichier. La valeur du pointeur est calculée en fonction de la position absolue du début des données (début = $000000).

La commande READ autorise deux types de lecture :
- lire un nombre déterminé de données à partir d'une position absolue dans un fichier texte ;
- lire un nombre de bytes par défaut, et, dans ce cas, un maximum de 224 caractères seront lus.

Paramètres adressables avec READ :

Fichiers du type séquentiel.

- Le paramètre F# permet de préciser le nombre de champs ou rubriques qu'il faut parcourir avant de lire les données. Le système compte le nombre de retours-chariot déclarés après F, pour se placer au début du champ à lire.

- Le paramètre B# permet de préciser le nombre de bytes à parcourir, avant de lire les données. Le système compte le nombre de bytes à partir de la position absolue de l'enregistrement déclaré - Champ de données.

Fichiers du type aléatoire.

- Pour les fichiers aléatoires, le paramètre R# permet de désigner le numéro de l'enregistrement à lire.

- **ENABLE MASK**

Ce paramètre est utilisé par la commande NEWLINE. Il nécessite un byte pour coder un masque permettant de forcer des bits à 1 lors de la saisie d'un caractère. Son utilisation en programmation machine pourrait avoir la syntaxe suivante :

```
LDA CARACTERE       Saisie d'un caractère du texte.
AND ENABLE-MASK     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 contient le code ASCII de Newline Character, la saisie des caractères du fichier est interrompue.

Valeurs du masque

Si la valeur du paramètre est #$7F (127), le bit 7 des caractères sera ignoré ce qui force l'interpréteur à évaluer d'une façon identique le code #$0D et #$8D - #$0D avec le bit 7 à 0, et #$8D avec le bit 7 à 1.
Si la valeur du paramètre est #$FF (255), chaque bit sera significatif, tandis que la valeur #$00 annulera le mode NEWLINE.

- **FILE REFERENCE NUMBER**

Ce paramètre, codé sur un byte, est utilisé par les commandes OPEN, CLOSE, FLUSH, NEWLINE, READ, WRITE, SET.MARK, GET.MARK, SET.EOF, GET.EOF, SET.BUF et GET.BUF.
La commande OPEN nécessite comme suffixe un nom de fichier. Le système copie dans la table des paramètres de OPEN - début de la table : $BECB -, un numéro de référence qui devra être utilisé par la suite (commandes citées auparavant). Cette pratique permet de traiter chaque fichier d'après son numéro de référence avec d'autres commandes, sans pour autant les confondre.

- **REQUEST COUNT - Bytes déclarés**

Ce paramètre, codé sur 2 bytes - LByte, HByte -, est utilisé par les commandes READ et WRITE. C'est le maximum de bytes ou caractères pouvant être transférés dans un tampon mémoire pointé par Data Buffer. Le nombre de bytes est limité par le nombre de pages mémoire libres. Lorsque la valeur de Newline Character 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.

- **TRANSFER COUNT - Bytes transférés**

Ce paramètre, codé sur 2 bytes - LByte, Hbyte -, est utilisé par les commandes READ et WRITE. Ce sont les bytes réellement lus ou écrits, par opposition à Request Count qui désigne les bytes à lire ou à écrire dans un fichier.
C'est le paramètre R# des commandes READ et WRITE, dont la valeur devra être un entier dans la limite d'un nombre de 0 à 65535 inclus. Si le paramètre n'est pas déclaré, PRODOS lira l'enregistrement 0 jusqu'à rencontrer un retour-chariot - code #$0D. Si R# désigne un enregistrement vide, une erreur surviendra au moment de l'instruction INPUT.
Avec la commande WRITE et le paramètre R# supérieur à la valeur de l'enregistrement précédent, PRODOS mettra à jour la zone EOF - End Of File - de la table des matières de la disquette (entrées relatives $15-$17 - 21-23 - de la table des matières).

- **UNIT NUMBER**

Ce paramètre, codé sur un byte, est utilisé par les commandes READ.BLOCK et WRITE.BLOCK. Le byte intégre l'image binaire des numéros de slot et drive déclarés.

Représentation du byte :

```
Bits : 7 6 5 4 3 2 1 0
       D S S S x x x x     D = drive.
                           S = slot.
                           x = Inutilisés.
```

DRIVE :
Bit 7 = 0 : drive 1
Bit 7 = 1 : drive 2

SLOT :
Bits : 654
001 = slot 1
100 = slot 4
110 = slot 6
111 = slot 7

Valeurs possibles de Unit Number

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 Unit Number est nul, tous les slots seront mémorisés et le résultat sera copié dans le tampon Data Buffer.

Exemples :

```
Bits : 7 6 5 4 3 2 1 0
       1 1 1 0 0 0 0 0    = $E0
                          = drive 2, slot 6

Bits : 7 6 5 4 3 2 1 0
       0 0 1 1 0 0 0 0    = $30
                          = drive 1, slot 3

Bits : 7 6 5 4 3 2 1 0
       0 0 0 0 0 0 0 0    = $00
                          = tous les slots seront mémorisés
```

- **BLOCK NUMBER**

$13-$14 (19-20): position relative dans une entrée de la table des matières d'un nom de fichier.
Ce paramètre, codé sur deux bytes (LByte et HByte), est utilisé par les commandes READ.BLOC et WRITE.BLOCK. Le couple de bytes désigne la taille d'un fichier qui regroupe un certain nombre de blocs index et de blocs de données. C'est la capacité logique d'un fichier.

La capacité standard d'une disquette souple 5 pouces 1/4 va de #$0000 à #$0117, ce qui fait un total de #$0118 blocs, soit 280 blocs. La surface d'une disquette se divise en 35 pistes qui, à leur tour, se subdivisent en 16 secteurs par piste. Cette structure représente la sectorisation physique d'une disquette.
ProDOS utilise sa structure propre, issue d'une conversion : c'est le format logique. Deux secteurs physiques sont l'équivalent d'un bloc logique. Le nombre de blocs que pourrait traiter un fichier se monte à #$FFFF, soit 65535 blocs.

Principe de lecture des données, par un DOS :

1- un secteur est lu à partir d'une disquette ;
2- les données sont ensuite placées dans un tampon mémoire : tampon de destination ;
3- s'il s'avère nécessaire de lire d'autres secteurs, le système branche à la fonction première (1).

Pendant le traitement des données - en lecture ou en écriture - et avant de poursuivre avec le prochain secteur, celui-ci passe au moins partiellement sous la tête de lecture/écriture du lecteur de disquettes. Si le déroulement des opérations nécessite un deuxième secteur, dont la position physique est directement à la suite du secteur précédent, le système devra attendre une rotation complète de la disquette. Par exemple, une image HGR d'une taille de 8 Ko, comptabiliserait un temps d'attente total d'environ six secondes, avant son implantation en mémoire centrale. Pour réduire ce temps d'attente, ProDOS a recours à une conversion des secteurs physiques en secteurs logiques, qui prennent la dénomination de blocs.

Lecture des blocs 0-7 :

bloc 00 : secteur 00 et 02
bloc 01 : secteur 04 et 06
bloc 02 : secteur 08 et 0A
bloc 03 : secteur 0C et 0E
bloc 04 : secteur 01 et 03
bloc 05 : secteur 05 et 07
bloc 06 : secteur 09 et 0B
bloc 07 : secteur 0D et 0F

Le même processsus reste valable pour les pistes suivantes. Maintenant, la routine RWTB - lecture/écriture d'un bloc - assemble les secteurs à l'intérieur d'un tampon et le temps nécessaire pour cette opération sera interne à la routine MLI. Toutes les opérations de conversion, secteur physique en secteur logique ou inversement, s'effectuent à l'intérieur de la routine RWTB, qui débute à l'adresse $F800. Le chapitre 2 traite plus spécialement l'organisation d'une disquette et ses différents formats.

La commande CAT ou CATALOG affiche le paramètre Block Number sous la rubrique BLOCKS ; il se trouve donc sauvegardé dans la table des matières du fichier.

- **INTERRUPT CODE POINTER**

Ce paramètre, codé sur 2 bytes - LByte, HByte -, est utilisé par la routine MLI ALLOCATE.INTERRUPT. Il pointe l'adresse de début du traitement des interruptions -

Interrupt-Handler. Quatre interruptions peuvent ainsi être traitées par ProDOS, qui gère les pointeurs dans la page globale, aux adresses $BF80-$BF87. Un numéro de référence de 1 à 4 est attribué à chaque pointeur et traité, par la suite, par la commande DEALLOC INTERRUPT.

- **INTERRUPT REFERENCE NUMBER**

Ce paramètre, codé sur un byte, est utilisé par les commandes ALLOCATE.INTERRUPT et DEALLOCATE.INTERRUPT. C'est un numéro de référence entre 1 et 4, attribué aux pointeurs de gestion d'interruptions, pour que ProDOS puisse les différencier les uns des autres. L'ordre de traitement des interruptions est imposé par la position des pointeurs stockés dans la page globale de PRODOS : le premier pointeur sera traité en numéro 1, le deuxième pointeur en numéro 2, et ainsi de suite.
Le numéro de référence est pris en considération par la commande DEALLOC INTERRUPT.

Appel du MLI par programme

Un programme Basic peut solliciter le module BASIC.SYSTEM qui est l'interface entre le système PRODOS et le langage AppleSoft, en faisant un appel à l'adresse $BF00, par un CALL 48896. En programmation machine, ce branchement se fera par un JSR, suivi de l'adresse du MLI, sous la forme de deux bytes, et dans l'ordre : byte poids faible, byte poids fort - LByte, HByte. Le code d'erreur éventuel, conséquence d'une exécution non aboutie, pourra être récupéré, et sa valeur affichée à l'écran. L'erreur renvoyée pourra par la suite être gérée grâce à un sous-programme mis en place par l'utilisateur. L'exemple ci-après vous montre un appel MLI type, et une gestion de l'erreur rencontrée, effectuée par un saut à l'adresse représentée par la variable ERR. Une table pourra être implantée à cette adresse pour gérer les erreurs ProDOS.

SYNTAXE D'UN APPEL MLI
Assembleur Big-Mac

JSR	MLI	où MLI équivaut à $BF00.
DFB	FICHIER	où FICHIER est le type de commande du fichier, à l'exception de Basic ($C0).
DA	PARMS	où PARMS est l'adresse de début de la table des paramètres pour créer le fichier.
BNE	ERR	où ERR est l'adresse d'une table d'erreurs. L'accumulateur est chargé du code de l'erreur rencontrée.

Remarque : à la fin de l'exécution de la routine ci-dessus, le compteur ordinal (PC) pointe l'adresse de retour et a comme valeur : JSR + 3 ; c'est la situation de notre exemple, après la déclaration de l'instruction BNE ERR. Si une erreur est rencontrée lors du traitement du sous-programme, l'indicateur C du registre d'état PS (bit 0) sera positionné, et le code d'erreur sauvegardé dans l'accumulateur.

Tableau récapitulatif des indicateurs d'état

Bits :	7 1 0 3 6 N Z C D V	ACC	PC	X Y Pile
Sans erreur :	0 1 0 0 x	0	JSR+3	Inchangés
Lors d'une erreur :	0 0 1 0 x	Code erreur	JSR+3	Inchangés

Légende :

- N, Z, C, D et V sont les indicateurs du registre d'état PS du microprocesseur (Statut).
- PC est le compteur ordinal (PCL, PCH)
- X et Y sont respectivement les registres d'index X et Y.
- Pile est bien entendu la pile - Stack - du microprocesseur.

Remarque : la valeur du bit 7 (N) du registre d'état PS est déterminée suivant le contenu de l'accumulateur. Si la valeur du code d'erreur est inférieure à #$80 (128), alors le bit 7 sera remis à zéro. La valeur du bit 6 (V) restera par contre indéterminée; elle est représentée par x dans le tableau récapitulatif.

En annexe 7 se trouve un court programme en assembleur, qui illustre parfaitement l'appel du MLI : il renomme un fichier existant sur une disquette, et affiche le code d'erreur lors d'une exécution infructueuse.

LES COMMANDES DU DIRECTORY

Le *Technical Manual* les appelle "Housekeeping Calls" : elles regroupent la famille des commandes pour créer, renommer et effacer des fichiers du Directory. Elles permettent un choix au niveau d'un périphérique en ligne, ou du préfixe.

Liste des commandes du Directory

$C0 CREATE	Créer un fichier programme ou Directory.
$C1 DESTROY	Détruire un nom de fichier - DELETE.
$C2 RENAME	Renommer un fichier.
$C3 SET.FILE.INFO	Permet de modifier des informations (date et heure par exemple).
$C4 GET.FILE.INFO	Permet de lire des informations (date et heure par exemple).
$C5 ON.LINE	Retourne les caractéristiques des Volumes en ligne ; drive, slot et nom de Volume.
$C6 SET.PREFIX	Permet de déclarer le préfixe.
$C7 GET.PREFIX	Affiche ou actualise le préfixe.

Commandes détaillées

• CREATE - $C0

Parameter Count	:	1 byte ($07).
Pathname Pointer	:	2 bytes - LByte, HByte.
Access	:	1 byte (Unlock File = $C3).
File Type	:	1 byte ("Font" File = $07).
Auxiliary Type	:	2 bytes (AppleSoft = $0801).
Storage Type	:	1 byte (Seedling = $01).
Create Date	:	2 bytes (00 00 pas de date).
Create Time	:	2 bytes (00 00 pas d'heure).

CREATE est la seule commande qui permet de créer un Volume-Subdirectory, ou table des matières auxiliaire.
Lors de la création d'un fichier, programme ou Directory, le système sauvegarde en même temps un byte d'identité. Celui-ci se compose des paramètres Storage type et Name Length et 4 bits sont nécessaires pour coder chacun d'eux. Ce byte est stocké à la position relative #$00 de l'entrée d'une table des matières. ProDOS permet, grâce à cet artifice, de différencier six types de sauvegardes, suivant la valeur attribuée au byte.

Signification de chaque type de sauvegarde :

$01 : fichier ne dépassant pas 1 bloc de données.
$02 : fichier avec au plus un bloc index, qui pointe au maximum 256 blocs de données.
$03 : fichier avec un bloc index principal - Master Block -, qui pointe au maximum 128 blocs index auxiliaires - Blocs Index -, dont chacun peut à nouveau pointer 256 blocs de données.
$0D : Subdirectory-File. Nom d'un fichier sauvegardé dans un Directory, qui pointe un Volume-Subdirectory. Le nom du fichier se trouve dans la table des matières qui est le parent de la table des matières du Subdirectory. Les blocs de ces tables des matières sont chaînés entre eux.
$0E : Volume-Subdirectory. Le nom du Volume se trouve en en-tête de la table des matières du Subdirectory pointée par le Subdirectory-File.
$0F : Volume-Directory. Le nom du Volume se trouve en en-tête de la table des matières principale de la disquette. C'est le nom du Volume de la disquette.

Création d'un Volume-Subdirectory :

CREATE Subdirectory effectue la sauvegarde du nom d'un fichier Subdirectory dans le Directory, détermine l'emplacement du premier bloc du Volume 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 Directory, 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 - à 1.

Codes d'erreurs possibles :

La table complète des erreurs MLI se trouve en annexe 3.

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$2B Disk write protected.
Disque protégé en écriture.
$40 Invalid pathname.
Syntaxe ou chemin incorrect.
$44 Directory not found.
Directory non trouvé.
$45 Volume not found.
Volume non trouvé.
$46 File not found.
Fichier non trouvé.
$47 Duplicate filename.
Nom de fichier existant.
$48 Volume full.
Disquette pleine, ou dépassement du pointeur EOF lors de la commande WRITE.
$49 Volume directory full.
Volume saturé (plus de 51 noms de fichiers).
$4B Unsupported storage type.
Type de fichier incorrect (sauvegarde).
$53 Invalid system call parameter.
Syntaxe ou nombre incorrect d'un paramètre MLI.
$5A File structure damaged.
La Volume Bit-Map est endommagée.

• DESTROY - $C1

Parameter Count	:	1 byte ($01).
Pathname Pointer	:	2 bytes - LByte, HByte.

L'instruction DELETE fait appel à la commande MLI DESTROY qui efface un fichier du Directory de la disquette et met à jour la table des matières correspondante.

Le nom du Volume de la disquette et un fichier ouvert ne peuvent être effacés de la table des matières.

Un fichier Subdirectory devra pointer un Volume vide avant de pouvoir être supprimé.
DESTROY met à 0 le byte Storage Type et laisse intact le nom du fichier. Le contenu du bloc index est mis à zéro.

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$2B Disk write protected.
Disque protégé en écriture.
$40 Invalid pathname.
Syntaxe ou chemin incorrect.
$44 Directory not found.
Directory non trouvé.
$45 Volume not found.
Volume non trouvé.
$46 File not found.
Fichier non trouvé.
$4A Incompatible file format.
Fichier : données incompatibles.
$4B Unsupported storage type.
Type de fichier incorrect (sauvegarde).
$4E File access error, also file locked.
Erreur d'accès au fichier : verrouillé.
$50 File is open.
Fichier ouvert (lors de la commande OPEN, etc.)
$5A File structure damaged.
La Volume Bit-Map est endommagée.

• RENAME - $C2

Parameter Count : 1 byte ($02).
Pathname Pointer : 2 bytes - LByte, HByte.
New Pathname : 2 bytes - LByte, HByte. Utilisé pour le deuxième nom lors de la commande RENAME.

Change le nom du fichier spécifié, à la suite de la commande RENAME, en un nouveau nom, dont la syntaxe est déclarée après la virgule. Avec la commande RENAME, les chemins déclarés doivent être les mêmes et faire partie d'une table des matières commune.

Exemples :

Syntaxe autorisée :

RENAME /UTILITAIRE/ANCIEN, /UTILITAIRE/NOUVEAU

Syntaxe correcte, car /UTILITAIRE/ est un préfixe commun aux deux noms de fichiers Directory.

Syntaxe interdite :

RENAME /UTILITAIRE/NOM, /CHEMIN/NOUVEAU

où /UTILITAIRE et /CHEMIN sont des préfixes différents.

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$2B Disk write protected.
Disque protégé en écriture.
$40 Invalid pathname.
Syntaxe ou chemin incorrect.
$44 Directory not found.
Directory non trouvé.
$45 Volume not found.
Volume non trouvé.
$46 File not found.
Fichier non trouvé.
$47 Duplicate filename.
Nom de fichier existant.
$4A Incompatible file format.
Fichier : données incompatibles.
$4B Unsupported storage type.
Type de fichier incorrect (sauvegarde).
$4E File access error, also file locked.
Erreur d'accès au fichier : verrouillé.
$50 File is open.
Fichier ouvert (lors de la commande OPEN, etc.)
$57 Duplicate volume.
Nom de Volume déjà en ligne.

• SET.FILE.INFO - $C3

Parameter Count : 1 byte ($07).
Pathname Pointer : 2 bytes - LByte, HByte.
Access : 1 byte.
File Type : 1 byte.
Auxiliary Type : 2 bytes.
Null Field : 3 bytes (format : 00 00 00).
Modified Date : 2 bytes.
Modified Time : 2 bytes.

Cette commande permet de modifier certains paramètres se rapportant à un fichier : il peut être soit ouvert, soit fermé. Par cette commande, le byte d'accès au fichier pourra par exemple être modifié.
Avec RENAME et SET.FILE.INFO, les paramètres suivants ne seront pas modifiés : date et heure de sauvegarde des données, pointeur du fichier, nombre des blocs occupés par le fichier, pointeur du Volume.
SET.FILE.INFO peut être associé à un fichier ouvert, mais reste 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 donc par principe difficilement adressable. Il sera plus sage d'utiliser SET.FILE.INFO avec un fichier fermé.

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$2B Disk write protected.
Disque protégé en écriture.
$40 Invalid pathname.
Syntaxe ou chemin incorrect.
$44 Directory not found.
Directory non trouvé.
$45 Volume not found.
Volume non trouvé.
$46 File not found.
Fichier non trouvé.
$4A Incompatible file format.
Fichier : données incompatibles.
$4B Unsupported storage type.
Type de fichier incorrect (sauvegarde).
$4E File access error, also file locked.
Erreur d'accès au fichier : verrouillé.
$53 Invalid system call parameter.
Syntaxe ou nombre incorrect d'un paramètre MLI.
$5A File structure damaged.
La Volume Bit-Map est endommagée.

• GET.FILE.INFO ($C4)

Parameter Count	:	1 byte ($0A).
Pathname Pointer	:	2 bytes - LByte, HByte.
Access	:	1 byte - mis en place par le MLI.
File Type	:	1 byte - mis en place par le MLI.
Auxiliary Type	:	2 bytes - mis en place par le MLI.
Storage Type	:	1 byte - mis en place par le MLI. Null Field avec la commande SET.FILE.INFO.
Blocks Used	:	2 bytes - LByte, HByte. Null Field avec la commande SET.FILE.INFO.
Modified Date	:	2 bytes - mis en place par le MLI.
Modified Time	:	2 bytes - mis en place par le MLI.
Create Date	:	2 bytes - mis en place par le MLI.
Create Time	:	2 bytes - mis en place par le MLI.

GET.FILE.INFO lit les paramètres attribués à un fichier, les modifie en mémoire, puis les sauvegarde en appelant SET.FILE.INFO.

Lorsque GET.FILE.INFO est associé à une commande de Volume, le MLI attribue à Auxiliary Type la capacité totale d'une disquette, évaluée en blocs, et à Blocks Used la somme des blocs occupés par les fichiers. Le nombre total des blocs occupés figure à un endroit précis de la disquette, sa valeur étant obtenue par calcul. Le MLI soustrait au nombre total des blocs de la disquette le nombre des blocs occupés par les fichiers.

Capacité de sauvegarde d'une disquette :

La capacité de sauvegarde d'une disquette se trouve stockée aux bytes $29-$2A (41-42) de l'entrée de la table des matières du Volume-Directory. Le pouvoir de sauvegarde d'une disquette est déterminé lors du formatage : pour un disque standard du type 5 pouces 1/4, elle est de #$0118 blocs, soit 280 blocs, par défaut.

Nombre des blocs occupés :

Ce nombre est évalué par le MLI et se trouve être la somme des blocs attribuée à chaque fichier sauvegardé sur la disquette. A cet effet, deux bytes sont réservés pour chaque entrée d'un nom de fichier dans la table des matières, pour stocker le nombre des blocs alloués. Leur position relative dans l'entrée de la table des matières se trouve aux bytes $13-$14 (19-20).

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$40 Invalid pathname.
Syntaxe ou chemin incorrect.
$44 Directory not found.
Directory non trouvé.
$45 Volume not found.
Volume non trouvé.
$46 File not found.
Fichier non trouvé.
$4A Incompatible file format.
Fichier : données incompatibles.
$4B Unsupported storage type.
Type de fichier incorrect (sauvegarde).
$53 Invalid system call parameter.
Syntaxe ou nombre incorrect d'un paramètre MLI.
$5A File structure damaged.
La Volume Bit-Map est endommagée.

• ON.LINE - $C5

Parameter Count	:	1 byte ($02).
Unit Number	:	1 byte.
Data Buffer Pointer	:	2 bytes - LByte, HByte.

ON.LINE mémorise slots et drives en ligne, ainsi que le nom du Volume et sa longueur en caractères. Cette recherche prend deux sens: d'une part, tous les noms de Volumes sont mémorisés ; d'autre part, une sélection des slots et drives est effectuée.
Lorsque le nom du Volume ou son emplacement, slot et drive, est déclaré dans la commande, le système utilise un tampon de 16 bytes pour la sauvegarde des

paramètres. Si Unit Number est nul - #$00 -, aucun slot ni drive ne seront mémorisés : tous les noms de Volumes seront recherchés et le tampon aura une taille de 256 bytes.

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$28 No device connected.
Pas de périphérique en ligne.
$2E Disk swichtched.
Disquettes inversées (les commandes WRITE, CLOSE, etc. ne peuvent aboutir).
$45 Volume not found.
Volume non trouvé.
$52 Not a ProDOS volume.
Pas de Volume ProDOS.
$55 Volume Control Block table full.
Table de gestion des blocs saturée : survient lors d'un traitement de plus de 8 fichiers, ou avec plus de 8 périphériques adressés.
$56 Bad buffer address.
Adresse du tampon non valide.
$57 Duplicate volume.
Nom de Volume déjà en ligne.

• SET.PREFIX ($C6)

Parameter Count : 1 byte ($01).
Pathname Pointer : 2 bytes - LByte, HByte.

La commande spécifie le Directory d'une disquette en ligne. Le chemin sera partiel ou complet avec une limitation de 64 caractères.
PREFIX/VOLUME effectue une recherche sélective du nom du Volume /VOLUME à travers tous les drives en ligne.
Lorsqu'un préfixe est déclaré nul par l'intermédiaire d'une chaîne de caractères, il aura comme longueur une valeur nulle (L = #$00).

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$40 Invalid pathname.
Syntaxe ou chemin incorrect.
$44 Directory not found.
Directory non trouvé.
$45 Volume not found.
Volume non trouvé.
$46 File not found.
Fichier non trouvé.

$4A Incompatible file format.
Données incompatibles d'un fichier.
$4B Unsupported storage type.
Type de fichier incorrect (sauvegarde).
$5A File structure damaged.
La Volume Bit-Map est endommagée.

• GET.PREFIX - $C7

Parameter Count : 1 byte ($01).
Data Buffer Pointer : 2 bytes - LByte, HByte.

Data Buffer Pointer pointe un tampon de 128 bytes de longueur : 64 bytes pour le chemin partiel, et 64 bytes pour le préfixe. Ce tampon est utilisé comme mémoire auxiliaire pour la sauvegarde temporaire des données relatives au chemin : la longueur et le nom du chemin, barre oblique (/) incluse, y sont placés.
L'instruction PREFIX appelle la commande GET.PREFIX et retourne le préfixe courant.
Si un préfixe nul est déclaré, GET.PREFIX affichera la valeur 0 comme longueur du préfixe.

Code d'erreur possible :

$56 Bad buffer address.
Adresse du tampon non valide.

LES COMMANDES DE GESTION DES FICHIERS

Le *Technical Manual* les appelle "Filing 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 disquettes 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

$C8	OPEN	Ouvre un fichier texte et alloue 1 024 octets à un tampon mémoire.
$C9	NEWLINE	Définit la marque de fin d'un champ d'enregistrement, ou de fin d'un fichier texte.
$CA	READ	Effectue la lecture d'un nombre de bytes définis au préalable (à partir d'une disquette par exemple).
$CB	WRITE	Effectue la sauvegarde d'un certain nombre de bytes définis au préalable (sur une disquette par exemple).
$CC	CLOSE	Ferme tout fichier texte encore ouvert.

$CD	FLUSH	Sauvegarde sur une disquette les données encore présentes dans le tampon mémoire alloué au fichier.
$CE	SET.MARK	Définit la position d'un pointeur à l'intérieur d'un fichier texte.
$CF	GET.MARK	Recherche dans un fichier texte la position définie par SET.MARK.
$D0	SET.EOF	Définit une position logique qui est la fin d'un fichier texte, EOF, End Of File.
$D1	GET.EOF	Recherche la marque EOF.
$D2	SET.BUF	Tampon d'entrée/sortie à définir.
$D3	GET.BUF	Recherche de l'adresse du tampon défini par SET.BUF.

Commandes détaillées

• OPEN - $C8

Parameter Count	:	1 byte ($03).
Pathname Pointer	:	2 bytes - LByte, HByte.
I/O Buffer Pointer	:	2 bytes - LByte, HByte. Exemple : 00 90 = $9000.
File Reference Number	:	1 byte - utilisé par le MLI.

La commande OPEN prépare un fichier texte à la lecture ou à l'écriture. Un maximum de 8 fichiers seront ouverts simultanément.
OPEN alloue au fichier nommé un tampon de 1 024 octets, utilisé comme mémoire temporaire, pour les échanges des données entre la disquette et la mémoire vive. Ce tampon restera actif jusqu'au moment où la commande CLOSE sera déclarée : le reste des bytes alors présents dans le tampon sera sauvegardé sur la disquette, l'espace mémoire sera libéré et le fichier fermé.
Les 512 premiers bytes du tampon sont utilisés pour placer les données du fichier, tandis que les 512 bytes suivants gèrent le répertoire des blocs alloués.
Après la commande OPEN, le MLI copie dans la table des paramètres MLI, à l'adresse $BED2, un nombre référence compris entre 1 et 8 - maximum 8 fichiers -, puis donne le contrôle à la routine de File Controle Blocks qui gère les paramètres du fichier. Le code référence alloué devra être identique pour toutes les commandes relatives au traitement de ce même fichier.

La page globale de PRODOS contient l'adresse $BF94, LEVEL, qui désigne le niveau pour la fermeture des fichiers. Ce paramètre est utilisé par OPEN, FLUSH et CLOSE. Si, à partir d'une programmation en langage machine, vous désirez fermer à un moment donné tous les fichiers ouverts, il faudra au préalable placer un byte nul ($00) à l'adresse $BF94.

Exemple :

```
LDA #$00
STA $BF94
```

Remarque : l'instruction CLOSE, sans suffixe et déclarée à partir de Basic, effectue la fermeture de tous les fichiers ouverts. L'interpréteur se charge de POKEr les paramètres nécessaires - Commande CLOSE + LEVEL #$00.

La commande CLOSE, déclarée seule, effectue d'elle-même la fermeture de tous les fichiers ouverts à cet instant.

Codes d'erreurs possibles :

$27	I/O error. Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$40	Invalid pathname. Syntaxe ou chemin incorrect.
$42	Maximum number of files open. Maximum de fichiers ouverts (8 est le maximum).
$44	Directory not found. Directory non trouvé.
$45	Volume not found. Volume non trouvé.
$46	File not found. Fichier non trouvé.
$4B	Unsupported storage type. Type de fichier incorrect (sauvegarde).
$50	File is open. Fichier ouvert (lors de la commande OPEN, etc.)
$53	Invalid system call parameter. Syntaxe ou nombre incorrect d'un paramètre MLI.
$56	Bad buffer address. Adresse du tampon non valide.
$5A	File structure damaged. La Volume Bit-Map est endommagée.

• NEWLINE - $C9

Parameter Count	:	1 byte ($03).
File Reference Number	:	1 byte.
Enable Mask	:	1 byte (#$7F,#$FF,etc.).
Newline Character	:	1 byte (#$0D = Return).

Lorsque le byte du masque est nul - Enable Mask = #$00 -, NEWLINE reste passif.
La commande MLI NEWLINE est utilisée pour définir, par programmation, un byte de masque, pour 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 texte, est réalisée avec Enable Mask comme valeur #$00 et Newline Character, équivalant à un retour-chariot. Le code ASCII de valeur #$0D est le caractère par défaut attribué par le système pour désigner la fin d'un enregistrement ou d'un champ de données.

Avec NEWLINE actif, un caractère, défini par le programmeur, sera recherché à travers le fichier texte. Sa valeur, équivalente au code ASCII, sera déterminée par le masque du paramètre Enable Mask et influencera en même temps Newline Character.

Opération avec Enable Mask actif :

D'abord, le caractère lu est placé dans le Data Buffer, puis un ET Logique (AND) est effectué sur ce caractère par l'intermédiaire d'un masque - Enable Mask, le résultat est ensuite comparé à Newline Character. Si le byte est nul, correspond au caractère comparé, le dernier caractère lu détermine la fin du texte. Exemple : en attribuant les valeurs #$7F au masque et #$0D à Newline Character, tout caractère ASCII testé ayant comme valeur #$0D ou #$8D sera reconnu comme caractère de fin des données. Ce procédé ne modifie en aucune façon la valeur ou le code ASCII du caractère lu.

Opération avec Enable Mask passif :

Le système utilise ses propres paramètres : ils ont des valeurs établies par défaut et deux pointeurs sont utilisés pour la circonstance - EOF et MARK.
EOF pointe un byte logique dans le fichier texte, tandis que MARK pointe un byte physique, avec #$0D comme valeur de Newline Character.

Valeurs du masque Enable Mask :

Celui-ci pourra prendre toute valeur, suivant les besoins et le but recherché par le programme : avec une valeur #$7F, il force à 0 le bit 7 du caractère ASCII considéré ; une valeur #$FF rend tous les bits significatifs et une valeur #$00 rend le masque passif.

Code d'erreur possible :

$43 Invalid reference number.
Fichier : numéro de référence incorrect.

• READ - $CA

Parameter Count	:	1 byte ($04).
File Reference Number	:	1 byte.
Data Buffer Pointer	:	2 bytes - LByte, HByte.
Request Count	:	2 bytes - LByte, HByte. Exemple : 00 10 = $2000.
Transfer Count	:	2 bytes - mis en place par le MLI.

Règles :

La fin d'un enregistrement ou champ de données ne détermine pas forcément la fin d'un fichier texte. Plusieurs champs ou enregistrements successifs peuvent faire partie du même fichier texte. Chaque fin d'un champ de données est matérialisée à l'intérieur d'un fichier texte par le code ASCII #$0D qui est un retour-chariot (CR).

Deux cas de figure peuvent se présenter avec READ : le caractère de fin de données du fichier texte a été défini au préalable ; le caractère n'a pas été défini. Cela implique que NEWLINE soit actif dans le premier cas et passif dans le second.

Newline actif :

Cette commande effectue à partir d'une disquette le transfert vers le tampon mémoire d'un certain nombre de bytes - Request Count -, à partir de la position courante, MARK, dans un fichier texte. Le fichier est identifié par un numéro de référence (File Reference Number) attribué par le MLI au moment de la déclaration de la commande OPEN. Le tampon mémoire où sont placées les données est pointé par Data Buffer Pointer. Le nombre de bytes effectivement transférés est attribué au paramètre Transfer Count.

Newline passif :

Si le caractère fixé par le paramètre Newline Character est rencontré avant que le nombre de bytes fixés par Request Count soit lu, Transfer Count prendra comme valeur le nombre de bytes transférés, Newline byte inclus.

Lecture d'un champ dans un fichier :

Si la fin d'un fichier est rencontrée avant que le nombre de bytes définis au préalable soit lu, Transfer Count aura comme valeur le nombre de bytes transférés.

Remarque : l'erreur End of file encountered : fin du fichier rencontrée (code $4C) - avertit qu'aucun byte n'a été transféré ; le paramètre Transfer Count aura dans ce cas la valeur #$00.

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$43 Invalid reference number.
Fichier : numéro de référence incorrect.
$4C End of file encountered.
Fin de fichier rencontrée (marque EOF rencontrée en association avec la commande READ).
$4E File access error, also file locked.
Erreur d'accès au fichier : verrouillé.
$56 Bad buffer address.
Adresse du tampon non valide.
$5A File structure damaged.
La Volume Bit-Map est endommagée.

• WRITE - $CB

Parameter Count	:	1 byte ($04).
File Reference Number	:	1 byte.
Data Buffer Pointer	:	2 bytes - LByte, HByte.
Request Count	:	2 bytes - LByte, HByte.
Transfer Count	:	2 bytes - mis en place par le MLI.

WRITE effectue la sauvegarde d'un nombre de bytes - Request Count - à partir d'un tampon mémoire pointé par Data Buffer Pointer, dans un fichier texte ouvert par la commande OPEN (File Reference Number), à partir d'une position déterminée (MARK). Le nombre de bytes de données transférés sera attribué à Transfer Count. La nouvelle position à l'intérieur du fichier sera mise à jour : elle aura comme valeur la position initiale + la valeur du paramètre Transfer Count.

Les autres paramètres du fichier seront mis à jour : un bloc index, si le fichier comporte plus de 512 bytes de données ; le pointeur EOF augmenté en conséquence, ce qui déterminera la taille du fichier (nombre de blocs).

La commande CATALOG affiche, sous la rubrique SUBTYPE, la longueur d'un enregistrement d'un fichier texte du type aléatoire, et équivaut au paramètre R#.

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$2B Disk write protected.
Disque protégé en écriture.
$43 Invalid reference number.
Fichier : numéro de référence incorrect.
$48 Volume full.
Disquette pleine, ou dépassement de la marque EOF lors de la commande WRITE.
$4E File access error, also file locked.
Erreur d'accès au fichier : verrouillé.
$56 Bad buffer address.
Adresse du tampon non valide.
$5A File structure damaged.
La Volume Bit-Map est endommagée.

• CLOSE - $CC

Parameter Count : 1 byte ($01).
File Reference Number : 1 byte.

CLOSE ferme le fichier dont le nom suit. S'ils n'ont pas de nom spécifié, tous les fichiers ouverts à cet instant seront fermés. Cette commande a deux fonctions : d'une part, elle sauvegarde sur une disquette le contenu du tampon mémoire pointé par Data Buffer Pointer et, d'autre part, elle libère de la place en mémoire (1 024 bytes).
Le paramètre File Reference Number détermine le fichier à fermer. Si celui-ci est mis à zéro (#$00), tous les fichiers ouverts seront fermés. Exemple : en supposant 3 fichiers texte ouverts, avec comme référence 0, 1 et 2 et qu'à l'adresse $BF94 - LEVEL - le byte du niveau est fixé à la valeur 1, tous les fichiers à partir du numéro de référence 1 inclus seront fermés. En clair, cela signifie que les fichiers 1 et 2 seront fermés, tandis que le fichier 0 restera ouvert. Pour fermer tous les fichiers, le paramètre LEVEL devra avoir comme valeur #$00.

A partir de Basic, la commande CLOSE, seule, ferme tous les fichiers, le paramètre LEVEL étant géré par le système. En programmation machine, il faudra avant la commande OPEN introduire le paramètre LEVEL par la méthode suivante :

```
LDA NUMERO    où NUMERO désigne le niveau
STA $BF94     $BF94 est l'adresse de LEVEL.
```

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$2B Disk write protected.
Disque protégé en écriture.
$43 Invalid reference number.
Fichier : numéro de référence incorrect.
$5A File structure damaged.
La Volume Bit-Map est endommagée.

• FLUSH - $CD

Parameter Count : 1 byte ($01).
File Reference Number : 1 byte.

FLUSH purge le tampon pointé par Data Buffer Pointer et sauvegarde les données sur la disquette, en appelant la commande WRITE. Si File Reference Number est à #$00, tous les fichiers ouverts seront sauvegardés, sinon, le paramètre niveau sera pris en considération.

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$2B Disk write protected.
Disque protégé en écriture.
$43 Invalid reference number.
Fichier : numéro de référence incorrect.
$5A File structure damaged.
La Volume Bit-Map est endommagée.

• SET.MARK - $CE

Parameter Count : 1 byte ($02).
File Reference Number : 1 byte.
Position : 3 bytes.

SET.MARK met à jour la position courante, MARK, spécifiée par le paramètre Position, à l'intérieur d'un fichier texte. La valeur de Position ne pourra cependant pas excéder celle de EOF (End Of File).

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$43 Invalid reference number.
Fichier : numéro de référence incorrect.
$4D Position out of range.
Débordement de la position (plus grand que EOF).
$5A File structure damaged.
La Volume Bit-Map est endommagée.

• GET.MARK - $CF

Parameter Count : 1 byte ($02).
File reference Number : 1 byte.
Position : 3 bytes - mis en place par le MLI.
LByte, MByte, HByte.

Retourne la position courante à l'intérieur d'un fichier ouvert. C'est la position absolue qui est prise en considération par les commandes READ et WRITE. Après OPEN, le pointeur est mis à #$00 et ne peut pas déborder la valeur de EOF.

Code d'erreur possible :

$43 Invalid reference number.
Fichier : numéro de référence incorrect.

• SET.EOF - $D0

Parameter Count : 1 byte ($02).
File Reference Number : 1 byte.
E0F : 3 bytes - LByte, MByte, HByte.

SET.EOF fixe, à l'intérieur d'un fichier texte désigné par un numéro de référence, ou File Reference Number, un pointeur de fin EOF qui indique une position logique - byte fictif.
Lorsque le pointeur EOF est diminué, la Bit-Map, répertoire du nombre de blocs occupés, sera mise à jour sur la disquette. Si, par contre, le pointeur EOF est augmenté, un certain nombre de bytes nuls seront comptabilisés (Ctrl-à). Ce sont des données de valeurs nulles stockées entre deux enregistrements de longueurs différentes. Ces données sont appelées enregistrements "creux" ou Sparse File : elles encombrent inutilement la surface de la disquette et sont comptabilisées par le système pour l'occupation des blocs.

Remarque : pour de plus amples détails, consulter le chapitre 4 qui traite plus particulièrement des fichiers.

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$43 Invalid reference number.
Fichier : numéro de référence incorrect.
$4D Position out of range.
Débordement de la position (plus grand que EOF).
$4E File access error, also file locked.
Erreur d'accès au fichier : verrouillé.
$5A File structure damaged.
La Volume Bit-Map est endommagée.

• GET.EOF - $D1

Parameter Count : 1 byte ($02).
File Reference Number : 1 byte.
EOF : 3 bytes - mis en place par le MLI (LByte, MByte, HByte).

GET.EOF retourne le nombre de bytes lus dans un fichier texte. Le pointeur EOF est positionné sur la fin logique d'un fichier et détermine sa longueur en bytes.

Code d'erreur possible :

$43 Invalid reference number.
Fichier : numéro de référence incorrect.

• SET.BUF - $D2 -

Parameter Count : 1 byte ($02).
File Reference Number : 1 byte.
I/O Buffer Pointer : 2 bytes - LByte, HByte.

SET.BUF met en place le pointeur du tampon mémoire - I/O Buffer Pointer - utilisé par un fichier ouvert et référencé par File Reference Number. Chaque tampon ainsi alloué occupe 1 024 bytes en mémoire centrale : il débute à la limite d'une page mémoire - Boundary Page - et sera protégé contre le débordement éventuel d'un programme. ProDOS gère à cet effet une table d'occupation de la mémoire, placée dans sa page globale, aux adresses $BF58-$BF6F (System Bit-Map). Il marque "occupé", les pages allouées à ses différents tampons. La commande CLOSE ferme le fichier : le tampon est libéré et la System Bit Map mise à jour.

Codes d'erreurs possibles :

$43 Invalid reference number.
Fichier : numéro de référence incorrect.
$56 Bad buffer address.
Adresse du tampon non valide.

• GET.BUF - $D3

Parameter Count : 1 byte ($02).
File Reference Number : 1 byte.
I/O Buffer Pointer : 2 bytes - LByte, HByte, mis en place par le MLI.

GET.BUF retourne la valeur du pointeur d'un tampon - I/O Buffer Pointer - utilisé par le fichier, avec comme numéro de référence, le paramètre attribué à File Reference Number.

Code d'erreur possible :

$43 Invalid reference number.
Fichier : numéro de référence incorrect.

LES COMMANDES DU SYSTEME

Le *Technical Manual* les appelle "System Calls" : elles regroupent la famille des commandes MLI traitant plus particulièrement le matériel de la configuration (Hardware) et la gestion des diverses interruptions.

Liste des commandes du système

$40	ALLOCATE INTERRUPT	Alloue les adresses d'interruptions sauvegardées dans une table de la page globale de PRODOS - MLI IRQ.
$41	DEALLOCATE INTERRUPT	Purge les adresses (codes) d'interruptions de la commande précédente par la gestion d'un numéro de référence - Interrupt Reference Number.
$65	REBOOT	Recharge un autre système à partir d'un sous-programme.
$80	READ BLOCK	Lecture d'un bloc (2 secteurs) à partir d'une disquette et transfert vers un tampon externe au système.
$81	WRITE BLOCK	Ecriture d'un bloc sur une disquette, à partir d'un tampon externe au système.
$82	GET.TIME	Gestion de la date et de l'heure à partir d'une interface carte horloge, Hardware. Les données sont sauvegardées, par la suite, dans la page globale de PRODOS - $BF90-$BF93.

Commandes détaillées

• ALLOCATE.INTERRUPT - $40

Parameter Count : 1 byte ($02).
Interrupt Reference Number : 1 byte - mis en place par le MLI.
Interrupt Code Pointer : 2 bytes - LByte, HByte.

ALLOCATE.INTERRUPT est la plupart du temps associé à un numéro de slot qui contient pour la circonstance une carte interface. Thunderclock, par exemple, permet de gérer la date et l'heure pour le traitement des fichiers d'une disquette.

Cette commande est aussi sollicitée lors d'une interruption en provenance du microprocesseur. La page globale de PRODOS peut contenir à cet effet quatre pointeurs, capables, suivant le programme en cours, d'appeler des sous-programmes de gestion des interruptions.

Le paramètre Interrupt Code Pointer désigne l'adresse du sous-programme. Le pointeur référencé par le paramètre Reference Number sera chargé au préalable à partir de la page globale de PRODOS, par la routine IRQ Handler qui débute à l'adresse $D13A.

Le paramètre Interrupt Reference Count représente un byte qui caractérise l'ordre de départ pour l'exécution des sous-programmes d'interruptions - 1, 2, 3 ou 4. La prise en compte des interruptions est réalisée suivant l'ordre où celles-ci sont placées dans la table : d'abord le pointeur en position 1, puis le pointeur en position 2, et ainsi de suite. Le numéro relatif au paramètre Interrupt Reference Number doit être mis en place par le programme intercepteur de l'interruption, car DEALLOCATE INTERRUPT gère ce paramètre.

Codes d'erreurs possibles :

$25 Interrupt table full.
Table saturée : plus de quatre interruptions formulées.
$53 Invalid system call parameter.
Syntaxe ou nombre incorrect d'un paramètre MLI.

• DEALLOCATE.INTERRUPT - $41

Parameter Count : 1 byte ($01).
Interrupt Reference Number : 1 byte.

DEALLOCATE.INTERRUPT est appelé par la routine d'interruption. Quatre pointeurs pour la gestion des interruptions peuvent être stockés aux adresses $BF80-$BF87 de la page globale de PRODOS. Le numéro de référence Interrupt Reference Number doit être mis en place par le programme intercepteur, DEALLOCATE.INTERRUPT nécessitant ce paramètre.

Code d'erreur possible :

$53 Invalid system call parameter.
Syntaxe ou nombre incorrect d'un paramètre MLI.

• REBOOT - $65

REBOOT ne comporte pas de paramètre. Le pointeur de cette commande est sauvegardé dans la page globale de PRODOS, à l'adresse $BF03. Un sous-programme Move, logé à partir de $EEDB, est associé à REBOOT. En appelant REBOOT, le sous-programme Relocator déplace la zone mémoire du MLI de $D100 vers $1000 et initialise le vecteur RESET à cette même adresse. PRODOS effectue alors un saut à l'adresse $1000 et exécute la routine relogée. REBOOT recherche ensuite le préfixe et tente par la suite de charger, à partir de la disquette, un programme système qui se compose du préfixe SYSTEM.

Appel de REBOOT à partir du langage machine :

```
LDA $C08B
LDA $C08B
JMP $BF03
```

Processus du MLI REBOOT

Pour mieux comprendre le mécanisme de la commande REBOOT, nous allons suivre le déroulement d'un tel appel en le commentant et en l'expliquant. Pour cela, un court sous-programme sera nécessaire : il sera implanté à l'adresse $0300 qui est un vecteur inutilisé par le système.

Mode opératoire :

```
CALL-151 <CR>
* 300 : AD 8B C0 AD 8B C0 4C 03 BF <CR>
<CR> = touche Return
```

Listing du sous-programme :

```
300 -  AD 8B C0   LDA $C08B  Sélectionne la RAM et la Bank 1
303 -  AD 8B C0   LDA $C08B  En lecture et écriture.
306 -  4C 03 BF   JMP $BF03  Appelle REBOOT.
```

A partir du Basic, tapez : CALL 768, qui appelle et exécute le sous-programme implanté à l'adresse $0300 (768 en décimal). ProDOS affiche alors en haut et à gauche de l'écran :

ENTER PREFIX (PRESS RETURN TO ACCEPT)

/PREFIX/

où PREFIX représente le dernier nom du Volume mémorisé.

Remarque : si un nom de Volume a été mémorisé auparavant, il sera affiché comme préfixe par défaut. Deux solutions possibles : taper la touche Return pour accepter - TO ACCEPT -, ou entrer un nouveau nom de Volume comme préfixe.

Tapez le préfixe de la disquette qui se trouve dans le lecteur en service : /PREFIX/ par exemple.

/PREFIX/ <CR>

ProDOS effectue une lecture sur le drive en ligne et vérifie si le nom du Volume correspond à celui que vous avez tapé au clavier. Si la recherche aboutit, l'écran affichera :

ENTER PATHNAME OF NEXT APPLICATION

ce qui vous invite clairement à taper le nom d'un fichier système : PRODOS, BASIC.SYSTEM, ou tout autre nom de fichier système.

PRODOS <CR>

Le fichier PRODOS se trouvant sur la disquette sera chargé en mémoire.

Remarque : si, lors de vos manipulations, l'ordre n'aboutit pas, le système vous le signalera et attendra un nouvel ordre à partir du clavier. Le fichier système doit obligatoirement être du type SYS, donc uniquement "bootable". A chaque recherche infructueuse au niveau de la disquette, le système retourne le texte d'erreur suivant :

FILE/PATH NOT FOUND

Fichier ou chemin non trouvé.

• READ.BLOCK - $80

```
Parameter Count     : 1 byte ($03).
Unit Number         : 1 byte ($50 = slot 6, drive 1).
Data Buffer Pointer : 2 bytes - LByte, HByte.
Block Number        : 2 bytes - LByte, HByte.
```

READ.BLOCK transfère un bloc de données, lues à partir d'une disquette, vers un tampon mémoire pointé par Data Buffer Pointer, dont les paramètres slot et drive sont mémorisés par Unit Number. Le tampon a au minimum une capacité de 512 bytes, car le système lit à chaque fois un minimum de 1 bloc (2 secteurs = 512 bytes).

Block Number désigne le nombre de blocs à lire, et se trouve sauvegardé dans la table des matières de l'entrée d'un fichier de données, à la position relative $13-$14 (19-20). Ce paramètre peut prendre toute valeur de #$0000 à #$0117, soit 0-279 = 280 blocs. La valeur déclarée ne doit pas dépasser 279, car la routine DisK-Driver la teste. Pour rappel, les pistes et secteurs composent la structure physique d'une disquette, tandis que les blocs sont spécifiques au système PRODOS et forment la partie logique. La transcription

logique/ physique est effectuée par la routine MLI Device-Driver. Le chapitre 2 donne de plus amples détails sur l'organisation de la disquette.

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$28 No device connected.
Pas de périphérique en ligne.

• WRITE.BLOCK - $81

Parameter Count : 1 byte ($03).
Unit Number : 1 byte.
Data Buffer Pointer : 2 bytes - LByte, HByte.
Block Number : 2 bytes - LByte, HByte.

WRITE.BLOCK sauvegarde, sur une disquette placée dans le drive spécifié par le paramètre Unit Number, un bloc de données à partir d'un tampon mémoire pointé par Data Buffer Pointer.
Le tampon mémoire a une capacité minimale de 512 bytes. Unit Number nécessite un byte, dont seuls les quatre bits de poids fort sont utilisés pour coder les caractéristiques relatives aux slots et drives en ligne.
Les autres caractéristiques sont semblables à celles de la commande READ.BLOCK.

Représentation du byte Unit Number :

Bits : 7 6 5 4 3 2 1 0
D S S S x x x x D = drive
S = slot
x = inutilisés

Bit 7 à 1 = drive 2
Bit 7 à 0 = drive 1

Bits 6, 5 et 4 représentent le numéro du slot, avec comme valeurs possibles : 1 à 7.

Codes d'erreurs possibles :

$27 I/O error.
Erreur d'entrées/sorties, ou lorsque l'erreur n'est pas reconnue.
$28 No device connected.
Pas de périphérique en ligne.
$2B Disk write protected.
Disque protégé en écriture.

• GET.TIME - $82

Parameter Count : #$00.

Cette routine MLI ne comporte aucun paramètre et ne génère pas d'erreur. Le sous-programme d'interception est mis en place par la ROM résidente d'une carte interface horloge du type Thunderclock qui loge dans un slot libre de l'Apple. Le but recherché est de sauvegarder la date et l'heure actuelles, lors de la création ou de la modification effectuée sur un fichier sauvegardé sur une disquette.

Lors de la création d'un fichier, la date et l'heure sont sauvegardées dans la table des matières, à la position relative $18-$1B (24-27) de l'entrée correspondante. La position relative $21-$24 (33-36) contiendra la date et l'heure de la dernière modification.

Pour ceux qui ne possèdent pas la carte horloge, un programme figure en annexe 10 et traite manuellement la fonction date de PRODOS - Année/ Mois et Jour.

Carte horloge Thunderclock :

La commande MLI GET.TIME comporte un saut interne à la page globale de PRODOS, sous la forme : JSR $BF06, où se trouve un RTS - retour au sous-programme appelant. Lorsque ProDOS est "booté", le système effectue une analyse des périphériques en ligne : si une carte horloge compatible avec le système est en place, le RTS à l'adresse $BF06 sera modifié en un JMP $F142. A cette adresse se trouve implantée la routine, GET TIME, de lecture de la date et de l'heure de l'interface horloge en place. Le RTS qui se trouve en fin de la routine GET TIME - adresse $F1AB - rebranche normalement au MLI : soit à l'adresse $D078 qui est la sortie MLI normale (MLI-EXIT), soit à la routine Command-Handler lors de l'appel des commandes $C0-$D3 (CREATE, CLOSE, etc.).

Format date et heure de la page globale :

BF90- DATE HEX 0000
BF92- HEURE HEX 0000

Bits représentatifs de la 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

Bits représentatifs de l'heure :

Bits : F E D C B A 9 8 7 6 5 4 3 2 1 0
x x x H H H H H x x M M M M M M
H = Heure
M = Minute
x = Inutilisés.

Remarque : si la carte horloge n'est pas compatible, elle sera purement et simplement ignorée par le système.

CHAPITRE 7

TRAITEMENT DES INTERRUPTIONS SOUS ProDOS-IRQ, NMI, RESET ET GET.TIME

INTERRUPTIONS DU MICROPROCESSEUR

"Interrupt" modes

Le microprocesseur permet de traiter trois types d'interruptions : IRQ, NMI et RESET.
Pour chaque signal d'interruption, il existe une ligne particulière d'accès au processeur : si le signal véhiculé est à un niveau bas (0), le processeur reçoit une demande d'interruption ; par contre, si le signal est à un niveau haut (1), rien ne se passe.

PRISE EN CHARGE DES INTERRUPTIONS

Le microprocesseur, de par sa structure interne, permet de traiter et de gérer une demande d'interruption, à condition que le bit d'inhibition (bit 2) du registre d'état PS soit mis à zéro et autorise ainsi l'interruption.

Déroulement type d'une interruption :

- l'instruction en cours est menée à terme ;
- l'octet de poids fort du compteur ordinal est empilé (PCH) ;
- l'octet de poids faible du compteur ordinal est empilé (PCL) ;
- le contenu du registre d'état PS est empilé ;
- le bit 2, inhibition des interruptions, est mis à 1 ;
- exécution de l'instruction qui se trouve à l'adresse $FFFE de la ROM Moniteur, ou saut indirect à l'adresse contenue dans les octets $FFFF, $FFFE.

Vecteurs d'interruption

Pour un Apple II, les vecteurs d'interruptions sont logés aux adresses $FFFA-$$FFFF de la ROM du Moniteur, et sont définis par le concepteur pour chaque type de machine. C'est ainsi que les pointeurs ne seront pas les mêmes pour un Apple IIe ou un Apple IIc, à cause de certaines différences technologiques internes.

Vecteurs pour un Apple IIe :

FFFA - FB 03 Pointeur $03FB - NMI.
FFFC - 62 FA Pointeur $FA62 - RESET.
FFFE - 40 FA Pointeur $FA40 - IRQ.

Vecteurs pour un Apple IIc :

FFFA - FB 03 Pointeur $03FB - NMI.
FFFC - 62 FA Pointeur $FA62 - RESET.
FFFE - 03 C8 Pointeur $C803 - IRQ.

La différence entre ces deux Apple réside dans le vecteur $FFFE-$FFFF, qui contient le pointeur $FA40 pour un IIe, et $C803 pour un IIc. Cette situation ne change en rien le déroulement et la gestion des interruptions, et ne perturbe donc pas la méthode de programmation adoptée jusqu'à présent.

Caractéristiques particulières à l'Apple IIc

Pour le IIc, qui est le modèle le plus récent de la gamme des Apple II, le sous-programme MOVEIRQ, logé à l'adresse $C8A2, transfère le contenu de $FFFE-$FFFF (vecteur IRQ) en mémoire vive. C'est ainsi que toute demande d'interruption est prise en charge par NEWIRQ qui débute à partir de l'adresse $C803 et va jusqu'à $C89F.

Les interruptions provoquées par la souris sont prises en compte par MOUSEINT qui débute à l'adresse $C4D5.
Une interruption déclenchée par le clavier est prise en compte par ACIAINT qui se trouve à l'adresse $C900.
Les adresses $03FE-$03FF de la page 3 contiennent un pointeur, qui est une adresse de saut redéfinissable par le programmeur. Ce vecteur est réservé au système d'exploitation ProDOS, et contient la valeur $BFEB, après le boot du système.

Caractéristiques communes aux Apple II

La société Apple spécifie, dans sa documentation, que ProDOS peut gérer quatre interruptions à l'aide de drivers ou routines mises en place par le système. Parmi celles-ci, une gestion de la date et heure de création ou de modification des fichiers d'une disquette (à condition que votre Apple soit équipé d'une carte horloge de la série Thunderclock). A chaque enregistrement d'un nouveau fichier, la date sera automatiquement estampillée sur la disquette, ce qui donnera un certain cachet à l'ensemble.

Pour intercepter un signal d'interruption émanant d'une carte horloge, ProDOS place au préalable en mémoire une petite routine qui a pour rôle de lire la date et l'heure dans la carte interface, et de sauvegarder les valeurs dans la page globale de PRODOS aux adresses $BF90-$BF93.

Certains périphériques standard de la série Apple, comme par exemple le lecteur de disquettes du type Disk II, gèrent au besoin leurs propres signaux d'interruptions.

Il est à noter que le système ProDOS ne permet pas une gestion des interruptions lors de l'accès de la tête de lecture aux pistes/secteurs, comme on pourrait l'attendre après avoir lu le *Technical Manual*. En effet, le système d'exploitation n'autorise pas ce genre d'interruption, le temps d'accès pour la lecture et l'écriture d'une piste physique de la disquette étant extrêmement critique.

Du point de vue technique, la gestion actuelle des interruptions par ProDOS est identique à celle du DOS 3.3. Au moment où le système attaque le secteur et rencontre l'adresse standard du marqueur de début de zone "D5 AA 96", le processeur prend en compte l'interruption par une instruction SEI qui est l'inhibition des interruptions. A ce stade, ProDOS ne se différencie pas du DOS 3.3.

LES INTERRUPTIONS EN GENERAL

Les interruptions et le microprocesseur

Le microprocesseur permet de traiter trois types d'interruptions :

NMI Non Maskable Interrupt : ce sont les interruptions non masquables.
RESET Monitor Interrupt : ce type d'interruption, gérée à partir de l'adresse $FA62, renvoie le système dans le Moniteur.
IRQ Interrupt Request : ce sont les interruptions masquables, dont le BReaK fait partie.

Pour chaque signal d'interruption mentionné plus haut, il existe une ligne particulière d'accès au processeur : si le signal véhiculé est à un niveau bas (0), le processeur reçoit une demande d'interruption ; par contre, si le signal est à un niveau haut (1), rien ne se passe.

Prise en charge des interruptions

Le microprocesseur est réceptif lors d'une demande d'interruption - IRQ actif -, à condition que le bit 2, inhibition des interruptions, du registre d'état PS soit mis à zéro, autorisant ainsi l'interruption.

Les différents types d'interruptions

Processus relatif aux interruptions :

- L'instruction en cours est menée à terme.

- Si le signal provient d'une ligne IRQ, le processeur teste un bit significatif - bit 2 - du registre d'état PS, pour vérifier son état. Si le bit est positionné à 0, l'interruption vient d'être menée à terme, et la ligne sera libérée ; si le bit est à 1, la demande n'est pas prise en compte et le déroulement des opérations en cours peut reprendre.

- Si le signal provient d'une ligne IRQ non masquée, ou si c'est un signal NMI, le processeur sauvegarde le registre d'état PS, et le compteur ordinal PC sur la pile. RESET n'utilise pas ce processus.

- Le processeur lit une adresse 16 bits à partir du vecteur des interruptions - $FFFA-$FFFF - et la copie dans le compteur ordinal ; l'exécution du programme en cours se poursuit à l'adresse lue.

Adresses d'interruptions :

$FFFA - $FFFB	pour une interruption	NMI	JMP ($FFFA)
$FFFC - $FFFD	pour une interruption	RESET	JMP ($FFFC)
$FFFE - $FFFF	pour une interruption	IRQ	JMP ($FFFE)

Processus d'une interruption type

Déroulement :

- l'interruption en cours est menée à terme ;
- l'octet de poids fort du compteur ordinal est empilé - PCH ;
- l'octet de poids faible du compteur ordinal est empilé - PCL ;
- le contenu du registre d'état PS est empilé ;
- le bit 2, inhibition des interruptions, est mis à 1 ;
- exécution de l'instruction qui se trouve à l'adresse $FFFE de la ROM Moniteur, ou saut indirect à l'adresse contenue dans les octets $FFFF, $FFFE.

INTERRUPTIONS - NMI, RESET ET IRQ

NMI - Interrupt

INTERRUPTION A PARTIR DU MONITEUR

ROM-Monitor commutée :

La ROM du Moniteur, adresses $FFFA-$FFFF, contient les pointeurs des vecteurs d'appel pour gérer la demande d'interruption adressée au processeur. Le premier pointeur 16 bits contenu dans $FFFA-$FFFB est réservé aux interruptions du type NMI. Pour un Apple IIe ou IIc, les adresses $FFFA-$FFFB contiennent : FB 03 = $03FB. Lors du boot de ProDOS, $30FB contient l'adresse de saut au Moniteur.

Vecteurs du NMI :

```
03FB -  4C 59 FF    JMP  $FF59

FF59 -  20 84 FE    JSR  $FE84    SETNORM. Mode normal
FF5C -  20 2F FB    JSR  $FB2F    INIT. Mode texte
FF5F -  20 93 FE    JRS  $FE93    SETVID. Mode PR#0
FF62 -  20 89 FE    JSR  $FE89    SETKBD. Mode IN#0
FF65 -  D8          CLD
FF66 -  20 3A FF    JSR  $FF3A    BELL. Sonnette
```

Après l'exécution de la routine implantée à partir de $FF59, le système effectue un saut dans le Moniteur, et affiche "*" comme prompt : l'étoile annonce à l'utilisateur qu'il a la main.

INTERRUPTION A PARTIR DE LA CARTE LANGAGE

Carte langage commutée :

La copie de l'adresse $03FB se trouve dans la carte langage, aux adresses $FFFA-$FFFB - FB 03 - pour être utilisée par le MLI lorsqu'une interruption NMI survient.
Comme la carte langage n'est pas protégée en lecture lors d'une interruption du type NMI, IRQ ou RESET, le processeur se branche à $03FB, puis effectue un saut à $FF59 et exécute le sous-programmme.
Si le pseudo-disque - RAM-Disk - est utilisé avec la carte auxiliaire 64 ko ou avec un Apple IIc, une interruption NMI plante le système, et renvoie l'utilisateur au Moniteur - prompt *.
L'instruction CALL 976 (3D0G) reconnecte le système Basic en effectuant un démarrage à chaud, tandis que CALL 39447 appelle la routine CONNECT qui se trouve à l'adresse $9A17.

RESET - Interrupt

INTERRUPTION A PARTIR DU MONITEUR

ROM-Monitor commutée :

Le point d'entrée de RESET se trouve à l'adresse $FA62. La copie de cette adresse se trouve au vecteur $FFFC-$FFFD - 62 FA - de la ROM du Moniteur.

RESET avec une ROM-AUTOSTART pour un Apple IIe

```
FA62 -  D8                          Annulation du mode décimal.
FA63 -  20 84 FE    JSR  $FE84      SETNORM. Mode normal.
FA66 -  20 2F FB    JSR  $FB2F      INIT. Mode texte.
FA69 -  20 93 FE    JSR  $FE93      SETKBD. Mode IN#0.
FA6C -  20 89 FE    JSR  $FE89      SETVID. Mode PR#0.
FA6F -  AD 58 C0    LDA  $C058      Annonciateur 0 à 1 - Read.
```

```
FA72 - AD 5A C0   LDA  $C05A    Annonciateur 0 à 1 - Write.
FA75 - A0 05      LDY  #$05     Initialise le registre Y à la valeur #$05.
FA77 - 20 B4 FB   JSR  $FBB4    Gestion du curseur - avance 5 positions.
FA7B - AD FF CF   LDA  $CFFF    CLRROM. Désactive les sous-programmes placés
                                entre $C800 à $CFFF.
FA7E - 2C 10 C0   BIT  $C010    KBDSTRB. Echantillonnage du clavier.
FA81 - D8         CLD
FA82 - 20 3A FF   JSR  $FF3A    BELL. Sonnette.
FA85 - AD F3 03   LDA  $03F3    RESET-Handler. Charge le pointeur HByte.
FA88 - 49 A5      EOR  #$A5     PRWRUP Byte.
FA8A - CD F4 03   CMP  $03F4
FA8D - D0 17      BNE  $FAA6    Démarrage à froid. Reboote le système
FA8F - AD F2 03   LDA  $03F2    RESET-Handler. Charge le pointeur LByte
FA92 - D0 0F      BNE  $FAA3    Si <> de 0, démarrage à chaud.
FA94 - A9 E0      LDA  #$E0
FA96 - CD F3 03   CMP  $03F3    Est-ce le vecteur Basic ? - $E000.
FA99 - D0 08      BNE  $FAA3    Non, démarrage à chaud.
FA9B - A0 03      LDY  #$03     Fixe RESET sur le vecteur Basic à froid.
FA9D - 8C F2 03   STY  $03F2    Démarrage en $E003 - Basic à chaud.
FAA0 - 4C 00 E0   JMP  $E000    Basic à froid - COLDSTART.

FAA3 - 6C F2 03   JMP  ($03F2)  Warmstart. Saut au Handler.

FAA6 - 20 60 FB   JSR  $FB60    COLDSTART. Entrée du démarrage à froid.
```

PROCESSUS DU RESET

Carte langage commutée :

L'interruption à partir d'une commande MLI PRODOS (venant donc 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, le vecteur $FFFC-$FFFD contient le pointeur $FFCB ; ce dernier, Moniteur commuté "on", contient un RTS - code 60.

```
:::: ::::::::  ::::::::::
FFC8- 8D 82 C0    STA $C082     ROMON. Normalement RTS à partir du Moniteur.
                  STA $C082     Carte langage commutée.
```

- RESET-Handler Entry
  Moniteur RESET = $FA62

```
FFCB - AD D7 FF   LDA  $FFD7    Charge le pointeur HByte,
FFCE - 48         PHA           et l'empile.
FFCF - AD D6 FF   LDA  $FFD6    Charge le pointeur LByte,
FFD2 - 48         PHA           et l'empile. Adresse de retour - $FA61.
FFD3 - 4C C8 FF   JMP $FFC8     Commute la ROM Moniteur.
FFD6 - 61 FA                    Pointeur $FA61 - 61 FA.
```


Processus :

A l'adresse $FFCB du Moniteur se trouve un RTS - code 60. Conséquence : le pointeur contenu aux adresses $FFD6,$FFD7 sera utilisé par le processeur pour effectuer un retour à l'adresse $FA62. L'adresse de retour, dépilée après une instruction RTS, sera augmentée de 1 : $FA61 + #$01 = $FA62 ; cela est une caractéristique propre au microprocesseur.

IRQ - Interrupt

Cette interruption n'est autorisée par le microprocesseur que si le bit d'inhibition (bit 2) du registre d'état PS est à 0. (Situation provoquée par l'instruction CLI du processeur.)

IRQ NTERRUPT
INTERRUPTION A PARTIR DU MONITEUR

Le vecteur d'interruption aux adresses $FFFE-$FFFF contient le pointeur $FA40 - 40 FA - pour un Apple IIe, et $C803 - 03 C8 - pour un IIc. Le listing qui suit débute à l'entrée de la prise en charge de l'interruption IRQ, et donne les explications nécessaires à sa compréhension.

IRQ Interrupt - pour un Apple IIe.

```
FA40 - 85 45      STA  $45      Sauvegarde de l'accumulateur.
FA42 - 68         PLA           Dépile le registre d'état PS.
FA43 - 0A         ASL           Bit 4 positionné à 1, si IRQ est transmis
FA44 - 0A         ASL           par une instruction BReaK - 00.
FA47 - 30 03      BMI  $FA4C    BReak avec affichage des registres.
FA49 - 6C 03 FE   JMP ($03FE)   Adresse IRQ-Handler.
```

- ProDOS positionne IRQ-Handler à la valeur $BFEB.

```
03FE - EB                       LByte du pointeur IRQ-Handler.
03FF - BF                       HByte du pointeur IRQ-Handler.

BFEB - AD 8B C0   LDA  $C08B    BankSelect. Bank 1 autorisation de lecture,
BFEE - AD 8B C0   LDA  $C08B    + écriture.
BFF1 - 4C 3A D1   JMP  $D13A    IRQ-Handler à partir du MLI.
```

Après avoir sélectionné la Bank 1 de la carte langage en lecture et écriture, le système se branche à l'entrée de la routine IRQ-Handler, adresse $D13A. Cette opération s'effectue à partir de l'adresse $BFF1. Le rôle principal de la Bankselect est d'aiguiller le système vers la routine de gestion, selon le type d'interruption rencontrée. La page globale de PRODOS se réserve une zone d'adresses - $BF80-$BF87 - où quatre pointeurs peuvent être placés par l'intermédiaire d'un programme, pour traiter par la suite les différentes interruptions.

ProDOS teste deux situations : tout d'abord, il vérifie si la source qui a déclenché l'interruption est bien du type IRQ ; puis, il vérifie le byte de poids fort de chacun des pointeurs qui se trouvent aux adresses $BF80-$BF87.
Si le test est négatif, c'est-à-dire si aucune des quatre adresses de gestion d'interruption n'est en place, le système se branche en $BF0C, par l'intermédiaire de l'adresse $D19A. En $BF0C se trouve un JMP $D1E6, qui appelle le sous-programme DEATH 1, et affiche le texte suivant :

INSERT SYSTEM DISK AND RESTART -ERR xx

Si par contre le système trouve une adresse de gestion pour l'interruption en cours, la routine implantée à partir de $D13A prend fin dans la page globale de PRODOS, en effectuant un saut en $BFD0 (voir le contenu de l'adresse $D1CB).

- IRQ-Handler Entry.

D13A -	A5 45	LDA	$45	Charge la valeur de l'adresse $45 dans l'accumulateur - Registre A.
D13C -	8D 88 BF	STA	$BF88	INTAREG. Sauvegarde de l'accumulateur.
D13F -	8E 89 BF	STX	$BF89	INTXREG. Sauvegarde de X dans la page globale.
D142 -	8C 8A BF	STY	$BF8A	INTYREG. Sauvegarde de Y dans la page globale.
D145 -	BA	TSX		Copie du pointeur de pile dans X.
D146 -	8E 8B BF	STX	$BF8B	INTSREG. Sauvegarde du pointeur de pile.
D149 -	68	PLA		Dépile la valeur du registre d'état PS.
D14A -	8D 8C BF	STA	$BF8C	INTPREG. Sauvegarde du registre d'état PS.
D14D -	68	PLA		Dépile l'adresse de retour - LByte.
D14E -	8D 8E BF	STA	$BF8E	INTADR. Sauvegarde de l'adresse de retour - LByte.
D151 -	68	PLA		Dépile l'adresse de retour - HByte.
D152 -	8D 8F BF	STA	$BF8F	Sauvegarde de l'adresse de retour - HByte.
D155 -	9A	TXS		Positionne le pointeur de pile.
D156 -	AD F8 07	LDA	$07F8	MSLOT. Numéro de l'interface, HByte.
D159 -	8D C4 D1	STA	$D1C4	Sauvegarde du numéro de l'interface.
D15C -	BA	TSX		
D15D -	30 09	BMI	$D168	Saut, si le pointeur de pile > #$7F, sinon,
D15F -	A0 0F	LDY	#$0F	initialise le registre Y à la valeur 16,
D161-	68	PLA		et dépile 16 valeurs.
D162 -	99 AF F0	STA	$F0AF,Y	Sauvegarde des valeurs dépilées.
D165 -	88	DEY		Décrémente Y d'une unité,
D166 -	10 F9	BPL	$D161	et continue à dépiler.
D168 -	A2 FA	LDX	#$FA	
D16A -	B5 00	LDA	$00,X	Sauvegarde des valeurs de la page zéro,
D16C -	9D A5 EF	STA	$EFA5,X	de #$FA à #$FF.
D16F -	E8	INX		Incrémente le registre X,
D170 -	D0 F8	BNE	$D16A	et continue avec la prochaine valeur.

- Interrupt Vectors.

D172 -	AD 81 BF	LDA	$BF81	Interrupt 1 : charge HByte de l'adresse de saut,
D17 5-	F0 05	BEQ	$D17C	adresse vide, charge le vecteur suivant.
D177 -	20 CE D1	JSR	$D1CE	Vecteur utilisé : appel du sous-programme.
D17A -	90 23	BCC	$D19F	Restaure la page zéro.
D17C -	AD 83 BF	LDA	$BF83	Interrupt 2 : charge HByte de l'adresse de saut,
D17F -	F0 05	BEQ	$D186	adresse vide, charge le vecteur suivant.
D181 -	20 D1 D1	JSR	$D1D1	Vecteur utilisé : appel du sous-programme.
D184 -	90 19	BCC	$D19F	Restaure la page zéro.
D186 -	AD 85 BF	LDA	$BF85	Interrupt 3 : charge HByte de l'adresse de saut,
D189 -	F0 05	BEQ	$D190	adresse vide, charge le vecteur suivant.
D18B -	20 D4 D1	JSR	$D1D4	Vecteur utilisé : appel du sous-programme.
D18E -	90 0F	BCC	$D19F	Restaure la page zéro.
D190 -	AD 87 BF	LDA	$BF87	Interrupt 4: charge HByte de l'adresse de saut,
D193 -	F0 05	BEQ	$D19A	adresse vide, charge le vecteur suivant.
D195 -	20 D7 D1	JSR	$D1D7	Vecteur utilisé : appel du sous-programme.
D198 -	90 05	BCC	$D19F	Restaure la page zéro.

- Error Handler.

D19A -	A9 01	LDA	#$01	Aucun des pointeurs n'est en place dans la page globale de PRODOS, aux adresses $BF80-$BF87. Charge l'accumulateur avec la valeur #$01 qui sera utilisée comme drapeau - Flag - pour le code d'erreur.
D19C -	20 0C BF	JSR	$BF0C	SYSTDEATH. Appel de la routine d'erreur : normalement contient un saut à $D1E6.

- Restaure la page zéro.

D19F -	A2 FA	LDX	#$FA	Initialise le registre X comme compteur.
D1A1-	BD A5 EF	LDA	$EFA5,X	Charge les valeurs des adresses indexées,
D1A4 -	95 00	STA	$00,X	et les sauvegarde dans la page zéro.
D1A6 -	E8	INX		Incrémente de 1 le registre X,
D1A7 -	D0 F8	BNE	$D1A1	et continue la sauvegarde.
D1A9 -	AE 8B BF	LDX	$BF8B	INTSREG. Charge le pointeur de pile,
D1AC -	30 0B	BMI	$D1D9	y a-t-il assez de place ? oui,
D1AE -	A0 00	LDY	#$00	sinon, empile les valeurs sauvegardées au préalable : voir à partir de $D161.

```
D1B0 - B9 AF F0    LDA  $F0AF,Y   Charge les valeurs des adresses indexées,
D1B3 - 48          PHA            et les empile.
D1B4 - C8          INY            Incrémente de 1 le registre Y,
D1B5 - C0 10       CPY  #$10      et vérifie si les 16 valeurs ont été
                                  sauvegardées.
D1B7 - D0 F7       BNE  $D1B0     Continue d'empiler.
D1B9 - AC 8A BF    LDY  $BF8A     INTYREG. Charge la valeur du registre Y.
D1BC - AE 89 BF    LDX  $BF89     INTXREG. Charge la valeur du registre X.
D1BF - AD FF CF    LDA  $CFFF     CLRROM. Déconnecte les sous-programmes
                                  des ROM, entre $C800-$CFFF.
D1C2 - AD 00 C1    LDA  $C100     REACTIV CARD. Charge le numéro du
                                  périphérique HByte - $D1C4 de l'adresse
                                  $D159 -,
D1C5 - AD C4 D1    LDA  $D1C4     et restaure MSLOT - $07F8.
D1C8 - 8D F8 07    STA  $07F8     INTEXIT. Sortie de la routine via la page
D1CB - 4C D0 BF    JMP  $BFD0     globale de PRODOS.
```

- Appel des sous-programmes, suite à une interruption.

Ces sous-programmes sont uniquement sollicités par un JSR à partir des adresses $D177, $D181, $D18B et $D195.

```
D1CE -  6C 80 BF   JMP  ($BF80)   Exécution de la routine Interrupt 1
D1D1 -  6C 82 BF   JMP  ($BF82)   Exécution de la routine Interrupt 2
D1D4 -  6C 84 BF   JMP  ($BF84)   Exécution de la routine Interrupt 3
D1D7 -  6C 86 BF   JMP  ($BF86)   Exécution de la routine Interrupt 4
```

- SET SYSERR.

Entrée par un JMP venant de l'adresse $BF09.

```
D1DA -  8D 0F BF   STA  $BF0F     Sauvegarde du code de l'erreur.
D1DD -  68         PLA            Dépile l'adresse de retour ; LByte.
D1DE -  68         PLA            HByte.
D1DF -  AD 0F BF   LDA  $BF0F     Charge le code de l'erreur.
D1E2 -  38         SEC            Utilisé comme indicateur de l'erreur.
D1E3 -  60         RTS            Retour : normalement à $D078.
```

- System DEATH Entry 2.

L'entrée du vecteur $D1E4 est appelée par la page globale de PRODOS, à savoir : REBOOT - $BF03 - et SYSDEATH2 - $BFF6.

```
D1E4 -  A9 00      LDA #$00       Donne la valeur #$00 au code d'erreur.
```


- System DEATH Entry 1.

A l'entrée de la routine, l'accumulateur contient le code de l'erreur rencontrée. L'appel vient de la page globale de PRODOS, à partir de l'adresse $BF0C.

```
D1E6 -  AA         TAX            Copie le code d'erreur dans X.
D1E 7-  8D 0C C0   STA  $C00C     Déconnecte la carte 80 colonnes - IIe et IIc.
D1EA -  AD 51 C0   LDA  $C051     Place le système en mode texte,
D1ED -  AD 53 C0   LDA  $C053     pleine page,
D1F0 -  AD 54 C0   LDA  $C054     page 1,
D1F3 -  AD 56 C0   LDA  $C056     et en basse résolution - LORES.
D1F6 -  A0 27      LDY  #$27      Initialise le registre Y, comme compteur
D1F8 -  A9 A0      LDA  #$A0      et charge la valeur du nombre d'espaces.
D1FA -  99 50 07   STA  $0750,Y   Affiche à l'écran le texte :
                                  INSERT SYSTEM DISK AND RESTART
D1FD -  B9 DE EF   LDA  $EFDE,Y   Charge les caractères stockés dans la
                                  carte langage,
D200 -  99 D0 07   STA  $07D0,Y   continue l'affichage des caractères à l'écran,
D203 -  88         DEY            décrémente d'une unité le compteur Y,
D204 -  10 F2      BPL  $D1F8     et continue l'affichage.
D206 -  8A         TXA            Recopie le code d'erreur dans A,
D207 -  F0 30      BEQ  $D239     si 00, le code ne sera pas affiché : -ERRxx.
D209 -  A9 AD      LDA  #$AD      Charge le code ASCII de "-",
D20B -  8D F1 07   STA  $07F1     et l'affiche à l'écran.
D20E -  A9 C5      LDA  #$C5      Charge le code ASCI de "E",
D210 -  8D F2 07   STA  $07F2     et l'affiche à l'écran.
D213 -  A9 D2      LDA  #$D2      Charge le code ASCII de "R",
D215 -  8D F3 07   STA  $07F3     et l'affiche deux fois à l'écran.
D218 -  8D F4 07   STA  $07F4     ==> R
                                  ==> -ERR
D21B -  8A         TXA            Code vers l'accumulateur,
D21C -  4A         LSR            et décale les bits, 4 fois vers la droite,
D21D -  4A         LSR            ce qui équivaut à ne garder que les 4 bits
D21E -  4A         LSR            de poids fort.
D21F -  4A         LSR
D220 -  09 B0      ORA  #$B0      Effectue un OU inclusif avec #$B0 (176),
D222 -  C9 BA      CMP  #$BA      et vérifie si le résultat est supérieur à 9.
D224 -  90 02      BCC  $D228
D226 -  69 06      ADC  #$06      Pour les valeurs supérieures à 9 => "A"-"F",
D228 -  8B F6 07   STA  $07F6     affiche la valeur correcte à l'écran.
D22B -  8A         TXA
D22C -  29 0F      AND  #$0F      Valeur des 4 bits de faible poids.
D22E -  09 B0      ORA  #$B0
D230 -  C9 BA      CMP  #$BA
D232 -  90 02      BCC  $D236
D234-   69 06      ADC  #$06      Correction pour les valeurs supérieures à 9,
D236-   8D F7 07   STA  $07F7     et affichage du résultat correct.
D239-   4C 39 D2   JMP  $D239     Attente d'un RESET, car le programme
                                  boucle sur lui-même - JMP $D239.
```

- INTEXIT.

Sortie de la routine de gestion d'interruption, lorsque ProDOS trouve un pointeur de sous-programme aux adresses $BF80-$BF87.

```
BFD0 - AD 8D BF   LDA  $BF8D   INTBANK1. Charge le statut de la ROM.
BFD3 - F0 0D      BEQ  $BFE2   #$00 = bank 1 connectée : elle le reste.
BFD5 - 30 08      BMI  $BFDF   #$FF = bank 2 à commuter.
BFD7 - 4A         LSR
BFD8 - 90 0D      BCC  $BFE7   Sans signification, valeur aléatoire
BFDA - AD 82 C0   LDA  $C082   #$01= ROMON ; IRQ à partir du Moniteur.
BFDD - B0 08      BCS  $BFE7   Sans signification, valeur aléatoire.
BFDF - AD 83 C0   LDA  $C083   Commute la bank 2 (protégée contre
                               l'écriture).
BFE2 - A9 01      LDA  #$01    Initialise l'accumulateur à la valeur #$01,
BFE4 - 8D 8D BF   STA  $BF8D   et fixe le mode de la ROM - ROM ou
                               Moniteur.
BFE7 - AD 88 BF   LDA  $BF88   Recharge la valeur de l'accumulateur à partir
                               de INTAREG.
BFEA - 40         RTI          Retour de l'interruption.
```

IRQ INTERRUPT
INTERRUPTION A PARTIR DE LA CARTE LANGAGE

Carte langage commutée :

Dans la carte langage, au vecteur $FFFE-$FFFF, se trouve le pointeur de l'adresse $FF9B. C'est le point d'entrée de IRQ-Handler, sollicité à partir de la carte langage.
ProDOS vérifie si l'interruption du type IRQ est issue d'une instruction BReaK (code 00). Dans l'affirmative, $FA42 sera utilisée comme adresse de retour, et sauvegardée au sommet de la pile, dans l'ordre HByte, LByte. Après commutation de la ROM, par ROMON, suite au RTS introduit par IRQ, le système branche à l'entrée Moniteur - $FA42. Une vérification préliminaire teste si l'interruption IRQ provient ou non d'un BReaK (code 00).
Si l'interruption ne provient pas d'un BReaK, mais d'un IRQ réel, le système consulte une série d'adresses pour connaître la bank commutée. C'est ainsi que la page globale de PRODOS comporte deux adresses spécialement mises à disposition : INTBANK1, adresse $BF8D, renseigne sur la Bank 1 ; INTBANK2, adresse $BF52, renseigne sur la Bank 2. Cette manœuvre explique enfin pourquoi à l'adresse $BFD0 - INTEXIT - se trouve toujours chargée la valeur #$01, qui est un drapeau - Flag - de INTBANK. La ROM du Moniteur ne comporte pas de routine semblable pour le test de la bank. Par ailleurs, à partir de la carte langage, la valeur de INTBANK1 est continuellement renouvelée par l'intermédiaire de l'adresse $FFB1. En fin de boot d'une disquette ProDOS, le vecteur INTBANK1 contient la valeur #$01 : astucieux, non ?

Avant que le mode IRQ-Handler du Moniteur soit actif (donc qu'il pointe l'adresse $FA42 comme adresse de retour) certaines valeurs sont sauvegardées sur la pile : la valeur #$04 comme drapeau - Flag - et l'adresse $BF50 - BF 50 - comme adresse de retour de

l'interruption vers la page globale. Ce pointeur de retour est placé sur la pile dans l'ordre HByte - BF -, LByte - 50 -, pour être, par la suite, dépilé dans le bon sens : LByte, HByte.

Le RTI de l'adresse $BFEA, de la routine INTEXIT - listée précédemment - amène la reflexion suivante : un RTI code 40 se différencie d'un RTS code 60, sur deux points.

Valeur du pointeur après un RTI :

1- le processeur récupère, sur la pile, le contenu du registre d'état PS, sauvegardé au préalable par le mécanisme d'interruption ;
2- L'adresse de retour - compteur ordinal - est elle aussi dépilée, mais non augmentée de 1 comme avec RTS.

Après une interruption IRQ venant de la carte langage, la situation à la sortie de la routine INTEXIT - après le RTI - sera la suivante : le registre d'état PS aura comme valeur #$04, fixée à l'adresse $FFBF, ce qui masque IRQ ; l'exécution se poursuivra dans la page globale de PRODOS, avec un saut à l'adresse $BF50. Ces différents vecteurs ont été mis en place à partir de l'adresse $FFB9 de la routine IRQ-Handler, logée dans la carte langage.

- IRQ-Handler.

```
FF9B - 48         PHA          Sauvegarde de l'accumulateur.
FF9C - A5 45      LDA  $45     L'accumulateur contient le pointeur RWTB.
FF9E - 8D 56 BF   STA  $BF56   INTACC. Sauvegarde du pointeur dans la page
                               globale de PRODOS.
FFA1 - 68         PLA          Dépile vers A la valeur initiale, contenue au
                               moment de l'interruption IRQ,
FFA2 - 85 45      STA  $45     et la sauvegarde.
FFA4 - 68         PLA
FFA5 - 48         PHA          Sauvegarde du registre d'état PS.
FFA6 - 29 10      AND #$10     Teste si IRQ provient d'un BReaK,
FFA8 - D0 18      BNE $FFC2    oui, alors branche à la routine BReaK.
FFAA - AD 00 D0   LDA $D000    Sinon, teste quelle bank est commutée.
FFAD - 49 D8      EOR #$D8     BANKID pour la bank 1.
FFAF - F0 02      BEQ $FFB3
FFB1 - A9 FF      LDA #$FF
FFB3 - 8D 8D BF   STA $BF8D    INBANK1. #$00 bank 1, ou #$FF bank 2,
FFB6 - 8D 57 BF   STA $BF57    sauvegarde :
                               $BF56-$BF57
                               - $45 pour LByte
                               - #$00 ou #$FF pour HByte.
                               Pointeurs possibles :
                               HByte, LByte
                               - $FF, contenu de $45
                               - $00, contenu de $45.
FFB9 - A9 BF      LDA  #$BF    Charge le pointeur HByte
```

```
FFBB - 48       PHA          et le place au sommet de la pile.
FFBC - A9 50    LDA #$50     Charge le pointeur LByte
FFBE - 48       PHA          et le place au sommet de la pile. Fixe
                             l'adresse $BF50 comme adresse RTI.
FFBF - A9 04    LDA  #$04    Charge le drapeau du bit d'inhibition (bit 2)
                             du registre d'état PS - IRQ masqué -,
FFC1- 48        PHA          et le sauvegarde au sommet de la pile.
FFC2 - A9 FA    LDA  # $FA   Charge le pointeur HByte,
FFC4 - 48       PHA          et le sauvegarde au sommet de la pile.
FFC5 - A9 41    LDA  #$41    Charge le pointeur LByte,
FFC7 - 48       PHA          et le sauvegarde au sommet de la pile
                             Moniteur BReaK, adresse de retour :
                             - $FA41 + #$01 = $FA42 (à cause du RTS : +1).
FFC 8- 8D 82 C0 STA  $C082   ROMON. Contient un RTS lorsque l'appel
                             provient du Moniteur.
```

- LANGIRQ.

Entrée de la routine, après une interruption IRQ appelée depuis la carte langage, et à la suite d'une instruction RTI détectée à l'adresse $BFEA de la page globale de PRODOS.

```
BF50 - 8D 8B C0 STA $C08B    BANKSELECT. Bank 1 : protégée contre
                             l'écriture.
BF53 - 4C D8 FF JMP  $FFD8   Saut dans la carte langage.
```

- LANGINTEXIT.

Entrée de la carte langage : provient d'une routine sollicitée par une interruption IRQ, avec, comme origine, la page globale de PRODOS, après la sélection en lecture ; protégée contre l'écriture.

```
FFD8 - 8D 88 BF  STA  $BF88   Sauvegarde du registre Accumulateur.
FFDB - AD 56 BF  LDA  $BF56   INTACCU. Charge la valeur,
FFDE - 85 45     STA  $45     et sauvegarde en A5H.
FFE0 - AD 8B C0  LDA  $C08B   BANKSELECT. Commute la Bank 1
FFE3 - AD 8B C0  LDA  $C08B   + écriture autorisée.
FFE6 - AD 57 BF  LDA  $BF57   #$00 = bank 1 ; #$FF = bank 2.
                              Utilisé comme pointeur HByte.
FFE9 - 4C D3 BF  JMP  $BFD3   INTEXIT2. Déconnecte la bank et actualise RTI.
```

- IRQ langage carte commutée.

Une interruption d'origine IRQ sollicite deux vecteurs de la page globale de PRODOS : INTBANK1 et INTBANK2. Par ailleurs, l'adresse $45 de la page zéro est utilisée pour rendre la valeur de l'accumulateur transparente au système.

Le contenu de l'accumulateur est sauvegardé sur la pile au moment de l'entrée dans la routine - $FF9B - et copié dans la page globale à l'adresse $BF56. Le saut à la routine IRQ-Handler du Moniteur se fera indifféremment à partir d'une interruption IRQ-BReaK ou IRQ-Réel, en fixant au préalable l'adresse de retour à $FA42, qui est, dans notre cas, l'adresse après le STA $45 - voir paragraphe : interrupution à partir du Moniteur.
Le listing ci-après donne le détail des différents points clefs.

```
FF9B - 48       PHA          Contenu de l'accumulateur au moment de
                             l'interruption IRQ.
FF9C - A5 45    LDA  $45     Charge A5H, qui contient le pointeur :
                             Read-Write-Tracks-Blocks - HByte.
FF9E - 8D 56 BF STA  $BF56   Sauvegarde dans INTACCU - Page globale.
FFA1 - 68       PLA          Dépile le contenu de l'accumulateur au moment
                             de l'interruption IRQ,
FFA2 - 85 45    STA  $45     et le sauvegarde dans A5H.

::::::::::::::::::::::::::::

FFB3 - 8D 8D BF STA  $BF8D   INTBANK1
FFB6 - 8D 57 BF STA  $BF57   INTBANK2
```

INTBANK1 et INTBANK2 sont positionnés à la même valeur : #$00 pour la "bank 1" ou #$FF pour la "bank 2". Avec une interruption IRQ, via un BReaK (code 00), les valeurs intermédiaires de INTBANK sont indéterminées, ce qui ne perturbe en rien le déroulement des opérations en cours.

Pour une interruption IRQ du Moniteur, le système effectue un saut à l'adresse $FA40 :

```
FA40 - 85 45    STA  $45     Sauvegarde du contenu de l'accumulateur.
FA42 - 68       PLA          Dépile vers A le contenu du registre d'état PS.
FA43 - ::::::::::::::::::
```

Lors d'une interruption IRQ réelle, le système branche *via* la page 3, par un JMP ($03FE), à l'adresse $BFEB - EB BF contenu de l'adresse $03FE. A partir de l'adresse $BFEB débute la routine de commutation : celle-ci connecte la "bank 1" de la carte langage, puis effectue un JMP $D13A.

- Commutation par INTERRUPT ENTRY.

```
BFEB - AD 8B C0 LDA  $C08B   Commute la carte langage
BFEE - AD 8B C0 LDA  $C08B   + écriture autorisée.
BFF1 - 4C 3A D1 JMP  $D13A   IRQ-Handler dans le MLI.
```

- Entrée dans IRQ-Handler.

```
D13A - A5 45    LDA  $45
D13C - 8D 88 BF STA  $BF88   INTAREG. Sauvegarde de la valeur
                             de A durant l'interruption IRQ.
D13F - ::::::::::::::::::::
```

En fin d'exécution de la routine $D13A, le système effectue un saut à la page globale de PRODOS, par le biais de l'adresse $BFD0 ; cet emplacement particulier se nomme INTEXIT. Le rôle de la routine implantée est de reconnecter la bank adéquate : par exemple la ROM du Moniteur.

- Commutation par INTEXIT.

```
BFD0 - AD 8D BF  LDA  $BF8D      INTBANK1.
BFD3 - F0 0D     BEQ  $BFE2      #$00 = bank 1 est (ou était) connectée.
BFD 5- ::::::::::::::::::::
```

GET.TIME - CARTE THUNDERCLOCK

La fonction MLI $82 - GET.TIME - génère, par l'intermédiaire de la ROM d'une carte horloge, un pointeur précédé d'un JMP, à l'intérieur de la page globale de PRODOS, et plus particulièrement à l'adresse $BF06. A cette adresse se trouve normalement un RTS - code 60 - qui est une instruction de retour pour le microprocesseur ; cette valeur est mise en place par défaut. Le MLI GET.TIME contient un appel interne à cette adresse sous la forme d'un JSR $BF06.

Lors du "boot" d'une disquette ProDOS, le système vérifie automatiquement le contenu de tous les slots en ligne : si une carte horloge du type "Thunderclock" est détectée, ProDOS change le RTS à l'adresse $BF06 en un JMP $F142.
A cette adresse, une routine débute qui effectue la lecture de la date et de l'heure actuelle, et sauvegarde les valeurs aux adresses $BF90-$BF93 : la date aux adresses $BF90-$BF91 et l'heure aux adresses $BF92-$BF93.

Le RTS qui se trouve en fin de la routine, $F1AB, renvoie à nouveau le système dans le MLI, et ceci soit à l'adresse $D078, si c'est uniquement GET.TIME qui est sollicité ; soit au Command-Handler pour les commandes $C0-$D3 qui nécessitent l'association d'autres fonctions, comme CREATE par exemple.

La routine de lecture date et heure sera toujours implantée à partir de l'adresse $F142 pour un système de 64 Ko, et ceci qu'une carte interface soit détectée ou non ; un court programme Move effectue le transfert de la routine vers son emplacement définitif au moment du boot.

Lorsqu'une carte horloge est détectée, la routine de recherche des slots place à l'intérieur du programme Move, à l'emplacement de la liste des paramètres slots, le numéro du slot qu'elle aura trouvé. Dans le listing du sous-programme, le byte de poids fort du slot, qui désigne le numéro du slot, est représenté par "xx" pour remplacer la valeur Cs, où "s" représente le numéro du slot où se trouve la carte horloge.

Routine GET.TIME - ($82)

- Routine de lecture de la date et de l'heure à partir de la ROM résidente (carte horloge). Sous-programme implanté à l'adresse $F142.


```
F142 - A2 50 F1  LDX  $F150      Charge le numéro du slot où se trouve la carte
                                 horloge : $Cs ou s est le numéro de slot.
F145 - BD 38 05  LDA  $0538,X    Bytes intermédiaires d'entrée/sortie de l'écran
F148 - 48        PHA             et sauvegarde sur la pile.
F149 - A9 A3     LDA  #$A3       Commande : Time format de ProDOS.
F14B - 20 0B xx  JSR  $xx0B      OUT de la commande résidente de la ROM.
F14E - 20 08 xx  JSR  $xx08      IN MM-YY-DD hh : mm. Sous-format ASCII.
                                 Mois-Année-Jour heure : minutes

F151 - 18        CLC
F152 - A2 04     LDX  #$04       Fixe la lecture des données en débutant par
F154 - A0 0C     LDY  #$0C       la fin, minutes d'abord.
F156 - B9 00 02  LDA  $0200,Y    Position des dizaines.
F159 - 29 07     AND  #$07       Fixe les limites des données de 0 à 7,
F15B - 85 3A     STA  $3A        et sauvegarde le résultat.
F15D - 0A        ASL
F15E - 0A        ASL             Multiplie la valeur des dizaines par 4,
F15F - 65 3A     ADC  $3A        et additionne sa propre valeur = dizaines * 5
F161 - 0A        ASL             * 2 = valeur initiale * 10.
F162 - 79 01 02  ADC  $0201,Y    Additionne la valeur basse.
F165 - 38        SEC
F166 - E9 B0     SBC  #$B0       #$B0 = 0 en code ASCII.
F168 - 95 3A     STA  $3A,Y      Adresses : $3E = minutes ; 3D = heures ;
                                 $3C = Jour ; $3B = Année ; $3A = Mois.

F16A - 88        DEY
F16B - 88        DEY             Saute les séparateurs de la date.
F16C - 88        DEY
F16D - CA        DEX
F16E- 10 E6      BPL  $F156
F170 - A8        TAY             Mois - vecteur $01 à $0C.
F171 - 4A        LSR
F172 - 6A        ROR             MMMM sera changé en format MMM0 0000.
F173 - 6A        ROR             MSB reste dans la Carry.
F174 - 6A        ROR             ORA avec Day = Jour.
F175 - 05 3C     ORA  $3C

                                 Bits :          7 6 5 4 3 2 1 0.
F177 - 8D 90 BF  STA  $BF90      Contenu de $BF90 : M M M D  D D  D D.

F17A - 08        PHP             Month = Mois ; Day = Jour.
                                 MSB du mois.

                                 Bits       F E D C B A 9 8.
F17B - 29 1F     AND  #$1F       Résultat = 0 0 0  D D D D D.
F17D - 79 AB F1  ADC  $F1AB,Y    Day = Jour ; 0 = bit à 0.
                                 La valeur du mois est indexée ;
F180 - 90 02     BCC  $F184      ADC avec la Carry = MSB du mois
F182 - 69 03     ADC  #$03
F184- 38         SEC             + 4 uniquement avec les Mois = 9 ; et Jour > 8.
```

```
F185 - E9 07     SBC  #$07
F187 - B0 FC     BCS  $F185
F189 - 69 07     ADC  #$07      Résultat allant de 0 à 7 ;
F18B - E5 3B     SBC  $3B       moins la valeur de l'année.
F18D - B0 02     BCS  $F191
F18F - 69 07     ADC  #$07      Résultat allant de 0 à 7.
F191 - A8        TAY
F192 - B9 B8 F1  LDA  $F1B8,Y   Charge la valeur de l'année, à partir
                                d'une table qui débute en $F1B8 - 2 digits -
F195 - 28        PLP            MSB du mois.
F196 - 2A        ROL            Effectué sur le bit 0 de la valeur de l'année
F197 - 8D 91 BF  STA  $BF91     Sauvegarde de l'année : Année + MSB du Mois,
                                dans le bit 0.
                                Bits : 7 6 5 4 3 2 1 0
                                       Y Y Y Y Y Y Y M
                                                Y = Year  = Année
                                                M = Month = Mois.

F19A - A5 3D     LDA  $3D       Charge la valeur des heures - Hours - hh -
F19C - 8D 93 BF  STA  $BF93     et sauvegarde dans $BF93 :
                                Bits : 7 6 5 4 3 2 1 0
                                       x x x h h h h h      h = Hours = Heures
                                                            x = bits inutilisés = 0

F19F - A5 3E     LDA  $3E       Charge la valeur des minutes - Minutes - mm -
F1A1 - 8D 92 BF  STA  $BF92     et sauvegarde dans $BF92 :
                                Bits : 7 6 5 4 3 2 1 0
                                       x x m mm m m m m   m = Minute = minutes
                                                            x = bits inutilisés =0.

F1A4 - 68        PLA            Restaure les vecteurs I/O sauvegardés au
                                préalable.
F1A5 - AE 50 F1  LDX  $F150     Format du slot : Cs ou s = numéro du slot
F1A8 - 9D 38 05  STA  $0538,X   et sauvegarde.
F1AB - 60        RTS            Retour au MLI.
```

Table de correction :

```
F1AC - 00 1F 3B 5A 78 97 B5 D3  Facteurs de correction.
F1B4 - F2 14 33 51
```

Table des années prédéfinies :

```
F1B8 - 54 54 53 52 57 56 55     Valeurs prédéfinies des années : 84 84 83 82 87
                                86 85.
```

ANNEXE 1

UTILISATION DES ADRESSES DE LA PAGE ZERO

Décimal		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
HEX		00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000	$00	AD	AD	AD	A	A	A			A	A	A	A	A	A		
016	$10	A	A	A	A	A	A	A	A								
032	$20	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	
048	$30	M	M	M	M	M	M	M	M	M	MD	MD	MD	MD	MD	MD	
064	$40	PM	DPM	DPM	DPM	DPM	DPM	DPM	DPM	PM	PM	P	P	P	P	PM	PM
080	$50	MA	MA	MA	MA	MA	MA	A	A	A	A	A	A	A	A	A	A
096	$60	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
112	$70	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
128	$80	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
144	$90	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
160	$A0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
176	$B0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
192	$C0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
208	$D0	A	A	A	A	A	A		A	A	A	A	A	A	A	A	
224	$E0	A	A	A	A	A	A	A	A	A	A	A					
240	$F0	A	A	A	A	A	A	A	A	A							

Légende: A = AppleSoft P = ProDOS - MLI
D = Disk-Driver M = Utilisation par le Moniteur

Remarque : ce tableau donne l'occupation mémoire de la page zéro par les différents sous-programmes des ROM et routines du système d'exploitation. La représentation reprend leurs initiales respectives. Certaines des adresses sont gérées par le système, et restaurées en fin d'exécution d'une routine.

ANNEXE 2

TABLE DES CODES D'ERREURS DE ProDOS

00 NO ERROR
Pas d'erreur : code pas affiché

02 RANGE ERROR
Valeur d'une option trop grande ou trop petite.

03 NO DEVICE CONNECTED
Pas de périphérique en ligne - Slot.

04 WRITE PROTECTED
Disquette protégée en écriture.

05 END OF DATA
Fin du fichier ou d'enregistrement.

06,07 PATH NOT FOUND
Chemin non trouvé, ou fichier non trouvé par le chemin spécifié.

08 I/O ERROR
Erreur d'entrée/sortie lors d'une tentative d'écriture/lecture sur la disquette.

09 DISK FULL
Le Volume est plein.

10 FILE LOCKED
Fichier verrouillé - LOCK.

11 INVALID OPTION
Option déclarée non valide.

12 NO BUFFER AVAILABLE
Plus de mémoire disponible pour ouvrir un fichier.

13 FILE TYPE MISMATCH
Type de fichier incorrect.

14 PROGRAM TOO LARGE
Le fichier dépasse la taille de la mémoire allouée.

15 NOT DIRECT COMMAND
Commande non valide en mode immédiat.

16 SYNTAX ERROR
Erreur de syntaxe dans l'écriture.

17 DIRECTORY FULL
Catalogue du volume plein : contient 51 fichiers.

18 FILE NOT OPEN
Fichier non ouvert.

19 DUPLICATE FILENAME
Le fichier existe déjà.

20 FILE BUSY
Le fichier est déjà ouvert.

21 FILE(S) STILL OPEN
Le ou les fichiers n'ont pas été fermés.

ANNEXE 3

CODES D'ERREURS ISSUES DU MLI

Table complète des erreurs pouvant être rencontrées lors de l'utilisation des routines MLI - Machine Language Interface - du système ProDOS. Ces différentes commandes sont appelées soit à partir d'un sous-programme, soit par le système lui-même, et retournent un message d'erreur lorsqu'une exécution n'aboutit pas. Le code ainsi renvoyé renseigne sur la cause de l'interruption de la routine.

$00 NO ERROR
Pas d'erreur : code non affiché.

$01 BAD SYSTEM CALL NUMBER
Erreur du numéro de la commande MLI : le byte de commande n'a pas été trouvé au niveau du MLI.

$04 BAD SYSTEM CALL PARAMETER COUNT
Erreur au niveau du nombre de paramètres alloués à une commande MLI : le nombre de paramètres déclaré ne correspond pas avec celui de la table des paramêtres, premier byte de la table.

$25 INTERRUPT TABLE FULL
Interruption : table saturée, 4 interruptions possibles seulement.

$27 I/O ERROR
Erreur d'entrée/sortie. Elle est aussi affichée lorsque l'erreur survenue n'est pas reconnue par le système.

$28 NO DEVICE CONNECTED
Pas de périphérique en ligne.

$2B DISK WRITE PROTECTED
Disque protégé en écriture.

$2E DISK SWICHTCHED
Disquettes inversées ; les commandes WRITE, CLOSE, etc., ne peuvent aboutir.

$40 INVALID PATHNAME
Chemin ou syntaxe incorrect.

$42 MAXIMUM NUMBER OF FILES OPEN
Maximum de fichiers ouverts : 8.

$43 INVALID REFERENCE NUMBER
Numéro de référence du fichier incorrect :
- "Reference Number" hors limite de la valeur 1-4 ;
- le File Control Block n'existe pas avec la valeur du numéro de référence cité.

$44 DIRECTORY NOT FOUND
Catalogue non trouvé.

$45 VOLUME NOT FOUND
Volume non trouvé.

$46 FILE NOT FOUND
Fichier non trouvé.

$47 DUPLICATE FILENAME
Nom de fichier existant.

$48 VOLUME FULL
Disquette pleine, ou dépassement de la marque EOF lors de la commande WRITE.

$49 VOLUME DIRECTORY FULL
Catalogue volume saturé, plus de 51 noms de fichiers.

$4A INCOMPATIBLE FILE FORMAT
Fichier : données incompatibles.

$4B UNSUPPORTED STORAGE TYPE
Type de fichier incorrect, sauvegarde.

$4C END OF FILE ENCOUNTERED
Fin de fichier rencontrée, marque EOF rencontrée en association avec la commande READ.

$4D POSITION OUT OF RANGE
Débordement de la position, plus grand que EOF.

$4E FILE ACCESS ERROR
Erreur d'accès au fichier : verrouillé.

$50 FILE IS OPEN
Fichier ouvert, lors de la commande OPEN, etc.

$51 DIRECTORY STRUCTURE DAMAGED
Structure du catalogue non conforme.

$52 NOT A PRODOS VOLUME
Pas de Volume ProDOS.

$53 INVALID SYSTEM CALL PARAMETER
Syntaxe ou nombre incorrect d'un paramètre MLI.

$55 VOLUME CONTROL BLOCK TABLE FULL
Table de gestion des blocs saturée: apparaît lors de la tentative d'ouverture de plus de 8 fichiers, ou plus de 8 périphériques adressés.

$56 BAD BUFFER ADDRESS
Adresse du tampon non valide.

$57 DUPLICATE VOLUME
Nom de Volume déjà en ligne.

$5A FILE STRUCTURE DAMAGED
La Volume Bit-Map et le nombre des blocs ne correspondent pas avec la structure du fichier actuel.

ERREURS VENANT DU SYSTEM DEATH

ERR#00
Pas d'erreur, donc aucun affichage.

ERR#01 INTERRUPT FROM UNKNOW SOURCE (VERS. 1.0.1)
Erreur dans la transmission des routines d'interruptions.

ERR#02 INTERRUPT FROM UNKNOW SOURCE (VERS. 1.0.2)
Identique à ERR#01, mais a changé de numéro.

ERR#0A NO VCB ENTRY FOR OPEN FILE
Lors de la recherche de la table VCB, il se révèle qu'aucun périphérique ne lui est alloué - Unit Number.

ERR#0B NO ASSIGNED FILE BUFFER FOR OPEN FILE
Aucun tampon mémoire n'est assigné à un fichier ouvert : l'adresse de début a été détruite dans la table des tampons pour une raison indéterminée.

ERR#0C NON EXISTING BLOCK IN FILE BUFFER
Le numéro du bloc a disparu de la table FCB pour une raison indéterminée.

ANNEXE 4

TABLE DES FICHIERS REDEFINISSABLES

TYPE	Signification
$00	Typeless file, SOS and PRODOS. Fichier non défini SOS et PRODOS.
$01	Bad Block file. Fichier contenant les blocs détruits.
$02	Pascal code file. Fichier Pascal, code.
$03	Pascal text file. Fichier Pascal, texte.
$04	ASCII text file, SOS and PRODOS. Fichier texte ASCII, SOS et PRODOS.
$05	Pascal data file. Fichier Pascal, données.
$06	General binary file, SOS and PRODOS. Fichier binaire, SOS et PRODOS.
$07	Font file. Fichier de caractères et matrices.
$08	Graphics screen file. Fichier graphique.
$09	Business Basic program file. Fichier programme en Basic, affaires.
$0A	Business Basic data file. Fichier Basic : données, affaires.
$0B	Word processor file. Fichier langage machine.
$0C	SOS system file. Fichier système, SOS.
$0D,0E	SOS reserved. Codes réservés au système SOS.
$0F	Directory File, SOS and PRODOS.Fichier catalogue, SOS et PRODOS.
$10	RPS data file. Fichier de données RPS.
$11	RPS index file. Fichier index RPS.
$12-BF	SOS reserved. Codes réservés au système SOS.
$C0-EF	PRODOS reserved. Codes réservés au système PRODOS.
$F0	PRODOS added command file. Fichier de commandes PRODOS.
$F1-F8	PRODOS user defined files 1-8. Fichiers redéfinissables par l'utilisateur : codes $F1-$F8, PRODOS.
$F9	PRODOS reserved. Réservés au système PRODOS.
$FA	Integer BASIC Programm file. Fichier BASIC Integer.
$FB	Integer BASIC variables file. Fichier BASIC Integer, variables.
$FC	AppleSoft program file. Fichier Programme AppleSoft.
$FD	AppleSoft variables file. Fichier AppleSoft, variables.
$FE	Relocatable code file, EDASM. Fichier relogeable, Codes EDASM.
$FF	PRODOS System file. Fichier système PRODOS.

Remarque : les codes $01-$03, $05, $07-0E, $10,$11 et $12-$BF sont réservés pour le système SOS avec l'Apple III. Il ne faudra en aucun cas les utiliser avec PRODOS.

ANNEXE 5

PARAMETRES OPTIONNELS

CHEMIN — C'est le chemin, Pathname, complet ou partiel pour la recherche d'un fichier.

— Spécifie une valeur en notation décimale lorsque le nombre qui suit # est seul. Est en notation hexadécimale lorsque le nombre spécifié a comme préfixe "$". La valeur attribuée au nombre, suivant son utilisation avec une commande bien déterminée, devra être conforme au type de périphérique cité, faute de quoi le système vous retourne un message d'erreur.

A# — Adresse mémoire. A# indique l'adresse de début d'une zone mémoire de la RAM. Lorsqu'il est utilisé en association avec les commandes BRUN, BLOAD et BSAVE, cette zone est celle où sera implanté le fichier. Pour BLOAD et BRUN, la valeur par défaut de A# est celle qui a été utilisée lors de la phase de sauvegarde du fichier sur la disquette, BSAVE.

B# — Nombre de bytes à sauter - Byte count. Sa valeur par défaut est 0 ; il indique une position dans le fichier, qui est la position courante +B# bytes. Lorsque ce paramètre est utilisé avec les commandes READ et WRITE, il est évalué après l'option F#. La valeur maximale de B# est la longueur du fichier, sa valeur minimale étant 1. Avec les commandes BRUN, BLOAD et BSAVE, il est toujours utilisé en faisant référence au début du fichier.

D# — Numéro du lecteur en ligne (1-2).

E# — Adresse mémoire de fin. Ce paramètre est un dérivé de l'option L# pour les commandes BRUN, BSAVE, et BLOAD, où E# indique la dernière adresse de la zone mémoire servant à l'implantation du programme. La commande BSAVE nécessite l'option L# ou E#. Si lors de la commande BRUN et BLOAD aucune des deux options n'est utilisée, les bytes seront placés les uns à la suite des autres en mémoire vive, jusqu'à la fin du fichier - option par défaut.

F# — Nombre de lignes de texte. C'est la position à l'intérieur d'un fichier texte à accès séquentiel ou aléatoire, et utilisé par les commandes READ, WRITE, POSITION et EXEC. Le numéro de position indique un champ, ou rubrique, dont la position dans le fichier est à F# champs en avant de la position du pointeur du fichier. Un champ est délimité par deux retours-chariots, code 13. La valeur par défaut de F# est 0, ce qui ne modifie pas la position du pointeur d'un fichier.

L# — Longueur déclarée. Utilisée avec les commandes OPEN et APPEND lors de la création d'un fichier à accès aléatoire, cette commande indique la longueur de l'enregistrement. Par défaut, L# est égal à 1. Si le fichier existe déjà, la valeur par défaut est celle qu'il avait lors de la création de celui-ci. Utilisé avec les commandes BRUN, BLOAD et BSAVE, L# spécifie le nombre de bytes à transférer du fichier vers la mémoire, ou vice versa.

R# — Numéro d'enregistrement déclaré. Ce paramètre est utilisé avec les commandes READ, WRITE et POSITION. Il désigne l'enregistrement numéro R# d'un fichier texte à accès aléatoire. La valeur par défaut est la dernière valeur de R# spécifiée pour ce fichier.

S# — Numéro de slot (1-7). Il est fixé à 6 sur le IIc.

Ttype — Où type indique le type de fichier déclaré par l'abréviation des trois premiers caractères (DIR, TXT, BIN, etc.).

à# — Numéro de la première ligne du programme à exécuter. Avec les commandes RUN et CHAIN, à# spécifie le numéro de la première ligne du programme à exécuter. Si la ligne indiquée par à# n'existe pas, l'exécution débutera à la première ligne existante suivante. La valeur par défaut de à# est le début du programme.

IN#,PR# — Désigne le périphérique utilisé par les commandes PR# et IN#, dont la valeur de l'indice # peut prendre une valeur entre 0 et 7 inclus.

COMMANDES DU SYSTEME ProDOS

Liste des commandes du système d'exploitation ProDOS avec sa syntaxe, ses paramètres autorisés et les différents chemins possibles.

nstruction	Syntaxe
APPEND	chemin, L#, S#, D#
BLOAD	chemin, A#, B#, L# ou B#, Ttype, S#, D#
BRUN	chemin, A#, B#, L# ou E#, S#, D#
BSAVE	chemin, A#, L# ou E#, B$, Ttype, S#, D#
CAT	chemin, S#, D#
CATALOG	chemin, S#, D#
CHAIN	chemin, à#, S#, D#
CLOSE	chemin
CREATE	chemin, Ttype, S#, D#
DELETE	chemin, S#, D#
EXEC	chemin, S#, D#
FLUSH	chemin
FRE	sans paramètre.
IN#	numéro du connecteur
LOAD	chemin, S#, D#
LOCK	chemin, S#, D#
OPEN	chemin, L#, Ttype, S#, D#
POSITION	chemin, F# ou R#
PREFIX	chemin, S#, D#
PR#	numéro du connecteur
READ	chemin, R#, F#, B#
RENAME	chemin (ancien nom), chemin (nouveau nom)
RESTORE	chemin, S#, D#
RUN	chemin, à#, S#, D#
SAVE	chemin, S#, D#
STORE	chemin, S#, D#
UNLOCK	chemin, S#, D#
WRITE	chemin, R#, F#, B#
-	chemin, S#, D#

ANNEXE 6

PLAN D'OCCUPATION MEMOIRE PAR APPLESOFT SOUS ProDOS

CARTE MERE

$F7FF-$FFFF	Moniteur en mémoire morte...
$D000-$F7FF	AppleSoft en mémoire morte...
$C000-$CFFF	Mémoires mortes des cartes périphériques d'entrées/sorties de la configuration.
$9600-$BFFF	Variables globales de PRODOS : $BF00-$BFFF. Variables globales du BASIC.SYSTEM : $BE00-$BEFF. BASIC.SYSTEM Tampon de 1 Ko.
$0800-$95FF	**HIMEM** Espace pour la programmation.
	LOMEM
$0400-$07FF	Page 1 texte.
$0000-$03FF	Tampon d'entrée du clavier. Page réservée pour la pile. Page zéro.

CARTE LANGAGE

$F000 - $FFFF	Routines disques, Disk-Driver.
$E000 - $EFFF	PRODOS 2e partie.
$D000 - $DFFF	Réservé pour le Reboot.
$D000 - $DFFF	Bank alternée de la carte langage. PRODOS 1re partie.

ANNEXE 7

EXEMPLE DE RECUPERATION DU CODE D'ERREUR

(ASSEMBLEUR BIG-MAC)

***TEST.FICHIER**

```
* COPYRIGHT NOV. 85
* Marcel COTTINI

BELL     EQU   $FF3A      ; routine BELL
COUT     EQU   $FDED      ; routine COUT
PRBYTE   EQU   $FDDA      ; routine PRBYTE
HOME     EQU   $FC58      ; routine HOME
MLI      EQU   $BF00      ; appel de PRODOS
CRCMD    EQU   $C0        ; code de commande

         ORG   $2000
```

***CREER LE FICHIER**

```
DEBUT    JSR   HOME       ; efface l'écran
         JSR   CREER      ; crée le fichier
         BNE   ERROR      ; erreur rencontrée
         RTS              ; sinon retour
```

***CODE ET POINTEUR**

```
CREER    JSR   MLI        ; appel de PRODOS
         DFB   CRCMD      ; code de commande
         DA    CRLIST     ; pointeur de CRLIST
         RTS              ; retour
```


***GESTION DE L'ERREUR**

```
ERROR    JSR   PRBYTE     ; affiche le code d'erreur
         LDX   #$00       ; charge X avec 00
ENCOR    LDA   TEXTE,X    ; charge un caractère
         BEQ   FINI       ; fin du texte
         JSR   COUT       ; affiche le caractère
         INX              ; incrémente X
         BNE   ENCOR      ; continue
FINI     JSR   BELL       ; sonnette
         JSR   BELL       ; sonnette
         JSR   BELL       ; sonnette
         RTS              ; retour
```

***TABLE DES PARAMETRES**

```
CRLIST   DFB   7          ; déclare 7 paramètres
         DA    FICH       ; pointeur nom du fichier
         DFB   $C3        ; accès normal du fichier
         DFB   $04        ; code d'un fichier texte
         DFB   $00,$00    ; type auxiliaire non utilisé
         DFB   $01        ; fichier standard
         DFB   $00,$00    ; date non utilisée
         DFB   $00,$00    ; heure non utilisée
```

***LONGUEUR DU NOM**

```
FICH     DFB   FIN-NOM    ; calcul de la longueur du nom
                          ; du fichier (10)
```

***CHEMIN DU FICHIER**

```
NOM      ASC   "/TEST/NOUVEAUNOM"
FIN      EQU   *
```

***TABLE ASCII**

```
TEXTE    ASC   " = CODE ERREUR RENCONTRE."
         HEX   00 ; fin de la table ASCII
```

Remarque : ce programme permet d'effectuer un appel à PRODOS par un JSR MLI, puis de créer un fichier texte du nom de "NOUVEAUNOM" dans un Volume qui a comme nom TEST. Si une erreur est rencontrée par le système, celui-ci émet un bruit de sonnette par l'intermédiaire de la routine BELL - JSR $FF3A. Ensuite, l'Apple affiche le

code d'erreur rencontrée. Ce programme peut facilement être adapté à tout assembleur, ou tout simplement rentré en mémoire par le Moniteur. Notre exemple a été assemblé sur BIG-MAC. L'exécution nécessite les fichiers PRODOS et BASIC.SYSTEM en mémoire, ainsi qu'une disquette avec comme nom de Volume TEST. Pour provoquer une erreur, il suffit d'ouvrir la porte du drive, ou tout simplement de retirer la disquette.

Méthode par le Moniteur :

```
CALL-151 <RETURN>

2000 : 20  58  FC  20  09  20  D0
       2A  20  60  20  DA  FD  A2
       20  ED  FD  E8  D0  F5  20
       3A  FF  60  07  36  20  C3
       00  00  10  AF  D4  C5  D3
       C5  C1  D5  CE  CF  CD  A0
       A0  C5  D2  D2  C5  D5  D2
       CE  D4  D2  C5  AE  00  <RETURN>



       08  60  20  00  BF  C0
       00  BD  47  20  F0  06
       3A  FF  20  3A  FF  20
       04  00  00  01  00  00
       D4  AF  CE  CF  D5  D6
       BD  A0  C3  CF  C4  C5
       A0  D2  C5  CE  C3  CF
```

Pour la sauvegarde sur une disquette, tapez :

```
BSAVE TEST.FICHIER, A$2000, L$61   <RETURN>
```

ANNEXE 8

TABLE DES APPELS MLI SOUS ProDOS

$40 ALLOC INTERRUPT	Fixe une routine de gestion d'interruption masquable - IRQ.
$41 DEALLOC INTERRUPT	Retrait d'une routine d'interruption.
$65 REBOOT	Demande de chargement d'un autre programme système - interpréteur.
$80 READ BLOCK	Lecture d'un bloc de 512 octets.
$81 WRITE BLOCK	Ecriture d'un bloc de 512 octets.
$82 GET TIME	Appel du sous-programme implanté à l'adresse $F142, à partir de la page globale de PRODOS, pour la mise à jour de la date et de l'heure - $BF90-$BF93.
$C0 CREATE	Création d'un nouveau fichier.
$C1 DESTROY	Destruction d'un fichier.
$C2 RENAME	Renomme un fichier avec le même nom de chemin - Pathname.
$C3 SET FILE INFO	Change les attributs d'un fichier.
$C4 GET FILE INFO	Lit les attributs d'un fichier.
$C5 ONLINE	Obtient, pour un lecteur physique, le nom du volume demandé.
$C6 SET PREFIX	Change le préfixe par défaut.
$C7 GET PREFIX	Lit le préfixe courant.
$C8 OPEN	Ouvre un fichier et retourne un numéro de référence.
$C9 NEWLINE	Spécifie le caractère de fin du champ d'un fichier (retour-chariot, code $0D).
$CA READ	Lit un ou plusieurs octets à partir de la position courante.
$CB WRITE	Ecrit un ou plusieurs octets à partir de la position courante.
$CC CLOSE	Ferme un ou tous les fichiers ouverts.
$CD FLUSH	Sauvegarde sur une disquette les tampons des fichiers ouverts.

$CE SET MARK	Change la position courante de lecture/ écriture.
$CF GET MARK	Recherche la position courante de lecture/ écriture.
$D0 SET EOF	Change la position de fin du fichier.
$D1 GET EOF	Recherche la position de fin du fichier.
$D2 SET BUF	Modifie l'adresse du tampon d'un fichier.
$D3 GET BUF	Recherche l'adresse du tampon d'un fichier.

ANNEXE 9

COMMUTATEURS D'AFFICHAGE ET DE MEMOIRE ENTREES/SORTIES ET COMMUTATEURS LOGIQUES

Noms	Etat	Adresse	Lec./Ecr.	Fonction
KEYBOARD	—	$C000	L	Lecture du clavier.
80STORE	off	$C000	E	Mise à 0 de 80STORE; PAGE2 est à nouveau commutateur des pages 1 et 2 (texte et GR). permet d'utiliser RAMRD/RAMWRT.
	on	$C001	E	Permet d'accéder aux pages d'affichage, par PAGE2 ; mise à 1 de 80STORE.
RAMRD	off	$C002	E	Permet la lecture en mémoire principale ; mise à 0 de RAMRD.
	on	$C003	E	Permet la lecture en mémoire auxiliaire ; mise à 1 de RAMRD.
RAMWRT	off	$C004	E	Permet d'écrire en mémoire principale ; mise à 0 de RAMWRT.
	on	$C005	E	Permet l'écriture en mémoire auxiliaire ; mise à 1 de RAMWRT.
SLOTCXROM	off	$C006	E	ROM résidente qui se trouve dans le slot $Cs00.
	on	$C007	E	ROM d'un périphérique qui se trouve en $Cs00.
ALTZP	off	$C008	E	La pile et la page 0 sont utilisées dans la mémoire principale ; mise à 0 de ALTZP.
	on	$C009	E	La pile et la page 0 sont utilisées dans la mémoire auxiliaire ; mise à 1 de ALTZP.

SLOTC3ROM	off	$C00A		E	ROM résidente utilisée à partir de $C300.
	on	$C00B		E	ROM d'un périphérique qui se trouve en $C300.
80COL	off	$C00C		E	Affichage en 40 colonnes et mise à 0 de 80COL.
	on	$C00D		E	Affichage en 80 colonnes et mise à 1 de 80COL.
ALTCHARSET	off	$C00E		E	Sélectionne l'ensemble primaire des caractères ; mise à 0 de ALTCHAR.
	on	$C00F		E	Sélectionne l'ensemble alternatif des caractères ; mise à 1 de ALTCHAR.
KBDSTRB	—	$C010		E	Remise à zéro du clavier;mise à 0 du bit 7.
	—	$C010	L		Permet ensuite de lire l'état de ALTCHARSET (adresse $C00E).
	—	$C011	L		Indicateur de la Bank utilisée (D0-DF).
	—	$C012	L		Indicateur de mémoire (principale ou auxiliaire). Mémoire entre les pages $D0-$FF.
	—	$C013	L		Permet de lire l'état de RAMRD.
	—	$C014	L		Permet de lire l'état de RAMWRT.
	—	$C015	L		Permet de lire l'état de SLOTCXROM.
	—	$C016	L		Permet de lire l'état de ALTZP.
	—	$C017	L		Permet de lire l'état de SLOTC3ROM.
	—	$C018	L		Permet de lire l'état de 80STORE.
	—	$C019	L		Permet de lire l'état de l'indicateur associé à VBL (effacement vertical).
	—	$C01A	L		Permet de lire l'état de TEXT.
	—	$C01B	L		Permet de lire l'état de MIXED.
	—	$C01C	L		Permet de lire l'état de PAGE2.
	—	$C01D	L		Permet de lire l'état de HIRES.
	—	$C01E	L		Indicateur du jeu de caractères.
	—	$C01F	L		Permet de lire l'état de 80COL.
TEXT	off	$C050	L	E	Mode graphique ; mise à 0 de TEXT.
	on	$C051	L	E	Mode texte ; mise à 1 de TEXT.
MIXED	off	$C052	L	E	Mode non mixte; mise à 0 de MIXED.
	on	$C053	L	E	Mode mixte; mise à 1 de de MIXED.
PAGE2	off	$C054	L	E	Accès à la page 1, si 80STORE off. Accès à la mémoire principale, si 80STORE est on ; mise à 0 de PAGE2.
	on	$C055	L	E	Accès à la page 2, si 80STORE est on ; Accès à la mémoire auxiliaire, si 80STORE est on ; mise à 1 de PAGE2.
HIRES	off	$C056	L	E	Accès basse résolution, avec 80 STORE off. PAGE2 commute la page texte si 80STORE est on ; mise à 0 de HIRES.
	on	$C057	L	E	Accès haute résolution, avec 80STORE off. PAGE2 commute la haute résolution, avec 80STORE on ; mise à 1 de HIRES.

Remarque : une série de commutateurs logiques permet un accès sélectif à la mémoire auxiliaire et principale. Les commutateurs d'accès aux pages d'écran ont priorité sur les commutateurs RAMRD et RAMRWRT. La mémoire morte contenant l'interpréteur AppleSoft ($D000-$F7FF) et la ROM Moniteur ($F800-$FFFF) est en parallèle avec la "bank switched memory". Son activation est indépendante de l'état des commutateurs : RAMRD, RAMWRT, ALTZP, 80STORE, PAGE2 et HIRES.
Les différents modes sont sélectionnés en écrivant ou en lisant à certaines adresses servant de points d'entrée aux commutateurs logiques.

Règles

MIXED et HIRES n'ont d'effet sur la visualisation que lorsque TEXT est débranché.
PAGE2 est différent si 80STORE est branché : PAGE2 n'a alors plus d'effet sur la visualisation. 80STORE permet au processeur d'adresser soit la page 1 de la mémoire principale, soit la page 1 de la mémoire auxiliaire.
En temps normal, les commutateurs sont gérés par les instructions et commandes permettant de sélectionner un des modes.
Pour actionner les commutateurs 80STORE et PAGE2, il est nécessaire de passer par un programme. En effet, dès que l'Apple est commuté en mode 80 colonnes, il utilise ces commutateurs logiques.
La page 1 du texte en mémoire principale a sa réplique en mémoire auxiliaire, et cela aux mêmes adresses.

ANNEXE 10

PROGRAMME DE GESTION DE LA DATE ProDOS ASSEMBLEUR BIG-MAC

** Gestion de la date de ProDOS*

* Marcel COTTINI 29/10/85

```
CURSEUR     EQU  $33          ; prompt, réceptionne les ordres.
BUFFER      EQU  $0200        ; tampon pour les données de la date.
DATE        EQU  $BF90        ; adresse de la Global Page de ProDOS.
COUT        EQU  $FDED        ; sous-programme sortie de caractères.
GETLIN      EQU  $FD6A        ; sous-programme entrée de caractères au
                              ; clavier, dans le tampon $0200 (IN).
            ORG  $6000
            LDA  CURSEUR      ; Charge la valeur du prompt
            STA  CURSEUR1     ; et le sauvegarde.
            LDA  #$A0         ; Charge la valeur d'un blanc (espace),
            STA  CURSEUR      ; et annule la valeur initiale du prompt.
INIT        LDX  #00          ; Initialise le registre X,
DATAS       LDA  CURSEUR2,X   ; et charge les caractères ASCII à l'adresse
                              ; indexée CURSEUR2,X.
            BEQ  SORTIE       ; Entrée des caractères dans le tampon
                              ; à l'adresse $0200,
            JSR  COUT         ; et visualisation sur l'écran.
            INX               ; Incrémente X, et pointe vers le prochain
                              ; caractère.
            BNE  DATAS        ; Charge le prochain caractère.
SORTIE      JSR  GETLIN       ; Stocke le caractère chargé par l'accu-
                              ;  mulateur, dans le tampon.
```


* Formate le byte de la date, et compare si sa valeur dépasse 8 bits :

```
* Bits :    01234567
* Format :  JJ/MM/AA     JJ  = Jour sur 2 bits
*                        /   = "/" de séparation sur 1 bit
*                        MM  = Mois sur 2 bits
*                        /   = "/" de séparation sur 1 bit
*                        AA  = Année sur 2 bits

            DEX               ; Décrémente X, pour comptabiliser le nombre
                              ; de caractères.
            CPX  #7           ; Compare si le bit 7 est codé ;
            BNE  INIT         ; dépassement de valeur, on recommence.
```

* Uniquement chiffres et "/" admissibles en entrée.

```
CHAR        LDA  BUFFER,X     ; Charge un caractère du tampon ;
            AND  #$7F         ; et force le bit 7 à 0.
            STA  DATE,X       ; Sauvegarde du résultat,
            CMP  #'/'         ; Est-ce le "/" ?
            BEQ  GETLIN2      ; oui, on continue.
            CMP  #'0'         ; Est-ce un 0 ? (Détermine la position du
                              ; premier chiffre dans la table des codes ASCII ;
                              ; le "0" étant le premier chiffre.)
            BCC  INIT         ; non, on recommence.
            CMP  #':'         ; Est-ce ":" ? (Détermine la position du dernier
                              ; chiffre dans la table des codes ASCII ; le ":"
                              ; étant après le chiffre 9.)
            BCS  INIT         ; non, on recommence.
GETLIN1     DEX
            BPL  CHAR
            BMI  LOOP         ; Détermine si la date est correcte ; pas de
                              ; dépassement dans le nombre de jours.
GETLIN2     CPX  #2
            BEQ  GETLIN1
            CPX  #5
            BEQ  GETLIN1
            BNE  INIT         ; On recommence, une erreur a été rencon-
                              ; trée dans la formulation de la date.
```

* Teste si le format de la date est correct :

```
LOOP        LDX  #$FF         ; Initialise le registre X,
            JSR  FORMAT       ; et formate les données dans un byte.
            STA  JJ           ; Sauvegarde du paramètre "JOUR".
```

```
            INX             ; Premier "/".
            JSR   FORMAT    ; Incorpore les données dans le byte.
            STA   MM        ; Sauvegarde du paramètre "MOIS".
            INX             ; Deuxième "/".
            JSR   FORMAT    ; Incorpore les données dans le byte.
            STA   AA        ; Sauvegarde du paramètre "ANNEE".
```

* Teste le chiffre de l'année (0-99) :

```
            CMP   #100      ; Est-ce que le chiffre de l'année est supérieur
                            ; à 99 ?
            BCC   ANNEE     ; non, alors on continue.
            BCS   INIT      ; on recommence.
ANNEE       CMP   #0        ; Est-ce que le chiffre de l'année est inférieur
                            ; à 0 ?
            BCS   MOIS1     ; non, alors on continue.
            BCC   INIT      ; oui, on recommence.
```

* Teste si le chiffre attribué au mois est correct (1-12) :

```
MOIS1       LDA   MM        ; Charge le chiffre du mois,
            CMP   #1        ; est-il supérieur à 0 ?
            BCS   MOIS2     ; oui, alors on continue.
            BCC   INIT      ; non, on recommence.
MOIS2       CMP   #13       ; Est-il supérieur à 12 ?
            BCC   JOUR1     ; non, alors on continue.
            BCS   INIT      ; oui, on recommence.
```

* Teste si le chiffre, attribué au jour (déterminé par la table des jours, suivant le mois
* considéré) est correct :

```
JOUR1       TAX
            DEX
            LDA   JJ        ; Charge la valeur du jour,
            CMP   #1        ; est-il supérieur à 0 ?
            BCS   JOUR2     ; oui, alors on continue.
            BCC   INIT      ; non, on recommence.
JOUR2       CMP   TABLE,X   ;Test du chiffre représentant le jour, par
                            ; rapport au mois indiqué.
            BCC   PLACE1    ; Place la valeur dans la page globale de
                            ; ProDOS, à partir de $BF90.
            BEQ   PLACE1
            JMP   INIT      ; Erreur, on recommence.
```


* Place les valeurs de la date dans la page globale de PRODOS, sur 2 bytes, aux adresses
* $BF90,$BF91.

* Format des 2 bytes :

```
* 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    Année/Mois/Jour
```

* $BF91 ==> PLACE1 ==> bits F E D C B A 9 8
                             A A A A A A A M

* $BF90 ==> PLACE2 ==> bits 7 6 5 4 3 2 1 0
                             M M M J J J J J

```
PLACE1      LDA   AA        ; Charge la valeur de l'année,
            ASL             ; et décale d'un bit vers la gauche.
            STA   SAUV1     ; Sauvegarde du résultat.
            LDA   MM        ; Charge la valeur du mois.
            ASL             ; Décale cinq fois vers la gauche,
            ASL
            ASL
            ASL
            ASL
            STA   SAUV3     ; et sauvegarde du résultat.
            BCC   PLACE2
            CLC
            LDA   SAUV1     ; Charge la valeur du byte,
            ADC   #1        ; et lui additionne 1 - correction -
            STA   SAUV1     ; puis sauvegarde du résultat.
PLACE2      CLC             ; Annule la retenue.
            LDA   SAUV3     ; Charge le résultat intermédiaire du byte du
                            ; mois,
            ADC   JJ        ; lui incorpore la valeur de "JOUR",
            STA   SAUV2     ; et sauvegarde du résultat.
```

*Positionne les bits dans le bons sens : *LByte, HByte = $BF90,$BF91; MM/JJ ; ensuite AA/MM :

```
            LDA   SAUV2     ; Charge la valeur de LByte MM/JJ,
            STA   DATE      ; et sauvegarde dans la page globale de
                            ; ProDOS, à l'adresse $BF90.
            LDA   SAUV1     ; Charge la valeur de HByte AA/MM
            STA   DATE+1    ; et sauvegarde à l'adresse $BF91.
            LDA   CURSEUR1  ; Charge la valeur initiale du prompt,
            STA   CURSEUR   ; et la sauvegarde à l'adresse $33.
            RTS             ; Retour au sous-programme appelant.
```

* Formate la date et incorpore les valeurs dans un byte :

```
FORMAT    INX
          LDA   DATE,X
          AND   #$0F
          ASL                 ; Multiplie le résultat par 2.
          STA   SAUV3
          ASL                 ; Multiplie par 2 le résultat.
          ASL                 ; Multiplie par 2 le résultat.
          CLC                 ; Annule la retenue.
          ADC   SAUV3
          STA   SAUV3         ; Sauvegarde du résultat.
          INX
          LDA   DATE,X
          AND   #$0F
          CLC
          ADC   SAUV3
          RTS
```

* Table des jours/mois. Fixe le nombre de jours, d'après le mois considéré :

```
TABLE     DFB   31            ; JANVIER
          DFB   29            ; FEVRIER
          DFB   31            ; MARS
          DFB   30            ; AVRIL
          DFB   31            ; MAI
          DFB   30            ; JUIN
          DFB   31            ; JUILLET
          DFB   31            ; AOUT
          DFB   30            ; SEPTEMBRE
          DFB   31            ; OCTOBRE
          DFB   30            ; NOVEMBRE
          DFB   31            ; DECEMBRE

JJ        HEX   00            ; JOUR ==> JJ
MM        HEX   00            ; MOIS ==> MM
AA        HEX   00            ; ANNEE ==> AA
SAUV3     HEX   00

SAUV1     HEX   00            ; Format : AA+MM = AA/MM
SAUV2     HEX   00            ; Format : MM+JJ = MM/JJ

CURSEUR1  HEX   00            ; Sauvegarde du prompt.
CURSEUR2  ASC   "Date (JJ/MM/AA)"  ; Format affiché de la date.
          HEX   00            ; Fin de la table ASCII.
DATE1     ASC   "JJ/MM/AA"    ; Sauvegarde du format.
```

ANNEXE 11

TABLEAU DES MODES D'ADRESSAGE

Modes d'adressage utilisés pour la série APPLE II

Modes d'adressage	Exemples	Codes machine	Cycles d'exécution (microsecondes)
Inhérent	TXA	8A	2
Immédiat	LDA #$50	A9 50	2
Absolu	LDA $800	AD 00 08	4
Page zéro	LDA $3C	A5 3C	3
Absolu indexé	LDA $800,X	BD 00 08	4 +1 si passage de page
Page zéro indexée	LDA $50,X	B5 50	4
Indirect indexé X	LDA ($50,X)	A1 50	6
indirect indexé Y	LDA ($50),Y	B1 50	5 +1 si passage de page
Relatif	BVC $1200	50 00 12	2 si pas branchement 3 si branchement même page 4 si différent
Indirect	JMP ($3400)	6C 00 34	5

ANNEXE 12

CODES OPERATIONS DU MICROPROCESSEUR : COMMUNS AUX 6502 et 65C02

00	BRK	
01	ORA	indirect,X
05	ORA	page 0
06	ASL	page 0
09	ORA	immédiat
0A	ASL	accumulateur
0D	ORA	absolu
0E	ASL	absolu
10	BPL	relatif
11	ORA	indirect,Y
15	ORA	page 0,X
16	ASL	page 0,X
18	CLC	inhérent
19	ORA	absolu,Y
1D	ORA	absolu,X
1E	ASL	absolu,X
20	JSR	absolu
21	AND	indirect,X
24	BIT	page 0
25	AND	page 0
26	ROL	page 0
28	PLP	inhérent
29	AND	immédiat
2A	ROL	accumulateur
2C	BIT	absolu
2D	AND	absolu
2E	ROL	absolu
30	BMI	relatif
31	AND	indirect,Y
35	AND	page 0,X
36	ROL	page 0,X
38	SEC	inhérent
39	AND	absolu,Y
3D	AND	absolu,X
3E	ROL	absolu,X
40	RTI	inhérent
41	EOR	indirect,X
45	EOR	page 0
46	LSR	page 0
48	PHA	inhérent
49	EOR	immédiat
4A	LSR	accumulateur
4C	JMP	absolu
4D	EOR	absolu
4E	LSR	absolu
50	BVC	relatif
51	EOR	indirect,Y
55	EOR	page 0,X
56	LSR	page 0,X
58	CLI	inhérent
59	EOR	absolu,Y
5D	EOR	absolu,X
5E	LSR	absolu,X
60	RTS	inhérent
61	ADC	indirect,X
65	ADC	page 0
66	ROR	page 0
68	PLA	inhérent
69	ADC	immédiat
6A	ROR	accumulateur
6C	JMP	indirect
6D	ADC	absolu
6E	ROR	absolu
70	BVS	relatif
71	ADC	indirect,Y
75	ADC	page 0,X
76	ROR	page 0,X
78	SEI	inhérent
79	ADC	absolu,Y
7D	ADC	absolu,X
7E	ROR	absolu,X
81	STA	indirect,X
84	STY	page 0
85	STA	page 0
86	STX	page 0
88	DEY	inhérent
8A	TXA	inhérent
8C	STY	absolu
8D	STA	absolu
8E	STX	absolu
90	BCC	relatif
91	STA	indirect,Y
94	STY	page 0,X
95	STA	page 0,X
96	STX	page 0,Y
98	TYA	inhérent
99	STA	absolu,Y

9A	TXS	inhérent
9D	STA	absolu,X
A0	LDY	immédiat
A1	LDA	indirect,X
A2	LDX	immédiat
A4	LDY	page 0
A5	LDA	page 0
A6	LDX	page 0
A8	TAY	inhérent
AA	TAX	inhérent
A9	LDA	immédiat
AC	LDY	absolu
AD	LDA	absolu
AE	LDX	absolu
B0	BCS	relatif
B1	LDA	indirect,Y
B4	LDY	page 0,X
B5	LDA	page 0,X
B6	LDX	page 0,Y
B8	CLV	inhérent
B9	LDA	absolu,Y
BA	TSX	inhérent
BC	LDY	absolu,X
BD	LDA	absolu,X
BE	LDX	absolu,Y
C0	CPY	immédiat
C1	CMP	indirect,X
C4	CPY	page 0
C5	CMP	page 0
C6	DEC	page 0
C8	INY	inhérent
C9	CMP	immédiat
CA	DEX	inhérent
CC	CPY	absolu
CD	CMP	absolu
CE	DEC	absolu
D0	BNE	relatif
D1	CMP	indirect,Y
D5	CMP	page 0,X
D6	DEC	page 0,X
D8	CLD	inhérent
D9	CMP	absolu,Y
DD	CMP	absolu,X
DE	DEC	absolu,X
E0	CPX	immédiat
E1	SEC	indirect,X
E4	CPX	page 0
E5	SEC	page 0
E6	INC	page 0
E8	INX	inhérent
E9	SEC	immédiat
EA	NOP	inhérent
EC	CPX	absolu
ED	SEC	absolu
EE	INC	absolu
F0	BEQ	relatif
F1	SEC	indirect,Y
F5	SEC	page 0,X
F6	INC	page 0,X
F8	SED	inhérent
F9	SEC	absolu,Y
FD	SEC	absolu,X
FE	INC	absolu,X

CODES OPERATIONS SPECIFIQUES AU 65C02

04	TSB	page zéro
0C	TSB	absolu
14	TRB	page zéro
1C	TRB	absolu
3A	DEA	accumulateur
5A	PHY	inhérent
64	STZ	page zéro
74	STZ	page zéro,X
7A	PLY	inhérent
80	BRA	relatif
9C	STZ	absolu
9E	STZ	absolu,X
DA	PHX	inhérent
FA	PLX	inhérent

ANNEXE 13

DICTIONNAIRE DES MOTS USUELS

L'évolution des différents systèmes d'exploitation et configurations Apple a bouleversé le monde des informaticiens amateurs. Certains mots usuels, devenus familiers avec le DOS 3.3, sont nouveaux avec ProDOS. Comme ce système est récent, bien des termes évoqués le sont à tort et prêtent souvent à confusion. J'ai estimé utile et nécessaire de faire une mise au point avec ce dictionnaire des mots usuels. Les explications seront souvent comparatives entre le DOS 3.3 et ProDOS, afin de lever le doute. Ce glossaire n'est nullement une liste exhaustive du vocabulaire informatique, mais a la prétention d'attirer l'attention du lecteur sur des cas d'interprétations tendancieuses. Lorsqu'un mot aura plusieurs sens, il sera traité sous ses différents aspects.

ADRESSE

Représente un nombre mathématique utilisé pour identifier un emplacement de la mémoire, que celle-ci soit principale ou auxiliaire. Pour la série des Apple II l'adressage s'effectue sur 16 bits (65 536 adresses possibles).

APPLESOFT

Version étendue du langage de programmation Basic, utilisé avec les ordinateurs de la famille Apple. Un interpréteur est intégré dans l'unité centrale afin de pouvoir exécuter les programmes. La série des Apple IIe et IIc ont la ROM AppleSoft résidente.

ASSEMBLEUR

Système ayant la faculté de pouvoir traduire un programme en langage d'assemblage, en un programme équivalent en langage machine.

AUTOSTART

C'est un circuit - ROM - renfermant des instructions qu'un micro-ordinateur peut interpréter et qui permet par la suite d'avoir accès à un langage Basic au moment de la mise sous tension du système.

ASCII

Abréviation d'American Standard Code for Information Interchange ; c'est un code de caractères utilisé en informatique. Exemple : 65 est le code ASCII de la lettre A. Dans ce code, les nombres de 0 à 127 représentent des caractères de texte comprenant les chiffres, les lettres de l'alphabet, les signes de ponctuation, des caractères spéciaux et des caractères de contrôle. Les codes ASCII spéciaux vont de 0 à 31 et sont réservés pour la gestion des fichiers ou la commande de l'imprimante.

BASIC

Langage de programmation évolué, d'une conception telle qu'il vous facilite le travail. L'AppleSoft est une de ses multiples versions.

BASIC.SYSTEM

C'est l'interface du système d'exploitation ProDOS. Il est sauvegardé sur la disquette ProDOS sous forme de fichier du type SYS, uniquement "bootable" et chargé en mémoire centrale par le fichier PRODOS. Son implantation se fait aux adresses $9A00 à $BDFF et ouvre un tampon par défaut, d'une valeur de 1 024 octets, aux adresses $9600-$99FF. Aux adresses $BE00-$BEFF se situe la page globale du BASIC.SYSTEM. C'est l'interface entre le système d'exploitation et le système Basic, dont le rôle premier est de prendre en relais toute commande non reconnue par PRODOS - MLI .

BAUD

Valeur qui exprime la vitesse de transmission d'un nombre de bits par seconde d'un périphérique. Elle s'exprime en bauds ou bits par seconde.

BINAIRE

Système de programmation primaire qui s'exprime à l'aide des chiffres 0 et 1, seule méthode de reconnaissance de l'ordinateur qui travaille d'après le système tout ou rien.

BIT

C'est le nom donné à un nombre qui ne peut prendre que deux valeurs : en général 0 et 1, tout ou rien.

BLOC

C'est une unité utilisée par le système d'exploitation ProDOS. Un bloc est un format logique de la disposition des données sur une disquette ; il équivaut à 2 secteurs ou 512 bytes. Les blocs sont gérés par le système d'exploitation lui-même, et la conversion bloc/secteur est réalisée par un sous-programme Devices-Driver, RWTB $F800. Une disquette standard 5 pouces 1/4 a une capacité de 280 blocs.

BOOT

C'est la phase d'implantation d'un système d'exploitation dans la mémoire de l'Apple. Le DOS 3.3, sauf cas spécial, se loge en mémoire centrale, en haut des 48 Ko, tandis que PRODOS - MLI - se loge dans la carte langage à la bank 1. Suivant que le boot est effectué par l'un ou l'autre des systèmes, le processus n'est pas le même. (Voir DOS 3.3 et ProDOS).

BYTE

Le byte est une unité de mesure correspondant soit à un octet (8 bits), soit à la longueur du mot du système utilisé. Exemple : 00000010 représente une valeur dont la longueur est de 1 byte.

CARTE PERIPHERIQUE

C'est un support de circuit imprimé, truffé de composants électroniques, dont l'ensemble forme un tout cohérent. Il se branche dans l'un des connecteurs disponibles de votre Apple afin d'étendre les possibilités de l'unité centrale. Il existe de nombreuses cartes dont l'interface imprimante, la carte contrôleur pour le lecteur de disquettes, ou encore la carte 80 colonnes qui permet une visualisation sur 80 colonnes, etc.

CAT

Instruction du système d'exploitation ProDOS, qui affiche le catalogue de la disquette sur 40 colonnes. Donne une version simplifiée du contenu de la disquette.

CATALOG

Instruction du DOS ou ProDOS, permettant d'afficher sur l'écran le contenu d'une disquette.

Sous DOS 3.3, le répertoire - catalogue - des fichiers se situe sur la disquette de la piste $11, secteur $0F, à la piste $11, secteur $01, et peut contenir normalement 105 fichiers ou programmes. L'écran affiche un certain nombre de renseignements sur la nature des fichiers : leurs noms, leurs tailles en secteurs, leurs types, s'ils sont verrouillés, et le numéro de Volume de la disquette.

Sous ProDOS, le Volume-Directory occupe les blocs 2 à 5 de la disquette, et la notion de pistes/secteurs a disparu. Le premier bloc contient le nom du Volume, c'est le Key-Block ; les blocs suivants sont appelés Non Key-Blocks. Un Directory principal peut contenir

51 noms de fichiers , et la commande CREATE peut créer un fichier Directory qui est une table des matières auxiliaire, chaînée au Volume-Directory. Un bloc lui est alloué au départ; un nouveau bloc est chaîné au premier, dès que la capacité du nombre de fichiers dépasse 13. Ce procédé permet une extension de la fonction répertoire - catalogue - et une gestion hiérarchisée des fichiers. La commande CATALOG, sous ProDOS, affiche un catalogue étendu sur 80 colonnes, plus riche en informations. Outre les informations communes avec DOS 3.3, ProDOS affiche le nombre total de blocs, les blocs occupés et libres. Dans certains cas, les informations sont étendues à la longueur d'un fichier, son implantation en mémoire, ses date et heure de création, ses date et heure de modification. Pour un fichier aléatoire, la rubrique Subtype affiche la longueur des enregistrements.

CATALOGUE (voir DIRECTORY.)

COMPILATEUR

C'est un traducteur de langage qui a la possibilité de traduire un programme écrit dans un langage évolué (Basic par exemple) en un langage de bas niveau (langage machine par exemple) afin d'obtenir une exécution plus rapide.

CONCATENER

C'est chaîner ou assembler deux chaînes de caractères pour n'en former qu'une, plus longue et contenant tous les caractères des chaînes initiales. Exemple : A$ = "MON" : B$ = " APPLE" : C$ = A$ + B$; où C$ représente la somme des chaînes de caractères A$ + B$. Cette opération s'appelle une concaténation. C$ = "MON APPLE".

CONNECTEUR

Les connecteurs - slots - sont les prises femelles, situées à l'intérieur de la carcasse de l'Apple, permettant de le relier à des périphériques, pour une extension de ses possibilités. Avec la venue sur le marché de l'Apple IIc, cette notion de connecteur a changé. Les slots internes ont totalement disparu pour laisser la place à des prises extérieures. Il est désormais impossible de connecter plus d'un lecteur de disquettes à un IIc - en plus du lecteur incorporé. Est-ce un signe de recul de la technique ? L'avenir nous le dira.

CONTROLEUR

Le contrôleur de disques est une carte interface permettant de relier des unités de disques à l'ordinateur. C'est l'organe intermédiaire qui assure le dialogue entre la "machine" et l'appareil d'extension connecté. Sur la série des Apple IIc, cette notion de contrôleur n'existe plus car il ne possède plus de slots.

CREATE

C'est une commande ProDOS qui permet de créer un fichier Subdirectory sur une disquette, dont le but est d'augmenter la possibilité de sauvegarde du nombre de fichiers. Une table des matières principale, Volume-Directory, permet de stoker 51 noms de fichiers. Cette structure permet aussi une gestion des fichiers par groupe ou genre de programmes ; elle est dite hiérarchisée, ou encore structurée.

CURSEUR

Symbole, matérialisé sur votre écran vidéo, qui vous permet d'avoir un repère constant de son emplacement actuel. Il sert à indiquer à l'opérateur que le système attend des instructions de sa part. C'est l'organe d'interrogation du système.

DEMARRAGE A FROID

C'est le procédé d'amorçage du système après sa mise sous tension. Il s'ensuit une mise en marche des activités de traitement par le chargement d'un système d'exploitation, dans la mémoire centrale pour le DOS, et dans la carte langage pour le ProDOS. Les programmes et variables sont purgés de la mémoire, et toute donnée détruite.

DEMARRAGE A CHAUD

Sous DOS 3.3, c'est le procédé d'amorçage du système, appareil déjà sous tension, en tapant CALL 976 ou 3D0G à partir du Moniteur. Les programmes en mémoire restent préservés. L'Apple IIe permet un démarrage à chaud en appuyant simultanément sur les touches CTRL et RESET.
Sous ProDOS, cette notion de démarrage à chaud a changé, pour devenir un procédé se situant entre un démarrage à chaud et à froid. En tapant 3D0G ou 3D3G à partir du Moniteur, un programme Basic implanté en mémoire centrale reste intact, mais ses variables sont purgées. Les vecteurs de la page 3 contiennent les paramètres suivants:

```
03D0- 4C 00 BE  JMP $BE00
03D3- 4C 00 BE  JMP $BE00
```

C'est chaque fois un démarrage à chaud qui sera effectué, en appelant les vecteurs de WARMSTART par $3D0G ou $3D3G à partir du moniteur.

DIRECTORY

Sous DOS 3.3, le Directory désigne le catalogue de la disquette : il a son point d'entrée à la piste $11, secteur $0F, et occupe 15 secteurs. Il peut contenir un maximum de 105 fichiers (15*7), avec 7 fichiers par secteur.
Sous ProDOS, Directory prend un sens différent : il désigne soit un Volume-Directory, si c'est une table des matières principale, soit un Volume-Subdirectory, si c'est une table des matières auxiliaire. Un Volume-Directory se trouve toujours sauvegardé aux blocs $02 à $05, avec son point d'entrée au bloc $02 - Key-Block. Un Volume-Subdirectory peut se trouver à n'importe quel autre bloc de la disquette, abstraction faite des blocs utilisés par le système. Le contenu d'un Directory peut être tout nom de fichier programme ou Subdirectory, ayant une syntaxe reconnue par le système. Le nom d'un fichier peut être du type fichier de données, ou Subdirectory. Un Volume-Directory peut comporter un maximum de 51 noms de fichiers divers. Un Volume-Subdirectory peut contenir au début 12 noms de fichiers (voir CATALOG).

DISQUETTE

Support magnétique pour stocker des informations. Il existe plusieurs modèles et formats, le plus répandu est le modèle 5 pouces 1/4. A partir de fin 1985, Apple a marqué le pas en sortant un nouveau lecteur de disquettes - Unidisk - au format 3 pouces 1/2, qui sera désormais le nouveau standard.

DOS

D'origine anglo-saxonne, "Disc Operating System" est un programme sytème permettant de gérer les fichiers sur disquettes. Il existe plusieurs versions dont la plus répandue est le DOS 3.3. Le système d'exploitation se trouve implanté sur la disquette, aux pistes $00, $01 et $02 et occupe 48 secteurs - 16*3. Il peut définir des disquettes "Master", ou "Slave", suivant que les secteurs $0A et $0B de la piste $00 contiennent ou non le "Relocator" du DOS (voir RELOCATOR).

ECRAN

C'est un périphérique du micro-ordinateur. Appelé tantôt moniteur (ne pas confondre avec la ROM Moniteur), tantôt écran vidéo, il peut être noir et blanc, monochrome ou en couleur. Il est caractérisé par sa bande passante, qui dénote un critère de choix et de qualité de la restitution des points - pixels - en graphique haute résolution. La qualité de l'écriture en dépend également. Un écran monochrome de bonne qualité doit avoir une bande passante de 18 mégahertz , si vous travaillez avec une carte 80 colonnes. Pour la couleur, cette valeur doit atteindre les 25 mégahertz , ce qui fera par la même occasion grimper le prix.

ECRIRE

C'est transférer des données, à partir de l'unité centrale, vers un organe de sortie : un lecteur de disquettes par exemple. Terme anglais équivalent : WRITE.

EDITER

C'est changer, modifier, insérer, effacer ou déplacer du texte dans un document. Cela se fait généralement avec un logiciel de traitement de texte, comme APPLEWRITER.

ENTREE

C'est une information transitée vers un ordinateur et provenant d'une source externe telle que clavier, lecteur de disquettes, etc.

ENTREE FICHIER

Dans une table des matières - principale ou auxiliaire - ce terme désigne le début de la sauvegarde des caractéristiques d'un fichier - 39 bytes réservés par fichier. La position du premier byte dans chaque entrée d'un fichier est donnée par un pointeur de valeur relative $00. Chaque bloc d'une table des matières peut stocker 13 noms de fichiers avec ses paramètres. Le bloc $02 - Key-Block - contient déjà le nom du Volume. ProDOS

attribue à chaque nom de fichier 39 bytes qui renseignent le système sur ses caractéristiques : le byte d'accès, la longueur ou encore l'adresse d'implantation en mémoire centrale. Le chapitre 3 traite le sujet et décrit un exemple de chaque type d'entrée.

EN-TETE

Désigne le début ou le point d'entrée dans une table des matières de la disquette : Volume-Directory ou Volume-Subdirectory (voir EN-TETE, VOLUME-DIRECTORY, VOLUME-SUBDIRECTORY, ou ENTREE FICHIER).

EN-TETE DU VOLUME-DIRECTORY

Désigne le point d'entrée de la table des matières principale avec ProDOS : le nom du Volume-Directory est le premier nom de Volume qui figure sur la disquette après le formatage. Le premier byte de l'entrée du nom d'un Volume-Directory se trouve à la 5^{e} position du bloc $02 de la disquette - Key-Block. Le nom du Volume occupe l'espace des bytes $05 à $13 (05 à 19) avec un maximum de 15 caractères autorisés.

EN-TETE DU VOLUME-SUBDIRECTORY

Désigne le point d'entrée de la table des matières auxiliaire avec ProDOS : le nom du Volume-Subdirectory est le premier nom de Volume auxiliaire, qui figure dans un Subdirectory. Le Volume-Subdirectory est créé par la commande CREATE. Le premier byte de l'entrée du nom d'un Volume-Subdirectory se trouve à la 5^{e} position du bloc alloué. Le nom du Volume occupe l'espace des bytes $05 à $13 (05 à 19) avec un maximum de 15 caractères autorisés.

ENTREE/SORTIE

Ce sont les moyens de communications et transmissions des données entre le micro-ordinateur et ses périphériques. Les adresses d'entrées/sorties se situent en $Cs00, où "s" représente le numéro du slot.

ESC

C'est la touche du clavier, communément appelée touche d'escapade, syntaxe dérivée du mot "s'échapper". Elle est utilisée pour mettre en œuvre certaines fonctions d'un éditeur de texte ou de tout autre programme. Elle peut être utilisée seule, ou conjointement avec d'autres touches - ESC-A, déplacement vers la droite.

FENETRE

Elle désigne une surface de visualisation précise sur l'écran vidéo. Elle est caractérisée par un haut, un bas, une marge gauche et une largeur. Elle peut être définie dans un programme Basic en modifiant les valeurs des adresses $20, $21, $22 et $23, à l'aide de POKE.

FICHIER

Contient des informations enregistrées sous un nom donné, sur un support périphérique tel qu'un lecteur de disquettes entre autres. Il existe divers types de fichiers, dont les plus connus sont :
- fichiers binaires : informations non transcrites (programmes binaires) ;
- fichiers aléatoires : informations ayant des enregistrements de taille fixe, directement accessibles ;
- fichiers séquentiels : informations ayant une longueur variable, et accessibles de façon séquentielle ;
- fichiers EXEC : lus par AppleSoft, et interprétés comme des instructions tapées au clavier ;
- fichiers AppleSoft : informations écrites dans un langage nommé Basic AppleSoft.

FIRMWARE

C'est un terme qui désigne des circuits électroniques contenant de petits programmes intégrés : ROM, EPROM, PROMs, etc.

FONCTION

Désigne toute instruction utilisant des parenthèses et transmettant des paramètres entre ces parenthèses : ABS (expression) ou ASC (chaînes), etc.

FORMAT

Il décrit la structure et la taille de certaines données.

FORMATER

Procédé qui consiste à diviser la surface d'un support de sauvegarde, en y plaçant un certain nombre d'informations pour pouvoir lire ou écrire des données par la suite (voir INITIALISER).

HEXADECIMAL

Système de numération à base 16, dans lequel les nombres sont représentés par des chiffres de 0 à 9 et des lettres de A à F. La numération hexadécimale est plus simple à comprendre que le binaire. Le système usuel que nous employons est de base 10.

IMPRIMANTE

C'est un périphérique de sortie qui permet de lister des données ou des programmes. Il existe une multitude de modèles ayant chacun des caractéristiques bien précises.
- imprimante matricielle ou à aiguilles : elle représente les caractères par des points, suivant une matrice ;
- imprimante à marguerite : de qualité courrier, elle comporte une boule qui supporte les caractères (marguerite) ;

- imprimante thermique : elle fonctionne par l'application de petits points de chaleur sur du papier spécial. Ces imprimantes peuvent être commandées par des interfaces parallèles ou séries.

INDEX BLOCKS

ProDOS alloue un bloc index - Index Block - à un fichier lorsque celui-ci dépasse la taille de 512 bytes - 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.

INITIALISER

C'est préparer une disquette vierge pour pouvoir, par la suite, stocker des informations en divisant sa surface en pistes et secteurs. Cette opération se retrouve encore sous le terme "formater".

INSTRUCTION

Unité d'un programme dans un langage donné, qui correspond à une seule action exécutable par le processeur ; par exemple, PRINT est une instruction.

INTERFACE

C'est un organe électronique qui autorise le dialogue d'un organe donné avec un autre, dans le système déterminé. Il permet la communication interactive entre les différents éléments constituant un ensemble - interface imprimante assurant la liaison entre le système et l'imprimante. Il peut être parallèle ; plusieurs bits, 8 pour l'Apple II, sont alors transmis simultanément. Par contre, s'il est série, les informations sont transmises de façon séquentielle bit par bit.

INTERPRETEUR

Traducteur de langage qui a la propriété de lire un programme dans un langage donné - évolué -¡ puis de le rendre compréhensible pour le microprocesseur. Le langage machine n'a pas besoin d'interpréteur.

LECTEUR DE DISQUES

Organe périphérique qui a la propriété de lire et d'écrire des informations sur le support magnétique qu'est la disquette. C'est un organe qui est utilisé en entrée et en sortie.

LIRE

Transférer des informations, venant d'une source extérieure, vers la mémoire de l'ordinateur (lecteur de disquettes), ou vers le processeur (clavier).

LOADER

Ne pas confondre avec l'action de charger en mémoire centrale une information à partir d'un support de sauvegarde.
C'est un programme d'aide pour charger en mémoire : le fichier SOS avec un Apple III, ou le fichier PRODOS avec un Apple II. Le Loader charge le fichier PRODOS à l'adresse $2000, puis le déplace dans la carte langage par un sous-programme Move - le Relocator est associé au fichier PRODOS. Le système se loge dans la bank 1 de la carte langage, à partir de $D000 et installe un programme Reboot dans la bank 2, à partir de $D100. Si la configuration possède une carte d'extension 64 Ko (Apple IIe) ou 64 Ko de mémoire auxiliaire incorporée (Apple IIc), une RAM Disk-Driver est installée (voir RAM Disk-Driver). Le Loader se trouve sur les secteurs $00-$03 de la piste $00 - Blocs 0-1.

LOGICIEL

Ce sont des éléments d'un système informatique appelés programmes qui ont la faculté d'activer les fonctions de l'ordinateur. Il existe toute une gamme très variée de logiciels dans la série Apple II : de l'application aux tableurs et utilitaires les plus divers.

MASTER

Disquette "maître", qui contient le sous-programme Relocator : il reloge le système à son adresse définitive, en haut des 48 Ko, capacité mémoire de 48 Ko (voir aussi RELOCATOR et SLAVE).

MASTER BLOCK

C'est un bloc répertoire principal alloué par ProDOS : il pointe d'autres Blocs Index. Un Master Block est nécessaire lorsque le nombre de blocs de données dépasse la valeur 256.

Structure type :

- 1 bloc index principal peut pointer un maximum de 128 blocs index ;
- chaque bloc index peut pointer à nouveau 256 blocs de données.

MLI

Machine Language Interface : sigle qui signifie langage machine interface. Ce sont les routines du fichier PRODOS : elles occupent l'espace mémoire des adresses $D000-$EFFF de la bank 1 de la carte langage. Pour appeler un module MLI, il faut passer par la page globale de PRODOS - adresses $BF00 à $BFFF.

MODEM

Dérivé du terme MOdulateur/DEModulateur, il désigne un périphérique effectuant la liaison entre un ou plusieurs micro-ordinateurs, au moyen du réseau téléphonique. En France, le modem doit être agréé par les PTT pour pouvoir être raccordé au réseau téléphonique.

NIBBLE

Il faut distinguer les nibbles bruts des nibbles tout court.
Nibbles bruts : ils comportent une masse de données dont le but est d'aider le système d'exploitation dans son travail. Ce sont des marques et des repères pour tous ceux qui travaillent avec des éditeurs de disquettes, genre DISKFIXER, ou autre.
Les autres types de nibbles sont l'équivalent de 4 bits : 2 nibbles = 1 byte.

PATHNAME

Désigne le chemin d'accès à un fichier quelconque. Il peut être partiel ou complet : dans le premier cas, il sera associé à un préfixe, tandis que dans le deuxième cas, il faudra donner le chemin complet pour avoir accès à un fichier.

OCTET

Unité d'informations composée d'un nombre fixe de bits. Sur la série Apple II, un octet contient 8 bits et peut mémoriser toute valeur comprise entre 0 et 255.

PERIPHERIQUE

Tout organe extérieur d'un système, qu'il s'agisse d'un élément physique (lecteur, imprimante, etc.) ou d'un élément logigue (carte interface). Pour l'Apple IIc, les interfaces pour les périphériques courants sont intégrées.

PISTE

Elément de la surface d'enregistrement d'une disquette qui consiste en une bande concentrique, située à une distance bien déterminée du centre. Cette bande n'est pas visible et ne forme pas un sillon comme sur un disque de phonographe. Sous DOS 3.3 ou ProDOS, la disquette comporte 35 pistes ; cependant les notions de pistes et secteurs ne sont plus les mêmes pour les deux systèmes.
ProDOS gère la disquette par l'intermédiaire de blocs logiques, avec au total 280 blocs par disquette 5 pouces 1/4. Un bloc se compose de 2 secteurs, et 16 secteurs forment 1 piste sur un disque.
Le DOS 3.3 gère la disquette avec des secteurs et pistes physiques, dont les valeurs sont respectivement 16 secteurs par piste, pour un total de 35 pistes.
La capacité en nombre de bytes reste la même pour les deux systèmes, mais l'organisation interne du Directory et de la table des matières a totalement changé.

POINTEUR

C'est un nombre mathématique qui désigne un emplacement de l'espace mémoire. Tout pointeur est une adresse, mais toute adresse n'est pas forcément appelée pointeur. Une adresse n'est un pointeur que si elle pointe un certain type de données, ou vers une information structurée (exemple: le pointeur de pile pointe la pile qui se trouve en page $01).

PREFIX

C'est la commande PRODOS permettant de fixer le préfixe qu'il faudra ajouter avant le nom d'un fichier. Il évite de fournir le nom complet du chemin - Pathname - lors de la formulation d'une instruction relative à un fichier. C'est la première partie d'un nom d'accès stocké en mémoire centrale.

PROCESSEUR

C'est le cerveau de l'ordinateur qui ne serait sans lui qu'un élément inerte. Il effectue les calculs nécessaires, en exécutant des instructions, dont le mode d'emploi en langage machine se trouve stocké dans la mémoire centrale.

ProDOS

C'est le nouveau système d'exploitation professionnel, écrit par Apple COMPUTER. ProDOS signifie : Professional Disk Operating System, ou système d'exploitation professionnel de disquettes. Il regroupe deux fichiers: PRODOS, qui est le système d'exploitation, et BASIC.SYSTEM, qui est une interface entre le système et le langage Basic. Toute commande non reconnue par PRODOS sera renvoyée au système Basic qui la reconnaîtra ou émettra un code d'erreur. Le système d'exploitation ne se trouve plus sauvegardé sur les premières pistes de la disquette, comme le DOS 3.3, mais enregistré sous forme d'un fichier binaire du type SYS, uniquement "bootable". Ce système a été conçu et écrit au départ pour un disque dur du type Profile, d'une capacité de 32 mégabytes. Pour les besoins de la technique, il a été adapté à l'Apple III sous la forme du système SOS, puis à l'Apple II sous PRODOS. La version nécessite une capacité mémoire minimale de 64 Ko, et la carte langage sur les anciens modèles - Apple II+.

PRODOS

Fichier binaire du type SYS, qui contient les commandes MLI - Machine Language Interface. Il regroupe 26 routines qui sont des sous-programmes écrits en langage machine, permettant la gestion des fichiers sur disquettes. Il autorise la gestion d'un disque dur, jusqu'à une capacité de 32 mégabytes. Un fichier peut avoir une taille maximale de 16 mégabytes.

PROGRAMME

C'est une série d'instructions qui décrit les actions que l'ordinateur doit exécuter afin de réaliser un certain travail. Un programme doit être écrit dans un langage de programmation spécifique pour être conforme à l'interpréteur.

PROGRAMMER

Consiste à écrire un programme en ayant soin de le rédiger en un langage adapté au système que l'on possède.

PUCE

Semi-conducteur sur lequel est réalisé un circuit intégré. Normalement, le mot "puce" ne désigne que le semi-conducteur (silicium) ; mais par déformation de langage, la dénomination s'est étendue au circuit intégré, ou même à un composant quelconque.

RAM

Random Access Memory : signifie mémoire à accès sélectif ou mémoire vive. On peut y lire, écrire, transférer des données et les déplacer. Une coupure de courant, ou l'extinction de votre Apple, fera disparaître le programme initialement stocké.

RAM DISK-DRIVER

Cette structure est mise en place après le boot du système d'exploitation ProDOS, avec une configuration de 64 Ko de mémoire auxiliaire. Il sera alors possible d'adresser le slot 3, drive 2, comme un lecteur de disquettes passif - disque virtuel. Une partie du dispositif de commande se trouvera dans les 64 Ko de la carte principale, aux adresses $FF00-$FF7E, et l'autre partie, dans la carte auxiliaire, aux adresses $0200-$03FD. Il sera possible de traiter un certain nombre de fichiers, sans dépasser un maximum de 120 blocs (61440 bytes ou 60 Ko). L'emplacement de ces 120 blocs se situe aux adresses $1000 à $FFFF (mémoire auxiliaire). La Volume Bit-Map se trouve aux adresses $0C00-$0DFF et le Directory-Block aux adresses $0E00-$0FFF (mémoire auxiliaire).

RELOCATOR

Avec le DOS 3.3, Relocator désigne un sous-programme associé à une disquette "maître", qui permet de reloger le système à sa place définitive, aux adresses $9D00-$BFFF - pour un 48 Ko. Le sous-programme translateur se trouve sur la piste $00, secteur $0A et $0B (MEV : $1B00-$1D00). N'existe pas sur une disquette "Slave".
Avec PRODOS, Relocator désigne un sous-programme associé à certaines routines MLI, pour les reloger à leurs adresses définitives. Le fichier PRODOS est relogé dans la carte langage, à partir de l'adresse $2000, après le boot de la disquette.

ROM

Read-Only Memory : signifie mémoire morte, où l'on ne peut que lire des données et où l'écriture est impossible. Les informations y sont stockées lors de la fabrication et y restent en permanence, même lors d'une coupure de courant.

RWTB

Read Write Track Block : c'est la routine d'écriture et de lecture d'un bloc, sur une disquette du système PRODOS. Cette expression est dérivée du sigle RWTS - Read Write Track Sector.

RWTS

Read Write Track System est une routine du DOS, chargée de lire et d'écrire les pistes et les secteurs. Sert également de filtre dans le sens passage des données de la disquette vers la mémoire centrale, il élimine des repères et données devenus inutiles et qui se trouvent sur la disquette sous forme de nibbles bruts.

SAUVEGARDER

C'est transférer des données de l'unité centrale vers un organe de mémoire périphérique, afin d'avoir la possibilité de les réutiliser.

SECTEUR

C'est une portion de piste se trouvant sur une disquette. Pour le DOS 3.3 ou PRODOS, un secteur a une capacité de 256 octets, et chaque piste est divisée en 16 secteurs. Un disque de 35 pistes comporte 560 secteurs. ProDOS n'utilise plus la notion de secteur (voir BLOC).

SLAVE

C'est une disquette "esclave", mot tiré de la traduction d'origine du mot anglais Slave. C'est une disquette dépourvue du sous-programme Relocator et initialisée, sur une configuration donnée, par la commande du DOS "INIT". Si la disquette est "bootée" sur un Apple d'une capacité mémoire supérieure, la mémoire au-delà sera bloquée et inutilisée par le système. HIMEM sera fixé d'après la configuration où la disquette aura été initialisée (voir aussi RELOCATOR).

SOUS-CATALOGUE (voir SUBDIRECTORY)

SUBDIRECTORY

C'est une table des matières auxiliaire pointée par un fichier Subdirectory, sauvegardé dans un Directory parent. Le contenu d'un Volume-Subdirectory est uniquement accessible en indiquant le chemin complet, ou en déclarant au préalable le préfixe. Un Subdirectory ne peut être créé que par la commande CREATE du système PRODOS.

SYSTEME

Ensemble des parties connectées entre elles pour former un tout actif qui peut fournir un certain travail.

SYSTEME D'EXPLOITATION

C'est un programme mis au point pour traiter plus spécialement certains périphériques de votre Apple. La fonction essentielle d'un tel système réside dans le traitement des fichiers par l'intermédiaire d'un lecteur de disquettes. Il interprète les commandes pour effectuer toutes les opérations qui seraient fastidieuses pour l'opérateur. Les systèmes les plus répandus sont : ProDOS, DOS 3.3 et PASCAL (voir DOS 3.3 ou ProDOS).

TRANSLATEUR (voir RELOCATOR)

VBM

Volume Bit-Map : ne pas confondre avec la System Bit-Map de PRODOS. C'est un bloc alloué à un Volume qui désigne l'occupation de la disquette. 35 bytes sont nécessaires pour la VBM d'une disquette standard de 35 pistes.

VOLUME

Le DOS 3.3 n'utilise guère la notion de Volume d'une disquette. Désigne un numéro de Volume, ou de disquette, et prend la valeur 254 par défaut, lors de la commande INIT. La commande CATALOG affiche le numéro de Volume d'une disquette, en haut et à gauche de l'écran.
Sous ProDOS, Volume est synonyme de disquette : le système voit le support de sauvegarde comme un Volume. Celui-ci se divise en blocs où l'on peut stocker un certain nombre de données, bytes. Le nom du Volume est sauvegardé dans le premier bloc du Volume-Directory - Key-Block, bloc 0002 - lors de la phase du formatage et occupe ainsi la première entrée. Le nombre de caractères ne devra pas excéder 15, en suivant une syntaxe particulière :
- commencer par une barre oblique, et être suivi d'une lettre ;
- être constitué de lettres, chiffres, ou points ;
- ne pas comporter d'espaces ou de caractères de ponctuation autre que le point ;
- ne pas dépasser 15 caractères au total, barre oblique non comprise.

VOLUME-DIRECTORY

C'est l'emplacement de la table des matières principale d'une disquette ProDOS. Le Volume-Directory est disponible par défaut, après le formatage de la disquette : il occupe les blocs $02 à $05. Il peut contenir 51 noms de fichiers qui peuvent se diviser en noms de fichiers programmes ou fichiers Directory. En tête du Volume-Directory se trouve stocké le nom du Volume qui est en fait le nom par défaut de la disquette. Le nom du Volume est attribué à une disquette par l'opération du formatage.

VOLUME-SUBDIRECTORY

C'est l'emplacement d'une table des matières auxiliaire, créé par la commande CREATE ; il peut se trouver à un endroit quelconque de la disquette. Le nom du fichier qui le pointe se trouve dans un Directory qui est le parent du Subdirectory. Il pourra à son tour contenir des fichiers programmes, ou Directory. En tête d'un Volume-Subdirectory se trouve stocké le nom du Volume du Subdirectory ; il lui est attribué lors de sa création, par le fichier qui le désigne.

VIDEO INVERSE

Affichage de données sur l'écran de visualisation, sous forme de points noirs sur fond blanc, l'affichage normal étant blanc sur fond noir.

CONSEILS DE LECTURE

Pour faire de la programmation de haut niveau en Basic ou en assembleur, et maîtriser le système de l'Apple IIe 6502 ou 65C02, et de l'Apple IIc, P.S.I. vous propose une palette d'ouvrages utiles.

Pour mieux programmer en Basic AppleSoft

☐ **Techniques de programmation sur Apple II** - René Belle (Editions du P.S.I.)

Pour programmer avec efficacité, voici des routines Basic d'édition, de tri, de recherche..., réutilisables et adaptables au contexte personnel de l'utilisateur.

☐ **La pratique de l'Apple II, tome 2** - Nicole Bréaud-Pouliquen (Editions du P.S.I.)

Ouvrage consacré au système d'exploitation disque, à la gestion des fichiers, à la carte Apple-clock, etc.

☐ **Basic plus, 80 routines sur Apple II** - Michel Martin (Editions du P.S.I.)

Pour pousser votre Apple au maximum de ses capacités : 80 routines de simulation d'instructions qui n'existent pas en Basic AppleSoft.

Pour programmer avec efficacité en assembleur et en langage machine

☐ **Assembleur de l'Apple II 6502 et 65C02** - Nicole Bréaud-Pouliquen et Daniel-Jean David (Editions du P.S.I.)

Une très bonne initiation à l'assembleur de l'Apple II sous DOS 3.3 et ProDOS. Tous les programmes proposés dans le livre sont assemblés sous Procode.